DISEÑO ARQUITECTURAL E IMPLEMENTACIÓN DE LAS FUNCIONALIDADES TRANSVERSALES COMUNES A LAS APLICACIONES WEB RSI BASADAS EN MICROSERVICIOS

ARLEY DAVID VELASCO BASTO - MARIO ANDRES ANAYA MERCHAN

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2023

DISEÑO ARQUITECTURAL E IMPLEMENTACIÓN DE LAS FUNCIONALIDADES TRANSVERSALES COMUNES A LAS APLICACIONES WEB RSI BASADAS EN MICROSERVICIOS

ARLEY DAVID VELASCO BASTO - MARIO ANDRES ANAYA MERCHAN

Trabajo de Grado para optar al título de
Ingeniero de Sistemas

Director:
GABRIEL RODRIGO PEDRAZA FERREIRA
Doctor en Ciencias de la Computación

Codirector:
DANNY FELIPE VERGEL PABA
Especialista en Gerencia de Proyectos

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2023

DEDICATORIA

A nuestra familia, sin ellos nada de esto hubiese sido posible.

AGRADECIMIENTOS

Primeramente a Dios, quien hizo que todo esto fuese posible.

A nuestros familiares y seres más cercanos que nos estuvieron brindando su cariño durante el desarrollo de toda nuestra etapa académica.

Al ingeniero Danny Felipe Vergel por todo su apoyo en la investigación y realización del proyecto de grado, además del compromiso y ayuda en nuestro desarrollo profesional.

Al profesor Gilberto Diaz, que con su contribución lidera el proyecto RSI y vela para que el desarrollo sea lo más óptimo y confiable, además de brindarnos la confianza y la oportunidad de contribuir en el.

Al profesor Gabriel Pedraza quien nos ha acompañado y orientado durante toda la etapa del proyecto de grado.

A nuestros profesores de Ingeniería de Sistemas por todo el apoyo y enseñanza durante el desarrollo del pregrado académico.

CONTENIDO

pág.

LISTA DE FIGURAS

pág.

RESUMEN

TÍTULO: DISEÑO ARQUITECTURAL E IMPLEMENTACIÓN DE LAS FUNCIONALIDADES TRANSVERSALES COMUNES A LAS APLICACIONES WEB RSI BASADAS EN MICROSERVICIOS [*]

AUTOR: ARLEY DAVID VELASCO BASTO - MARIO ANDRES ANAYA MERCHAN [**]

PALABRAS CLAVE: Flexibilidad, mantenibilidad, escalabilidad (sistema software).

DESCRIPCIÓN:

El objetivo del proyecto RSI es renovar los sistemas actuales de la universidad, que llevan mucho tiempo en uso y utilizan tecnologías en desuso, como Informix de IBM y Java 5, principalmente porque ahora se dispone de tecnologías mejores. Para conseguirlo, se propone la implantación de nuevas y mejores tecnologías, y se ha propuesto una arquitectura basada en microservicios como base para el desarrollo de servicios web. Esta arquitectura permite que los microservicios compartan características no funcionales y trabajen conjuntamente como software para dar soporte a los procesos requeridos por la universidad. Se considera esencial mantener unos buenos patrones de desarrollo, diseñar estrategias eficaces y garantizar el correcto crecimiento y funcionamiento del software.

El enfoque principal del proyecto es diseñar e implementar las diferentes funcionalidades transversales comunes de los diferentes microservicios backend, lo que garantiza la mantenibilidad, escalabilidad y flexibilidad del software en conjunto. Además, se busca asegurar que el software sea fácilmente mantenible, pueda adaptarse a un mayor volumen de datos y maneje cambios futuros en los requisitos del usuario.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: GABRIEL RODRIGO PEDRAZA FERREIRA. Codirector: DANNY FELIPE VERGEL PABA.

ABSTRACT

TITLE: Architectural design and implementation of cross-cutting functionalities common to microservices-based RSI web applications [*]

AUTHOR: ARLEY DAVID VELASCO BASTO - MARIO ANDRES ANAYA MERCHAN [**]

KEYWORDS: Coded quadratic measurements, phase retrieval, coding masks, deep learning, object classification.

DESCRIPTION:

The RSI project aims to update the university's existing systems, which have been in use for a considerable period and now rely on outdated technologies, such as IBM's Informix and Java 5. The project proposes implementing new and superior technologies, including a microservices-based architecture, as the foundation for web service development. This architecture enables microservices to share non-functional characteristics and collaborate as software to support the university's processes. The project considers it critical to maintain proper development standards, devise effective strategies, and ensure that the software grows and operates correctly.

The main objective of the project is to design and execute the various common transversal functions of the different backend microservices, ensuring the software's maintainability, scalability, and flexibility. Furthermore, the project aims to ensure that the software is easily maintainable, can cope with a larger volume of data, and accommodate future user requirements changes.

---

[*] Bachelor Thesis

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería de Sistemas e Informática. Advisor: GABRIEL RODRIGO PEDRAZA FERREIRA. Co-advisor: DANNY FELIPE VERGEL PABA

INTRODUCCIÓN

La Universidad Industrial de Santander soporta muchos de sus procesos en sus sistemas de información, razón por la cual ha decidido mejorar aspectos críticos en los sistemas actuales como lo son el rendimiento, la usabilidad, la seguridad y la extensibilidad. Las mejoras propuestas implican una actualización tecnológica y arquitectural que permita implementar las propiedades requeridas.

Los sistemas actuales tienen muchos años y manejan tecnologías que ya no se usan, es por esto que requieren una mejora necesaria para soportar los procesos y servicios de la universidad actualmente. En este contexto, el proyecto RSI se enfoca en desarrollar los nuevos sistemas de información de la universidad bajo una arquitectura de microservicios[1] que permita un crecimiento eficiente y rápido, así como la liberación ágil de funcionalidades.

La arquitectura de microservicios es una técnica de diseño de software que se enfoca en la construcción de aplicaciones mediante la división de la funcionalidad en pequeños servicios independientes. Cada servicio se ejecuta en su propio proceso y se comunica con los demás servicios a través de una API[2] en caso de ser necesario. Esta técnica permite que cada servicio sea escalable y desplegable de forma independiente, lo que facilita la integración de nuevas funcionalidades y mejoras sin afectar el resto del sistema.

---

[1] Atlassian. *Microservices architecture*. URL: https://www.atlassian.com/es/microservices/microservices-architecture. (accedido: 12.12.2022).

[2] Wikipedia. *API*. URL: https://en.wikipedia.org/wiki/API. (accedido: 11.01.2023).

Además, para garantizar el éxito del software, es fundamental establecer buenas bases de desarrollo, diseñar estrategias efectivas y mantener patrones de calidad que permitan su correcto crecimiento y funcionamiento. El objetivo principal de este proyecto es diseñar e implementar las funcionalidades comunes de los diferentes microservicios backend, lo que garantiza la mantenibilidad[3], escalabilidad[4] y extensibilidad[4] del software en conjunto.

Para lograr estos objetivos, se ha trabajado en aspectos clave como la seguridad, la comunicación entre microservicios, la auditoría, la gestión de librerías, las buenas prácticas y metodologías de desarrollo, las pruebas de código y de rendimiento, y el desarrollo de configuraciones transversales en las redes de microservicios.

En resumen este proyecto se enfoca en el desarrollo de una arquitectura de software basada en microservicios y en la creación de funcionalidades transversales que servirán como modelo para la implementación de los diferentes microservicios en el proyecto. Esto garantizará una mayor calidad en el software desarrollado, ya que se prioriza la escalabilidad, la extensibilidad y la mantenibilidad del producto.

---

[3] D. Budgen. “Software design: Creating solutions for ill-structured problems”. En: CRC Press, Taylor amp; Francis Group, 2021, pág. 72.

[4] Apiumhub. *Atributos de Calidad de Arquitectura de software*. URL: https://apiumhub.com/es/tech-blog-barcelona/atributos-calidad-arquitectura-software/. (accedido: 12.01.2023).

1. OBJETIVOS

Objetivo general

- Realizar el diseño arquitectónico e implementación de las funcionalidades transversales que mejoren la escalabilidad, el rendimiento y la calidad de las aplicaciones web del proyecto RSI-UIS.

Objetivos específicos

1. Realizar un proceso de vigilancia tecnológica que permita identificar y seleccionar las tecnologías que se adapten a la arquitectura definida y permitan implementarla adecuadamente junto con sus mejoras.

2. Diseñar una arquitectura software que soporte la implementación adecuada de los aspectos transversales y que además exhiba las propiedades de mantenibilidad, flexibilidad y escalabilidad requeridas por el sistema software RSI.

3. Establecer los requerimientos y condiciones en términos de aspectos transversales que deben soportar las diversas aplicaciones que hacen parte de la implementación del sistema software del proyecto RSI.

4. Implementar las tecnologías y desarrollar las funcionalidades transversales con los estándares adecuados para suplir las necesidades comunes a las diferentes aplicaciones del sistema software del proyecto RSI.

5. Validar la arquitectura definida a través de la realización de una implementación de referencia de esta, que pueda ser utilizada por las diferentes aplicaciones del sistema software del proyecto RSI.

2. MARCO DE REFERENCIA

2.1. ARQUITECTURA DE MICROSERVICIOS

La Arquitectura de Microservicios [1] es un enfoque de diseño de software que se basa en la creación de pequeños servicios independientes, que se comunican mediante mecanismos ligeros. En esta arquitectura, cada servicio tiene su propia base de código y puede contar con su propia base de datos, lo que minimiza el acoplamiento entre ellos. Cuando un cliente necesita realizar una petición, se comunica con el microservicio correspondiente y este procesa la solicitud y responde. Además, los microservicios pueden comunicarse entre sí para satisfacer las necesidades del cliente. Este modelo contrasta con la arquitectura cliente-servidor, en la que existe un único servicio que responde a todas las solicitudes, también conocido como monolito.

2.1.1. CARACTERÍSTICAS DE LA ARQUITECTURA DE MICROSERVICIOS La Arquitectura de Microservicios presenta una serie de características que la hacen una opción atractiva para el diseño de software.

- Los servicios son pequeños e independientes, lo que les permite estar acoplados de forma flexible.
- Cada servicio cuenta con su propio código fuente, lo que significa que puede estar desarrollado en diferentes tecnologías.
- La comunicación entre servicios se realiza mediante API's, lo que garantiza una interacción fluida y segura.
- Cada servicio tiene responsabilidades diferentes, lo que permite una mayor modularidad y escalabilidad del sistema.

- La libertad para el despliegue permite que cada servicio pueda estar desplegado de manera diferente, lo que brinda una mayor flexibilidad y tolerancia a fallos.

2.1.2. VENTAJAS DE LA ARQUITECTURA DE MICROSERVICIOS La Arquitectura de Microservicios presenta una serie de ventajas que la hacen una opción atractiva para el diseño de sistemas de software. A continuación, se detallan las principales:

- **Modularidad:** La modularidad es una de las principales ventajas de la Arquitectura de Microservicios. Al tratarse de servicios independientes, cada uno puede ser desarrollado y desplegado de manera independiente, lo que facilita la gestión del sistema y permite una mayor flexibilidad en su escalabilidad.
- **Escalabilidad horizontal:** Otra ventaja importante de la Arquitectura de Microservicios es su capacidad de escalar horizontalmente. Es decir, pueden agregarse tantos módulos como sean necesarios para satisfacer las necesidades del sistema, sin afectar el rendimiento general.
- **Versatilidad:** La Arquitectura de Microservicios permite un desarrollo versátil, ya que cada servicio puede desarrollarse en el lenguaje y tecnología más óptima y que se adapte mejor a las necesidades. Esto permite una mayor eficiencia en el uso de recursos y una mayor flexibilidad en el diseño del sistema.
- **Responsabilidades diferentes:** Cada servicio en la Arquitectura de Microservicios tiene responsabilidades diferentes, lo que permite una mayor modularidad y escalabilidad del sistema.
- **Mantenimiento simple y barato:** Por último, la Arquitectura de Microservicios tiene la ventaja de tener un mantenimiento simple y barato. Si se necesita

modificar una parte del sistema, basta con ir hasta el módulo que contiene la funcionalidad y no hay necesidad de modificar todo el sistema, lo que se traduce en un menor costo y tiempo de mantenimiento.

2.1.3. DESVENTAJAS DE LA ARQUITECTURA DE MICROSERVICIOS Aunque la arquitectura de microservicios presenta múltiples ventajas, también es importante tener en cuenta sus desventajas. A continuación, se presentan algunas de las desventajas más relevantes de esta arquitectura.

- Se consumen más recursos de CPU y memoria debido a que la aplicación se divide en diferentes microservicios, lo que implica asignar recursos específicos a cada uno.
- La implementación de una arquitectura de microservicios implica una inversión de tiempo adicional para definir de manera clara la estructura de la aplicación en las diferentes partes.
- Gestionar todo el sistema se vuelve más complejo al ser diferentes servicios, lo que dificulta el despliegue y el control de cada uno.
- Es más difícil realizar pruebas ya que las funcionalidades están distribuidas, lo que dificulta la realización de pruebas y tests globales.
- Tiene un coste de implantación alto debido a los costes de infraestructura y de pruebas necesarias.

2.2. ARQUITECTURA DEL PROYECTO

La siguiente figura ilustra la arquitectura actual del proyecto RSI. Más adelante se describen de manera detallada los conceptos que se mencionan en la figura, así como las tecnologías empleadas.

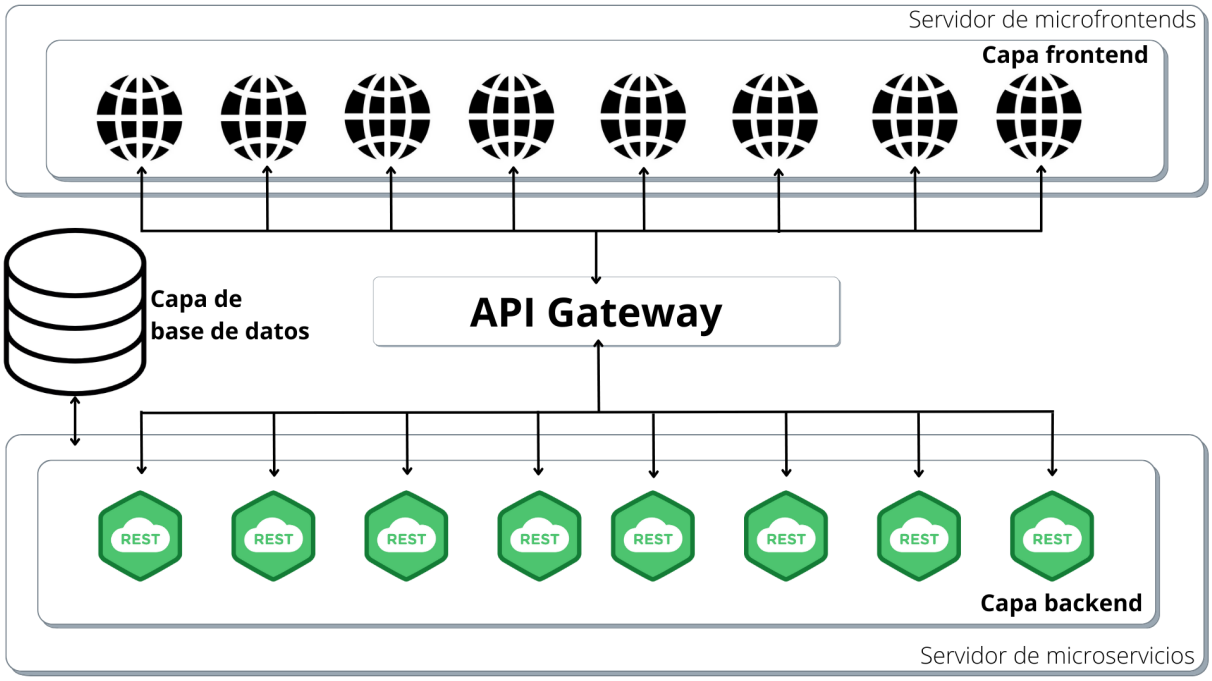

Figura 1. Esquema de arquitectura de microservicios actual en RSI.

2.3. CAPA BACKEND

En la actualidad, el proyecto RSI está enfocado en el desarrollo de microservicios para atender diversas necesidades.

Los microservicios ofrecen la ventaja de ser independientes, lo que les permite utilizar diferentes lenguajes de programación, tecnologías y frameworks[5] según las necesidades particulares de cada microservicio. Aunque en RSI existe una tendencia hacia ciertas tecnologías, como Spring Boot y Angular, no hay una regla absoluta y se utilizan varios lenguajes de programación.

---

[5] Wikipedia. *Framework*. URL: https://es.wikipedia.org/wiki/Framework. (accedido: 10.01.2023).

