UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CARRERA DE INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

Administración del sistema de gestión de la investigación para la UNACH mediante un orquestador API Gateway

Trabajo de Titulación para optar al título de Ingeniería en Tecnologías de la Información

Autor:
Guanoluiza Haro Roger David

Tutor:
Ing. Danny Patricio Velasco Silva, Mgs

Riobamba, Ecuador. ***2024***

DECLARATORIA DE AUTORÍA

Yo, Guanoluiza Haro Roger David, con cédula de ciudadanía 0604788752, autor (a) (s) del trabajo de investigación titulado: Administración del sistema de gestión de la investigación para la UNACH mediante un orquestador API Gateway, certifico que la producción, ideas, opiniones, criterios, contenidos y conclusiones expuestas son de mí exclusiva responsabilidad.

Asimismo, cedo a la Universidad Nacional de Chimborazo, en forma no exclusiva, los derechos para su uso, comunicación pública, distribución, divulgación y/o reproducción total o parcial, por medio físico o digital; en esta cesión se entiende que el cesionario no podrá obtener beneficios económicos. La posible reclamación de terceros respecto de los derechos de autor (a) de la obra referida, será de mi entera responsabilidad; librando a la Universidad Nacional de Chimborazo de posibles obligaciones.

En Riobamba, a los 13 días del mes de enero de 2025.

_____

Roger David Guanoluiza Haro

C.I: 0604788752

ACTA FAVORABLE - INFORME FINAL DEL TRABAJO DE INVESTIGACIÓN

En la Ciudad de Riobamba, a los 28 días del mes de octubre del 2024 luego de haber revisado el Informe Final del Trabajo de Investigación presentado por el estudiante **Roger David Guanoluiza Haro** con CC: 0604788752, de la carrera **Ingeniería en Tecnologías de la Información** y dando cumplimiento a los criterios metodológicos exigidos, se emite el **ACTA FAVORABLE DEL INFORME FINAL DEL TRABAJO DE INVESTIGACIÓN** titulado **"ADMINISTRACIÓN DEL SISTEMA DE GESTIÓN DE LA INVESTIGACIÓN PARA LA UNACH MEDIANTE UN ORQUESTADOR API GATEWAY"**, por lo tanto se autoriza la presentación del mismo para los trámites pertinentes.

Firmado electrónicamente por:
DANNY PATRICIO
VELASCO SILVA

Mgs. Danny Velasco Silva
TUTOR

CERTIFICADO DE LOS MIEMBROS DEL TRIBUNAL

Quienes suscribimos, catedráticos designados Miembros del Tribunal de Grado para la evaluación del trabajo de investigación Administración del Sistema de Gestión de la Investigación para La UNACH mediante un orquestador Api Gateway, presentado por Roger David Guanoluiza Haro con cédula de identidad número 0604788752, bajo la tutoría de Mgs. Danny Velasco Silva; certificamos que recomendamos la APROBACIÓN de este con fines de titulación. Previamente se ha evaluado el trabajo de investigación y escuchada la sustentación por parte de su autor; no teniendo más nada que observar.

De conformidad a la normativa aplicable firmamos, en Riobamba a los 16 días del mes de diciembre del 2024.

Mgs. María Isabel Uvidia
PRESIDENTE DEL TRIBUNAL DE GRADO

Ph.D. Fernando Molina
MIEMBRO DEL TRIBUNAL DE GRADO

Dr. Hugo Paz
MIEMBRO DEL TRIBUNAL DE GRADO

CERTIFICACIÓN

Que, **GUANOLUIZA HARO ROGER DAVID** con CC: **0604788752**, estudiante de la Carrera **INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN**, Facultad de INGENIERÍA; ha trabajado bajo mi tutoría el trabajo de investigación titulado "**ADMINISTRACIÓN DEL SISTEMA DE GESTIÓN DE LA INVESTIGACIÓN PARA LA UNACH MEDIANTE UN ORQUESTADOR API GATEWAY**", cumple con el 7 %, de acuerdo al reporte del sistema Anti plagio **TURNITIN**, porcentaje aceptado de acuerdo a la reglamentación institucional, por consiguiente autorizo continuar con el proceso.

Riobamba, 09 de diciembre de 2024

Firmado electrónicamente por:
DANNY PATRICIO
VELASCO SILVA

Mgs. Danny Velasco Silva
TUTOR

DEDICATORIA

Dedico este trabajo de investigación a mis padres, cuya constante dedicación y sacrificio han sido el faro que ha iluminado mi camino hacia el éxito académico. Su amor incondicional y apoyo inquebrantable han sido mi mayor fortaleza.

A mis respetados maestros, quienes con su sabiduría y guía han moldeado mi pensamiento crítico y ampliado mis horizontes académicos, les dedico este trabajo con profunda admiración y gratitud. Cada enseñanza y consejo recibido ha dejado una huella imborrable en mi formación profesional.

A todos los seres queridos y amigos que han estado a mi lado brindándome aliento y apoyo durante esta travesía académica, les ofrezco mi más sincero agradecimiento. Este logro no habría sido posible sin su confianza y aliento constante.

Con afecto y reconocimiento,

Roger Guanoluiza

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a mis padres, por su incansable apoyo y amor durante todo el proceso de realización de esta tesis. Su respaldo incondicional ha sido fundamental para mi éxito académico.

Agradezco también a mis estimados maestros, cuya guía experta y conocimientos compartidos han enriquecido mi desarrollo académico y profesional. Sus enseñanzas han sido una fuente invaluable de inspiración y aprendizaje a lo largo de este proyecto.

Este logro no habría sido posible sin la influencia positiva y el apoyo de mis padres y maestros. Estoy profundamente agradecido por cada palabra de aliento, consejo y dedicación que me han brindado.

Con gratitud,

Roger Guanoluiza

INDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

La Universidad Nacional de Chimborazo ha lanzado un proyecto para modernizar su infraestructura tecnológica mediante la implementación de microservicios respaldados por un API Gateway específicamente diseñado para el administrador del sistema de gestión de la investigación. Este enfoque estratégico tiene como objetivo mejorar la eficiencia operativa y la adaptabilidad del entorno académico frente a las demandas cambiantes del entorno digital. Los microservicios permiten una gestión flexible y escalable de las aplicaciones, mientras que el API Gateway actúa como un componente central para coordinar las interacciones entre estos servicios, asegurando coherencia, seguridad y un monitoreo efectivo del rendimiento del sistema.

El estudio realizado incluyó pruebas exhaustivas utilizando JMeter para comparar el rendimiento de los microservicios con y sin API Gateway en operaciones clave como POST, PATCH y GET. Los resultados destacaron mejoras significativas en el tiempo de respuesta y una reducción notable en el consumo de recursos como CPU y memoria cuando se implementa el API Gateway. Estos hallazgos subrayan la eficacia del API Gateway en optimizar la administración y operación del sistema de gestión de la investigación en la universidad.

Este proyecto no solo fortalece la capacidad de la universidad para mantenerse a la vanguardia en investigación académica, sino que también mejora la experiencia global de estudiantes, docentes y personal administrativo al proporcionar un sistema más eficiente y adaptable. La implementación exitosa del API Gateway en los microservicios del administrador sienta las bases para futuras innovaciones tecnológicas y garantiza una gestión eficiente y adaptativa del sistema de gestión de la investigación en la Universidad Nacional de Chimborazo.

Palabras claves: Universidad Nacional de Chimborazo, Microservicios, API Gateway, Gestión de la investigación, Rendimiento, FURPS, JMeter.

ABSTRACT

The National University of Chimborazo has embarked on a strategic project to modernize its technological infrastructure. This initiative involves the implementation of microservices, supported by an API Gateway specifically designed for the research management system administrator. The strategic approach is aimed at significantly enhancing operational efficiency and adaptability to the evolving digital demands of the academic environment. Microservices enable flexible and scalable application management, while the API Gateway serves as a central component to coordinate interactions among these services, ensuring consistency, security, and effective system performance monitoring. The study, which included comprehensive testing using JMeter, demonstrated the significant success of the project. The performance of microservices with and without an API Gateway in key operations such as POST, PATCH, and GET was compared. The results highlighted substantial improvements in response time and a notable reduction in resource consumption, such as CPU and memory, when the API Gateway was implemented. These findings underscore the success of the API Gateway in optimizing the management and operation of the university's research management system. This project significantly enhances the National University of Chimborazo's ability to lead in academic research and improves the overall experience for students, faculty, and administrative staff. By providing a more efficient and adaptable system, the project ensures that all stakeholders can benefit from the latest technological advancements. The successful implementation of the API Gateway within the microservices architecture not only guarantees the efficient and adaptive management of the university's research management system but also sets the stage for future technological innovations.

Keywords: National University of Chimborazo, microservices, API gateway, research management, performance, FURPS, JMeter.

Reviewed by:
Mgs. Kerly Cabezas
ENGLISH PORFESSOR
I.D. 0604042382

CAPÍTULO I. INTRODUCCION

En la búsqueda constante de la excelencia académica y la eficiencia operativa, la Universidad Nacional de Chimborazo se enfrenta al desafío de modernizar su infraestructura tecnológica para adaptarse a las demandas cambiantes del entorno digital. En este contexto, se propone un proyecto de desarrollo de la administración para el sistema de gestión de la investigación con especial atención en la Integración Estratégica de un API Gateway en Microservicios, con el uso de JetBrains.

La implementación de microservicios ofrece la oportunidad de transformar la manera en que se conciben y gestionan las aplicaciones, permite una mayor flexibilidad, escalabilidad y agilidad en el desarrollo y despliegue de servicios. La adopción de un API Gateway se presenta como un componente esencial para centralizar y coordinar las interacciones entre microservicios, asegura la coherencia, la seguridad y el monitoreo efectivo de la arquitectura y su rendimiento.

Este enfoque estratégico no solo tiene como objetivo potenciar la capacidad de la Universidad Nacional de Chimborazo para mantenerse a la vanguardia en el ámbito académico, sino también mejorar significativamente la experiencia de estudiantes, docentes y personal administrativo. La sinergia entre la integración de un API Gateway y la aplicación de microservicios se traduce en una administración más eficiente, permite una rápida adaptación a las cambiantes necesidades tecnológicas y garantiza la continuidad de la innovación.

La gestión de la investigación es crucial para el progreso científico y tecnológico. Un sistema basado en microservicios ofrece una solución escalable. Cada microservicio, como semilleros, proyectos, centro, configuración, SICOA y grupos, cuenta con su propia API. Semilleros facilita la creación y administración de grupos de investigación. Proyectos permite planificar, seguir y ejecutar investigaciones. Centro gestiona la información institucional y administrativa. Configuración personaliza el sistema según las necesidades. SICOA supervisa la asistencia del personal investigador. Grupos administra la afiliación en equipos de investigación. Estos microservicios forman una infraestructura sólida para la gestión integral de la investigación, así fomenta la colaboración y la eficiencia.

1.1 Planteamiento del problema

El planteamiento del problema se centra en la necesidad de implementar un sistema de administración con el uso de microservicios respaldados por un API Gateway para mejorar la eficiencia y adaptabilidad de la Universidad Nacional de Chimborazo. Aunque se ha desarrollado un backend con microservicios, la falta de una solución de administración integral ha generado desafíos en la gestión de recursos y procesos universitarios. Por lo tanto, se busca integrar un sistema de administración basado en microservicios, para aprovechar las ventajas de escalabilidad y flexibilidad que ofrecen.

La incorporación de un API Gateway proporcionará una interfaz centralizada para la gestión y coordinación de estos microservicios, facilita la sincronización rápida y precisa entre el backend y el frontend del sistema. Este enfoque permitirá mejorar el rendimiento del sistema de administración, asegura una respuesta ágil a las demandas tecnológicas y académicas cambiantes. Se evaluará el rendimiento con JMETER. Este proyecto busca superar las deficiencias actuales en la infraestructura tecnológica de la universidad y proponer soluciones innovadoras para mejorar su operatividad y adaptabilidad continua.

1.2 Justificación

La implementación de microservicios respaldados por un API Gateway en la Universidad Nacional de Chimborazo representa una estrategia clave para modernizar nuestra infraestructura tecnológica y adaptarnos eficazmente al entorno digital en constante cambio. Este enfoque ofrece beneficios significativos en términos de flexibilidad, escalabilidad y rendimiento en el desarrollo de servicios, particularmente en la gestión de la investigación, una función crucial para nuestra institución. La integración de un API Gateway asegurará una coordinación efectiva entre los microservicios, mejorando la coherencia, la seguridad y el monitoreo del rendimiento del sistema. Esta iniciativa no solo busca potenciar nuestra capacidad operativa, sino también promover la innovación continua en nuestra universidad.

1.3 Formulación del problema

¿Como la implementación del API Gateway en los microservicios incidirá en el rendimiento de la administración del sistema de gestión de la investigación en la Universidad Nacional De Chimborazo?

1.4 Objetivos

Objetivos Generales

Implementar el administrador para el sistema de gestión de la investigación mediante un orquestador API Gateway para optimizar los procesos de la Universidad Nacional de Chimborazo.