2.3.1. SERVIDOR DE MICROSERVICIOS El servidor de microservicios utilizado en el proyecto es un servidor de aplicaciones que aloja los archivos war de cada microservicio para su despliegue. Este servidor proporciona un control de los recursos utilizados por las aplicaciones, también ofrece una flexibilidad en el despliegue y configuración de los microservicios.

2.3.1.1. SERVIDOR DE APLICACIONES Un servidor de aplicaciones [6] es una plataforma de software que proporciona servicios y herramientas para el desarrollo, ejecución y gestión de aplicaciones empresariales, incluyendo la gestión de conexiones con bases de datos, la seguridad de las aplicaciones, la gestión de transacciones, la integración de servicios, entre otros.

EL servidor de aplicaciones utilizado por RSI es Wildfly[7]. Es un servidor de aplicaciones robusto y flexible que facilita la construcción de aplicaciones. Se enfoca en la experiencia del usuario y ofrece diversas opciones de configuración. Wildfly permite el control de métricas de cada aplicación, el análisis de conexiones en tiempo real y la configuración de parámetros, entre otras funciones. Además, ofrece diferentes métodos de acceso, como la línea de comandos (CLI), la API REST en formato JSON[8] y la API nativa para Java.

Una de las ventajas de Wildfly es su capacidad de aislamiento, ya que trabaja únicamente con los JARs de nuestras aplicaciones (jar, war, etc.). También cuenta con opciones de configuración de reglas y otros aspectos importantes para el desarrollo de aplicaciones.

---

[6] IBM. *Servidores de aplicaciones*. URL: https://www.ibm.com/docs/es/i/7.2?topic=serving-application-servers. (accedido: 11.02.2023).

[7] WildFly. *Wildfly Docs*. URL: https://docs.wildfly.org/27/. (accedido: 06.01.2023).

[8] Wikipedia. *JSON*. URL: https://es.wikipedia.org/wiki/JSON. (accedido: 14.01.2023).

2.3.2. FRAMEWORK CAPA BACKEND Spring[9] es un framework[5] maduro y robusto de desarrollo de aplicaciones web para Java, con una amplia comunidad detrás que siempre está en constante mejora, brindando soporte y adaptándose conscientemente al ecosistema Java para integrarse muy bien con las necesidades actuales.

El propósito de Spring, como cualquier framework[5], es facilitar el trabajo y ofrecer un marco de trabajo que defina las opciones y formas de solventar nuestras necesidades. Su belleza radica en que se encarga de todas las funciones de bajo nivel, tediosas y complejas, para que la aplicación sea más fácil de iniciar, configurar y desarrollar, y que en esencia solo debamos preocuparnos por la lógica de negocio y aplicar de manera fácil las características que necesitemos para cumplir con el desarrollo de la aplicación web.

Además, contamos con Spring Boot, que es una capa a nivel más alto útil para crear aplicaciones independientes de manera más sencilla, ahorrándonos trabajo desde la creación y ofreciendo muchas características (starters) que vienen con configuraciones establecidas para que el desarrollo, los tests y el despliegue de nuestras aplicaciones sean más rápidos y cómodos.

En conclusión, Spring Boot es una extensión de Spring y las principales diferencias entre ambas son:

- Mientras que en Spring las configuraciones se hacen manualmente en muchos casos, en Spring Boot no es necesario, ya que el propósito es eliminar el código repetitivo.
- Spring Boot ofrece dependencias "starters"que facilitan la adición de dependencias para funcionalidades específicas, como tests, seguridad, etc.

---

[9] Spring. *Spring Framework*. URL: `https://spring.io/projects/spring-framework`. (accedido: 10.01.2023).

- Mientras que Spring requiere una configuración propia para el despliegue, Spring Boot viene con un servidor integrado que facilita el desarrollo, y además viene con su propio contenedor para que el despliegue en servidor sea fácil de configurar.

Por estas razones y ventajas se decidió optar por Spring-boot como una de las tecnologías principales usadas en los microservicios de desarrollo ya que es capaz de suplir multiples necesidades.

2.3.3. SEGURIDAD CAPA BACKEND La seguridad [10] es fundamental en cualquier empresa y debe ser abordada con gran atención y cuidado. Proteger la información y el procesamiento de la misma, garantizando la autorización y autenticación de los usuarios, es esencial para evitar cualquier tipo de acceso no autorizado.
Para lograr una buena seguridad de la información, es necesario considerar algunos pilares clave, como la confidencialidad, que garantiza que la información se mantenga privada y solo se comparta con el usuario o procesos internos correspondientes; la integridad, que se refiere a la consistencia de los datos y su coherencia, lo que asegura una información correcta y evita que los datos sean alterados sin autorización; y la disponibilidad, que asegura que los datos estén disponibles cuando se necesiten.
Desde el nivel de desarrollo de las aplicaciones, es común implementar la protección mediante un login que, generalmente, requiere el ingreso del usuario y su contraseña. Sin embargo, hoy en día existen soluciones de autenticación de doble factor o múltiples factores, que aumentan la seguridad y disminuyen el riesgo de acceso no autorizado. La autenticación de doble factor incluye un segundo factor, además

---

[10] Thalesgroup. *¿Qué es la seguridad del software y por qué es tan importante ahora?* URL: `https://cpl.thalesgroup.com/es/software-monetization/what-is-software-security`. (accedido: 11.02.2023).

del login, que puede ser un SMS, un correo electrónico, una aplicación de autenticación, etc. De esta manera, además de la información que se sabe (contraseña), se requiere poseer algo (teléfono, correo electrónico, etc.) para poder acceder.
Esta visión de seguridad debe ser ampliada bajo algunos conceptos y tecnologías que emplea la capa de backend que es mayormente dominada por Spring-boot.

2.3.3.1. Tecnologías para la seguridad en los microservicios La seguridad es un tema crítico en cualquier aplicación, y Spring Boot ofrece varias herramientas para abordarla. Una de ellas es Spring Security[11], que proporciona funcionalidades para la autenticación[12], autorización[12] y protección contra ataques comunes. Esta dependencia permite la configuración global de la seguridad mediante la anotación de clases de configuración o mediante el uso de @Bean en la clase principal.
En el ámbito de la seguridad con RBAC (Control de Acceso Basado en Roles)[13], se puede modelar la seguridad de la aplicación mediante una base de datos que contenga las definiciones y relaciones entre usuarios, roles, permisos y acciones en lenguaje natural. De esta manera, se pueden definir los accesos de los diferentes usuarios a los servicios ofrecidos por la aplicación, según sus roles y acciones.
Para implementar la seguridad con RBAC, Spring Security ofrece dos opciones: la configuración a nivel de métodos o clases, que es manejada por el desarrollador, y la autenticación y autorización a nivel global. Ambas opciones pueden combinarse

---

[11] Spring. *Spring Security*. URL: https://docs.spring.io/spring-security/reference/index.html. (accedido: 11.02.2023).

[12] Linkedin.alexandrerocha. *Diferencia entre Autenticación y Autorización*. URL: https://es.linkedin.com/pulse/diferencia-entre-autenticaciÃşn-y-autorizaciÃşn-alex-rocha. (accedido: 11.02.2023).

[13] Digital Guide IONOS. *¿Qué es y cómo funciona el RBAC?* URL: https://www.ionos.es/digitalguide/servidores/seguridad/que-es-el-role-based-access-control-rbac/. (accedido: 11.02.2023).

para lograr una configuración más específica o básica, según las necesidades del proyecto.

Otra herramienta que se puede utilizar para la autenticación y autorización de servicios es JWT (JSON Web Tokens)[14](JSON Web Token), un estándar que ofrece una mayor flexibilidad y simplicidad en el manejo de la comunicación entre servicios en aplicaciones web. Los JWT están compuestos por tres partes: el header, el payload y la firma del token, que garantiza la inmutabilidad del token. Es importante destacar que la encriptación [15] y la codificación [15] son técnicas importantes en la seguridad de las aplicaciones, y se utilizan para cifrar datos y transformarlos en otro formato legible pero no visual.

En el contexto de RBAC, la asignación de roles y funciones permite una gestión eficiente y escalable de los permisos de los usuarios. Esto ayuda a mejorar la seguridad de la aplicación y a mantener el control de acceso a los recursos de forma organizada y efectiva. En resumen, Spring Boot ofrece diversas herramientas y opciones para implementar la seguridad en una aplicación, y es importante elegir la mejor opción según las necesidades del proyecto y las características de la aplicación.

2.3.3.2. Lenguaje y framework encargado de la autenticación En el apartado de seguridad, los microservicios que manejan la autenticación y la capa del login no están trabajados en spring boot, esto es una de las ventajas de los microservicios y para el apartado de seguridad se optó por el lenguaje de programación nodejs y el framework express.

---

[14] Auth0.com. *JSON web tokens introduction*. `https://jwt.io/introduction`.

[15] Hackwise. *¿Cuál es la diferencia entre codificación, cifrado y hashing?* URL: `https://hackwise.mx/cual-es-la-diferencia-entre-codificacion-cifrado-y-hashing/#:~:text=La%20codificaciÃ³n%20es%20un%20proceso,una%20cadena%20de%20longitud%20fija.` (accedido: 11.02.2023).

Node.js[16] es un lenguaje de programación basado en javascript que se ejecuta en el lado del servidor, lo que permite que el mismo lenguaje que se utiliza para el desarrollo web en el lado del cliente (navegador) se utilice también en el ecosistema RSI. Debido a que javascript es uno de los lenguajes más utilizados en el desarrollo web, cuenta con una comunidad muy grande y un desarrollo muy amplio de funcionalidades.

En el proyecto RSI, se utiliza el framework[5] express, que es un framework minimalista web para algunos microservicios, y se escogió debido a su comodidad para trabajar con funcionalidades puntuales como aspectos de rutas y seguridad. La elección de Node.js y express se realizó cuidadosamente para garantizar la eficiencia y escalabilidad del sistema, así como para asegurar su mantenibilidad y flexibilidad en el futuro. Además, se establecieron prácticas de desarrollo y metodologías de trabajo efectivas para optimizar el uso de estas tecnologías en el proyecto.

2.3.4. COMUNICACIÓN CON CAPA DE BASE DE DATOS La capa de backend de una aplicación es fundamental para el procesamiento eficiente de todas sus funcionalidades, y para lograrlo, se requiere una comunicación adecuada con la base de datos. Esta comunicación implica no solo el acceso a los datos almacenados, sino también una gestión eficiente de las conexiones y los recursos asociados.

En este sentido, es común utilizar estrategias como el pool de conexiones y la configuración de datasource para optimizar el rendimiento y la eficiencia de la aplicación. Es importante destacar que cuando una aplicación se conecta a una base de datos, se puede establecer una conexión dedicada que implica un acceso único con un usuario y contraseña durante el tiempo necesario. Si bien esta opción es útil para aplicaciones de escritorio con pocos usuarios, no resulta eficiente para aplicaciones

---

[16] Node.js. *Node JS Docs*. URL: `https://nodejs.org/es/docs`. (accedido: 13.01.2023).

web con múltiples usuarios debido al alto costo de abrir y cerrar conexiones. Para resolver este problema, se recurre a una estrategia llamada pool de conexiones, que es un conjunto de conexiones compartidas entre diferentes usuarios para procesar sus solicitudes de manera eficiente.

Los pools de conexiones [6] son una estrategia comúnmente utilizada en aplicaciones web para maximizar el rendimiento optimizando los recursos. El pool se define con un mínimo y un máximo de conexiones, donde el mínimo son las conexiones que el sistema crea al iniciar y mantiene activas para abordar las solicitudes al servidor. Cuando llega una solicitud, el pool le asigna una conexión disponible, y cuando se completa la solicitud, la conexión queda libre para ser usada por otra solicitud. Si se requieren más conexiones y no se ha alcanzado el máximo, se crean nuevas conexiones.

Esto se logra desde el servidor de aplicaciones el cual permite configurar el datasource con los parametros requeridos por la aplicación. En conjunto la administración del datasource proporcionada por el servidor de aplicaciones y la implementación propia basada en HikariCp permiten esta comunicación eficiente a base de datos.

En cuanto al desarrollo, es crucial contar con herramientas que simplifiquen la manipulación de objetos de la base de datos. El uso de un ORM [17] que significa Object-Relational Mapping, es una técnica de programación que nos permite representar las entidades y relaciones de una base de datos en objetos de programación,lo cual resulta muy útil. Esta herramienta acelera el proceso de desarrollo al hacer más fácil la comunicación con la base de datos a través de la manipulación de objetos.

Un ORM permite realizar operaciones en la base de datos de manera indirecta, sin necesidad de escribir código SQL. El ORM se encarga de generar y ejecutar las

---

[17] Deloitte. *¿Qué es un ORM?* URL: `https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html`. (accedido: 11.02.2023).

consultas automáticamente, lo que nos permite enfocarnos en la lógica de negocio y olvidarnos de la sintaxis y complejidad de las consultas SQL.
En Spring Boot, el ORM se compone de dos tecnologías clave: Hibernate y JPA. JPA es una especificación que define reglas y patrones para el manejo de objetos que interactúan con la base de datos, mientras que Hibernate es una de las implementaciones más populares de JPA. Ambas trabajan en conjunto para lograr la correcta manipulación de objetos de base de datos, por lo tanto es una herramienta muy poderosa para el desarrollo de aplicaciones en spring boot.

2.3.5. AUDITORÍA CAPA BACKEND La auditoría [18] es una práctica clave en la seguridad de la información, ya que permite tener un registro completo de los eventos que suceden en nuestra aplicación y detectar cualquier actividad sospechosa o anomalía. La auditoría a nivel de microservicios se logra utilizando tecnologías como entity listener y event listener para capturar los eventos a nivel de entidades y transacciones, y almacenar un registro único por transacción con la información del usuario que origina una petición en el esquema de autorización. Dicha información permite almacenar algunos detalles como información del navegador y sistema operativo del usuario, fecha, hora y otros detalles importantes de la modificación o acción.
Para el desarrollo de aplicaciones, es importante tener un registro detallado de las operaciones que se realizan en la base de datos para garantizar la integridad y seguridad de la información. Para ello, se utilizan tecnologías como Event Listener y Entity Listener.
Un Event Listener [19] nos permite capturar eventos específicos en la aplicación, como

---

[18] HubSpot. *5 pasos para hacer una auditoría de base de datos*. URL: https://blog.hubspot.es/marketing/auditoria-base-de-datos. (accedido: 11.02.2023).

[19] Baeldung. *Spring Events*. URL: https://www.baeldung.com/spring-events. (accedido:

el cierre de una transacción, y realizar acciones en respuesta a ellos. En este caso, se utiliza para registrar la auditoría antes de que se haga el commit de la transacción, lo que nos permite rastrear cambios específicos realizados por un usuario.
Por otro lado, Entity Listener [20] es una tecnología que se encarga de monitorear cualquier cambio persistente en la base de datos en todas las entidades de la aplicación. Esto se logra utilizando anotaciones como @PreUpdate, @PrePersist y @PreRemove. De esta manera, se tiene control inmediato de los cambios que se realizan en la base de datos antes de que sean efectuados.
Ambas tecnologías en conjunto nos permiten mantener un registro detallado de las operaciones realizadas en la base de datos a nivel de microservicio, lo que nos ayuda a garantizar la integridad y seguridad de la información en nuestra aplicación.

2.3.6. LIBRERÍAS COMPARTIDAS CAPA BACKEND El uso de librerías compartidas se refiere a la práctica de crear y utilizar bibliotecas de código compartido entre varios microservicios o aplicaciones de software. En lugar de tener múltiples copias del mismo código en diferentes lugares, se crea una biblioteca de código compartido que puede ser utilizada por varios microservicios o aplicaciones.
Una dependencia [21] es un archivo que un proyecto necesita para poder compilarse, construirse, realizar el test o el run del mismo, o cualquiera de las fases del ciclo de vida. Al correr el build o ejecutar un objetivo de Maven, estas dependencias se resuelven y se cargan del repositorio local. Si no se encuentran allí, se buscan en

---

11.02.2023).

[20] Baeldung. *JPA Entity Lifecycle Events*. URL: https://www.baeldung.com/jpa-entity-lifecycle-events. (accedido: 11.02.2023).

[21] Google Cloud. *Administración de dependencias*. URL: https://cloud.google.com/software-supply-chain-security/docs/dependencies?hl=es-419#:~:text=Una%20dependencia%20de%20software%20es,descargas%20o%20instalas%20tu%20software.. (accedido: 11.02.2023).

el repositorio remoto y se cargan. Luego, las dependencias quedan almacenadas en el archivo war o jar de nuestro proyecto para que nuestra aplicación las utilice y funcione. Además, estas dependencias tienen un ámbito (scope) que especifica en qué momentos son accesibles. Los diferentes ámbitos disponibles son: compile, provided, runtime, test, system e import.

Por otro lado, los perfiles son un concepto importante asociado a Maven que permiten definir configuraciones que se ejecutan dependiendo del perfil. Es decir, un conjunto o sección del POM que es opcional y que, en función del perfil, ejecutará algo específico. Los perfiles pueden ser útiles para manejar configuraciones asociadas a diferentes entornos de nuestra aplicación (por ejemplo, desarrollo, producción, etc.) y también pueden facilitar la realización de pruebas.

2.3.7. COMUNICACIÓN ENTRE MICROSERVICIOS La comunicación entre microservicios es esencial para que una arquitectura de microservicios funcione correctamente, ya que permite que los diferentes servicios se comuniquen entre sí y compartan información, lo que a su vez permite que la aplicación en su conjunto proporcione una funcionalidad más compleja y completa.

2.3.7.1. Tecnología para la comunicación Feign[22] es una herramienta cliente de servicio web declarativo que facilita la comunicación entre microservicios. Esta herramienta permite declarar clientes de servicios web con la información necesaria para invocar servicios de otros microservicios y capturar las respuestas que se obtienen de ellos. Feign es una herramienta propia de Spring Boot que es ampliamente utilizada por aplicaciones como Netflix.

Dada la arquitectura de microservicios presente en el sistema desarrollado por RSI,

---

[22] Spring cloud Netflix. *Declarative REST Client: Feign*. URL: `https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html`. (accedido: 010.01.2023).

resulta imprescindible que exista una comunicación fluida entre los diferentes microservicios. En este contexto, el uso de Feign se convierte en una herramienta fundamental para lograr una comunicación efectiva y eficiente entre los distintos componentes del sistema.

2.4. CAPA FRONTEND

La capa de microfrontends es la parte de la arquitectura que se enfoca en el desarrollo y despliegue de las interfaces de usuario. Esta capa está compuesta por módulos independientes y escalables que trabajan de manera conjunta para construir la interfaz de usuario completa. Cada microfrontend se encarga de una tarea específica en la presentación de la información, y todos juntos conforman la experiencia de usuario completa de la aplicación.

2.4.1. SERVIDOR DE MICROFRONTEND Teniendo claras las tecnologías, la forma de trabajar los microservicios, los microfrontends, es necesario tener claro como estos van a ser administrados para su posterior despliegue, acá es dónde recae la importancia de los servidores, que se encargan de ofrecer dichos servicios.
En el proyecto RSI, se utilizan dos servidores distintos para el despliegue de los microservicios y microfrontends: WildFly como servidor de aplicaciones y Nginx como servidor web, respectivamente.
Un servidor web [23] es un software diseñado para procesar las solicitudes que recibe un cliente a través de los protocolos HTTP/HTTPS. Cuando el cliente solicita recursos, el servidor web los entrega, como por ejemplo una página web con todo su contenido, incluyendo texto, imágenes y videos. Los servidores web son esen-

[23] Webempresa. *¿Qué es un servidor Web?* URL: `https://www.webempresa.com/hosting/que-es-servidor-web.html`. (accedido: 11.02.2023).

ciales para alojar y distribuir contenido en la web, y existen una gran variedad de servidores disponibles, tanto gratuitos como de pago, que pueden ser configurados para distintas necesidades y especificaciones técnicas.
El servidor web utilizado por RSI es Nginx[24], un servidor web de código abierto que se destaca por su alto rendimiento y la capacidad de procesar un gran número de solicitudes. Aunque su enfoque principal es servir contenido estático, también es muy eficiente en el manejo de aplicaciones web y tiene un bajo consumo de memoria. Debido a estas características, Nginx es ampliamente utilizado como servidor web en la industria.

2.4.2. LENGUAJE Y TECNOLOGÍAS FRONTEND En este proyecto, se decidió utilizar el framework Angular en conjunto con el lenguaje de programación typescript para el desarrollo de las interfaces web. La elección de estas tecnologías se debe principalmente a su amplia popularidad en el ámbito de la programación web, así como a la gran cantidad de recursos y herramientas disponibles para su uso.
Además, se consideró que el framework Angular posee características que lo hacen una excelente opción para el desarrollo de aplicaciones web, tales como su robustez, madurez, alto rendimiento, facilidad de mantenimiento, buena documentación y una gran comunidad de desarrolladores detrás. Todo esto contribuye a un proceso de desarrollo más ágil y eficiente, permitiendo ofrecer una experiencia de usuario de calidad en la aplicación final.

---

[24] Nginx. *Nginx Documentation*. URL: `http://nginx.org/en/docs/`. (accedido: 010.01.2023).

2.5. CAPA DE BASE DE DATOS

Al hablar de bases de datos [25], se hace referencia a un conjunto de datos que son almacenados para su posterior uso y gestión. Estos datos pueden estar relacionados con una organización, dominio o negocio específico. Cada sistema de base de datos cuenta con un conjunto de reglas y parámetros que rigen el manejo de los datos, incluyendo la forma en que se relacionan entre sí y el control de las tablas en las que están almacenados.

Existen diferentes modelos de datos y una variedad de operaciones que pueden ser realizadas, incluyendo las operaciones básicas de agregar, editar, eliminar y consultar información.

2.5.1. TECNOLOGÍA DE BASE DE DATOS Hay muchas tecnologías y bases de datos, en particular en el Proyecto RSI se optó por utilizar Oracle Database[26] el cual es un sistema de gestión de base de datos desarrollado por Oracle que se utiliza ampliamente en el mundo empresarial debido a su capacidad para manejar grandes volúmenes de datos. Esta herramienta de gestión de datos es de tipo objeto-relacional, lo que permite almacenar y manipular tanto datos estructurados como no estructurados.

La seguridad es uno de los componentes principales de Oracle Database. Este sistema de gestión de base de datos cuenta con características de seguridad avanzadas como la autenticación de usuarios, la gestión de permisos y el cifrado de datos, lo que garantiza la integridad y confidencialidad de la información almacenada en la

---

[25] Oracle. *¿Qué es una base de datos?* URL: https://www.oracle.com/co/database/what-is-database/. (accedido: 11.02.2023).

[26] Oracle. *Oracle Database Documentation*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/index.html. (accedido: 09.01.2023).

base de datos. Además, Oracle Database ofrece una amplia gama de herramientas de administración de base de datos que permiten una gestión eficiente y confiable de los datos almacenados en ella.

2.5.2. AUDITORIA EN CAPA DE BASE DE DATOS La auditoría consta de dos partes, una de ellas se enfoca en la base de datos y utiliza una tecnología llamada Flashback de Oracle. Esta tecnología permite relacionar las transacciones mediante el ID del commit y rastrear la información modificada analizando las tablas en su versión anterior. De esta manera, se puede contrastar los cambios con la información almacenada en la tabla propia de auditoría y trazar la información cuando sea necesario.

2.5.2.1. Tecnología de base de datos para auditoría Oracle Flashback[27] es una tecnología de Oracle que permite recuperar versiones anteriores de los datos en una base de datos. Permite ver cómo eran los datos en un momento determinado en el pasado, de manera similar a un historico. Es por esto que el flashback de Oracle se puede utilizar para comparar y analizar los cambios realizados en una tabla en diferentes momentos. Esto puede ser útil para detectar errores o inconsistencias en los datos y para auditar los cambios realizados por los usuarios.

---

[27] Oracle. *1.4 About Oracle Flashback Technology*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/bradv/introduction-backup-recovery.html. (accedido: 11.02.2023).

3. MARCO METODOLOGICO

En esta sección se describe la metodología utilizada para el desarrollo del proyecto, incluyendo el flujo de trabajo interno y otras particularidades del mismo. Se detallará el enfoque metodológico utilizado, así como las herramientas y técnicas empleadas para la gestión del proyecto. También se abordarán aspectos relevantes como el sistema de control de versiones utilizado y la realización de pruebas, tanto internas como externas, basadas en código o no.

3.1. FLUJO DE TRABAJO

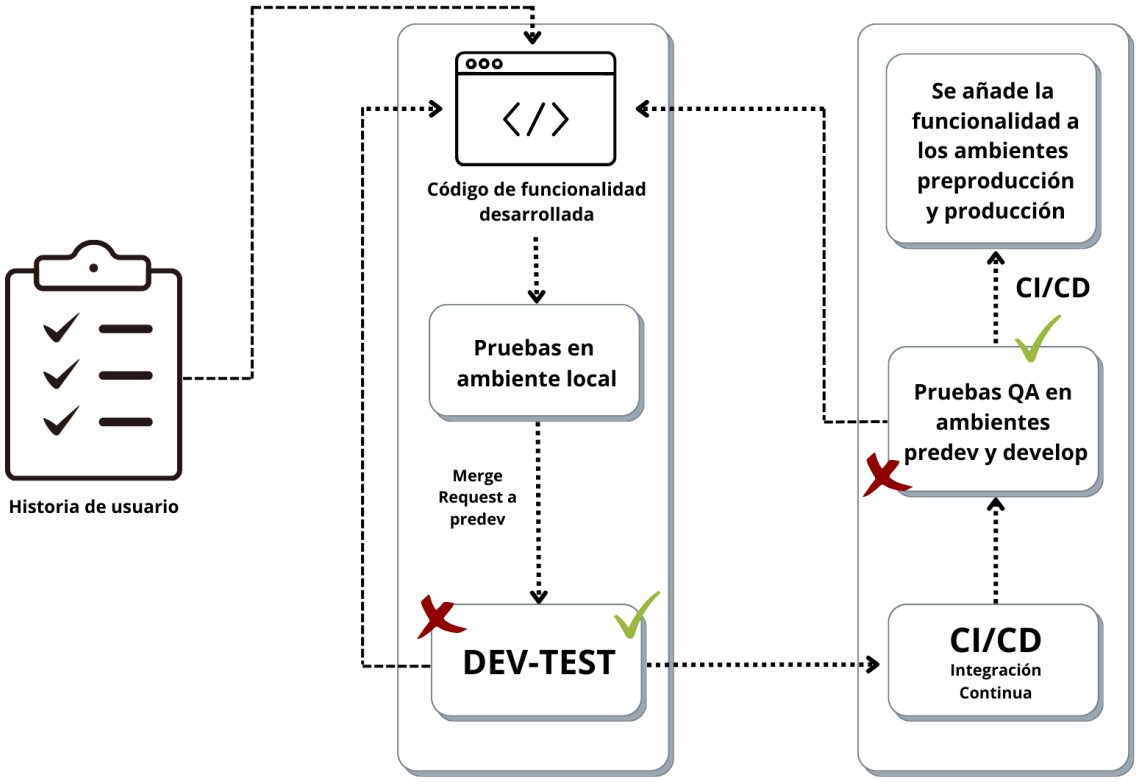

Figura 2. Flujo de trabajo para la adición de funcionalidades en RSI.

Cada vez que se desarrolla una nueva funcionalidad en RSI, es necesario seguir una serie de pasos para garantizar que su integración sea exitosa.
La figura anterior ilustra el proceso de flujo de trabajo en RSI, el cual requiere de algunos conceptos clave para su comprensión adecuada. En primer lugar, se necesita contar con un sistema de control de versiones para cargar y modificar el código, así como repositorios remotos. Además, se utilizan herramientas automáticas de integración y despliegue de código para facilitar el proceso de desarrollo.
Es importante destacar el concepto de dev-test en este proceso, el cual implica la creación de un entorno de pruebas separado del entorno de producción para que los desarrolladores puedan probar la funcionalidad y detectar errores antes de implementar el código en producción. Además, las pruebas de QA son esenciales para verificar y validar la calidad del software y garantizar una experiencia de usuario satisfactoria.
Otro aspecto relevante es la metodología ágil utilizada en el desarrollo de las historias de usuario, lo que permite una gestión más eficiente del proyecto y una mejor colaboración entre los miembros del equipo. Con estos elementos en su lugar, se logra un flujo de trabajo más eficiente y efectivo en RSI.

3.2. SISTEMA DE CONTROL DE VERSIONES

El Sistema de Control de Versiones (SCV) [28] es una herramienta fundamental para la gestión y control de los cambios en los archivos de los diferentes módulos de un proyecto. Su principal función es permitir tener una copia maestra en un repositorio remoto, de tal forma que cada desarrollador pueda tener acceso a la misma y a su vez mantener una copia local en su ordenador para realizar cambios y actualiza-

[28] Atlassian. *¿Qué son las pruebas unitarias y cómo llevar una a cabo?* URL: https://www.atlassian.com/es/git/tutorials/what-is-version-control. (accedido: 11.02.2023).

ciones de forma autónoma. Esto resulta en una mayor eficiencia y optimización del trabajo en equipo, evitando errores en la gestión de los cambios y permitiendo un seguimiento detallado del progreso del proyecto en todo momento.
El mas conocido y popular es Git [29] y en base a este, se utiliza en el proyecto GitLab [30] que es la herramienta web utilizada en el proyecto para el control de versiones y las características Dev-ops, el cual está basado en git, ofrece varias opciones pero las más relevantes son un una integración continua y un despliegue continuo (CI/CD), manejo de test, pipelines y reglas para las integraciones a las ramas protegidas expuestas en el modelo de negocio (ejemplo: ramas desarrollo y producción), todo esto sobre la totalidad de microservicios del proyecto.

3.3. CI/CD

La implementación de CI/CD [31] permite automatizar los diferentes procesos involucrados en el desarrollo de una aplicación. Estas siglas hacen referencia a la Integración Continua (CI) y a la Entrega Continua/Implementación Continua (CD), respectivamente. El propósito de CI/CD es facilitar la integración de nuevo código y evitar problemas en el proceso de implementación.
Para lograr esta automatización, se emplean pipelines que contienen definiciones claras sobre cómo se debe trabajar con el código antes de su despliegue final. Estas definiciones pueden incluir desde la compilación del código, el análisis de calidad, la ejecución de pruebas y la implementación en la rama final del proyecto.

---

[29] Git. *Git Documentation*. URL: https://git-scm.com/docs/git. (accedido: 07.01.2023).

[30] GitLab. *GitLab Docs*. URL: https://docs.gitlab.com/ee/. (accedido: 07.01.2023).

[31] Red Hat. *La integración y la distribución continuas (CI/CD)*. URL: https://www.redhat.com/es/topics/devops/what-is-ci-cd#:~:text=La%20CI%2FCD%20es%20un,distribuciÃşn%20y%20la%20implementaciÃşn%20continuas.. (accedido: 11.02.2023).

3.4. TESTS

Los tests o pruebas de software [32] son procesos fundamentales para verificar el estado de un programa, validar que éste funciona correctamente y garantizar su calidad.

La realización de tests sobre nuestro software es crucial ya que nos permite detectar problemas de manera temprana, facilita la integración de nuevos módulos y comprueba que el sistema sigue funcionando correctamente después de aplicar cambios, actualizaciones o nuevas implementaciones. Además de brindar calidad al software, las pruebas lo hacen más eficiente.

Los principales objetivos de las pruebas de software son:

- Detectar y corregir errores.
- Asegurar la calidad del sistema.
- Detectar errores en cualquier fase del proyecto, tanto presente como futura.
- Ayudar a garantizar los requerimientos del software.
- Existen diferentes tipos de pruebas de software, siendo las principales las pruebas de funcionalidad y las pruebas de rendimiento.

En las pruebas de funcionalidad se verifica cada función del software, ya sea funcionalidad unitaria o integrada. Algunos ejemplos de estas pruebas son:

- Pruebas unitarias.
- Pruebas de integración.

---

[32] IBM. *¿Cómo funcionan las pruebas de software?* URL: `https://www.ibm.com/es-es/topics/software-testing`. (accedido: 11.02.2023).

- Pruebas de sistema.
- Pruebas de interfaz, entre otras.

Por otro lado, en las pruebas de rendimiento se tienen en cuenta aspectos como la confiabilidad, la usabilidad y el rendimiento del software. Algunos ejemplos de estas pruebas son:

- Pruebas de rendimiento.
- Pruebas de carga.
- Pruebas de estrés.
- Pruebas de volumen.
- Pruebas de seguridad, entre otras.

3.4.1. Test Unitarios Las pruebas unitarias [33] son una técnica de testing que se utiliza para comprobar que una unidad de código, generalmente una función o un método, funciona correctamente de forma aislada. Su objetivo es detectar posibles errores en la lógica del código en una etapa temprana del desarrollo, antes de que se integre con el resto del sistema. De esta forma, nos ayuda a garantizar la funcionalidad y detectar errores antes de que se propaguen a otras partes del software.
Entre las ventajas de las pruebas unitarias se encuentran:
Demuestran que la lógica del código está bien y que cumple con todos los casos de prueba. Aumentan la legibilidad del código, ya que obligan a factorizar y modularizar el código. Son rápidas de ejecutar, lo que nos permite ejecutar un gran volumen de ellas en muy poco tiempo. Mejoran la calidad del código y facilitan el mantenimiento

[33] Yeeply. *¿Qué son las pruebas unitarias y cómo llevar una a cabo?* URL: `https://www.yeeply.com/blog/que-son-pruebas-unitarias`. (accedido: 11.02.2023).

a futuro. En resumen, las pruebas unitarias son una técnica de testing esencial para garantizar la calidad del código y detectar errores de forma temprana.