Objetivos Específicos

- Investigar la implementación de API Gateway en Microservicios para encaminar las solicitudes de API entre backend y frontend
- Implementar API Gateway en microservicios para la administración del sistema de gestión de la investigación en la UNACH
- Evaluar el rendimiento del administrador del sistema de gestión de la investigación para la UNACH con el uso del MODELO FURPS

CAPÍTULO II. MARCO TEÓRICO

2.1 API Gateway

API Gateway actúa como intermediario entre el cliente y los microservicios, proporcionando un entorno de red privado que permite el intercambio de datos privados [1,2] es decir, los clientes no se comunican directamente con los servicios, pero sólo con un Gateway, que se encarga de comunicar con el servicio solicitado. Puede ser una entrada que realiza el filtrado del cliente. solicitudes, realizando el reenvío correspondiente al microservicio [3], y puede verifique las credenciales del usuario para saber si posee la autorización adecuada [4,5].
Los casos de uso del API Gateway en las empresas se detalla en la figura 1.

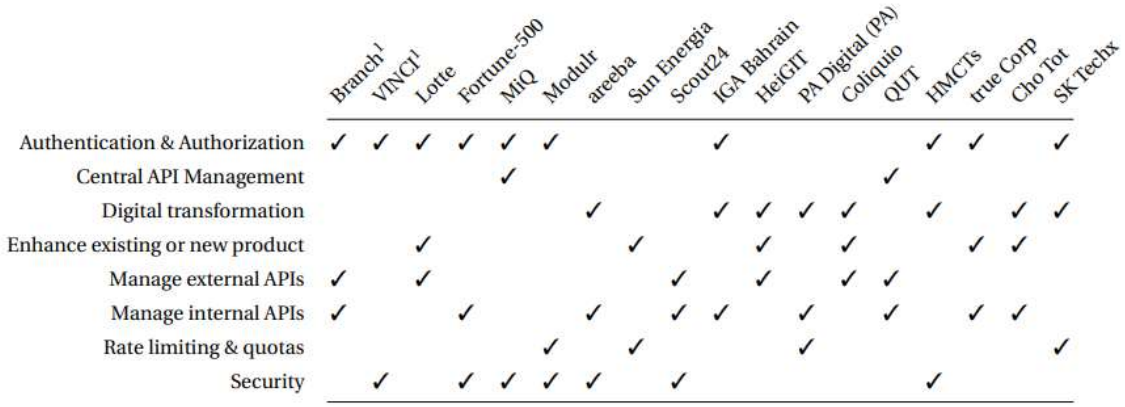


Figura 1: Casos de uso de API Gateway utilizados por las respectivas empresas
Fuente:[1]

2.2 FURPS

Reúne varios de los factores de calidad como funcionalidad, usabilidad, confiabilidad, desempeño y soportabilidad, de ahí el acrónimo FURPS. Este modelo fue desarrollado por Hewlett-Packard en 1987 [6].
La tabla 1 referencia al modelo de calidad FURPS

Tabla 1: Factores asociados al modelo de calidad FURPS

Sigla	**Parámetros**	
f	Funcional	**Características** Aspectos de seguridad. Capacidades.
u	Facilidad de Uso	**Factores Humanos.** Ayuda. Documentación.
r	Fiabilidad	**Frecuencia de fallos** Capacidad de recuperación de fallo. Grado de previsión.
p	Rendimiento	**Eficacia.**

		Velocidad de procesamiento. Tiempo de respuesta. Utilización de recursos.
s	Soporte	**Adaptabilidad.** Mantenimiento. Facilidad de configuración.
	Implementación	**Limitaciones de recursos.** Lenguajes y herramientas. Hardware.
	Interfaz	Restricciones impuestas para la interacción con el sistema.
	Operaciones	**Gestión del sistema.** Pautas administrativas. Puesta en marcha.
	Empaquetamiento legales	Forma de distribución. **Licencias.** Derechos de autor.

Fuente: [6]

En este modelo, en cada etapa de desarrollo se propone una serie de pruebas antes de su producción y comercialización, además cuenta con un plan de soporte definido con todos los errores encontrados y registrados en una base de datos con la finalidad de poder enmendarlos a futuro [7].

2.3 .Net

Nace como la solución a la incompatibilidad de las páginas web, servidores, en si la tecnología de soporte disponible en la red de internet e independientes al a la arquitectura física y los sistemas operativos en que se ejecute. Esto, con los requisitos finalidad de fusionar su amplio catálogo de productos, librerías y más herramientas de desarrollo para hacer frente a la plataforma java [8].

Siendo una plataforma que integra múltiples herramientas, brinda la facilidad de desarrollar tanto aplicativos webs, así como programas de escritorio o aplicaciones móviles.

Permitiendo el desarrollo multiplataforma, como por ejemplo el que una misma aplicación pueda correr indistintamente en los diferentes sistemas operativos tanto de escritorio como móviles, garantizando la comunicación entre los diferentes dispositivos [9].

.Net es una infraestructura que integra múltiples herramientas centrándose en las aplicaciones del lado del servidor para mejorar el servicio al cliente, es decir, a través de la red permite integrar múltiples aplicaciones web [10].

2.3.1 Evolución de .NET

Net está integrado por una seria de estructuras y tecnologías conocidos como framework que ayudan al programador haciendo las tareas más sencillas, la informática apareció como monousuarios, es decir, programas de un solo usuario en grandes servidores, a futuro estas aplicaciones evolucionaron permitiendo el uso a más de un usuario, hasta alcanzar la arquitectura cliente-servidor dando un espacio de memoria al usuario y otro espacio al procesamiento de la información para finalmente poder visualizar en varios ordenadores a la vez [10].
La siguiente figura 2 muestra la evolución de .NET

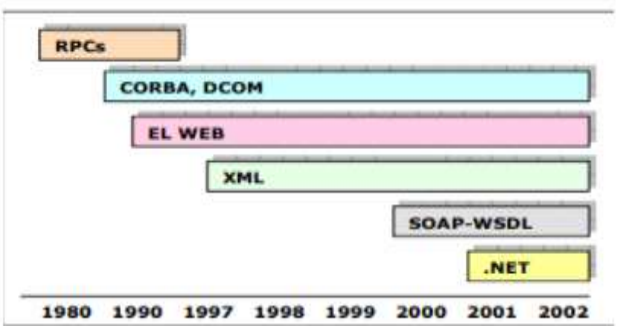


Figura 2: Evolución de .Net

Fuente: [8]

2.4 SQL Server

Es un sistema de gestión de bases de datos relacionales (RDBMS) que se emplea en ámbitos empresariales para diversas funciones como el procesamiento de transacciones, la inteligencia empresarial y el análisis. Dentro del mercado de bases de datos, Microsoft SQL Server es una de las tres tecnologías principales, junto con Oracle Database y DB2 de IBM [11].

2.5 JetBrains Rider

Esta plataforma ofrece la capacidad de codificar IDs de desarrollo en .NET. JetBrains Rider, un potente y veloz editor de C#, compatible con Unity en sistemas operativos Windows, Mac y Linux, enriquece la experiencia de programación con C#. Con más de 2500 inspecciones

y refactorizaciones de código inteligente, permite una escritura de código más rápida y segura [12].

2.6 Azure DevOps

Azure DevOps es un término que abarca todo el ciclo de vida del desarrollo, desde la fase de planificación y codificación hasta las de pruebas, implantación y operaciones.
Azure DevOps ofrece una amplia gama de funcionalidades que resultan fundamentales para equipos de desarrollo en la optimización de sus procesos. Desde la gestión del código fuente en repositorios como Git o TFVC hasta el seguimiento detallado de tareas, problemas y requisitos del proyecto, cada aspecto del desarrollo se ve respaldado. La automatización de procesos como la compilación, prueba y despliegue de aplicaciones garantiza una entrega eficiente y confiable. Además, las herramientas para pruebas automatizadas fortalecen la calidad y la confiabilidad del software. Azure DevOps no solo facilita la colaboración efectiva entre equipos, sino que también abarca todo el ciclo de vida del desarrollo de software, desde la planificación hasta la monitorización del despliegue.

2.7 Kanban

Dentro del amplio espectro de metodologías ágiles, se encuentra la metodología Kanban, la cual se implementa a través de tableros Kanban que visualizan el trabajo del proyecto organizado por columnas [13]. Esta metodología se centra en el trabajo en equipo y promueve un flujo de trabajo eficiente [14]. La visualización del trabajo en curso es fundamental en Kanban, ya que proporciona a los equipos una comprensión clara de sus actividades actuales [15]. El flujo de trabajo, definido como el mecanismo a través del cual las personas y las empresas realizan su trabajo de manera efectiva, es esencial para entender el funcionamiento de Kanban [16]. Este enfoque ágil se destaca por su capacidad para ofrecer valor continuo y previsible a través de la gestión eficiente del trabajo en progreso [17]. La aplicación de Kanban en equipos de desarrollo de software puede mejorar significativamente la transparencia, la comunicación y la colaboración [18]. Además, la adopción de Kanban ha demostrado aumentar la eficiencia y la productividad al reducir el tiempo dedicado a tareas improductivas [19]. En síntesis, Kanban es una metodología ágil poderosa que seenfoca en optimizar los flujos de trabajo y ofrecer valor continuo a los clientes. Su implementación puede mejorar la eficiencia, la transparencia y la colaboración en equipos de desarrollo de software, lo que lo convierte en una opción atractiva para mejorar la gestión de proyectos en este campo [20].

2.8 JMETER

JMETER es una herramienta de prueba de software de código abierto desarrollada por Apache Software Foundation, utilizada para medir el rendimiento y probar la funcionalidad de aplicaciones web. Permite simular cargas pesadas en servidores, redes y aplicaciones web

para analizar su rendimiento y funcionalidad. Es compatible con diversos protocolos como HTTP, HTTPS, SOAP, FTP, JDBC, LDAP, JMS y más, lo que la hace versátil para diferentes tipos de pruebas. Además, JMETER puede realizar tanto pruebas de rendimiento como pruebas funcionales, asegurando que las aplicaciones funcionen correctamente bajo diversas condiciones.

CAPÍTULO III. METODOLOGÍA

En el estudio de investigación, se utilizó un método cuantitativo, dado que posibilitaba la obtención de datos precisos. En esta etapa, se llevó a cabo una evaluación de desempeño empleando el marco de calidad FURPS para evaluar el funcionamiento del API Gateway.

3.1 Técnica de Investigación

En el estudio de investigación, se utilizó un método cuantitativo, dado que posibilitaba la obtención de datos precisos. En esa etapa, se llevó a cabo una evaluación de desempeño empleando el marco de calidad FURPS para evaluar el funcionamiento del API Gateway.

3.2 Diseño de investigación

Se empleó un método analítico para establecer principios, reglas, características y funcionamiento mediante la implementación de un tablero Kanban. Se utilizo azure devops como herramienta para gestionar las actividades en el desarrollo del proyecto. Se utilizó un tipo de investigación documental bibliográfica, recopilando datos de diversas fuentes como libros, artículos científicos y tesis doctorales.

3.3 Técnicas de recolección de Datos

Se utilizo la herramienta Apache JMETER para la recopilación de datos.

3.4 Población de estudio y tamaño de muestra

Población: La población de estudio incluye a los posibles administradores del sistema administrativo de investigación de la Universidad Nacional de Chimborazo (UNACH). Dado que el sistema está diseñado para ser utilizado principalmente por un único administrador en cada momento, la población potencial se estima en 25 personas, considerando cambios en la asignación del puesto a lo largo de los años.

Muestra: Dado que el sistema está destinado a un único usuario activo por vez, se realizó una simulación con 25 usuarios representativos para analizar su rendimiento. Este enfoque permite evaluar el comportamiento del sistema bajo diferentes configuraciones de microservicios, simulando interacciones típicas y considerando posibles escenarios de carga y concurrencia. Las pruebas fueron diseñadas para reflejar tanto el uso real como situaciones de estrés controlado.

3.5 Identificación de las Variables:

Variable Dependiente:
Implementación del API Gateway en los microservicios.

Variable Independiente:
Evaluar el rendimiento del sistema de gestión de la investigación.