3.5. DEV-TEST

Es común necesitar agregar, modificar o eliminar código para mejorar la funcionalidad del software. Sin embargo, estos cambios pueden afectar el rendimiento y la calidad del proyecto, lo que puede generar problemas en el despliegue del mismo y en la experiencia del usuario final. Para evitar estos problemas, los desarrolladores pueden realizar una prueba conocida como "devtest", que consiste en verificar que los cambios a integrar cumplen con los criterios de calidad establecidos y reducen los riesgos de errores en la integración.

3.6. PRUEBAS QA

Las pruebas de Aseguramiento de la Calidad (QA, por sus siglas en inglés de Quality Assurance) son un conjunto de actividades realizadas durante el proceso de desarrollo de software con el fin de garantizar que el producto final cumpla con los requisitos de calidad y funcionalidad esperados.
Las pruebas QA [34] pueden incluir tanto pruebas manuales como automáticas, y se realizan en diferentes etapas del ciclo de vida del software para detectar y corregir errores o defectos en el código. Algunos ejemplos de pruebas que se pueden realizar en el proceso de QA son pruebas de integración, de regresión, de carga, de usabilidad, de seguridad, entre otras.
El objetivo principal de las pruebas de QA es asegurar que el software cumpla con

---

[34] Inesdi. *¿Qué es un QA testing y cómo se hace?* URL: https://www.inesdi.com/blog/que-es-un-qa-testing/#:~:text=Un%20QA%20testing%20es%20un,se%20correspondan%20con%20lo%20precisado.. (accedido: 11.02.2023).

los requisitos del cliente y con los estándares de calidad esperados, lo que a su vez ayuda a reducir los costos de mantenimiento y aumenta la satisfacción del usuario final.

3.7. METODOLOGÍAS AGILES

Las metodologías ágiles [35] son estrategias y herramientas que permiten a las organizaciones gestionar sus proyectos de manera flexible y rápida. Permiten organizar los flujos de trabajo, dividir los proyectos en partes y adaptarse a los contratiempos que puedan surgir durante el proceso. Además, estas metodologías permiten resolver las etapas de los proyectos en tiempos reducidos.

3.7.1. Scrum El Scrum [36] es una metodología ágil que se basa en un conjunto de buenas prácticas para fomentar la colaboración y el control de las diferentes etapas del desarrollo, así como para el seguimiento de los objetivos establecidos en cada período de tiempo. Esta metodología se caracteriza por su estrategia de desarrollo incremental y su enfoque en equipos auto-dirigidos y auto-organizados. El Scrum se destaca por la celebración de reuniones diarias de no más de 15 minutos para obtener retroalimentación acerca del progreso de las tareas y los obstáculos presentados. Además, esta metodología se compone principalmente de períodos de tiempo llamados "sprints", donde se definen objetivos y metas a alcanzar por el equipo Scrum.

---

[35] Salesforce. *¿Qué son las metodologías ágiles y cómo pueden ayudar a tus equipos de trabajo?* URL: https://www.salesforce.com/mx/blog/2021/12/que-son-metodologias-agiles-y-como-pueden-ayudar-a-tus-equipos-de-trabajo.html#:~:text=Las%20metodologÃŋas%20Ãągiles%20no%20son,proyectos%20con%20rapidez%20y%20flexibilidad.. (accedido: 11.02.2023).

[36] Atlassian. *¿Qué es scrum?* URL: https://www.atlassian.com/es/agile/scrum#:~:text=Â¿QuÃl%20es%20scrum%3F,de%20valores%2C%20principios%20y%20prÃącticas.. (accedido: 11.02.2023).

Los principales roles manejados en Scrum son:

- **Product owner (Propietario del producto)**: Es el encargado de que el equipo Scrum trabaje de forma adecuada acorde a la perspectiva del negocio.
- **Scrum master (facilitador)**: Es el responsable del cumplimiento de las reglas dentro del marco scrum, elimina los obstáculos para el cumplimiento de ellas, y sirve de mediador la mayor parte del tiempo entre el product owner y los desarrolladores.
- **Desarrollador**: Son cada uno de los profesionales encargados de hacer el incremento del producto generado en cada sprint, y son los que constantemente están trabajando en base a historias de usuario.

En Scrum, existen una serie de fases o procesos cíclicos que son característicos de esta metodología y que contribuyen a la flexibilidad y rapidez en el desarrollo del proyecto. La primera fase es la Planificación del Sprint, una reunión en la que se establecen las metas a alcanzar durante el próximo sprint, el cual tiene una duración máxima de 4 semanas. Después de decidir las metas y las historias de usuario que se trabajarán, se lleva a cabo una reunión para evaluar el progreso durante el sprint, y durante el desarrollo del mismo, hay reuniones diarias de no más de 15 minutos, donde cada miembro del equipo aporta su progreso y discute posibles obstáculos. Al finalizar el sprint, se lleva a cabo una reunión llamada Revisión del Sprint, donde se presenta el progreso realizado durante todo el sprint y se muestra la traza de los objetivos alcanzados que se plantearon inicialmente. Finalmente, sigue la Retrospectiva del Sprint, una reunión enfocada en analizar los aspectos positivos y negativos del sprint para mejorar de cara al siguiente.

3.7.2. Historias de Usuario Las historias de usuario [37] son una representación escrita de los requerimientos o funcionalidades del software, y son muy útiles para especificar los requisitos del sistema. En el marco de Scrum, son uno de los pilares fundamentales, ya que permiten la gestión y el avance de funcionalidades dentro del sistema. Las historias de usuario expresan las necesidades del cliente y son abordadas por los equipos de desarrolladores, quienes evalúan la complejidad del trabajo y la desarrollan en consecuencia. En resumen, las historias de usuario son esenciales para garantizar que el software se adapte a las necesidades del cliente y se desarrolle de manera eficiente.

[37] Atlassian. *Historias de usuario con ejemplos y plantilla*. URL: https://www.atlassian.com/es/agile/project-management/user-stories. (accedido: 11.02.2023).

4. FUNCIONALIDADES TRANSVERSALES IMPLEMENTADAS

Para el desarrollo de las funcionalidades del proyecto, se adoptó la metodología ágil Scrum, la cual permitió una gestión eficiente del equipo de desarrollo y una mayor adaptabilidad a los cambios y requerimientos del cliente. Durante las diferentes etapas del proyecto, se identificaron los objetivos relacionados con la configuración transversal del sistema y se definieron herramientas, técnicas y tecnologías para diseñar aspectos no funcionales y arquitecturales, de modo que su implementación lograra una mejora en el producto final.

La calidad del desarrollo no solo se basó en la metodología del equipo, sino que también se llevaron a cabo pruebas exhaustivas para garantizar la funcionalidad y calidad del software desarrollado. Además, se contó con un microservicio conocido como ”training” dedicado a realizar pruebas sin ninguna repercusión en caso de falla, para que dichas funcionalidades pudieran luego ser desplegadas sin problemas.

Para el despliegue del código desarrollado, se contó con dos ambientes principales: predev y develop. En el ambiente de predev, el desarrollador debía subir su código previamente probado en su entorno local y precedido por las pruebas del dev-tester asignado, para que se pudiera probar la funcionalidad por el equipo de pruebas QA. Una vez que el resultado de la prueba por el equipo QA fuese exitoso, se subía este código al ambiente de develop, donde la funcionalidad pasaba por una etapa de maduración y de pruebas adicionales con datos más acordes a la lógica de negocio del aplicativo. Finalmente, una vez que la funcionalidad estaba completamente validada, se agregaba a los ambientes de preproducción y producción.

En esta sección, se expondrá la ejecución del proyecto y los resultados obtenidos a través de las historias de usuario planteadas por el equipo RSI, con el objetivo de abordar los temas transversales relacionados con los microservicios y el software en general.

Además cada implementación cuenta con su objetivo, sus criterios de aceptación que son aspectos obligatorios,y una fase de ejecución que muestra el proceso de desarrollo.

4.1. Aplicar auditoría backend principal a cada uno de los microservicios

4.1.1. Objetivo El objetivo de esta historia de usuario consiste en investigar y evaluar las posibles herramientas o tecnologías que se pueden utilizar para implementar una auditoría en cada uno de los microservicios presentes en el proyecto RSI. Dicha auditoría debe ser capaz de registrar todas las transacciones que se realicen en la base de datos, con el objetivo de tener un control adecuado del correcto funcionamiento de la aplicación.

Para lograr este objetivo, se requiere una auditoría que permita identificar las modificaciones realizadas por los usuarios en la base de datos, lo cual puede llevarse a cabo mediante el uso de la tecnología Flashback en una primera fase a nivel de base de datos. Para complementar esta auditoría, se requerirá una contraparte a nivel de los microservicios, que permita recolectar y registrar información en una tabla específica para poder hacer los cruces entre ambas.

Es fundamental contar con una auditoría completa y eficiente para garantizar la integridad y seguridad de los datos del proyecto RSI, así como para poder detectar posibles errores y anomalías en el sistema. Por lo tanto, se llevará a cabo una investigación exhaustiva de herramientas y tecnologías disponibles en el mercado, para determinar la opción más adecuada que cumpla con los requisitos específicos del proyecto y su implementación.

4.1.2. Criterios de aceptación

- Revisar exhaustivamente cada uno de los microservicios del proyecto RSI, con el objetivo de garantizar que la auditoría se esté realizando correctamente en

todos ellos. Esto implica verificar que se estén registrando todas las transacciones relevantes y que los datos estén siendo almacenados de forma adecuada en las tablas correspondientes.

- Realizar pruebas exhaustivas para validar el correcto funcionamiento de la auditoría en diferentes tipos de transacciones que puedan ocurrir en la base de datos. Se deben comprobar las diferentes operaciones que se realizan sobre la información del sistema, tanto de lectura como de escritura, y verificar que la auditoría esté registrando correctamente todas las acciones realizadas por los usuarios.

- Generar un documento detallado que explique el funcionamiento de la auditoría implementada y los procesos involucrados en su implementación. El documento debe contener información relevante acerca de las herramientas y tecnologías utilizadas, así como una guía paso a paso que permita a otros miembros del equipo comprender cómo se implementó la auditoría en el sistema. Además, se debe incluir información sobre cómo interpretar los datos de la auditoría para poder identificar posibles anomalías y errores en el sistema.

4.1.3. Ejecución Para llevar a cabo el desarrollo de la auditoría se realizó una exhaustiva investigación de las tecnologías Entity listener y Event Listener en Spring, ya que se utilizan Hibernate y JPA como ORM para la comunicación con la base de datos. En este modelo, las entidades son fundamentales para la auditoría y los Listeners permiten tener un control más detallado sobre los cambios en las mismas. Además, se descubrió que la tecnología Event Listener es clave debido a las particularidades del Flashback de Oracle, que requiere una única transacción a nivel de base de datos para que todo quede guardado en un mismo commit.
Una vez investigadas las tecnologías necesarias, se exploró cómo agrupar la lógica sin la necesidad de modificar cada entidad/repositorio. Se encontró que Spring

maneja ciertas configuraciones desde archivos xml, como el archivo orm.xml que permite definir la configuración para el Entity Listener global.

Para la recolección de la información, se identificó qué datos eran necesarios, como la IP del usuario, el navegador, el sistema operativo, etc. La solución para manejar toda esta información de forma atómica y una única vez por transacción fue utilizar otra tecnología auxiliar conocida como MDC. MDC es un almacenamiento clave-valor Thread-Local, lo que significa que estos datos se guardan a nivel del hilo que ejecuta el código y pueden ser usados en cualquier servicio o parte del código que se requiera.

Al tener un almacenamiento global, se pudo manejar el estado de la auditoría para las transacciones, registrando un valor para cada inicio de transacción y limpiándolo al finalizar la misma utilizando los TransactionalEventListener, esto debido a que un endpoint puede llegar a manejar varias transacciones y es necesario realizar la auditoría por cada transacción y no únicamente por cada endpoint.

En la función principal de auditoría, se realizó una verificación del valor de la variable correspondiente para evitar problemas de bucles innecesarios y garantizar una sola inserción de auditoría. De esta manera, se asegura la eficiencia y la exactitud del registro de auditoría.

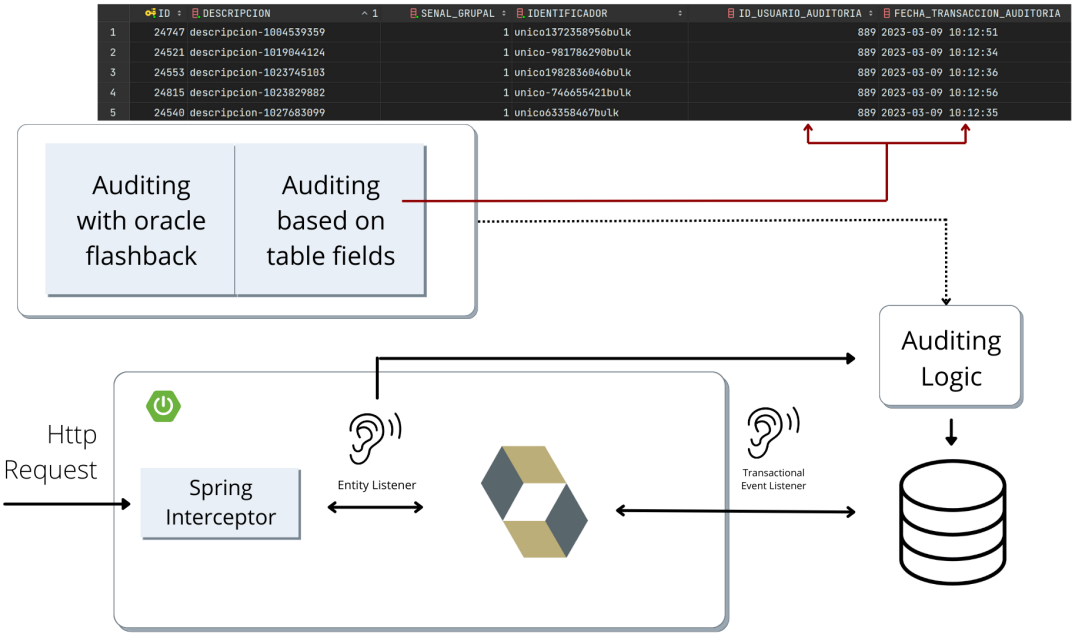

Figura 3. Esquema que ilustra el flujo de la lógica de la auditoría implementada

Durante la fase de pruebas y validación, el equipo se enfocó en realizar pruebas exhaustivas en los ambientes locales y predev. Para ello, el equipo de base de datos proporcionó una vista que permitía verificar que los cambios realizados en la base de datos estuvieran bajo el mismo ID de transacción que la inserción en la auditoría. Se llevaron a cabo múltiples pruebas bajo una amplia variedad de escenarios, incluyendo excepciones, transacciones nativas, uso del EntityManager y múltiples transacciones.

Una vez que las pruebas locales fueron satisfactorias, se migró la implementación al ambiente de predev, desplegado en el servidor de aplicaciones. Sin embargo, se encontró que el comportamiento era diferente en este ambiente. A pesar de mantener la misma lógica, los IDs de transacciones no correspondían, lo que impedía el correcto funcionamiento de la auditoría. Para corregir este problema, se instaló el servidor de aplicaciones en un entorno local y se realizaron pruebas adicionales en el servidor, lo que permitió comprobar que el problema residía en la configuración

del servidor.

Finalmente, se investigó el manejo de datasource por parte del servidor de aplicaciones y la integración con los microservicios en Java mediante la propiedad Jndi, y esto solucionó el problema. Al solventar el problema, se permitió una evaluación correcta de la auditoría en ambos ambientes de pruebas, cumpliendo así con los criterios de aceptación. Una vez comprobado dicho comportamiento, se migró la auditoría a los demás ambientes y se evaluó en producción en los microservicios. Con esto se logró implementar una auditoría en los microservicios de RSI, garantizando un control de los datos que eran modificados y permitiendo un rastreo en los mismos en caso de anomalias encontradas a nivel de base de datos, algo que hasta el momento no existía en el proyecto RSI.

Se adjunta la wiki en la carpeta auditoría para una mayor comprensión del proceso.

4.2. Monitoreo, configuración y testeo de peticiones masivas a fuentes de datos involucrados en proyectos RSI

4.2.1. Objetivo El objetivo de la historia de usuario es abordar unos inconvenientes que surgieron en el proyecto RSI en relación con la auditoría, la cual no estaba funcionando correctamente debido a problemas con la tecnología flashback de Oracle. La lógica de negocio y la lógica de auditoría estaban ubicados en una misma transacción, pero se estaban generando cambios independientes sin respetar dicha comportamiento transaccional. Para solucionar este problema, se propuso cambiar la implementación de las conexiones a una administrada por el servidor de aplicaciones Wildfly, ya que se sospechaba que la falta de configuración del datasource y la mala sincronización de las conexiones estaban alterando el comportamiento final de los microservicios y arruinando la auditoría.

Aunque la solución propuesta al problema de auditoría funcionó, surgió la duda y la necesidad de explorar tecnologías, alternativas y opciones de manejo de conexio-

nes para garantizar un correcto funcionamiento y optimizar el uso de recursos. Se realizaron investigaciones sobre alternativas de manejo de conexiones en Spring Boot, y quedaron condensadas en las siguientes:

- Configuración automática por parámetros (application.properties)
- Configuración manual de datasource
- Wildfly auto-booteable o embebido
- Wildfly datasource y jndi en properties
- Hikaricp (implementación de pool de conexiones)

El objetivo final es explorar y evaluar estas alternativas para determinar cuál es la más eficiente y adecuada para garantizar el correcto funcionamiento de la auditoría y optimizar el uso de los recursos disponibles en el proyecto RSI.

4.2.2. Criterios de aceptación

- Analizar cada una de las cinco opciones de manejo de conexiones en Spring Boot y Wildfly para comprender su funcionamiento, ventajas y desventajas.
- Realizar pruebas exhaustivas con cada una de las opciones para evaluar su desempeño en relación con los criterios de aceptación establecidos.
- Documentar el proceso de implementación, configuración, caracteristicas y parametros para la aplicación de cada una de las alternativas.
- Documentar los resultados de las pruebas, incluyendo el tiempo de respuesta, la estabilidad y el uso de recursos de cada opción.
- Identificar la opción que cumple con todos los criterios de aceptación y proponer su implementación en el proyecto RSI.

- Establecer un plan de implementación y seguimiento para la solución seleccionada, asegurando que se cumplan todos los requisitos y se minimice cualquier impacto negativo en la operación del sistema.
- Proporcionar formación y documentación detallada sobre la solución seleccionada para garantizar que el equipo de desarrollo pueda entender y mantener adecuadamente el código.

4.2.3. Ejecución Durante la ejecución de esta historia de usuario, se exploraron varias alternativas y se documentaron cuidadosamente los aspectos relevantes en la página de documentación del proyecto (wiki). Al evaluar las opciones, se descubrió que la implementación por defecto en Spring Boot 2.x, HikariCP, era una solución óptima y ligera para el manejo de pool de conexiones. Como resultado, se decidió unificar las alternativas de configuración automática por parámetros, configuración manual de datasource y la implementación de HikariCP.

Además, se revisó el tema de la programación orientada a aspectos implicado en anotaciones como @Transactional, @Async, y @Cacheable para interactuar con el código contenido bajo estas anotaciones. Se encontró que era necesario profundizar en el manejo interno de Spring Boot para la transaccionalidad, la inyección de servicios, el uso de proxis y la programación orientada a aspectos para lograr una configuración adecuada que soportara las transacciones correctamente. A pesar de que la implementación de HikariCP resultó ser una muy buena opción, se registraron los resultados de las pruebas para todas las alternativas evaluadas y esta no fue finalmente aplicada en todos los proyectos, únicamente en un proyecto implicado con multiples fuentes de datos.

En cuanto a la alternativa de Wildfly como administrador del pool de conexiones y del datasource, se concluyó que era una opción ideal y óptima para el proyecto RSI, ya que proporciona una configuración sin la necesidad de configuración adicional

para el manejo de conexiones, transacciones y sincronización. Además, se acopla perfectamente al perfil trabajado a lo largo del proyecto, ya que dicho servidor es el encargado de administrar los diferentes microservicios del proyecto. Se llevaron a cabo pruebas de rendimiento y comportamiento para validar esta elección.
Por último, se consideró la alternativa de Wildfly booteable, que genera instancias para cada nuevo despliegue del proyecto y permite que cada vez que se construye el proyecto se genere un servidor embebido para su despliegue, el cual inicializa la configuración del datasource en base a un script en formato CLI para cumplir con las necesidades de auditoría y demás.
Con esto quedó desarrollado a profundidad el analisis de alternativas, otorgando 3 opciones para el manejo eficiente de los recursos relacionados con las conexiones y que permiten un correcto funcionamiento de auditoría entre otros aspectos de rendimiento, además se documentaron todos estos escenarios, junto con los aspectos técnicos, ventajas, desventajas e implementación, para ofrecer opciones futuras para mejoras del proyecto y facilitar la toma de decisiones en caso de cambios.
Se adjunta la wiki en la carpeta Datasources para una mayor comprensión del proceso.

4.3. Desarrollo y aplicación de librería compartida entre microservicios para configuraciones transversales

4.3.1. Objetivo En los proyectos back de RSI, se ha identificado la existencia de un código transversal que se replica en todos los proyectos. Este código común, que no aumenta de forma exponencial en el tiempo, necesita ser unificado en una o varias dependencias para apuntar a una arquitectura limpia y eficiente. Para lograr esto, es necesario generar una librería compartida entre los proyectos que contenga todo el código transversal relacionado con excepciones, logs, auditoría, internacionalización, seguridad, utilería y cualquier otro código que pueda ser alojado allí. Al

centralizar este código común, se evita la duplicación innecesaria de trabajo y se fomenta una mayor eficiencia y calidad en el desarrollo de los proyectos. La creación de esta librería compartida es un paso importante para mejorar la gestión de los proyectos y garantizar la consistencia y coherencia del código en todas las áreas de RSI.

4.3.2. Criterios de aceptación

- Investigar la tecnología necesaria para una correcta integración de la librería con los diferentes microservicios de spring-boot.
- Realizar las correspondientes pruebas en el ambiente de predev.
- Implementar la librería en cada uno de los microservicios.
- Documentar el proceso de la generación de la librería en la wiki RSI.
- Posiblemente esto involucre cambios en la manera en la que se desarrolla, hacer una breve explicación de esto a los compañeros.

4.3.3. Ejecución Para llevar a cabo el desarrollo de la historia, se realizó una investigación previa acerca de las posibles formas de integrar la librería con los microservicios existentes. Dado que el código implicado contenía anotaciones relacionadas con componentes administrados por el contenedor de Spring[9], era necesario generar una librería que pudiera ser consumida por los proyectos principales y que, al mismo tiempo, pudiera administrar estos componentes para asegurar el correcto funcionamiento de las configuraciones transversales.
Después de realizar varias pruebas, se determinó que la forma más factible de generar esta librería era a través de un plugin de Maven[38], una herramienta que permite

[38] Apache Maven Project. *Maven Documentation*. URL: https://maven.apache.org/guides/.

generar proyectos empaquetados en formatos como .jar o .war. Para comenzar, se creó un proyecto Spring Boot[39] con la misma versión que los microservicios existentes, y se incorporó todo el código necesario para la configuración transversal.
A continuación, se configuró el archivo pom.xml del proyecto. En la sección de "build", se llevó a cabo la configuración necesaria para que el plugin de Maven generara un archivo .jar que contuviera únicamente el código necesario para la librería, excluyendo las dependencias. Esto era importante para evitar la duplicación de dependencias cuando el proyecto padre consumiera la librería.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <id>repackage</id>
                    <configuration>
                        <skip>true</skip>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Además de permitir la generación de la librería, el uso de un plugin de Maven tam-

(accedido: 010.01.2023).

[39] Baeldung. *Learn Spring boot*. URL: https://www.baeldung.com/spring-boot. (accedido: 10.01.2023).

bién facilita el versionamiento semántico[40] de la misma. Esto es esencial para generar diferentes versiones que se adapten a los distintos microservicios y garantizar una fácil mantenibilidad en el tiempo.

Con el tiempo, los proyectos pueden requerir cambios en sus configuraciones propias, como la seguridad o la auditoría, y contar con una librería versionada permite realizar dichas adaptaciones de forma sencilla y ordenada. De esta forma, se logra una mayor eficiencia en la gestión de la librería y se simplifica el proceso de mantenimiento de los microservicios.

```
<dependency>
  <groupId>co.edu.uis</groupId>
  <artifactId>library</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
```

Durante el proceso de implementación de una librería para integrar configuraciones transversales en microservicios, surgieron varios imprevistos que requirieron soluciones y adaptaciones para asegurar el correcto funcionamiento de la misma. Uno de estos desafíos fue la configuración de la auditoría, la cual requirió una reestructuración de archivos para asegurar un flujo adecuado de configuración y permitir cambios futuros. Además, fue necesario adaptar la generación de excepciones en los diferentes proyectos y asegurar que los mensajes de error manejados por internacionalización se encontraran en cada microservicio y no en la librería. Esto fue importante para permitir que cada proyecto tuviera la libertad de crear, gestionar y mantener sus propios mensajes de error y mensajes de utilería utilizados en DTO's. Luego de varios procesos de prueba, se logró generar con éxito la primera versión de la librería. Ésta se integró gradualmente en cada uno de los microservicios y se documentó exhaustivamente todo el trabajo realizado. Debido a cambios en la

[40] Semver. *Versionado Semántico*. URL: https://semver.org/lang/es/. (accedido: 11.01.2023).

forma en que cada desarrollador backend RSI desplegaba los proyectos de manera local, se elaboró un documento en la wiki de RSI que explicaba cómo desplegar el proyecto de manera local con la inclusión de la librería. Aunque los pasos implicados son sencillos, involucran la generación de un token y la adición de un archivo de configuración al repositorio de maven local.

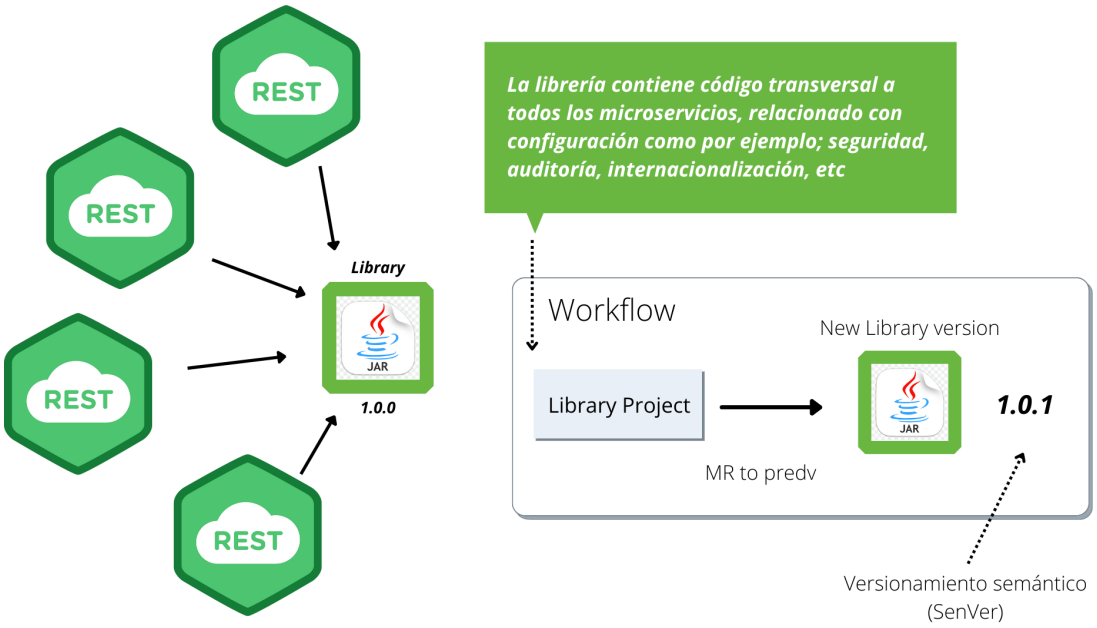

Figura 4. Librería compartida entre microservicios

Finalmente, se logró implementar la librería de manera exitosa en cada uno de los microservicios. Esto facilitó el desarrollo relacionado con la arquitectura transversal de los mismos, reduciendo la duplicidad de código en los diferentes microservicios y disminuyendo significativamente los tiempos en que las diferentes funcionalidades transversales eran agregadas a cada microservicio, ya que dicho código quedaba implementado en la librería y no en cada microservicio. Este trabajo ha demostrado la importancia de una planificación adecuada, la adaptabilidad y una documentación exhaustiva para garantizar el éxito en la implementación de una librería compartida. Se adjunta la wiki en la carpeta Librería para una mayor comprensión del proceso.

4.4. Codificación de caracteres, configuración de internacionalización de mensajes e interpolación de mensajes de errores

4.4.1. Objetivo Con el avance del proyecto RSI es necesario implementar la internacionalización en el proyecto RSI para soportar múltiples lenguajes, incluyendo español e inglés, y solucionar un problema de codificación que afecta la visualización de los mensajes en la interfaz de usuario. Para lograrlo, se debe identificar la causa del problema de codificación y aplicar una solución temprana para evitar que se vuelva más difícil de corregir y mantener en el futuro. Al mismo tiempo, se debe implementar la internacionalización para que los usuarios puedan interactuar con la interfaz en el idioma que prefieran. Es fundamental que la solución al problema de codificación se realice antes de la implementación de la internacionalización, ya que este aspecto afecta directamente la calidad de los mensajes visibles en la interfaz de usuario y puede generar confusiones o malentendidos.

4.4.2. Criterios de aceptación

- La implementación de la internacionalización en el proyecto RSI debe permitir que los usuarios interactúen con la interfaz en español e inglés.

- Los mensajes visibles en la interfaz de usuario deben ser legibles y mostrar correctamente los caracteres especiales y símbolos, independientemente del idioma seleccionado.
- Se deben realizar pruebas exhaustivas para garantizar que la solución implementada para el problema de codificación no afecte negativamente otros aspectos del proyecto.
- Debe generarse los manuales para la solución del problema de codificación en los diferentes IDE's en caso de ser necesario, además de la documentación pertinente a la implementación de la internacionalización y las configuraciones aplicadas.

4.4.3. Ejecución En primer lugar, se llevó a cabo una investigación exhaustiva sobre la codificación de caracteres y los estándares actuales en relación con la internacionalización o multilenguaje. Después de considerar varias opciones, se decidió utilizar UTF-8 como codificación del proyecto y de los archivos properties, garantizando así la representación adecuada de todos los caracteres de los idiomas involucrados.

Posteriormente, se realizó una configuración en los IDE's utilizados en el proyecto, Intellij y Eclipse, para que trabajaran con la codificación UTF-8 y se asegurara que los mensajes se mostraran correctamente en la interfaz de usuario. Además, se creó una guía para los demás desarrolladores que explicaba cómo realizar dicha configuración en los IDE's.

Para abordar la internacionalización del proyecto, se evaluó la necesidad de almacenar los mensajes en el archivo "messages.properties" y mostrarlos de forma adecuada, a fin de tener un mayor control sobre la información que se presenta y asegurar una lectura y comprensión precisas. Se identificó la necesidad de resolver el problema de la falta de internacionalización en algunos mensajes de error, que se

presentaban por defecto y no eran suficientemente claros.

Para solventar esta situación, se realizó la configuración de una serie de beans para que Spring pudiera resolver los mensajes y, así, encontrar la ubicación de los archivos de mensajes y poder interpolarlos en los mensajes de error o validaciones. Se revisaron las dependencias necesarias para el manejo de validaciones, las cuales se encontraron presentes.

Posteriormente, se añadió en los archivos messagesxx.properties que deben estar ubicados en el directorio src/main/resources, logrando resolver los mensajes de error con las validaciones de forma correcta.

Finalmente, se implementó la configuración del bean localeResolver, que permitió utilizar una lógica propia para el manejo de la internacionalización. Este analizador, de forma predeterminada, utiliza el campo Accept-Language del encabezado de la solicitud para determinar el entorno de la solicitud actual y proporcionar la respuesta en el lenguaje correspondiente. Se crearon archivos de mensajes específicos para cada lenguaje deseado, tales como messageses.properties y messagesen.properties.

A continuación las dependencias involucradas y los beans de configuración.