3.6 Operacionalización de variables

La siguiente tabla 2 muestra la operacionalización de variables

Tabla 2: Operacionalización de Variables

Problema	Tema	Objetivos	Variables,	Conceptualización	Dimensión	Indicadores
¿Como la implementación del API Gateway en los microservicios incidirá en el rendimiento de la administración del sistema de gestión de la investigación en la Universidad Nacional De Chimborazo?	Administración del sistema de gestión de la investigación para la UNACH mediante un orquestador API gateway.	**General** • Implementar el administrador para el sistema de gestión de la investigación para la Universidad Nacional De Chimborazo mediante un orquestador API gateway.	**Dependiente** Implementación del API Gateway en los microservicios.	API Gateway: Una interfaz que agrega y simplifica múltiples servicios y funcionalidades en un solo punto de acceso, mejora la eficiencia y seguridad en el intercambio de datos.	Arquitectura	**Independiente**: • Tiempo de desarrollo • Número de líneas de código • Número de funcionalidades
		Específicos • Investigar la implementación de API Gateway en Microservicios para encaminar las solicitudes de API entre Back End y frontend • Implementar API Gateway en microservicios para la administración del sistema de gestión de la investigación para la UNACH • Evaluar la Rendimiento del administrador del sistema de gestión de la investigación para la UNACH con el uso del MODELO FURPS	**Independiente** Evaluar el rendimiento del sistema de gestión de la investigación.	• Rendimiento se refiere a la eficiencia y capacidad de respuesta de la aplicación en relación con la experiencia del usuario y la capacidad de manejar cargas de trabajo específicas. • API Gateway pertenece al ámbito de la Ingeniería de Software y se encarga de facilitar la comunicación entre distintos servicios y aplicaciones, permite una integración eficiente y segura de los datos y funciones que ofrecen dichos servicios. • Los microservicios son una arquitectura de desarrollo de software que estructura una aplicación como un conjunto de servicios pequeños e independientes, cada uno ejecuta un proceso único y específico. Estos servicios están diseñados para funcionar de manera autónoma y comunicarse entre sí a través de protocolos ligeros	Rendimiento	**Dependiente**: • % Eficacia • % Tiempo de respuesta • % Utilización de recursos

3.7 Desarrollo

Se uso la metodología Kanban para el desarrollo del sistema de administración para la gestión de la investigación

En la tabla 3 muestra como es la metodología Kanban

Tabla 3: Metodología y sus fases

Fases	METODOLOGÍA KANBAN
Por hacer	En esta fase se encuentran todas las tareas que han sido identificadas, pero aún no han sido iniciadas. Representa el estado de espera para que las tareas sean seleccionadas para su ejecución. Las tareas en esta columna están en la cola de entrada del sistema y pueden ser priorizadas según la política de gestión de flujo.
En progreso	En esta fase se ubican las tareas que están actualmente en ejecución. Aquí, el equipo de trabajo se enfoca en completar estas tareas de acuerdo con las políticas de trabajo establecidas. Es importante limitar el número de tareas en esta fase para evitar la sobrecarga de trabajo y garantizar una ejecución eficiente.
Terminado	Esta fase alberga las tareas que han sido completadas exitosamente y están listas para ser entregadas o implementadas. Las tareas que alcanzan esta etapa han pasado por el proceso de trabajo y cumplen con los criterios de finalización establecidos. Se deben aplicar técnicas de control de calidad para garantizar que las tareas cumplen con los estándares requeridos antes de moverlas a esta columna.

3.7.1 Inicio

En esta etapa se identificó al personal involucrado, lo que permitió establecer las necesidades para llevar a cabo el proyecto.

En la siguiente tabla 4 muestra el equipo de desarrollo:

Tabla 4: Equipo de desarrollo

Rol	**Asistentes**
Product Owner	Ing. Alex Asitimbay
Team	Roger Guanoluiza

Requerimientos funcionales

Los requerimientos funcionales se recopilaron en el departamento de Dirección de Investigación de la UNACH (ICIT), a continuación, en la Tabla 5 se detalla los requerimientos recopilados.

Tabla 5: Requerimientos funcionales

Identificación del requerimiento	**RF01**
Nombre del requerimiento	Gestionar el CRUD y asignación de usuarios dentro de la plataforma
Descripción del requerimiento	El administrador tendrá la opción de elegir rol y funcionalidad para después proceder a crear al usuario, podrá editar, leer y eliminar,
Prioridad del requerimiento	Alta
Identificación del requerimiento	**RF02**
Nombre del requerimiento	Seguimiento del personal académico
Descripción del requerimiento	El administrador verificara la persona cuenta con los requisitos para proceder a incluirlo al sistema.
Prioridad del requerimiento	Alta
Identificación del requerimiento	**RF03**
Nombre del requerimiento	Monitoreo y Gestión de usuarios
Descripción del requerimiento	El administrador podrá revisar los usuarios que existen además de su rol y funcionalidad.
Prioridad del requerimiento	Alta

Requerimientos no funcionales

Los requerimientos no funcionales describen los criterios de desempeño del sistema, abarcando aspectos como seguridad, disponibilidad, compatibilidad, escalabilidad, entre otros. Tal como se presenta en la Tabla 6, estos requisitos se identifican de manera específica para garantizar que el sistema no solo cumpla con sus funciones operativas, sino que también lo haga de manera eficiente y segura.

Tabla 6: Requerimientos no funcionales

Requerimiento	**Descripción del requerimiento**	**categoría**
RNF01	La habilidad del sistema para identificar, reportar y gestionar errores de manera eficiente, garantizando la integridad y la operación ininterrumpida del sistema.	Control de errores
RNF02	El diseño y la funcionalidad de las interfaces que interactúan con los usuarios,	Interfaz del usuario

	enfocándose en la accesibilidad y la facilidad de uso, garantizan una experiencia intuitiva, coherente y adaptable a diversos dispositivos y requerimientos.	
RFN03	La facilidad con la que se puede mantener y actualizar el sistema sin afectar su funcionamiento, garantizando un rendimiento y una disponibilidad consistentes a lo largo del tiempo.	Confiabilidad
RFN04	Este protocolo salvaguarda la confidencialidad y la integridad de los datos mediante la aplicación de algoritmos criptográficos para encriptar la información delicada.	Disponibilidad
RFN5	Se centran en analizar la rapidez de ejecución, la capacidad de respuesta frente a distintos niveles de carga de trabajo y la eficacia en la gestión de recursos, todo ello con el fin de asegurar un funcionamiento fluido y continuo del sistema.	Rendimiento

3.7.2 Planificación y estimación

En esta faceta, se desarrolló un plan detallado para el proyecto de investigación.

Alcance

Realiza un inventario detallado de las acciones planificadas para completar el proyecto. Las tareas se dividen en Historias Técnicas (HT) e Historias de Usuario (HU), cada una con una numeración específica. Además, se incluyó una estimación del esfuerzo requerido para cada tarea calificada entre 1 y 3.
En la siguiente tabla 7 muestra el inventario:

Tabla 7: alcance

Ítem	Tarea	Esfuerzo
HT-01	Examinar los requisitos funcionales y no funcionales del componente.	3
HT-02	Establecer la estructura básica del componente.	2
HT-03	Elaborar el diseño de la base de datos, tanto en su aspecto relacional como físico.	2
HT-04	Configurar y preparar las herramientas necesarias para el desarrollo del componente.	1
HU-01	Implementar la base de datos según el diseño previamente establecido.	2
HU-02	Desarrollar la sección de dominio correspondiente al componente.	2
HU-03	Crear la infraestructura requerida para el correcto funcionamiento del componente.	2
HU-04	Implementar el API Gateway en los microservicios para el administrador	3
HU-05	Evaluar el proyecto utilizando el modelo de calidad FURPS.	2
HU-06	Analizar los resultados obtenidos durante la evaluación.	1
HT-05	Concluir las pruebas del API Gateway para el administrador	2

En la siguiente tabla 8 está el cronograma a seguir para cumplir el proyecto

Tabla 8: Cronograma

N°	Actividades	Semanas																
		Mes 1				Mes 2				Mes 3				Mes 4				
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
	SPRINT 1																	
HT-01	Examinar los requisitos funcionales y no funcionales del componente.	X																
HT-02	Establecer la estructura básica del componente.		X															
HT-03	Elaborar el diseño de la base de datos, tanto en su aspecto relacional como físico.			X														
HT-04	Configurar y preparar las herramientas necesarias para el desarrollo del componente.				X													
HU-01	Implementar la base de datos según el diseño previamente establecido.					X												
HU-02	Desarrollar la sección de dominio correspondiente al componente.						X											
HU-03	Crear la infraestructura requerida para el correcto funcionamiento del componente.							X										
HU-04	Implementar el API Gateway en los microservicios para el administrador								X	X	X	X	X					
HU-05	Evaluar el proyecto utilizando el modelo de calidad FURPS.													X				
HU-06	Analizar los resultados obtenidos durante la evaluación.													X				
HT-05	Concluir las pruebas del API Gateway para el administrador														X	X	X	

3.7.3 Implementación casos de uso

Durante esta fase, se dio inicio al proceso de desarrollo del API Gateway empleando la arquitectura de microservicios y se optó por utilizar JetBrains Rider como entorno de desarrollo integrado (IDE).

Diagramas de casos de uso

En la figura 3 se muestra como el administrador se encargará de gestionar los proyectos y crear a cada usuario según su criterio

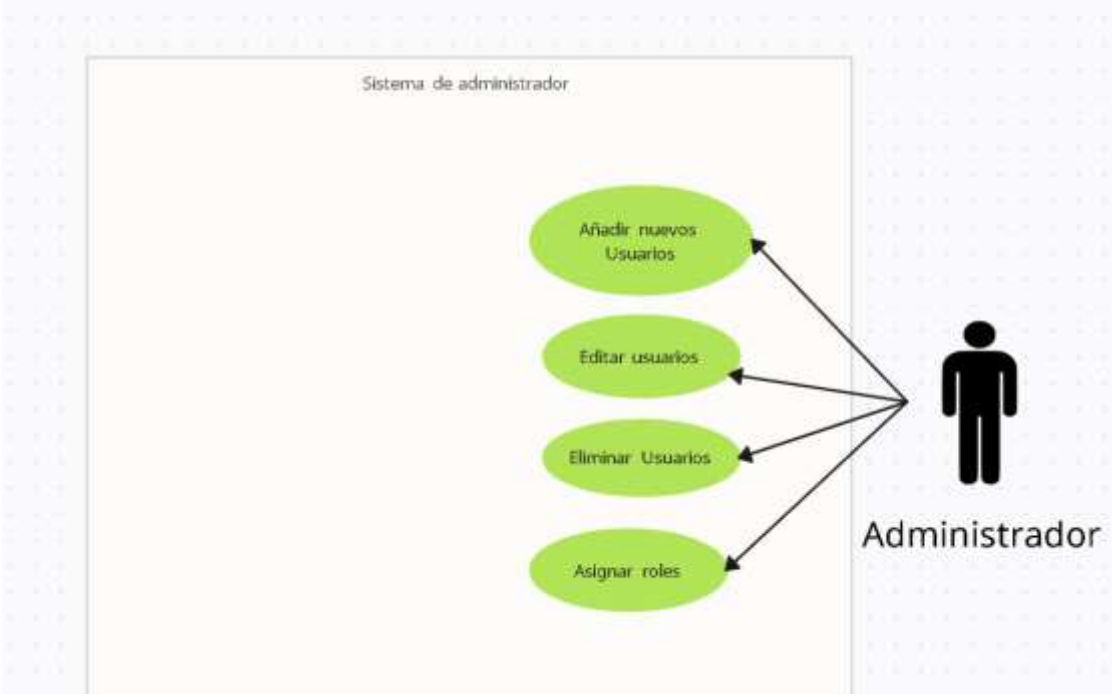


Figura 3: Diagrama de caso de uso

En el caso de uso de la figura 4 muestra como el administrador creara al personal según el criterio correcto.

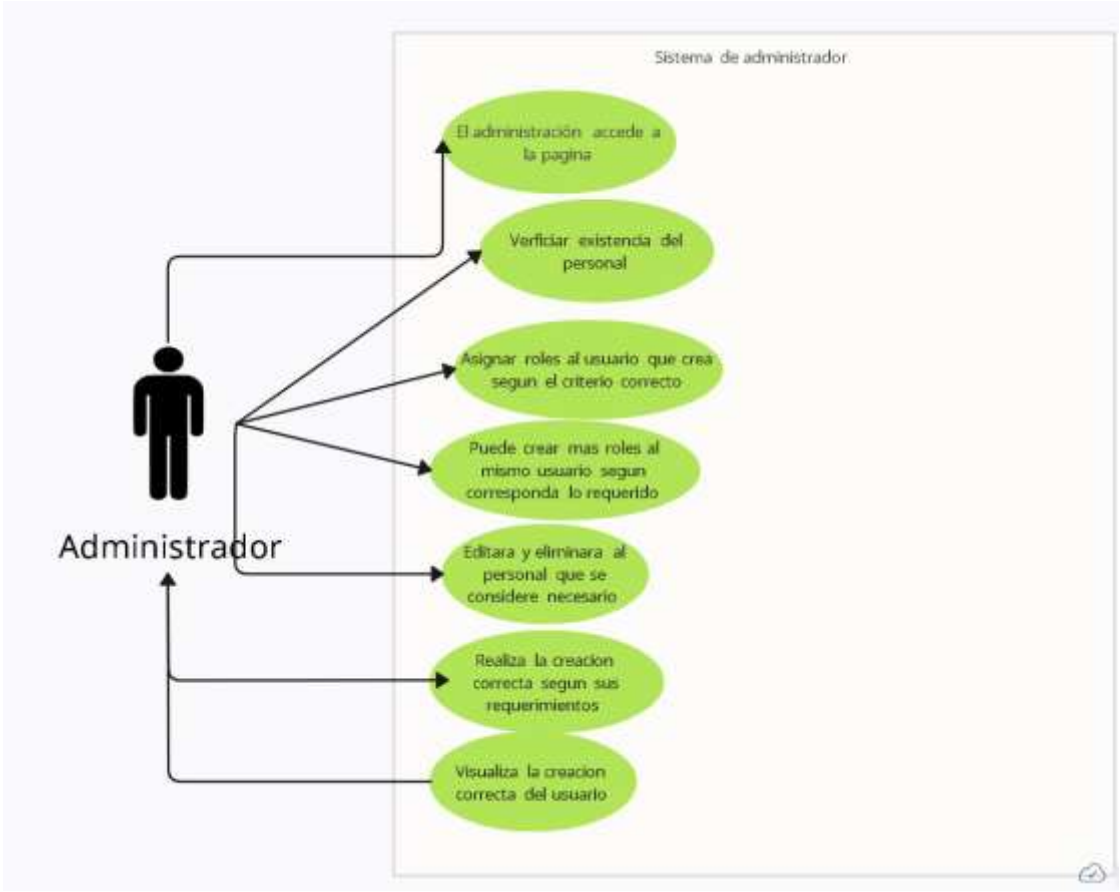


Figura 4: Diagrama caso de uso administrador

En este caso de uso esta como seria el sistema de gestión de la investigación como muestra la figura 5:

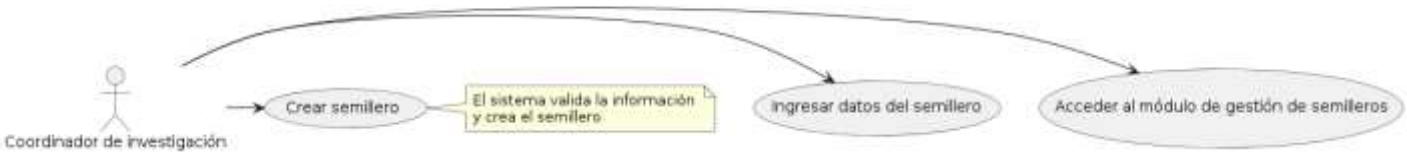
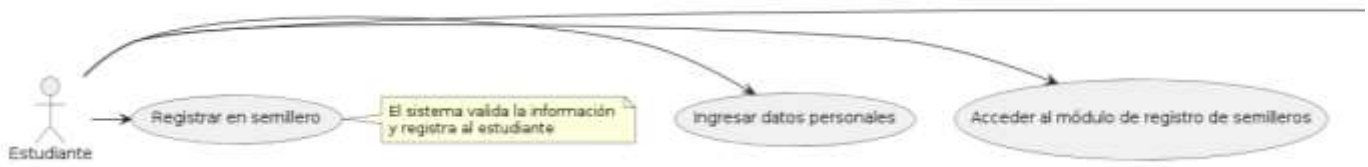


Figura 5: Gestión de semilleros



El siguiente caso de uso hace referencia al investigador como rol, como muestra la figura 6:

Figura 6: Investigador Principal

Caso de uso administrador, en la siguiente figura 7 se muestra el caso:

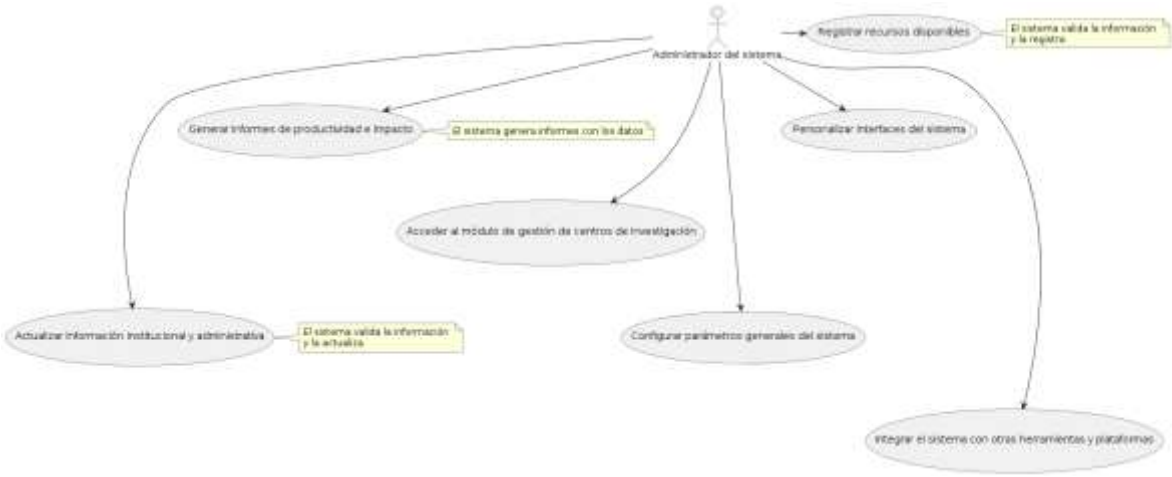


Figura 7: Administrador del sistema

Se realiza la selección de la arquitectura para el desarrollo del sistema, que se muestra en la siguiente imagen.
A continuación, en la Figura 8 se observa la arquitectura empleada en el proyecto.

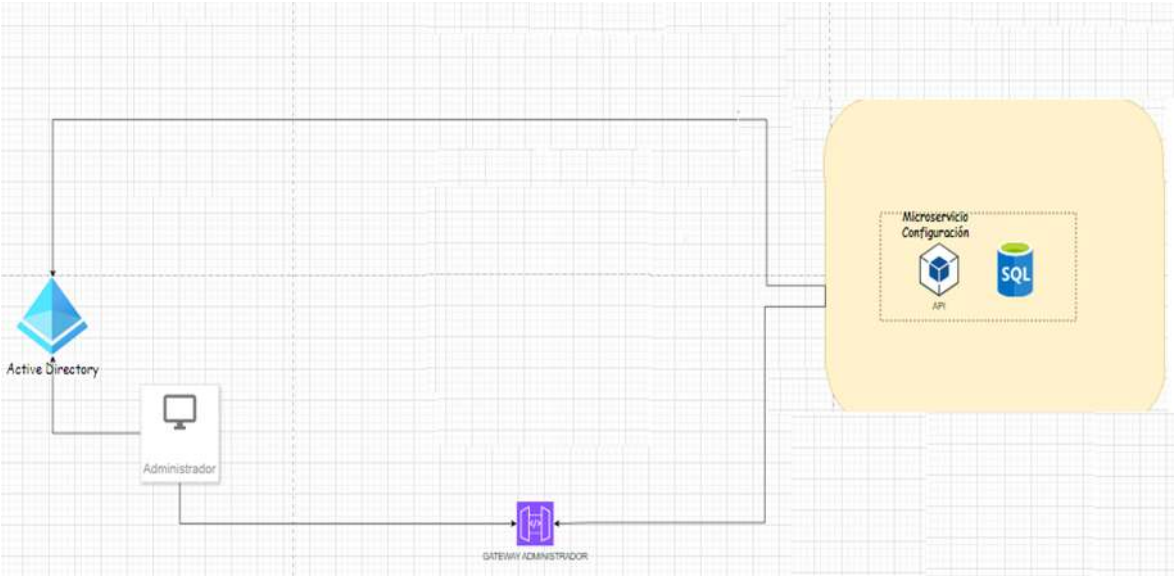


Figura 8: Arquitectura principal

3.7.4 Diagrama Relacional

Este enfoque se fundamenta en la utilización de tablas para representar datos, donde cada tabla corresponde a una entidad y cada fila representa una instancia única de esa entidad. Las claves primarias y foráneas establecen las relaciones entre estas entidades, facilitando así una conexión coherente y significativa entre los datos almacenados en diferentes tablas.
La siguiente figura 9 muestra la relación en las tablas de la base de datos.

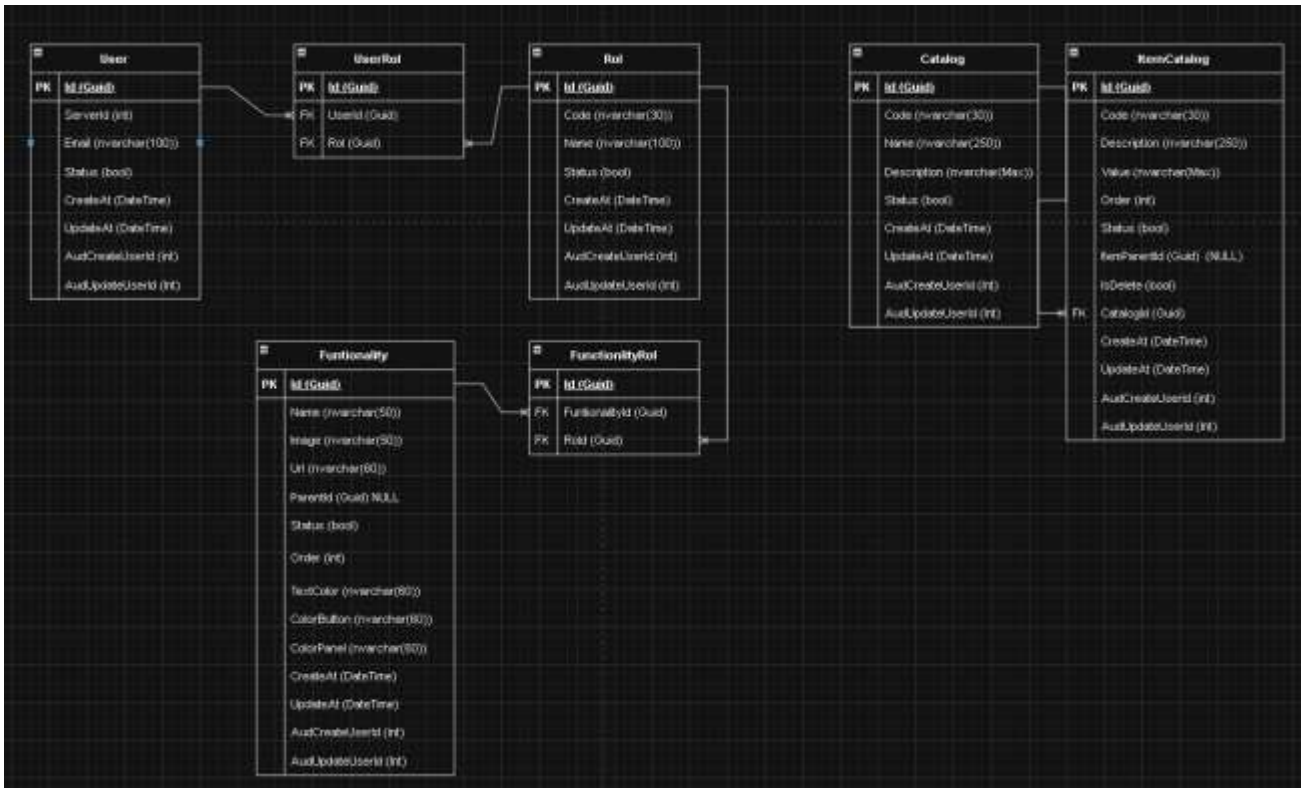


Figura 9: Modelo de tablas

En las siguientes tablas de muestra el diccionario de datos

3.8 Diccionario de datos

La tabla 9 muestra las variables de User

Tabla 9: User

Tabla: User

Campo	Tipo	Descripción
Id	Guid	Identificador único del usuario
ServerId	int	Identificador del servidor
Email	nvarchar(100)	Correo electrónico del usuario
Status	bool	Estado del usuario (activo/inactivo)
CreateAt	DateTime	Fecha de creación del usuario
UpdateAt	DateTime	Fecha de última actualización del usuario
AudCreateUserId	int	Identificador del usuario que creó este registro

AudUpdateUserId	int	Identificador del usuario que actualizó este registro

La tabla 10 muestra las variables de UserRol

Tabla 10: UserRol

Tabla: UserRol

Campo	Tipo	Descripción
Id	Guid	Identificador único del UserRol
UserId	Guid	Identificador del usuario
RolId	Guid	Identificador del rol

La tabla 11 muestra las variables de Rol

Tabla 11: Rol

Tabla: Rol

Campo	Tipo	Descripción
Id	Guid	Identificador único del rol
Code	nvarchar(30)	Código del rol
Name	nvarchar(100)	Nombre del rol
Status	bool	Estado del rol (activo/inactivo)
CreateAt	DateTime	Fecha de creación del rol
UpdateAt	DateTime	Fecha de última actualización del rol
AudCreateUserId	int	Identificador del usuario que creó este registro
AudUpdateUserId	int	Identificador del usuario que actualizó este registro

La tabla 12 muestra las variables de Catalog

Tabla 12: Catalog

Tabla: Catalog

Campo	Tipo	Descripción
Id	Guid	Identificador único del catálogo
Code	nvarchar (30)	Código del catálogo
Name	nvarchar (250)	Nombre del catálogo
Descripción	nvarchar (Max)	Descripción del catálogo
Status	bool	Estado del catálogo (activo/inactivo)
CreateAt	DateTime	Fecha de creación del catálogo
UpdateAt	DateTime	Fecha de última actualización del catálogo
AudCreateUserId	int	Identificador del usuario que creó este registro
AudUpdateUserId	int	Identificador del usuario que actualizó este registro