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
</dependency>

@Bean
public MessageSource messageSource() {
    var messageSource = new ReloadableResourceBundleMessageSource();
    messageSource.setBasename("classpath:messages");
    messageSource.setDefaultEncoding("UTF-8");
    return messageSource;
}
```

```
@Bean
public LocalValidatorFactoryBean getValidator() {
    var bean = new LocalValidatorFactoryBean();
    bean.setValidationMessageSource(messageSource());
    return bean;
}

@Bean
public LocaleResolver localeResolver() {
    return new CustomLocaleResolver();
}
```

Con esta solución, se resolvió el problema de codificación de caracteres, lo que permitió una visualización adecuada y agradable de los mensajes finales en la interfaz del cliente. Además, los mensajes estaban disponibles en inglés y español, para que el usuario pudiera elegir el idioma de su preferencia.

Por último, se elaboró una guía detallada sobre el funcionamiento de la internacionalización en los microservicios, con el objetivo de difundir y clarificar el uso de esta funcionalidad entre los demás miembros del equipo RSI.

Se adjunta la wiki en la carpeta Internacionalización para una mayor comprensión del proceso.

4.5. Investigación, análisis e implementación de tecnología que permita una eficiente comunicación entre los microservicios

4.5.1. Objetivo En el contexto de los proyectos back de RSI, es común que se necesite acceder a otros endpoints, ya sean internos o externos, que se denominan clientes o proxys. Actualmente, utilizamos una tecnología llamada RestTemplate[41]

[41] Baeldung. *A Guide to the RestTemplate*. URL: https://www.baeldung.com/rest-template. (accedido: 11.02.2023).

para este propósito, pero Spring Boot ha declarado su obsolescencia y su salida en futuras versiones, por lo que debemos buscar alternativas.
Además, es fundamental que cumplamos con los principios de una arquitectura limpia, que implica escribir un código limpio, reutilizable y escalable, y que nos permita orientar nuestros desarrollos tanto en on premise como en la nube. En este sentido, on premise se refiere a la ejecución de aplicaciones en servidores y dispositivos físicos ubicados en las instalaciones de la empresa, mientras que la nube se refiere a la ejecución de aplicaciones en servidores y dispositivos virtuales alojados en Internet. Para asegurar el éxito de nuestros proyectos y la satisfacción de nuestros clientes, es imprescindible que adoptemos una nueva herramienta que nos proporcione un código de mayor calidad y que se ajuste a los requerimientos de una arquitectura limpia. Con esta estrategia, podremos garantizar la escalabilidad, reutilización y facilidad de mantenimiento de nuestras aplicaciones, independientemente del entorno de ejecución en que se encuentren, ya sea on premise o en la nube.

4.5.2. Criterios de aceptación

- Para llevar a cabo una investigación y lectura efectiva, es importante tener en cuenta diversas consideraciones y la forma en la que actualmente se realizan las llamadas a otros microservicios. Por lo tanto, es necesario abordar todos los apartados de investigación y lectura con esta perspectiva en mente.
- Dejar plasmado en el proyecto de training los ejemplos e investigación que complemente todo lo leído.
- Dejar plasmado en la wikiRSI, todo lo investigado y leído.
- Implementar la nueva tecnología con las correspondientes pruebas en el ambiente de predev.

- A través de un documento plasmado en la wikiRSI dejar clara la nueva metodología para invocar microservicios.

4.5.3. Ejecución Se inició una investigación sobre una herramienta ofrecida por la documentación de Spring llamada FEIGN[22], la cual es una solución robusta que facilita la reutilización de código mediante el uso de interfaces que corresponden a los clientes que serán consumidos por el microservicio que los utiliza. Para analizar su comportamiento y evaluar sus posibles usos, se realizaron pruebas con FEIGN. Luego se realizaron las configuraciones necesarias en cada proyecto para habilitar la comunicación entre microservicios con esta herramienta, añadiendo las dependencias necesarias y habilitando el uso de FEIGN en la capa principal de la aplicación. Se realizaron pruebas en predev y en el proyecto de training para evaluar diferentes peticiones, tiempos de respuesta, posibles operaciones que involucran transacciones y se documentó adecuadamente el proceso.
Se presenta a continuación un ejemplo de cómo se utiliza la interfaz declarativa con el cliente. Esto permite que cada vez que se necesite invocar un servicio externo, solo sea necesario importar una instancia de la interfaz y acceder al método correspondiente.

```
@FeignClient(value = "rol-admonpersonal-service", url = "${admonpersonal.
    service.url}")
@RequestMapping("/api/rol")
public interface IRolAdmonPersonalClient {

    @GetMapping("/all")
    ResponseEntity<List<RolDTO>> findAll();
}
```

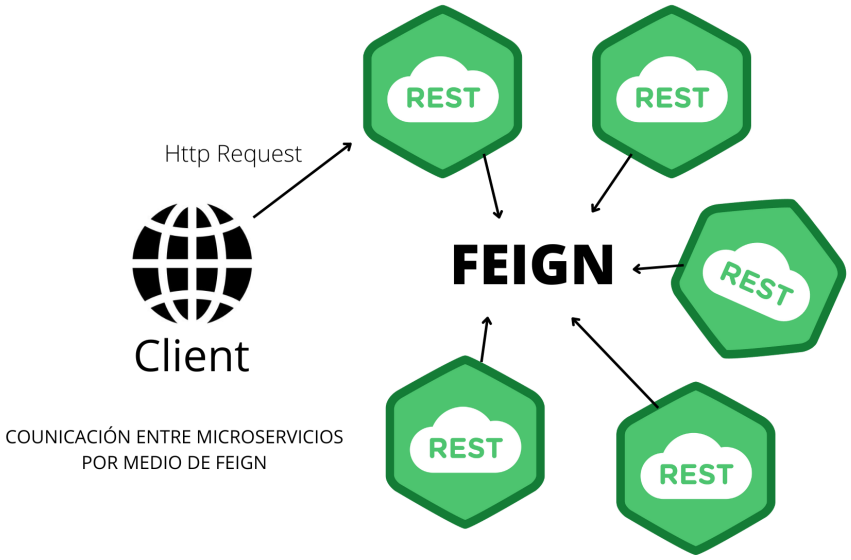

Figura 5. Comunicación entre microservicios por medio de FEIGN.

La imagen ilustra cómo cada microservicio utiliza la tecnología FEIGN para comunicarse con otros. Es importante destacar que esto no es algo común en una arquitectura de microservicios, ya que el objetivo es minimizar el acoplamiento y permitir que cada servicio funcione de manera independiente.

Se llevó a cabo una tarea importante consistente en migrar las tecnologías utilizadas en los microservicios. Anteriormente, se utilizaba RestTemplate, pero fue necesario revisar cada uno de los microservicios para migrarlos a la nueva tecnología. Durante este proceso, se descubrió la cantidad de código que se puede reutilizar al usar FEIGN. Con RestTemplate, era necesario definir cada uno de los parámetros de la petición en cualquier lugar donde se usara, pero con FEIGN, esto se hace una sola vez en la interfaz declarativa del cliente, lo que simplifica y agiliza el desarrollo. Se documentaron las políticas necesarias para el uso de esta tecnología, de manera que cada desarrollador pueda comprender claramente cómo trabajar con ella, esto se puede evidenciar en la wikiRSI.

Se adjunta la wiki en la carpeta FEIGN para una mayor comprensión del proceso.

4.6. Definición de estándares y reglas relacionadas a las políticas de pruebas del código desarrollado (dev-test)

4.6.1. Objetivo Se plantea la necesidad de mejorar la calidad de revisión del código antes de integrarlo en las ramas de desarrollo predev. Además, se ha estado trabajando en la creación de documentos en la wiki relacionados con la arquitectura, patrones, estándares backend, seguridad, test unit, entre otros. Se considera importante contar con una guía detallada que complemente las funciones de un dev test backend, a fin de mejorar la metodología de trabajo del equipo RSI. Esta iniciativa se enmarca dentro del objetivo de optimizar los procesos de desarrollo y aumentar la eficiencia del equipo en la implementación de soluciones de software.

4.6.2. Criterios de aceptación

- Crear un documento con las soluciones al objetivo mencionado en la historia de usuario.
- El documento que se debe crear debe contener los puntos clave necesarios para realizar un dev-test de manera efectiva, asegurando la detección y minimización de errores de integración.
- Garantizar la difusión del documento para promover la conciencia de su existencia y la correcta realización de las pruebas de desarrollo y dev-test.

4.6.3. Ejecución Tras analizar los pipelines de GitLab que se encargan de la integración y el despliegue automático en el proyecto RSI, se definieron conceptos claves para una revisión efectiva del código desarrollado. Se estableció un procedimiento que describe paso a paso lo que debe hacer un dev-tester al probar la funcionalidad, lo que incluye el despliegue local de la misma, la revisión del código y la verificación de que cumpla con los parámetros de código y arquitectura estable-

cidos por el proyecto RSI. Para esta tarea se detalla el uso de herramientas como SonarQube, que facilitan el proceso de revisión del código. Además, se detallan los archivos que no deben ser subidos, y se indica al dev-tester qué revisar para asegurar que no haya errores en el despliegue tras aprobar el merge request.
El documento resultante de este proceso se subió a la wiki de RSI, donde se detallan todos los pasos mencionados anteriormente. De esta manera, se logró mejorar los desarrollos de dev-test, esto redujo los posibles retrasos generados por malas prácticas en la fase de pruebas y optimizó tanto el desarrollo como la adición de las nuevas funcionalidades a los sistemas desarrollados por RSI. El documento se encuentra disponible en los anexos de este trabajo para su consulta y seguimiento.
Se adjunta la wiki en la carpeta DevTest para una mayor comprensión del proceso.

4.7. Investigación y análisis de alternativas para implementación de test unitarios

4.7.1. Objetivo Durante el desarrollo del proyecto RSI, se evidencia la falta de implementación de pruebas unitarias en el código, lo que afecta la calidad del mismo y en caso de fallas no ofrece ninguna opción para una fácil y posible identificación de errores. Por esta razón, se plantea como necesidad implementar pruebas unitarias en el proyecto para garantizar la calidad del código desarrollado, reducir la cantidad de errores y mejorar la robustez del software.
Para lograrlo, es necesario investigar sobre los conceptos relacionados con pruebas unitarias, sus ventajas y desventajas, así como las metodologías y tecnologías necesarias para aplicarlas de manera efectiva en el proyecto RSI.
El objetivo de esta historia es definir una estrategia para la implementación de pruebas unitarias en el proyecto, y preparar una charla de capacitación para el equipo de desarrollo, con el fin de establecer buenas prácticas y mejorar la calidad del código desarrollado en el futuro.

4.7.2. Criterios de aceptación

- Investigar y comprender los conceptos clave sobre test unitarios, su importancia en el desarrollo de software y cómo se aplican en el proyecto RSI.
- Identificar las tecnologías necesarias para la implementación de test unitarios en el proyecto RSI, incluyendo librerías y herramientas de testing.
- Implementar ejemplos de tests unitarios en el proyecto RSI, asegurándose de que se cubren los casos más relevantes y se logra una adecuada cobertura de código.
- Documentar de manera clara y concisa la información recolectada sobre test unitarios, incluyendo definiciones, casos de uso, y detalles de implementación.
- Realizar una charla de capacitación sobre test unitarios para el equipo de desarrollo del proyecto RSI, explicando los conceptos clave, las tecnologías necesarias, los ejemplos de tests unitarios implementados y cómo se pueden aplicar en el futuro.

4.7.3. Ejecución Para la ejecución de esta historia, se inició con la definición de conceptos básicos sobre test unitarios, con el objetivo de tener una comprensión clara sobre su importancia, ventajas, fundamentos y tipos de pruebas. Además, se investigaron otras metodologías basadas en pruebas, como la TDD, para tener una visión más amplia y enriquecedora.
Posteriormente, se profundizó en los aspectos específicos relacionados con las tecnologías que se utilizan en los microservicios de RSI, incluyendo las diferentes opciones y la estructuración de los test en Spring. También se trabajó en la definición de la capa esencial sobre la cual se desarrollarían los test unitarios en esta fase inicial.

Se creó un documento que recopila toda la información procesada y se desarrollaron ejemplos en el proyecto de pruebas para poner en práctica todo lo aprendido. Además, se realizó una charla de capacitación al equipo para presentar los resultados y dar inicio a la fase 1 de pruebas unitarias en RSI.
Con esta implementación, se lograron establecer los inicios de las pruebas de código en el proyecto RSI, las cuales no habían sido consideradas anteriormente. De esta forma, se comienza a mejorar el código desarrollado y se aumenta la calidad del proyecto en aspectos como la mantenibilidad y la escalabilidad.
El documento correspondiente se encuentra disponible en los anexos de este trabajo, en la carpeta Test donde se pueden encontrar más detalles sobre el proceso de implementación de las pruebas de código en el proyecto.

4.8. Revisión de latencia en microservicios y ajustes necesarios para la mejora en la duración en los tiempos de respuesta

4.8.1. Objetivo El objetivo de esta historia es mejorar la experiencia de usuario al momento de invocar el microservicio de integración, el cual maneja múltiples fuentes de datos y permite el acceso a información de sistemas anteriores y de información renovada. Durante pruebas previas, se ha identificado que los tiempos de respuesta son altos, lo que puede ser un problema para los usuarios.
Para abordar este problema, se debe investigar la causa principal de estos largos tiempos de respuesta. Se considera que podría haber falta de optimización en la configuración manual del manejo de transacciones y conexiones en el microservicio. Se debe estudiar el comportamiento de la transaccionalidad y el manejo de conexiones en el microservicio, y definir y aplicar las mejores prácticas en su configuración. También se debe tener en cuenta la configuración manual existente y las múltiples fuentes de datos para garantizar una mejora significativa en la duración de los tiempos de respuesta al momento de invocar el microservicio. El resultado

esperado es una mejor experiencia de usuario y mayor eficiencia en la operación del sistema.

4.8.2. Criterios de aceptación

- Se debe revisar los archivos de configuración manual relacionados con las fuentes de datos u otros aspectos para verificar si la lentitud en dicho proyecto se debe a la ausencia, falta de configuración o configuración no idónea de los datasources.
- Se deben realizar pruebas para verificar la efectividad de las estrategias de optimización aplicadas en el manejo de transacciones y conexiones.
- Se deben establecer y documentar las mejores prácticas en la configuración del microservicio para lograr una mejora significativa en la duración de los tiempos de respuesta.
- Se deben realizar pruebas de carga para validar que el microservicio pueda manejar una cantidad significativa de solicitudes sin afectar su rendimiento.
- Se debe presentar un informe detallado sobre los cambios y mejoras aplicadas, así como los resultados obtenidos en las pruebas realizadas.
- Se debe garantizar que la experiencia de usuario al momento de invocar el microservicio haya mejorado significativamente y que los tiempos de respuesta sean acordes a los estándares establecidos por la organización.

4.8.3. Ejecución Para abordar esta historia, se inició realizando un análisis del comportamiento del microservicio, lo que implicó el testeo de las peticiones y comparando el tiempo de respuesta con otros microservicios que no tenían múltiples fuentes de datos. Luego, se realizaron múltiples peticiones utilizando herramientas

como webstress y jmeter para testear los endpoints y determinar en qué escenarios se presentaba la alta latencia.

A partir de las pruebas realizadas, se llegó a la conclusión de que las llamadas a base de datos estaban tomando más tiempo de lo normal después de múltiples peticiones. Para abordar este problema, se comenzó a realizar pruebas en la configuración de los datasources y se encontró una implementación más eficiente para las conexiones a base de datos.

En consecuencia, se implementaron los cambios necesarios para el manejo de pool de conexiones basadas en HikariCP y se realizaron múltiples pruebas para validar la eficacia de los cambios implementados. Como resultado, se observó un aumento drástico en el rendimiento del microservicio. Luego de la implementación y la carga de los cambios en el proyecto, se activó en producción.

Pasando de tener el siguiente rendimiento.

Figura 6. Rendimiento inicial, con problemas de latencia

a tener uno mucho más óptimo como lo es este final.

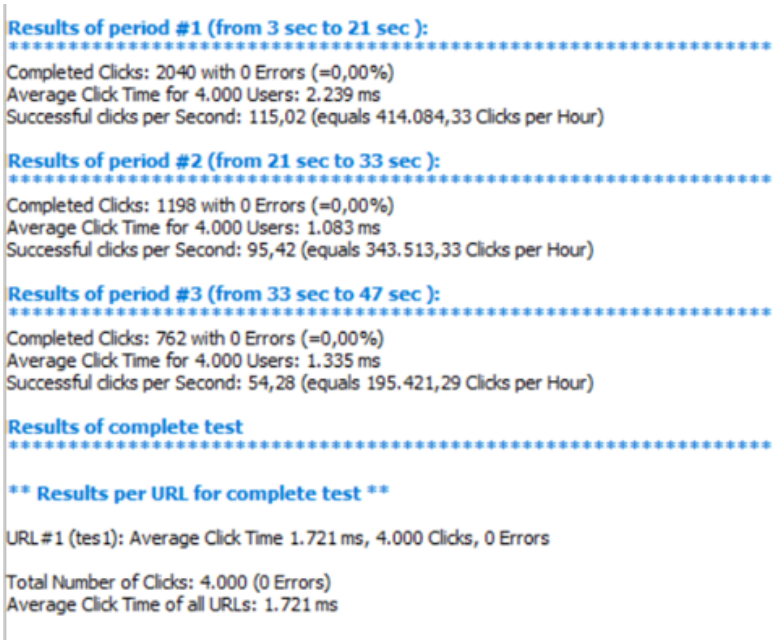

Figura 7. Rendimiento final, sin problemas de latencia

Tras la implementación de los cambios en la configuración del manejo de pool de conexiones basadas en HikariCP, se observó una significativa mejora en el rendimiento del microservicio. Se pasó de un tiempo promedio de respuesta de alrededor de 68 ms a cerca de 2 ms en la métrica de Average click time of all URLs. Aunque esta métrica no es completamente rigurosa, sí es una indicación útil del tiempo promedio en el que los usuarios tienen acceso al endpoint en el escenario documentado.

Vemos incluso disminución total de los errores presentados a causa del tiempo de espera agotado en ciertas peticiones, las cuales dependen de la magnitud de solicitudes y la capacidad del datasource, por tanto la mejora en el rendimiento del microservicio fue evidente y se demostró cómo la configuración anterior estaba afectando su desempeño.

El anexo en la carpeta Datasources sirvió como guía para la implementación de dicha configuración óptima del microservicio.

4.9. Gestión, codificación y encriptación de contraseñas de personales administrativos, estudiantes y funcionarios entre múltiples sistemas

4.9.1. Objetivo Dado que actualmente se utilizan dos tipos de sistemas, uno basado en Informix y otro en desarrollo basadeo en Oracle, es necesario implementar un API que permita mantener la encriptación de las contraseñas de los nuevos usuarios en ambos sistemas. Es decir, al registrar un usuario en uno de los sistemas, se deberá replicar la información en el otro. Esta funcionalidad deberá ser desarrollada para ser consumida en las funcionalidades de registro o creación de usuarios y cambio de contraseñas.

4.9.2. Criterios de aceptación

- Se requiere la implementación de funcionalidades específicas en el microservicio de integration, con el fin de satisfacer las diferentes necesidades de inserción y actualización de contraseñas con encriptación correspondiente entre los sistemas basados en informix y en oracle.
- Estas funcionalidades deben estar debidamente adaptadas a ambas plataformas y deben permitir la realización de tareas como el registro y cambio de contraseñas de los usuarios. Además, es importante garantizar que estas funcionalidades sean totalmente funcionales y confiables para asegurar la integridad de los datos en ambos sistemas.
- Realizar las correspondientes verificaciones y pruebas de lo implementado en el ambiente de predev.

4.9.3. Ejecución Para llevar a cabo la tarea de garantizar la compatibilidad de las contraseñas de los usuarios en las bases de datos antiguas y nuevas, se realizó un análisis detallado de los algoritmos de codificación y encriptación que se estaban

utilizando para guardarlas.

Después de haber completado el análisis, se procedió a implementar estos algoritmos en el nuevo servicio. En este sentido, se desarrolló un API que permitió la recepción de la información del usuario una vez se registraba o actualizaba su contraseña, para que de esta manera los campos de las contraseñas en ambas bases de datos pudieran ser guardados y/o actualizados de manera simultánea, permitiendo al usuario acceder a los sistemas antiguos y nuevos sin problemas de compatibilidad.

Finalmente, se realizaron exhaustivas pruebas para garantizar que los servicios desarrollados funcionaran correctamente y cumplieran con los objetivos esperados brindando así una mayor adaptabilidad, soportando los procesos anteriores y teniendolos en cuenta para las funcionalidades actuales.

4.10. Desarrollo transversal para manipulación de plantillas de mensajes de correos

4.10.1. Objetivo El objetivo es desarrollar un módulo transversal que permita la manipulación de plantillas y correos, así como la visualización en tiempo real de los mensajes predefinidos, con el fin de simplificar la edición del cuerpo de los correos que se enviarán a la comunidad UIS mediante el API de notificaciones en situaciones específicas, como cambios de contraseñas, alertas, notificaciones, entre otros.

Este módulo debe tener una interfaz intuitiva que permita a los usuarios ajustar fácilmente el contenido de los correos electrónicos, adaptándolos a las necesidades de cada situación. Además, se espera que el módulo sea flexible y escalable, de manera que pueda adaptarse a futuros cambios en los requerimientos de notificación de la comunidad UIS.

La visualización en tiempo real permitirá a los usuarios verificar el aspecto final del correo electrónico antes de enviarlo, lo que garantizará que la información se

presente de manera clara y efectiva. En resumen, el módulo transversal será una herramienta valiosa para simplificar el proceso de envío de correos electrónicos y garantizar la eficacia de las notificaciones que se envíen a la comunidad UIS.

4.10.2. Criterios de aceptación

- Test unitarios de backend para la historia de usuario: se deben desarrollar pruebas unitarias exhaustivas para verificar que el backend de la aplicación funcione correctamente y cumpla con los requisitos definidos en la historia de usuario. Estas pruebas deben cubrir todas las funcionalidades del backend y garantizar su correcto funcionamiento.

- Cumplimiento de estándares en desarrollo back y front: se debe asegurar que el desarrollo del back y front se realice siguiendo las mejores prácticas y estándares de codificación. Esto incluye, entre otros, la implementación de patrones de diseño, la estructuración adecuada del código, la documentación clara y concisa, y la utilización de herramientas de análisis de calidad de código.

- Correcto manejo de validaciones, mensajes de error y estados de los procesos: se debe garantizar que la aplicación maneje correctamente las validaciones de datos, mensajes de error y estados de los procesos. Esto incluye la validación de los datos ingresados por el usuario, la notificación clara de los errores ocurridos durante el proceso y la actualización de los estados de los procesos en tiempo real.

- Permitir modificar plantillas de correos, de mensajes predefinidos, así como su visualización bajo una implementación de un visor HTML propio o de terceros: la aplicación debe permitir la modificación de las plantillas de correos y mensajes predefinidos de manera fácil e intuitiva, y debe contar con un visor HTML propio o de terceros que permita visualizar el resultado final de los correos

electrónicos antes de su envío. Además, se debe asegurar que la visualización de estos correos sea adecuada en diferentes dispositivos y navegadores.

4.10.3. Ejecución Para el desarrollo de esta historia se realizó un análisis detallado de la descripción de la funcionalidad a desarrollar, prestando especial atención a los detalles y particularidades específicas. En cuanto al apartado de front-end, se llevaron a cabo revisiones de estándares y buenas prácticas para garantizar que el desarrollo fuera de alta calidad.

Uno de los mayores desafíos del desarrollo fue la implementación del Visor HTML, ya que esta funcionalidad es bastante diferente a cualquier otra. Después de considerar diversas alternativas, se optó por desarrollar un visor HTML propio utilizando el componente HTML iframe. Este componente permite la inserción de un documento HTML dentro de otro, lo cual resulta muy útil en este caso, ya que tanto las plantillas de correos como los mensajes predefinidos son en formato HTML. De esta manera, se logra una visualización adecuada de los contenidos. Posteriormente, se aplicaron los estilos y la lógica necesarios para obtener la interfaz con la funcionalidad correcta.

Es importante mencionar que en el desarrollo del back-end se siguió un proceso lineal y se aplicaron las buenas prácticas y estándares recomendados para garantizar la calidad del código y su correcto funcionamiento.

El desarrollo meticuloso de la funcionalidad para optimizar el proceso de creación de mensajes predefinidos en formato HTML ha mejoró significativamente la usabilidad del software en este área, al implementar una interfaz de visualización en tiempo real de los cuerpos de los correos electrónicos. Se aplicaron buenas prácticas de programación para garantizar la calidad del código y su mantenimiento a largo plazo, lo que hace que el software sea fácil de mantener y evolucionar. En general el desarrollo mejora aspectos de usabilidad del software y se ha garantizado su facilidad de mantenimiento y evolución.

4.10.4. Resultado final A continuación se muestra una figura del resultado visible del desarrollo de esta historia de usuario.

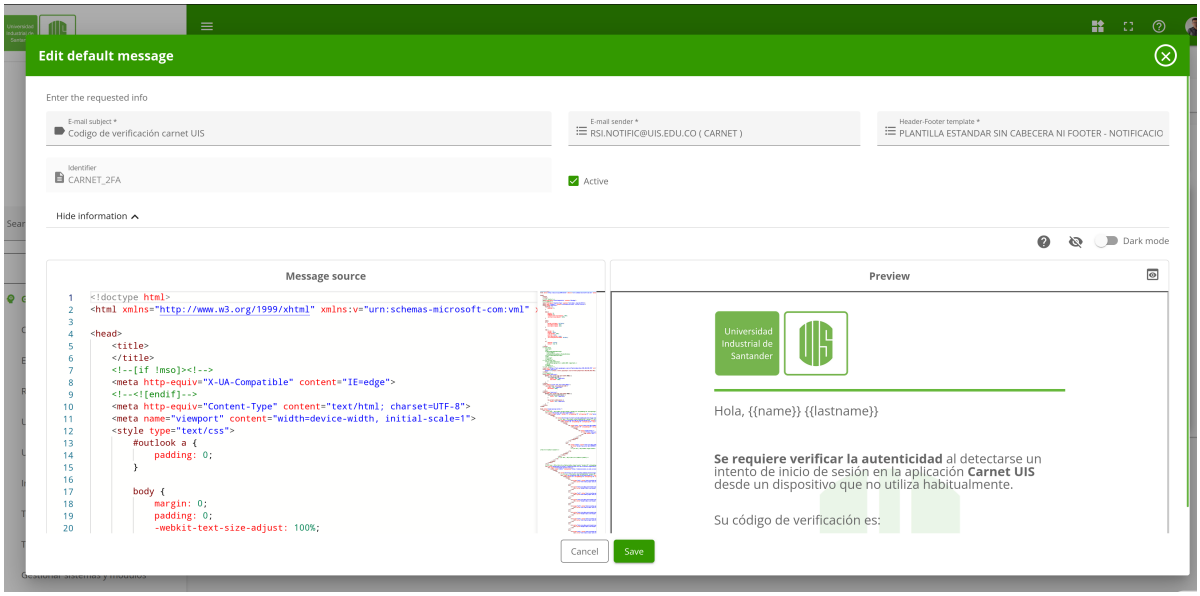

Figura 8. Funcionalidad de editor de plantillas de correos en el modulo de configuración

4.11. Aplicar RBAC (seguridad basada en roles) a cada microservicio

4.11.1. Objetivo Durante el desarrollo de los sistemas, se reconoce la necesidad de contar con una estructura de seguridad más robusta y estructurada. Por esta razón, se debe investigar posibles maneras de incluir una seguridad basada en roles y/o acciones con el fin de garantizar un mayor nivel de protección de los datos y recursos en los diferentes proyectos.

La mejor manera de abordar esta necesidad es a través de la implementación de un modelo de control de acceso basado en roles (RBAC), el cual permite definir clara y específicamente los permisos y acciones que los usuarios pueden realizar en los sistemas. Esto permite establecer una estructura de seguridad sólida que puede ser fácilmente administrada y actualizada en caso de ser necesario.

El objetivo principal es aplicar RBAC en los sistemas existentes, teniendo en cuenta los posibles impactos que esta implementación pueda generar, para garantizar una seguridad más robusta y estructurada en los sistemas.

4.11.2. Criterios de aceptación

- Investigar y documentar los conceptos y metodologías de RBAC para su aplicación en los proyectos Spring Boot.
- Implementar y configurar RBAC en el proyecto de prueba (Training), incluyendo roles y permisos para diferentes usuarios y acciones.
- Realizar pruebas exhaustivas para garantizar que la seguridad basada en roles y acciones se está aplicando correctamente en el proyecto de prueba.
- Analizar los posibles impactos y riesgos de la implementación de RBAC en los diferentes proyectos, tomando en cuenta los roles y permisos de los usuarios para garantizar que solo tengan acceso a las funciones que les corresponden.

- Documentar y presentar los resultados de las pruebas y la implementación de RBAC en la wiki del proyecto.

4.11.3. Ejecución La implementación de seguridad basada en roles y acciones en la arquitectura de microservicios como objetivo central se llevó a cabo mediante una investigación exhaustiva acerca de los diferentes métodos de implementación disponibles, considerando detalladamente sus ventajas y desventajas, para luego de un análisis riguroso, implementar dos niveles de RBAC (Role-Based Access Control): el primero basado en el acceso a los microservicios que cada rol puede tener, y el segundo basado en acciones específicas con nombres concretos.
La implementación se llevó a cabo utilizando Spring Security. En el primer nivel, fue necesario definir la autorización de acceso a un microservicio para un usuario en concreto, y esto se logró mediante la asignación de roles a dicho usuario, junto con la definición de una tabla en la base de datos que almacena los roles que tienen acceso a un microservicio específico. Esta combinación permitió manejar eficientemente la política de acceso a los microservicios, y autorizar la petición de un usuario en el primer nivel. La clase SecurityConfig es responsable de definir este primer nivel de autorización, y a continuación se muestra un ejemplo de dicha clase de seguridad:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    /**
     * Ignorar rutas publicas de la cadena de seguridad, security chain
     */
    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers(API_URI_PUBLIC_PATTERN);
    }
```

```
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable().authorizeRequests()
                    .antMatchers("/api/v1/**").authenticated()
                    .antMatchers("/api/secured/**").access("@RouteGuard.
                        checkAccess()")
                         .anyRequest().authenticated();
            }
            http.addFilterBefore(filterRequests,
                UsernamePasswordAuthenticationFilter.class);
    }
}
```