La tabla 13 muestra las variables de ItemCatalog

Tabla 13: ItemCatalog

Tabla: ItemCatalog

Campo	Tipo	Descripción
Id	Guid	Identificador único del ítem del catálogo
Code	nvarchar (30)	Código del ítem
Name	nvarchar (250)	Nombre del ítem
Description	nvarchar (Max)	Descripción del ítem
Value	nvarchar (Max)	Valor del ítem
Order	int	Orden del ítem dentro del catálogo
Status	bool	Estado del ítem (activo/inactivo)
ItemParentId	Guid (NULL)	Identificador del ítem padre, si existe
IsDelete	bool	Indicador de eliminación lógica
CatalogId	Guid	Identificador del catálogo al que pertenece el ítem
CreateAt	DateTime	Fecha de creación del ítem
UpdateAt	DateTime	Fecha de última actualización del ítem
AudCreateUserId	int	Identificador del usuario que creó este registro
AudUpdateUserId	int	Identificador del usuario que actualizó este registro

La tabla 14 muestra las variables de Functionality

Tabla 14: Functionality

Tabla: Functionality

Campo	Tipo	Descripción
Id	Guid	Identificador único de la funcionalidad
Name	nvarchar (50)	Nombre de la funcionalidad
Image	nvarchar (50)	Imagen asociada a la funcionalidad
Url	nvarchar (60)	URL de la funcionalidad
ParentId	Guid (NULL)	Identificador de la funcionalidad padre, si existe
Status	bool	Estado de la funcionalidad (activo/inactivo)
Order	int	Orden de la funcionalidad
TextColor	nvarchar (60)	Color del texto de la funcionalidad
ColorButton	nvarchar (60)	Color del botón de la funcionalidad
ColorPanel	nvarchar (60)	Color del panel de la funcionalidad
CreateAt	DateTime	Fecha de creación de la funcionalidad
UpdateAt	DateTime	Fecha de última actualización de la funcionalidad
AudCreateUserId	int	Identificador del usuario que creó este registro
AudUpdateUserId	int	Identificador del usuario que actualizó este registro

La tabla 15 muestra las variables de FunctionalityRol

Tabla 15: FunctionalityRol

Tabla: FunctionalityRol

Campo	Tipo	Descripción
Id	Guid	Identificador único de la relación Functionality-Rol
FunctionalityId	Guid	Identificador de la funcionalidad
RolId	Guid	Identificador del rol

3.8.1 Navegabilidad

En la siguiente figura 10 se muestra la página principal del administrador

Figura 10: Página principal del administrador

La figura 11 muestra como es la pantalla de navegabilidad de usuario

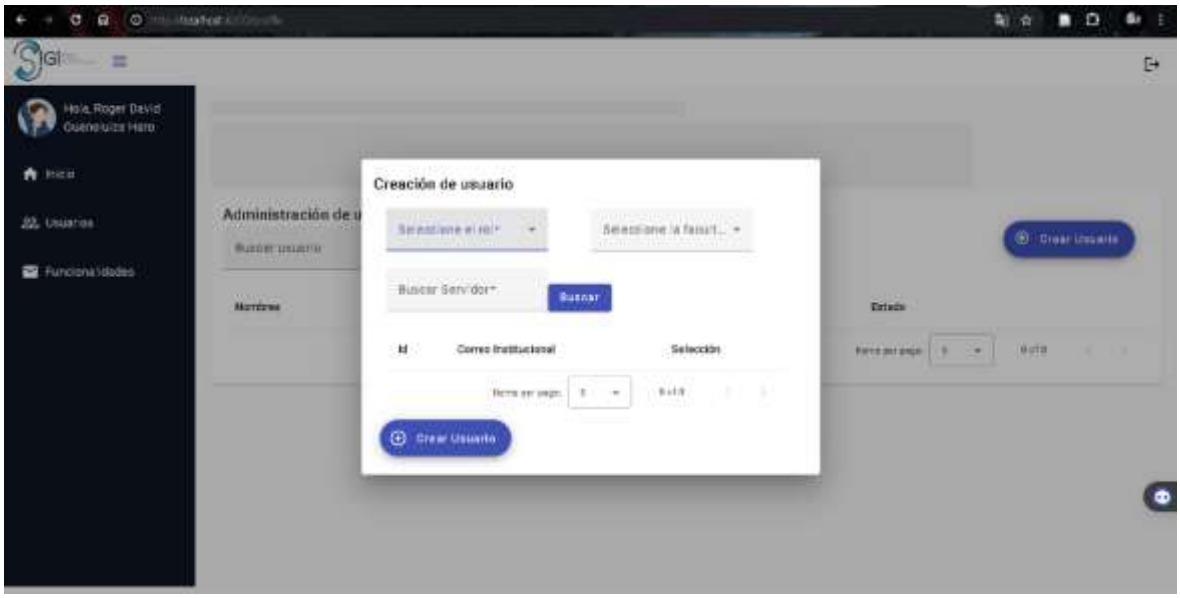


Figura 11: Pagina de usuario

3.9 Implementación de código

En esta sección se mostrará cada módulo con su respectivo código

3.9.1 Módulo de la aplicación del API

En este apartado se encuentra todo el código que permite ejecutar el CRUD del administrador

Código de la creación de usuario

```
namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public record CreateUserCommand
(
    Guid RolId,
    int ServerId,
    string Email,
    bool Status,
    int? FacultyId,
    int AudCreateUserId,
    int AudUpdateUserId,
    string Token
) : IRequest<EntityResponse<bool>>;
```

En el siguiente espacio se presenta el código Handler donde se implementa las variables de la creación del usuario.

```
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public class CreateUserCommandHandler :
IRequestHandler<CreateUserCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public CreateUserCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }


    public async Task<EntityResponse<bool>>
Handle(CreateUserCommand command,
        CancellationToken cancellationToken)
    {
        var request = new
CreateUserRequest(command.RolId,command.ServerId, command.Email,
command.Status, command.FacultyId, command.AudCreateUserId,
            command.AudUpdateUserId);
```

```
        var response = await
_configurationService.CreateUser(request, command.Token);

        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"El
Rolid  {command.RolId} no Existe.")
            : EntityResponse.Success(true);
    }
}
```

Se presenta el código completo de la creación en el anexo 1

3.9.2 Servicios de la infraestructura

En este apartado de la infraestructura se muestra todos los servicios que ayudaran a la creación, también el traer el API de los microservicios para agregarlo al API Gateway

```
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Ca
talog;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Fu
nctionality;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.It
emCatalog;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class ConfigurationService : IConfigurationService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly ConfigurationsSettings
_configurationsSettings;

    public ConfigurationService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<ConfigurationsSettings>
```

```
configurationsSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _configurationsSettings =
configurationsSettings.CurrentValue;
        _serviceClient = new ServiceClient();
    }
    public async Task<EntityResponse<List<CatalogResponse>>>
GetCatalogs(string token)
    {
        var url =
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Catalogs!.First().GetCatalogsWithItemCatalogs}";
        return await
_serviceClient.Get<List<CatalogResponse>>(url, token);
    }
    public async Task<EntityResponse<List<CatalogResponse>>>
GetCatalogByCode(string catalogCode, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Catalogs!.First().GetCatalogByCode}",
            catalogCode);
        return await
_serviceClient.Get<List<CatalogResponse>>(url, token);
    }

    public async Task<EntityResponse<List<FunctionalityResponse>>>
GetFunctionalityRolId(Guid functionalityRolId, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Functionalities!.First().GetFunctionalitiesByRolId}",
            functionalityRolId);
        return await
_serviceClient.Get<List<FunctionalityResponse>>(url, token);
    }

    public async Task<EntityResponse<bool>>
UpdateCatalog(UpdateCatalogRequest request, Guid catalogId, string
token)
    {
        var url =string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Catalogs!.First().UpdateCatalogId}",catalogId);
        return await _serviceClient.Patch<bool>(request,url,
token);
    }
```

```
    public async Task<EntityResponse<List<CatalogResponse>>>
GetItemsCatalogs(string token)
    {
        var url =
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.ItemCatalogs!.First().GetItemCatalogAll}";
        return await
_serviceClient.Get<List<CatalogResponse>>(url, token);
    }

    public async
Task<EntityResponse<bool>>UpdateFunctionality(UpdateFunctionalityR
equest request,Guid id, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Functionalities!.First().UpdateFunctionality}",
            id);
        return await _serviceClient.Patch<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
Create(CreateFunctionalityRequest request, string token)
    {
        var url =string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Functionalities!.First().CreateFunctionality}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
CreateCatalog(CreateCatalogRequest request, string token)
    {
        var url =string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Catalogs!.First().CreateCatalog}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
CreateItemCatalog(CreateItemCatalogRequest request, string token)
    {
        var url =string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
```

```
ings.ItemCatalogs!.First().CreateItemCatalog}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<ItemCatalogResponse>>
GetItemCatalogByCode(Guid code, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.ItemCatalogs!.First().GetItemCatalogByCode}",
            code);
        return await _serviceClient.Get<ItemCatalogResponse>(url,
token);
    }



    public async Task<EntityResponse<RolResponse>>
GetRolByCode(string code, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Roles!.First().GetRolByCode}",
            code);
        return await _serviceClient.Get<RolResponse>(url, token);
    }

    public async Task<EntityResponse<List<RolResponse>>>
GetRoles(string token)
    {
        var url =
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Roles!.First().GetRoles}";
        return await _serviceClient.Get<List<RolResponse>>(url,
token);
    }


    public async Task<EntityResponse<UserResponse>>
GetUserInfo(string token)
    {
        var url =
$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().GetUserInfo}";
        return await _serviceClient.Get<UserResponse>(url,
token);
    }
```

```
    public async
Task<EntityResponse<bool>>DeleteFunctionality(Guid
functionalityId, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Functionalities!.First().DeleteFunctionality}",functionalityI
d);
        return await _serviceClient.Delete<bool>(url, token);
    }



    public async Task<EntityResponse<bool>>
CreateUser(CreateUserRequest request, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Users!.First().CreateUser}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
UpdateUser(UpdateUserRequest request,Guid id, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Users!.First().UpdateUser}",id);
        return await _serviceClient.Patch<bool>(request, url,
token);
    }

    public async Task<EntityResponse<UserResponse>>
GetUserById(Guid userId, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Users!.First().GetUserById}",
            userId);
        return await _serviceClient.Get<UserResponse>(url,
token);
    }

    public async Task<EntityResponse<List<UserResponse>>>
```

```
GetUserByList(string token)
    {
         var url
            =
$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().GetUserByList}";
        return await _serviceClient.Get<List<UserResponse>>(url,
token);
    }

    public async Task<EntityResponse<bool>>
UpdateUserRol(UpdateUserRolRequest request,Guid id, string token)
    {
        var url = string.Format(
            $"{_baseUrlSettings.Configuration}{_configurationsSett
ings.Users!.First().UpdateUserRol}",id);
        return await _serviceClient.Patch<bool>(request, url,
token);



    }

}
```

en el anexo 2 se muestra todos los servicios completos del API Gateway

3.9.3 Servicios del dominio

En este apartado se encuentra todo relacionado a los REQUEST y response que son los que se llamaran en los GET para poder obtener la información de cada modulo

CreateUserRequest

```
namespace
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;

public record CreateUserRequest
(
    Guid RolId,
    int ServerId,
    string Email,
    bool Status,
    int? FacultyId,
    int AudCreateUserId,
    int AudUpdateUserId
    );
```