En adición, el bean llamado RouteGuard y su método checkAccess son responsables de consultar la base de datos para verificar si el usuario tiene los permisos necesarios para acceder al microservicio en cuestión. A continuación se presenta un ejemplo de este bean.

```
@Component("RouteGuard")
@Slf4j
public class RouteGuard {

    public boolean checkAccess() {
        return this.rolLibraryRepository.
            checkAccessToMicroserviceByAuthorities(this.getIdUsuario(),
                initialsMicroservice);
    }
```

Para implementar el segundo nivel de RBAC, se definió una abstracción que asigna nombres en lenguaje natural a las acciones y se las asoció a los métodos o endpoints que necesitan seguridad. La información de acceso se guardó en una tabla de la base de datos y se utilizó una anotación personalizada de Spring Security para permitir o denegar el acceso al método correspondiente (@PreAuthorize).

Para activar esta seguridad de segundo nivel, es necesario configurar previamente la clase de seguridad, que tendría el siguiente aspecto:

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    ...

    //los @preauthorize tambi n deben excluirse en local y para esa
       parte debemos configurar din micamente la anotaci n que habilita
        o deshabilita el preauthorize, de manera condicional, la
       seguridad ser  activada

    @ConditionalOnProperty(prefix = "authorization.enable", name = "on",
       havingValue = "true")
    @EnableGlobalMethodSecurity(prePostEnabled = true)
    static class GlobalMethodSecurity {
    }
}
```

A continuación se presenta un ejemplo de cómo se puede aplicar seguridad a un método específico.

```
@PreAuthorize("@RouteGuard.checkAccessAction('generar-nomina')")
@GetMapping
public ResponseEntity<String> generarNomina(){
    ....
    ....
    return  ResponseEntity.ok("Nomina");
}
```

Finalmente, después de una investigación exhaustiva y la implementación de dos niveles de seguridad basados en roles y acciones en la arquitectura de microser-

vicios, se llevaron a cabo pruebas exhaustivas para analizar su comportamiento. A medida que se incluyó en cada microservicio, se logró una mejora significativa en la calidad del proyecto al contar con una seguridad a dos niveles que limita el acceso a los recursos únicamente a personal autorizado. La implementación exitosa de esta funcionalidad garantiza un entorno seguro para los usuarios y una mayor protección de los datos en el sistema.
En los anexos en la carpeta Seguridad se encuentran los documentos de la wiki que abordan estos aspectos.

4.12. Optimización en los tiempos de inserciones masivas a base de datos

4.12.1. Objetivo Se requiere investigar la mejora u optimización de los tiempos de inserciones masivas a base de datos, ya que en un futuro se prevee hacer uso de dichas inserciones.
Actualmente las inserciones masivas se hacen con el método saveAll() de la implementación de los repositorios de JPA, pero es necesario investigar y profundizar sobre nuevas tecnologías que permitan un mejor rendimiento en estos tiempos.

4.12.2. Criterios de aceptación

- Investigar y documentar los conceptos y metodologías de inserciones masivas a base de datos en Spring Boot.
- Implementar y desarrollar las diferentes soluciones relacionadas a esta historia en el proyecto de training.
- Realizar pruebas y analizar los tiempos con las diferentes tecnologías usadas.
- Documentar, socializar y presentar los resultados de las pruebas y su implementación.

4.12.3. Ejecución Las inserciones masivas también conocidas por su término en inglés “bulk insert” consisten basicamente en insertar muchos registros en la base de datos de una sola vez. En base a lo investigado e implementado a continuación se presentan las opciones que se tienen para mejorar los tiempos en los que ocurren estas inserciones, ya que muchas veces el tipico método saveAll() que encontramos con los repositorios de JPA no llega a ser suficiente para suplir esta necesidad.

4.12.3.1. Hibernate Batch Insert Para hacer la inserción de multiples registros con el método saveAll() Hibernate crea un statement por cada registro y va envíando la instrucción de la inserción de los registros uno por uno, esto es el indicio de la demora cuando se trata de inserciones masivas. Existe un concepto llamado batch insert, y consiste en que estos statements no sean envíados uno por uno, sino que se puedan envíar más a la base de datos, de esta manera las inserciones se harán agrupadas por el número del batch. Por ejemplo si el batch es 30, se envíaran 30 statements de inserciones a la base de datos en lugar de envíarlos uno por uno.
Para que esta caracteristica se active de manera correcta es muy importante que la forma en la que se genera el id de la entidad en cuestión sea de tipo SEQUENCE, para las entidades que no tengan este tipo de estrategia se seguirá haciendo la inserción de la manera habitual.

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
@Column
private Long id;
```

También se tuvieron que añadir configuraciones al archivo de application.properties del proyecto que activa el uso de esta tecnología, Finalmente utilizando este método se redujeron los tiempos en un 50 %, es decir, si antes para insertar 1000 registros de la manera clásica se tardaba 60 segundos, ahora se tarda 30 segundos.