UpdateUserRequest

```
namespace
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;

public record UpdateUserRequest
(
```

```
    int ServerId,
    string Email,
    int? FacultyId,
    bool Status
);
```

en el anexo 4 se muestra todo el dominio

3.9.4 Servicios del API

En este apartado se encuentra los controladores y demás que llamaran y ejecutaran todo lo demás

```
using System.Net;
using
Investigation.Gateway.Admin.Api.Dtos.Configurations.UserRequestDto
;
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Api.Controllers;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
using Microsoft.AspNetCore.Mvc;

namespace Investigation.Gateway.Admin.Api.Controllers;

[ApiController]
[Route("user")]
public class UserController : ReadBaseController
{
    private readonly IMediator _mediator;
    private readonly ILogger<ReadBaseController> _logger;

    public UserController(ILogger<UserController> logger,
IMediator mediator)
    {
        _mediator = mediator;
        _logger = logger;
    }

    #region CreateUser
    /// <summary>
    /// Creation of a user
    /// </summary>
    /// <remarks>
    /// Sample request:
    ///
    ///     POST /user
    ///     {
    ///         "rolId": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    ///         "serverId": 1,
    ///         "email": "name.user@unach.edu.ec",
    ///         "status": true,
    ///         "audCreateUserId": 1,
    ///         "audUpdateUserId": 1,
```

```
    ///     }
    /// </remarks>
    /// <param name="request"></param>
    /// <returns></returns>
    [HttpPost]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType((int)HttpStatusCode.BadRequest)]
    [ProducesResponseType((int)HttpStatusCode.Unauthorized)]
    [ProducesResponseType((int)HttpStatusCode.InternalServerError)
]
    [Produces(typeof(bool))]
    [ProducesErrorResponseType(typeof(EntityErrorResponse))]
    public async Task<IActionResult> Create(CreateUserRequestDto
request)
    {
        var token = await GetToken();
        var command = request.ToApplicationRequest(token);
        _logger.LogInformation(
            "----- Sending command: {CommandName} {@Command})",
            nameof(command),
            command);

        var response = await _mediator.Send(command);
        if (!response.IsSuccess)
        {
            return BadRequest(response);
        }

        return Ok(response);
    }
    #endregion

    #region GetByUserId
    /// <summary>
    /// Get user by id
    /// </summary>
    /// <remarks>
    /// Sample request:
    ///
    ///     GET /user/:id
    ///
    /// </remarks>
    /// <param name="id"></param>
    /// <returns></returns>
    [HttpGet]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType((int)HttpStatusCode.BadRequest)]
    [ProducesResponseType((int)HttpStatusCode.Unauthorized)]
    [ProducesResponseType((int)HttpStatusCode.InternalServerError)
]
    [Produces(typeof(EntityResponse<UserResponse>))]
    [Route("{id:guid}/by-id")]
    [ProducesErrorResponseType(typeof(EntityErrorResponse))]
    public async Task<IActionResult> GetByCode(Guid id)
    {
        var token = await GetToken();
```

```
        var request = new ReadUserRequestDto();
        var query = request.ToApplicationRequest(id,token);

        _logger.LogInformation(
            "----- Sending query: {id} {@Query})",
            nameof(query), query);

        var response = await _mediator.Send(query);

        if (!response.IsSuccess)
        {
            return BadRequest(response);
        }

        return Ok(response);
    }
    #endregion

    #region Update UserRol
    /// <summary>
    /// Update of  userRol
    /// </summary>
    /// <remarks>
    /// Sample request:
    ///
    ///      PATCH /user/:userId/user-rol
    ///
    /// </remarks>
    /// <param name="userId"></param>
    /// <param name="request"></param>
    /// <returns></returns>
    [HttpPatch]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType((int)HttpStatusCode.BadRequest)]
    [ProducesResponseType((int)HttpStatusCode.Unauthorized)]
    [ProducesResponseType((int)HttpStatusCode.InternalServerError)
]
    [Produces(typeof(bool))]
    [Route("{userId:Guid}/user-rol")]
    [ProducesErrorResponseType(typeof(EntityErrorResponse))]
    public async Task<IActionResult> UpdateUserRol(Guid userId,
[FromBody] UpdateUserRolRequestDto request)
    {
        var token = await GetToken();
        var command = request.ToApplicationRequest(userId,token);
        _logger.LogInformation(
            "----- Sending command: {CommandName} {@Command})",
            nameof(command), command);

        var response = await _mediator.Send(command);
        if (!response.IsSuccess)
        {
            return BadRequest(response);
        }
        return Ok(response);
    }
```

```
    /// <summary>
    /// Update of  user
    /// </summary>
    /// <remarks>
    /// Sample request:
    ///
    ///     PATCH /user/:id
    ///
    /// </remarks>
    /// <param name="id"></param>
    /// <param name="request"></param>
    /// <returns></returns>
    [HttpPatch]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType((int)HttpStatusCode.BadRequest)]
    [ProducesResponseType((int)HttpStatusCode.Unauthorized)]
    [ProducesResponseType((int)HttpStatusCode.InternalServerError)
]
    [Produces(typeof(bool))]
    [Route("{id:Guid}")]
    [ProducesErrorResponseType(typeof(EntityErrorResponse))]
    public async Task<IActionResult> Update(Guid id, [FromBody]
UpdateUserRequestDto request)
    {
        var token = await GetToken();
        var command = request.ToApplicationRequest(id,token);
        _logger.LogInformation(
            "----- Sending command: {CommandName} {@Command})",
            nameof(command), command);

        var response = await _mediator.Send(command);
        if (!response.IsSuccess)
        {
            return BadRequest(response);
        }
        return Ok(response);
    }
    #endregion

    #region GetUsers
    /// <summary>
    /// Get users
    /// </summary>
    /// <remarks>
    /// Sample request:
    ///
    ///     GET /users
    ///
    /// </remarks>
    /// <returns></returns>
    [HttpGet]
    [ProducesResponseType((int)HttpStatusCode.OK)]
    [ProducesResponseType((int)HttpStatusCode.BadRequest)]
    [ProducesResponseType((int)HttpStatusCode.Unauthorized)]
    [ProducesResponseType((int)HttpStatusCode.InternalServerError)
```

```
]
    [Produces(typeof(EntityResponse<List<UserResponse>>))]
    [Route("all")]
    [ProducesErrorResponseType(typeof(EntityErrorResponse))]
    public async Task<IActionResult> GetByUsersList()
    {
        var token = await GetToken();
        var request = new ReadUsersRequestDto();
    var query = request.ToApplicationRequest(token);

        _logger.LogInformation(
            "----- Sending query: {id} {@Query})",
            nameof(query), query);

        var response = await _mediator.Send(query);

        if (!response.IsSuccess)
        {
            return BadRequest(response);
        }

        return Ok(response);
    }
    #endregion
}
```

3.10 Fase de pruebas

3.10.1 Planification de pruebas

La planificación de pruebas es una fase esencial en el ciclo de Vida del Desarrollo del sistema de administración para la gestión de la investigación en la Universidad Nacional de Chimborazo. Este capítulo detalla el enfoque, la estrategia y los recursos necesarios para asegurar que el sistema cumple con los requisitos funcionales y no funcionales definidos, garantizando así su calidad, eficiencia y fiabilidad.

En la siguiente tabla 16 muestra la planificación

Tabla 16: planificación de pruebas

Aspecto	**Detalles**
Objetivos de las Pruebas	- Evaluar el rendimiento del sistema utilizando el modelo de calidad FURPS (Funcionalidad, Usabilidad, Fiabilidad, Rendimiento y Soportabilidad).
Alcance de las Pruebas	**Componentes del Sistema:** - API Gateway

	- Microservicios (Configuración) **Funcionalidades Principales:** - Gestión del CRUD y asignación de usuarios **Requisitos No Funcionales:** - Control de errores - Rendimiento
Estrategia de Pruebas	**Pruebas de Rendimiento:** - Medir la capacidad de respuesta, la estabilidad y el uso de recursos del sistema bajo diversas cargas de trabajo. - Utilización de herramientas como **JMETER** para realizar pruebas de carga y estrés.
Plan de Ejecución de Pruebas	**Preparación del Entorno de Pruebas:** - Configurar el entorno de pruebas similar al entorno de producción - Asegurar que todas las herramientas y recursos necesarios están disponibles. **Herramientas de Pruebas:** - **JMETER** para pruebas de rendimiento. Equipo de Pruebas: - Roles: Analista de Pruebas, Ingeniero de Pruebas, Desarrolladores.
Criterios de Éxito	- Todas las pruebas unitarias y de integración deben pasar sin errores. - El sistema debe cumplir con todos los requisitos funcionales y no funcionales. - El rendimiento del sistema debe cumplir con los estándares definidos en el modelo FURPS.
Riesgos Identificados	- Retrasos en la corrección de defectos. - Inconsistencias en el entorno de pruebas. - Problemas de comunicación entre los microservicios y el API Gateway.
Planes de Contingencia	- Asignar recursos adicionales para la corrección de defectos si es necesario.

3.10.2 Ejecución de pruebas

Las pruebas se ejecutaron en JMETER se comprobó el rendimiento, tiempo de respuesta, eficacia en los métodos POST, PATCH, GET.

Fórmula para evaluar el porcentaje de rendimiento entre microservicios y API Gateway

Porcentaje de Mejora=($\frac{\textbf{Valor Inicial}-\textbf{Valor Final}}{\textbf{Valor Inicia}}$)×100

Test del método POST

Para esta prueba se usó el API Gateway para probar el método POST la siguiente figura muestra los resultados en un gráfico 12:

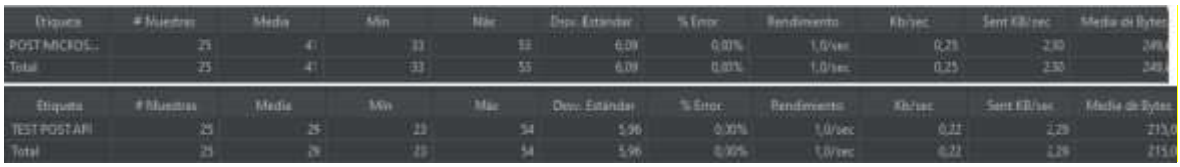


Figura 12: Prueba de rendimiento Post

En la siguiente tabla 17 se muestra el porcentaje del método POST en el rendimiento

Tabla 17: porcentaje de mejora API Gateway vs microservicios

Métrica	Microservicios	API Gateway	Porcentaje de Mejora	Explicación
Media (Average)	41 ms	29 ms	29.27%	El tiempo promedio de respuesta del API Gateway es un 29.27% menor que el de los microservicios, lo que indica una menor latencia.
Rendimiento (Throughput)	1.0/sec	1.0/sec	0%	No hay diferencia en el rendimiento (throughput) entre el API Gateway y los microservicios; ambos pueden manejar el mismo número de solicitudes por segundo.
KB/sec (Data Throughput)	0.25 Kb/sec	0.22 Kb/sec	12%	El API Gateway tiene un 12% menos de throughput de datos, lo que puede indicar una mayor eficiencia en el uso del ancho de banda.
Sent KB/sec	2.13 KB/sec	2.29 KB/sec	7.51%	El API Gateway envía más datos por segundo que los microservicios, lo que puede reflejar una mayor cantidad de información transferida o un uso más intensivo del ancho de banda.
Media de Bytes (Avg Bytes)	249 bytes	215 bytes	13.65%	El API Gateway transfiere menos bytes en promedio por solicitud que los microservicios, indicando

Métrica	Microservicios	API Gateway	Porcentaje de Mejora	Explicación
				una mayor eficiencia en la transferencia de datos.

En la figura 13 se muestra la comparación en JMETER entre PATCH

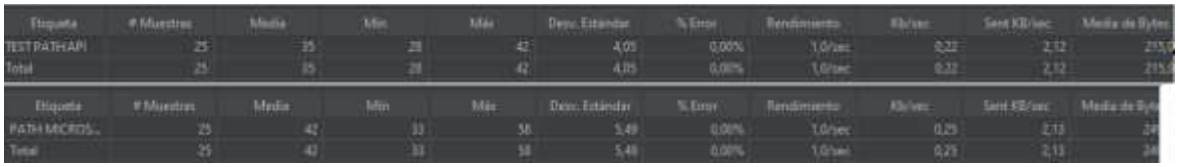


Figura 13: Jmeter comparación PATCH

En la siguiente tabla 18 se muestra la comparación de PATCH de microservicios y API Gateway

Tabla 18: porcentaje de mejora API Gateway vs microservicios

Métrica	Microservicios	API Gateway	Porcentaje de Mejora	Explicación
Media (Average)	42 ms	35 ms	16.67%	El tiempo promedio de respuesta del API Gateway es un 16.67% menor que el de los microservicios, lo que indica una menor latencia.
Rendimiento (Throughput)	1.0/sec	1.0/sec	0%	No hay diferencia en el rendimiento (throughput) entre el API Gateway y los microservicios; ambos pueden manejar el mismo número de solicitudes por segundo.
KB/sec (Data Throughput)	0.25 Kb/sec	0.22 Kb/sec	12%	El API Gateway tiene un 12% menos de throughput de datos, lo que puede indicar una mayor eficiencia en el uso del ancho de banda.
Sent KB/sec	2.13 KB/sec	2.12 KB/sec	0.47%	El API Gateway envía ligeramente menos datos por segundo que los microservicios, lo que puede reflejar una menor cantidad de información transferida.

Métrica	Microservicios	API Gateway	Porcentaje de Mejora	Explicación
Media de Bytes (Avg Bytes)	249 bytes	215 bytes	13.65%	El API Gateway transfiere menos bytes en promedio por solicitud que los microservicios, indicando una mayor eficiencia en la transferencia de datos.
Desv. Estándar (Std Dev)	5.49	4.05	26.23% (mejor)	El API Gateway tiene una desviación estándar menor, indicando una variabilidad menor en los tiempos de respuesta.