4.12.3.2. Jdbc Batch Insert JDBC así como JPA es un API que se comunica con la base de datos, pero a diferencia de JPA, se le deben dar las instrucciones de la base de datos como queries nativas, pero para este método de inserción la estrategia sigue siendo la misma, insertar una gran cantidad de registros, pero dividiendo el número de inserciones y agrupandolas por el tamaño del batch. Se implementó este API junto con su estrategia de inserción por "batch".

```
@Override
@Transactional
public void saveAllJdbcBatch(List<ClaseSituacionAdministrativa>
    cSAEntities, int batchSize) {
    String sql = String.format(
            "INSERT INTO %s (descripcion, senal_grupal, identificador) "
                +
                    "VALUES (?, ?, ?)",
            ClaseSituacionAdministrativa.class.getAnnotation(Table.class)
                .name()
    );
    try (Connection connection = hikariDataSource.getConnection();
         PreparedStatement statement = connection.prepareStatement(sql)
    ) {
        int counter = 0;
        for (ClaseSituacionAdministrativa cSA : cSAEntities) {
            statement.clearParameters();
            statement.setString(1, cSA.getDescripcion());
            statement.setBoolean(2, cSA.getSenalGrupal());
            statement.setString(3, cSA.getIdentificador());
            statement.addBatch();
            if ((counter + 1) % batchSize == 0 || (counter + 1) ==
                cSAEntities.size()) {
                statement.executeBatch();
                statement.clearBatch();
            }
```

```
                counter++;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
}
```

Esta es una de las maneras más optimas de hacer inserciones masivas, las mejoras son bastante considerables. Por ejemplo en las pruebas realizadas para insertar 1000 registros con el método saveAll (sin Hibernate batch) se tarda 60 segundos en promedio, y con el método jdbcBatch, con un batch size de 30 se tarda 3 segundos.

4.12.3.3. Use connection pool with threading Otra forma de hacer las inserciones masivas es aprovechando el pool de conexiones y usando hilos o subprocesos que pueden llegar a ser ejecutados al mismo tiempo según el tamaño del pool de conexiones.

A continuación se muestra un ejemplo del código implementado para dos casos distintos, uno será este método con el saveAll clásico de Hibernate y el otro ejemplo con el jdbcBatch.

```
@Override
@Transactional
public void saveAllThreadsCallable(List<ClaseSituacionAdministrativa>
    cSAEntities, int batchSize) {
    ExecutorService executorService = Executors.newFixedThreadPool(
        hikariDataSource.getMaximumPoolSize());
    List<List<ClaseSituacionAdministrativa>> listOfBookSub = this.
        createSubList(cSAEntities, batchSize);
    List<Callable<Void>> callables = listOfBookSub.stream().map(sublist
        ->
            (Callable<Void>) () -> {
                saveAllHibernate(sublist);
```

```
                return null;
            }).collect(Collectors.toList());
    try {
        executorService.invokeAll(callables);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Para el ejemplo anterior se obtiene una mejora notable utilizando el SaveAll clásico, por ejemplo, para insertar 1000 registros se pasa de tardar alrededor de 60 segundos a tardar alrededor de 15 segundos en promedio, esta es una mejora bastante significativa, pero no se dispara la auditoría, un truco podría ser tomar el primer registro de la data y hacer un .save() para que se dispare la auditoría clasica y el resto de la data si se guarda con este método.

```
@Override
 @Transactional
 public void saveAllThreadsCallable(List<ClaseSituacionAdministrativa>
    cSAEntities, int batchSize) {
     ExecutorService executorService = Executors.newFixedThreadPool(
        hikariDataSource.getMaximumPoolSize());
     List<List<ClaseSituacionAdministrativa>> listOfBookSub = this.
        createSubList(cSAEntities, batchSize);
     List<Callable<Void>> callables = listOfBookSub.stream().map(sublist
        ->
             (Callable<Void>) () -> {
                 saveAllJdbcBatch(sublist, batchSize);
                 return null;
             }).collect(Collectors.toList());
     try {
         executorService.invokeAll(callables);
     } catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
    }
```

De acuerdo a las pruebas realizadas en este caso se obtiene la combinación más eficiente para las inserciones masivas, pudiendo insertar 50.000 registros con un tamaño del batch de 5.000 en apenas 5 segundos en promedio. Una mejora bastante significativa a comparación del método saveAll() clásico que se venía utilizando.

La implementación exitosa de estas mejoras ha logrado optimizar significativamente los tiempos de inserciones masivas, lo que se traduce en una mayor eficiencia y eficacia en el manejo de los datos. Como resultado, se mejora el rendimiento y la capacidad de respuesta de la aplicación, lo que se traduce en una mejor experiencia del usuario y una mayor satisfacción con el servicio ofrecido. Además, esto contribuye a la calidad del software al mejorar su capacidad de procesamiento de grandes cantidades de datos de manera eficiente y efectiva.

En los anexos en la carpeta Tiempos-Peticiones se encuentran los documentos relacionados con la optimización correspondiente.

5. CONCLUSIONES

Al actualizar la herramienta de comunicación entre microservicios, se logró eliminar la duplicidad de código en diferentes servicios, lo que resultó en una reducción de los costos de desarrollo y mantenimiento asociados con la tecnología anterior. Además, la implementación de una librería de microservicios optimizó los tiempos de mantenimiento de funcionalidades compartidas y eliminó la duplicidad de código en diferentes microservicios, facilitando la integración de nuevas funcionalidades.

Asimismo, se implementó una estrategia de auditoría en la base de datos para tener un control detallado de los datos de interés transaccionados en ella. Al establecer una estrategia de seguridad basada en roles se le brindó confiabilidad al software.

La definición de estándares y manuales para los procedimientos de dev-test y test unitarios mejoró la metodología de desarrollo de RSI, lo que disminuyó los retrasos producidos por mala calidad en dev-test en la entrega del producto. Además, se mejoraron los tiempos de inserciones masivas en la base de datos, logrando una disminución del tiempo final hasta en un 97 %. En la optimización de microservicios con alta latencia, se disminuyeron los tiempos de respuesta hasta en un 97.5 %.

La constante vigilancia tecnológica permitió la implementación de estas soluciones eficientes en los aspectos transversales de la aplicación para lograr un proyecto exitoso y sostenible en el tiempo teniendo en cuenta aspectos de arquitectura. Los resultados obtenidos en este trabajo de grado son una muestra de cómo la mejora continua y la innovación en el desarrollo de software son clave para alcanzar el éxito en proyectos de esta índole.

6. TRABAJO FUTURO

En el proceso de desarrollo de una aplicación de gran envergadura, el aspecto transversal puede ser bastante amplio y complejo. Sin embargo, en este proyecto se ha logrado avanzar significativamente en este sentido, gracias a la implementación de una metodología ágil y la definición de herramientas y tecnologías adecuadas para diseñar aspectos no funcionales del software.

A pesar de ello, es importante reconocer que el proyecto aún tiene margen de mejora y se requiere seguir invirtiendo esfuerzos para lograr un despliegue más eficiente de la aplicación. Para ello, se propone la implementación de tecnologías actuales y eficientes como Docker para las aplicaciones y Kubernetes como orquestador de contenedores, lo que permitiría garantizar la estabilidad y confiabilidad de la aplicación en su despliegue.

Asimismo, se identifican oportunidades de mejora en el proceso de integración continua y entrega continua (CI/CD) del software, en especial en lo que respecta a la realización de pruebas y el despliegue automático de ajustes en la librería. Esto permitiría garantizar una mayor calidad en el software entregado y reducir los tiempos de respuesta ante posibles errores o fallos.

En conclusión, aunque se ha avanzado en el aspecto transversal del proyecto RSI, es importante continuar invirtiendo en tecnologías y procesos que permitan mejorar la eficiencia y calidad en el despliegue de la aplicación, con el fin de garantizar su estabilidad y confiabilidad en el largo plazo.

BIBLIOGRAFÍA

Apiumhub. *Atributos de Calidad de Arquitectura de software*. URL: https://apiumhub.com/es/tech-blog-barcelona/atributos-calidad-arquitectura-software/. (accedido: 12.01.2023) (vid. pág. 14).

Atlassian. *Historias de usuario con ejemplos y plantilla*. URL: https://www.atlassian.com/es/agile/project-management/user-stories. (accedido: 11.02.2023) (vid. pág. 42).

— *Microservices architecture*. URL: https://www.atlassian.com/es/microservices/microservices-architecture. (accedido: 12.12.2022) (vid. págs. 13, 16).

— *¿Qué es scrum?* URL: https://www.atlassian.com/es/agile/scrum#:~:text=Â¿QuÃ©%20es%20scrum%3F,de%20valores%2C%20principios%20y%20prÃ¡cticas.. (accedido: 11.02.2023) (vid. pág. 40).

— *¿Qué son las pruebas unitarias y cómo llevar una a cabo?* URL: https://www.atlassian.com/es/git/tutorials/what-is-version-control. (accedido: 11.02.2023) (vid. pág. 35).

Auth0.com. *JSON web tokens introduction*. https://jwt.io/introduction (vid. pág. 24).

Baeldung. *A Guide to the RestTemplate*. URL: https://www.baeldung.com/rest-template. (accedido: 11.02.2023) (vid. pág. 59).

— *JPA Entity Lifecycle Events*. URL: https://www.baeldung.com/jpa-entity-lifecycle-events. (accedido: 11.02.2023) (vid. pág. 28).

Baeldung. *Learn Spring boot*. URL: https://www.baeldung.com/spring-boot. (accedido: 10.01.2023) (vid. pág. 53).

— *Spring Events*. URL: https://www.baeldung.com/spring-events. (accedido: 11.02.2023) (vid. pág. 27).

Budgen, D. "Software design: Creating solutions for ill-structured problems". En: CRC Press, Taylor amp; Francis Group, 2021, pág. 72 (vid. pág. 14).

Cloud, Google. *Administración de dependencias*. URL: https://cloud.google.com/software-supply-chain-security/docs/dependencies?hl=es-419#:~:text=Una%20dependencia%20de%20software%20es,descargas%20o%20instalas%20tu%20software.. (accedido: 11.02.2023) (vid. pág. 28).

Deloitte. *¿Qué es un ORM?* URL: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html. (accedido: 11.02.2023) (vid. pág. 26).

Git. *Git Documentation*. URL: https://git-scm.com/docs/git. (accedido: 07.01.2023) (vid. pág. 36).

GitLab. *GitLab Docs*. URL: https://docs.gitlab.com/ee/. (accedido: 07.01.2023) (vid. pág. 36).

Hackwise. *¿Cuál es la diferencia entre codificación, cifrado y hashing?* URL: https://hackwise.mx/cual-es-la-diferencia-entre-codificacion-cifrado-y-hashing/#:~:text=La%20codificaciÃşn%20es%20un%20proceso,una%20cadena%20de%20longitud%20fija.. (accedido: 11.02.2023) (vid. pág. 24).

Hat, Red. *La integración y la distribución continuas (CI/CD)*. URL: https://www.redhat.com/es/topics/devops/what-is-ci-cd#:~:text=La%20CI%2FCD%20es%

20un,distribuciÃşn%20y%20la%20implementaciÃşn%20continuas.. (accedido: 11.02.2023) (vid. pág. 36).

HubSpot. *5 pasos para hacer una auditoría de base de datos*. URL: https://blog.hubspot.es/marketing/auditoria-base-de-datos. (accedido: 11.02.2023) (vid. pág. 27).

IBM. *Servidores de aplicaciones*. URL: https://www.ibm.com/docs/es/i/7.2?topic=serving-application-servers. (accedido: 11.02.2023) (vid. págs. 20, 26).

— *¿Cómo funcionan las pruebas de software?* URL: https://www.ibm.com/es-es/topics/software-testing. (accedido: 11.02.2023) (vid. pág. 37).

Inesdi. *¿Qué es un QA testing y cómo se hace?* URL: https://www.inesdi.com/blog/que-es-un-qa-testing/#:~:text=Un%20QA%20testing%20es%20un,se%20correspondan%20con%20lo%20precisado.. (accedido: 11.02.2023) (vid. pág. 39).

IONOS, Digital Guide. *¿Qué es y cómo funciona el RBAC?* URL: https://www.ionos.es/digitalguide/servidores/seguridad/que-es-el-role-based-access-control-rbac/. (accedido: 11.02.2023) (vid. pág. 23).

Linkedin.alexandrerocha. *Diferencia entre Autenticación y Autorización*. URL: https://es.linkedin.com/pulse/diferencia-entre-autenticaciÃşn-y-autorizaciÃşn-alex-rocha. (accedido: 11.02.2023) (vid. pág. 23).

Netflix, Spring cloud. *Declarative REST Client: Feign*. URL: https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html. (accedido: 010.01.2023) (vid. págs. 29, 61).

Nginx. *Nginx Documentation*. URL: http://nginx.org/en/docs/. (accedido: 010.01.2023) (vid. pág. 31).

Node.js. *Node JS Docs*. URL: https://nodejs.org/es/docs. (accedido: 13.01.2023) (vid. pág. 25).

Oracle. *1.4 About Oracle Flashback Technology*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/bradv/introduction-backup-recovery.html. (accedido: 11.02.2023) (vid. pág. 33).

— *Oracle Database Documentation*. URL: https://docs.oracle.com/en/database/oracle/oracle-database/index.html. (accedido: 09.01.2023) (vid. pág. 32).

— *¿Qué es una base de datos?* URL: https://www.oracle.com/co/database/what-is-database/. (accedido: 11.02.2023) (vid. pág. 32).

Project, Apache Maven. *Maven Documentation*. URL: https://maven.apache.org/guides/. (accedido: 010.01.2023) (vid. pág. 52).

Salesforce. *¿Qué son las metodologías ágiles y cómo pueden ayudar a tus equipos de trabajo?* URL: https://www.salesforce.com/mx/blog/2021/12/que-son-metodologias-agiles-y-como-pueden-ayudar-a-tus-equipos-de-trabajo.html#:~:text=Las%20metodologÃŋas%20Ãągiles%20no%20son,proyectos%20con%20rapidez%20y%20flexibilidad.. (accedido: 11.02.2023) (vid. pág. 40).

Semver. *Versionado Semántico*. URL: https://semver.org/lang/es/. (accedido: 11.01.2023) (vid. pág. 54).

Spring. *Spring Framework*. URL: https://spring.io/projects/spring-framework. (accedido: 10.01.2023) (vid. págs. 21, 52).

Spring. *Spring Security*. URL: https://docs.spring.io/spring-security/reference/index.html. (accedido: 11.02.2023) (vid. pág. 23).

Thalesgroup. *¿Qué es la seguridad del software y por qué es tan importante ahora?* URL: https://cpl.thalesgroup.com/es/software-monetization/what-is-software-security. (accedido: 11.02.2023) (vid. pág. 22).

Webempresa. *¿Qué es un servidor Web?* URL: https://www.webempresa.com/hosting/que-es-servidor-web.html. (accedido: 11.02.2023) (vid. pág. 30).

Wikipedia. *API*. URL: https://en.wikipedia.org/wiki/API. (accedido: 11.01.2023) (vid. pág. 13).

— *Framework*. URL: https://es.wikipedia.org/wiki/Framework. (accedido: 10.01.2023) (vid. págs. 19, 21, 25).

— *JSON*. URL: https://es.wikipedia.org/wiki/JSON. (accedido: 14.01.2023) (vid. pág. 20).

WildFly. *Wildfly Docs*. URL: https://docs.wildfly.org/27/. (accedido: 06.01.2023) (vid. pág. 20).

Yeeply. *¿Qué son las pruebas unitarias y cómo llevar una a cabo?* URL: https://www.yeeply.com/blog/que-son-pruebas-unitarias. (accedido: 11.02.2023) (vid. pág. 38).