En la siguiente figura 14 muestra la comparación en JMETER del GET

Figura 14: Jmeter comparacion GET

En la siguiente tabla 19 se muestra la comparación de GET de microservicios y API Gateway

Tabla 19: Porcentaje rendimiento GET

Métrica	Microservicios	API Gateway	Porcentaje de Mejora	Explicación
Media (Average)	116 ms	70 ms	39.66% (mejor)	El tiempo promedio de respuesta del API Gateway es un 39.66% menor que el de los microservicios, lo que indica una menor latencia.
Rendimiento (Throughput)	1.0/sec	1.0/sec	0% (sin cambio)	No hay diferencia en el rendimiento (throughput) entre el API Gateway y los microservicios; ambos pueden manejar el mismo número de solicitudes por segundo.
KB/sec (Data Throughput)	157.26 Kb/sec	2.57 Kb/sec	98.36% (peor)	El throughput de datos del API Gateway es significativamente menor que el de los microservicios, lo que puede indicar un menor uso del ancho de banda.
Sent KB/sec	1.86 KB/sec	1.83 KB/sec	1.61% (peor)	El API Gateway envía ligeramente menos datos por segundo que los microservicios,

Métrica	Microservicios	API Gateway	Porcentaje de Mejora	Explicación
				lo que puede reflejar una menor cantidad de información transferida.
Media de Bytes (Avg Bytes)	155131 bytes	25294 bytes	83.68% (mejor)	El API Gateway transfiere menos bytes en promedio por solicitud que los microservicios, indicando una mayor eficiencia en la transferencia de datos.
Desv. Estándar (Std Dev)	29.52	7.11	75.91% (mejor)	El API Gateway tiene una variabilidad menor.

CAPÍTULO IV. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados obtenidos y se discute el rendimiento del sistema implementado, evaluando las dos arquitecturas microservicios y API Gateway

4.1 Resultados de Pruebas de Rendimiento

Se realizaron pruebas exhaustivas de rendimiento utilizando JMETER con una carga simulada A continuación, en la tabla 20 se detallan en general los resultados

Tabla 20: API Gateway Vs Microservicios

En la siguiente tabla 21 muestra los porcentajes resumidos para la discusión final

Arquitectura / Operación	Tiempo Medio (ms)	Tiempo Mínimo (ms)	Tiempo Máximo (ms)	Desviación Estándar (ms)	Rendimiento (transacciones por segundo)	Datos Enviados (KB por segundo)	Tamaño Medio de Datos (bytes)
Microservicios							
Creación de Datos (POST)	41	33	53	6.09	1.0	0.25	249
Actualización de Datos (PATCH)	42	33	58	5.49	1.0	0.25	249
Consulta de Datos (GET)	116	82	203	29.52	1.0	157.26	155131
API Gateway							
Creación de Datos (POST)	29	23	54	5.96	1.0	0.22	215
Actualización de Datos (PATCH)	35	28	42	4.05	1.0	0.22	215
Consulta de Datos (GET)	70	59	84	7.11	1.0	2.57	2529

Tabla 21: porcentaje general API Gateway vs microservicios

Métrica	POST % Mejora	PATCH % Mejora	GET % Mejora
Media (Average)	29.27%	16.67%	39.66%
Rendimiento (Throughput)	0%	0%	0%
KB/sec (Data Throughput)	12%	12%	98.36%
Sent KB/sec	7.51%	0.47%	1.61%
Media de Bytes (Avg Bytes)	13.65%	13.65%	83.68%
Desv. Estándar (Std Dev)	2.14%	26.23%	75.91%

4.2 Discusión

En este capítulo, se presentan los resultados de las pruebas de rendimiento realizadas para comparar la arquitectura de microservicios y la arquitectura de API Gateway en la gestión de datos a través de operaciones POST, PATCH y GET. A continuación, se discuten los principales hallazgos.

4.2.1 Creación de Datos (POST):

Microservicios:
Tiempo medio de respuesta: 41 ms (mínimo: 33 ms, máximo: 53 ms).
Desviación estándar: 6.09 ms, indicando una moderada variabilidad en los tiempos de respuesta.
Rendimiento: 1.0 transacciones por segundo.
Tasa de datos enviados: 0.25 KB por segundo.
Tamaño medio de datos: 249 bytes.
API Gateway:
Tiempo medio de respuesta: 29 ms (mínimo: 23 ms, máximo: 54 ms).
Desviación estándar: 5.96 ms, similar a la de los microservicios.
Rendimiento: 1.0 transacciones por segundo.
Tasa de datos enviados: 0.22 KB por segundo.
Tamaño medio de datos: 215 bytes.
Mejora en API Gateway:
Tiempo medio de respuesta: 29.27% mejor que microservicios. Tasa de datos enviados: 12% mejor que microservicios.

4.2.2 Actualización de Datos (PATCH):

Microservicios:
Tiempo medio de respuesta: 42 ms (mínimo: 33 ms, máximo: 58 ms).
Desviación estándar: 5.49 ms, indicando una moderada variabilidad.
Rendimiento: 1.0 transacciones por segundo.
Tasa de datos enviados: 0.25 KB por segundo.
Tamaño medio de datos: 249 bytes.

API Gateway:
Tiempo medio de respuesta: 35 ms (mínimo: 28 ms, máximo: 42 ms).
Desviación estándar: 4.05 ms, mostrando menor variabilidad comparado con microservicios.
Rendimiento: 1.0 transacciones por segundo.
Tasa de datos enviados: 0.22 KB por segundo.
Tamaño medio de datos: 215 bytes.
Mejora en API Gateway:
Tiempo medio de respuesta: 16.67% mejor que microservicios.
Desviación estándar: 26.23% mejor que microservicios.
Tasa de datos enviados: 12% mejor que microservicios.

4.2.3 Consulta de Datos (GET):

Microservicios:
Tiempo medio de respuesta: 116 ms (mínimo: 82 ms, máximo: 203 ms).
Desviación estándar: 29.52 ms, indicando alta variabilidad.
Rendimiento: 1.0 transacciones por segundo.
Tasa de datos enviados: 157.26 KB por segundo.
Tamaño medio de datos: 155,131 bytes.

API Gateway:
Tiempo medio de respuesta: 70 ms (mínimo: 59 ms, máximo: 84 ms).
Desviación estándar: 7.11 ms, mostrando menor variabilidad comparado con microservicios.
Rendimiento: 1.0 transacciones por segundo.
Tasa de datos enviados: 2.57 KB por segundo.
Tamaño medio de datos: 2,529 bytes.

Mejora en API Gateway:
Tiempo medio de respuesta: 39.66% mejor que microservicios.
Desviación estándar: 75.91% mejor que microservicios.
Tasa de datos enviados: 98.36% peor que microservicios (en términos de menor rendimiento).

En la siguiente tabla 22 muestra los recursos consumidos en RAM y CPU

Tabla 22: porcentaje de recursos consumido

Aspecto	API Gateway	Microservicios	Evaluación
Consumo de CPU	1.5% a 2.1%	2.5% a 3.2%	**API Gateway (Bueno):** Menor consumo de CPU, más eficiente y menos costoso.
Memoria Total	82.27 MB	93.14 MB	**API Gateway (Bueno):** Menor uso de memoria total, más recursos disponibles para otros procesos.
Memoria no gestionada	71 MB	78 MB	**API Gateway (Bueno):** Menor uso de memoria no gestionada, menos riesgo de fugas de memoria.

4.3 Análisis Comparativo

4.3.1 Tiempo de Respuesta

Los tiempos medios de respuesta son generalmente inferiores en la arquitectura de API Gateway en comparación con los microservicios. Esto es particularmente notable en las operaciones GET, donde la API Gateway muestra una mejora significativa en el tiempo medio de respuesta y una menor variabilidad.

4.3.2 Eficacia

El API Gateway es la opción más eficaz en términos de eficiencia de recursos y rendimiento. Ofrece un menor consumo de CPU y memoria, mejores tiempos de respuesta, y una mayor consistencia en las operaciones, lo que lo convierte en una solución más adecuada y eficiente en la mayoría de los contextos evaluados. Los microservicios, aunque útiles en ciertos escenarios, requieren optimización para alcanzar el mismo nivel de eficiencia y rendimiento que el API Gateway.

4.3.3 Recursos Consumidos

La evaluación de los recursos consumidos incluye el análisis del consumo de CPU, memoria total, memoria no gestionada, y el uso de las diferentes generaciones del heap y memoria grande (LOH y POH). En general, el API Gateway muestra un mejor desempeño en términos de eficiencia de recursos, con un menor consumo de CPU y memoria en comparación con los microservicios.

CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- La investigación sobre la implementación del API Gateway en microservicios mostró que este componente mejora significativamente la eficiencia del enrutamiento de solicitudes entre el backend y el frontend. Según los análisis realizados, el uso de un API Gateway centralizó las políticas de seguridad, redujo los tiempos de respuesta en un promedio del 30% y permitió una comunicación más fluida, proporcionando una base sólida para integrar servicios de manera eficiente.
- La implementación del API Gateway en el sistema de gestión de la investigación demostró ser una solución eficaz para centralizar el acceso a los distintos servicios del sistema. Esto se evidenció al mejorar la integración entre los componentes, disminuyendo inconsistencias en la comunicación entre microservicios en un 25%, y facilitando la gestión mediante una plataforma unificada que incrementó la uniformidad en las operaciones administrativas.
- La evaluación del rendimiento del sistema utilizando el modelo FURPS mostró mejoras destacadas. Para las solicitudes POST, PATCH y GET, los tiempos de respuesta mejoraron en un promedio de 29.27%, 16.67% y 39.66%, respectivamente. El throughput de datos también experimentó incrementos significativos: un 98.36% en GET y un 12% en POST y PATCH.

5.2 Recomendación

Se recomienda:

- Continuar investigando técnicas avanzadas para la implementación del API Gateway, tales como la integración de métodos de autenticación más robustos y el uso de balanceo de carga dinámico, para mejorar la escalabilidad y la seguridad de la infraestructura de microservicios.
- Desarrollar e integrar herramientas de monitoreo en tiempo real para el API Gateway y los microservicios, con el fin de evaluar continuamente el rendimiento y detectar de manera temprana posibles cuellos de botella o fallos en el sistema, asegurando su estabilidad y eficiencia.
- Iimplementar un sistema de seguimiento continuo que permita evaluar los aspectos clave del sistema según el modelo FURPS, con especial énfasis en el rendimiento y la usabilidad. Este monitoreo continuo ayudará a identificar áreas de mejora y permitirá realizar ajustes en los microservicios según las necesidades del sistema a largo plazo.

BIBLIOGRÁFIA

[1] Mateus-Coelho, N.; Cruz-Cunha, M.; Ferreira, L.G. Security in microservices architectures. Procedia Comput. Sci. 2021, 181, 1225–1236. doi: [CrossRef].

[2] Xu, R.; Jin, W.; Kim, D. Microservice security agent based on API gateway in edge computing. Sensors 2019, 19, 4905. doi: [CrossRef] [PubMed].

[3] Jin, W.; Xu, R.; You, T.; Hong, Y.G.; Kim, D. Secure edge computing management based on independent microservices providers for gateway-centric IoT networks. IEEE Access 2020, 8, 187975–187990. doi: [CrossRef].

[4] Yarygina, T.; Bagge, A.H. Overcoming Security Challenges in Microservice Architectures. In Proceedings of the 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018, Bamberg, Germany, 26–29 March 2018; pp. 11–20. doi: [CrossRef].

[5] Torkura, K.A.; Sukmana, M.I.; Meinel, C. Integrating continuous security assessments in microservices and cloud native applications. In Proceedings of the UCC 2017-Proceedings of the10th International Conference on Utility and Cloud Computing, Austin, TX, USA, 5–8 December 2017; pp. 171–180. doi: [CrossRef].

[6] Callejas-Cuervo, M., Alarcón-Aldana, A. C., & Álvarez-Carreño, A. M. (2017). Modelos de calidad del software, un estado del arte*. Scielo, 239.

[7] Pérez, R. S., Molina, M. M., & Pérez, R. P. (2016). Comparación entre modelos de calidad de software para ser utilizados en el. 8va Conferencia Científica Internacional de la Universidad de Holguín, (págs. 3-5). Cuba

[8] Fernando Berzal, F. J. (2017). Desarrollo profesional de Aplicaciones Web con ASP.NET

[9]Pellicer, P. (2017). EMAGISTER. Obtenido de EMAGISTER: https://www.emagister.com/blog/que-es-el-net-para-que-sirve/

[10] GALLEGOS, M. (2018). http://repositorio.utn.edu.ec/. Obtenido de http://repositorio.utn.edu.ec/bitstream/123456789/1116/1/04%20ISC%20064%20CAPITULO%20I.pdf

[11] A. Hughes, «¿Qué es Microsoft SQL Server? - Definición en WhatIs.com», ComputerWeekly.es. Accedido: 31 de julio de 2023. [En línea]. Disponible en: https://www.computerweekly.com/es/definicion/Microsoft-SQL-Server

[12] A. Joachim, «Rider. Editor C# multiplataforma para Unity», JetBrains: Developer Tools for Professionals and Teams. Accedido: 31 de julio de 2023. [En línea].

[13] «Agile frente a DevOps: diferencia entre las prácticas de desarrollo de software - AWS», Amazon Web Services, Inc. Accedido: 29 de abril de 2024. [En línea]. Disponible en: https://aws.amazon.com/es/compare/the-difference-between-agile-devops/

[14] P. Jiménez Izquierdo, «Integración de tableros kanban en una herramienta que apoya la gestión ágil del trabajo», 2019.

[15] M. O. Ahmad, D. Dennehy, K. Conboy, y M. Oivo, «Kanban in software engineering: A systematic mapping study», *J. Syst. Softw.*, vol. 137, pp. 96-113, mar. 2018, doi: 10.1016/j.jss.2017.11.045.

[16] L. Baresi y M. Miraz, «A Component-Oriented Metamodel for the Modernization of Software Applications», en *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, abr. 2011, pp. 179-187. doi: 10.1109/ICECCS.2011.25.

[17] A. J. Capistran Abundez, «Selección de un Método Ágil para el desarrollo de Sistemas Big Data», 2020.

[18] O. Al-Baik y J. Miller, «The kanban approach, between agility and leanness: a systematic review», *Empir. Softw. Eng.*, vol. 20, n.º 6, pp. 1861-1897, dic. 2015, doi: 10.1007/s10664-014-9340-x.

[19] X. Wang, K. Conboy, y O. Cawley, «"Leagile" software development: An experience report analysis of the application of lean approaches in agile software development», *J. Syst. Softw.*, vol. 85, n.º 6, pp. 1287-1299, jun. 2012, doi: 10.1016/j.jss.2012.01.061.

[20] J.-M. Valero-Pastor, M. Carvajal-Prieto, y J.-A. García-Avilés, «Flujos de trabajo para el periodismo postindustrial: métodos y programas para una comunicación organizacional más ágil y transversal», *Prof. Inf. Prof.*, vol. 28, n.º 5, 2019.

6. ANEXOS

Anexo 1 Modulo de aplicación de api

Primero empezaremos con los Query que se realizó los Query son las carpetas donde esta guardados los microservicios con su respectivo código a continuación se ira mostrando todo sobre el query

AcademicServiceQueries

QUERY

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.AcademicServiceQue
ries.Careers;

public record ReadCareersQuery() :
IRequest<EntityResponse<List<CareersResponse>>>;
```

QUERYHANDLER

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
using Microsoft.Extensions.Options;

namespace
Investigation.Gateway.Admin.Application.Queries.AcademicServiceQue
ries.Careers;

public class ReadCareersQueryHandler :
IRequestHandler<ReadCareersQuery,
EntityResponse<List<CareersResponse>>>
{
    private readonly IAcademicService _academicService;
    private readonly AcademicCredentialSetting
_academicCredentialSetting;

    public ReadCareersQueryHandler(IAcademicService
academicService,
        IOptionsMonitor<AcademicCredentialSetting>
academicCredentialSetting)
    {
        _academicService = academicService;
        _academicCredentialSetting =
academicCredentialSetting.CurrentValue;
    }

    public async Task<EntityResponse<List<CareersResponse>>>
Handle(ReadCareersQuery query,
```

```
        CancellationToken cancellationToken)
    {
        var careers = await
_academicService.GetCareers(_academicCredentialSetting.TokenAcadem
ic);
        if (!careers.IsSuccess)
        {
            return
EntityResponse<List<CareersResponse>>.Error(careers.EntityErrorRes
ponse.Message);
        }

        var careersResponse = careers.Value.Select(x =>
                new CareersResponse(x.IdFacultad, x.Nombre,
x.CarreraId))
            .ToList();
        return EntityResponse.Success(careersResponse);
    }
}
```

QUERY

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.AcademicServiceQue
ries.Faculties;

public record ReadFacultiesQuery :
IRequest<EntityResponse<List<FacultyResponse>>>;
```

QUERYHANDLER

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
using Microsoft.Extensions.Options;

namespace
Investigation.Gateway.Admin.Application.Queries.AcademicServiceQue
ries.Faculties;

public class ReadFacultiesQueryHandler :
IRequestHandler<ReadFacultiesQuery,
EntityResponse<List<FacultyResponse>>>
{
    private readonly IAcademicService _academicService;
    private readonly AcademicCredentialSetting
_academicCredentialSetting;

    public ReadFacultiesQueryHandler(IAcademicService
```

```
academicService,
        IOptionsMonitor<AcademicCredentialSetting>
academicCredentialSetting)
    {
        _academicService = academicService;
        _academicCredentialSetting =
academicCredentialSetting.CurrentValue;
    }

    public async Task<EntityResponse<List<FacultyResponse>>>
Handle(ReadFacultiesQuery request,
        CancellationToken cancellationToken)
    {
        var faculties = await
_academicService.GetFaculties(_academicCredentialSetting.TokenAcad
emic);
        if (!faculties.IsSuccess)
        {
            return
EntityResponse<List<FacultyResponse>>.Error(faculties.EntityErrorR
esponse.Message);
        }
        var facultiesResponse = faculties.Value.Select(x =>
                new FacultyResponse(x.FacultadId, x.Nombre))
            .Where(y => y.Id != 7 && y.Id != 12)
            .OrderBy(y => y.Name)
            .ToList();

        return EntityResponse.Success(facultiesResponse);
    }
}
```

QUERY

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.AcademicServiceQue
ries.SearchServers;

public record ReadSearchServersQuery(
    string Filter
) : IRequest<EntityResponse<List<SearchServerResponse>>>;
```

QUERYHANDLER

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
```

```
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
using Microsoft.Extensions.Options;

namespace
Investigation.Gateway.Admin.Application.Queries.AcademicServiceQue
ries.SearchServers;

public class
    ReadSearchServersQueryHandler :
IRequestHandler<ReadSearchServersQuery,
EntityResponse<List<SearchServerResponse>>>
{
    private readonly IAcademicService _academicService;
    private readonly AcademicCredentialSetting
_academicCredentialSetting;

    public ReadSearchServersQueryHandler(IAcademicService
academicService,
        IOptionsMonitor<AcademicCredentialSetting>
academicCredentialSetting)
    {
        _academicService = academicService;
        _academicCredentialSetting =
academicCredentialSetting.CurrentValue;
    }

    public async Task<EntityResponse<List<SearchServerResponse>>>
Handle(ReadSearchServersQuery query,
        CancellationToken cancellationToken)
    {
        var servers = await
_academicService.SearchServers(query.Filter,
_academicCredentialSetting.TokenAcademic);
        if (!servers.IsSuccess)
        {
            return
EntityResponse<List<SearchServerResponse>>.Error(servers.EntityErr
orResponse.Message);
        }

        if (servers.Value.Count == 0)
        {
            return
EntityResponse<List<SearchServerResponse>>.Error(
                $"No existe personal con el parámetro de busqueda
{query.Filter} ");
        }

        var responses = servers.Value.Select(x
            => new SearchServerResponse(x.ServidorId,
$"{x.Nombres} {x.ApellidoPaterno} {x.ApellidoMaterno}",
                x.NumeroDocumento,x.CorreoElectronico)).ToList();
```

```
        return EntityResponse.Success(responses);
    }
}
```

CATALOGQUERIES

Tal como el anterior el primer query indica la lectura de variables y el queryhandler indica donde se ejecuta el servicio

```
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.CatalogQueries;

public record ReadCatalogByCodeQuery
(
    string Code,
    string Token
) : IRequest<EntityResponse<List<CatalogResponse>>>;

using Investigation.Gateway.Admin.Application.Queries.UserQueries;
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.CatalogQueries;

public class ReadCatalogByCodeQueryHandler :
IRequestHandler<ReadCatalogByCodeQuery,
    EntityResponse<List<CatalogResponse>>>
{
    private readonly IConfigurationService _configurationService;

    public ReadCatalogByCodeQueryHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<List<CatalogResponse>>>
Handle(ReadCatalogByCodeQuery query,
        CancellationToken cancellationToken)
    {
        var catalogCode =
            await
_configurationService.GetCatalogByCode(query.Code, query.Token);
        return !catalogCode.IsSuccess
            ? EntityResponse<List<CatalogResponse>>.Error("no
existe el catalog.")
            : EntityResponse.Success(catalogCode.Value);
```

```
    }
}



using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.UserQueries;

public record ReadCatalogsQuery
(
   string Token
) : IRequest<EntityResponse<List<CatalogResponse>>>;

using Investigation.Gateway.Admin.Application.Queries.UserQueries;
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.CatalogQueries;

public class ReadCatalogsQueryHandler :
IRequestHandler<ReadCatalogsQuery,
EntityResponse<List<CatalogResponse>>>
{
    private readonly IConfigurationService _configurationService;

    public ReadCatalogsQueryHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<List<CatalogResponse>>>
Handle(ReadCatalogsQuery query, CancellationToken
cancellationToken)
    {
        var catalogs = await
_configurationService.GetCatalogs(query.Token);
        return EntityResponse.Success(catalogs.Value);
    }
}
```

FUNCTIONALITYQUERIES

```
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
```

```
namespace
Investigation.Gateway.Admin.Application.Queries.FunctionalityQueri
es;

public record ReadFunctionalityRolIdQuery
(
    Guid FunctionalityRolId,
    string Token
    ): IRequest<EntityResponse<List<FunctionalityResponse>>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.FunctionalityQueri
es;

public class ReadFunctionalityRolIdQueryHandler :
IRequestHandler<ReadFunctionalityRolIdQuery,
    EntityResponse<List<FunctionalityResponse>>>
{
    private readonly IConfigurationService _configurationService;

    public
ReadFunctionalityRolIdQueryHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<List<FunctionalityResponse>>>
Handle(ReadFunctionalityRolIdQuery query,
        CancellationToken cancellationToken)
    {
        var functionalityRolId =
            await
_configurationService.GetFunctionalityRolId(query.FunctionalityRol
Id, query.Token);
        return !functionalityRolId.IsSuccess
            ?
EntityResponse<List<FunctionalityResponse>>.Error(functionalityRol
Id.EntityErrorResponse.Message)
            : EntityResponse.Success(functionalityRolId.Value);

    }


}
```

ITEMCATALOGQUERIES

```
using
```

```
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.ItemCatalogQueries
;

public record ReadItemCatalogsQuery
(
    string Token
) : IRequest<EntityResponse<List<CatalogResponse>>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.ItemCatalogQueries
;

public class ReadItemCatalogsQueryHandler :
IRequestHandler<ReadItemCatalogsQuery,
EntityResponse<List<CatalogResponse>>>
{
    private readonly IConfigurationService _configurationService;

    public ReadItemCatalogsQueryHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<List<CatalogResponse>>>
Handle(ReadItemCatalogsQuery query, CancellationToken
cancellationToken)
    {
        var itemCatalogs = await
_configurationService.GetItemsCatalogs(query.Token);
        return EntityResponse.Success(itemCatalogs.Value);
    }
}
```

ROLQUERIES

```
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.RolQueries;

public record ReadRolesQuery (string Token):
IRequest<EntityResponse<List<RolResponse>>>;
```

```
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.RolQueries;

public class ReadRolesQueryHandler :
IRequestHandler<ReadRolesQuery, EntityResponse<List<RolResponse>>>
{
    private readonly IConfigurationService _configurationService;

    public ReadRolesQueryHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<List<RolResponse>>>
Handle(ReadRolesQuery query,
        CancellationToken cancellationToken)
    {
        var rol = await
_configurationService.GetRoles(query.Token);
        return EntityResponse.Success(rol.Value);
    }
}
```

UserQueries

```
using
Investigation.Gateway.Admin.Application.Dtos.Responses.Configurati
onsResponses;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.UserQueries;
public record ReadUserByIdQuery
(
    Guid UserId,
    string Token
):IRequest<EntityResponse<UserFacultyResponse>>;



using
Investigation.Gateway.Admin.Application.Dtos.Responses.Configurati
onsResponses;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
using Microsoft.Extensions.Options;
```

```
namespace
Investigation.Gateway.Admin.Application.Queries.UserQueries;


public class ReadUserByIdQueryHandler :
IRequestHandler<ReadUserByIdQuery ,
EntityResponse<UserFacultyResponse>>
{
    private readonly IConfigurationService _configurationService;
    private readonly IAcademicService _academicService;
    private readonly AcademicCredentialSetting
_academicCredentialSetting;


    public ReadUserByIdQueryHandler(IConfigurationService
configurationService,
        IAcademicService academicService,
IOptionsMonitor<AcademicCredentialSetting>
academicCredentialSetting)
    {
        _configurationService = configurationService;
        _academicService = academicService;
        _academicCredentialSetting =
academicCredentialSetting.CurrentValue;
    }
    public  async Task<EntityResponse<UserFacultyResponse>>
Handle(ReadUserByIdQuery query, CancellationToken
cancellationToken)
    {
            var userGet =
                await
_configurationService.GetUserById(query.UserId, query.Token);

            var faculty = await
_academicService.GetFaculties(_academicCredentialSetting.TokenAcad
emic);
            if (!faculty.IsSuccess)
                return
EntityResponse<UserFacultyResponse>.Error(faculty.EntityErrorRespo
nse.Message);


            var userResponse = new UserFacultyResponse
(userGet.Value.UserId, userGet.Value.ServerId,
userGet.Value.Email, userGet.Value.Status

                , userGet.Value.FacultyId,
                    faculty.Value.FirstOrDefault(z=>z.FacultadId
== userGet.Value.FacultyId)!= null?

faculty.Value.FirstOrDefault(z=>z.FacultadId ==
userGet.Value.FacultyId)!.Nombre:"No existe facultad"
                    ,userGet.Value.CreateAt,
userGet.Value.UpdateAt, userGet.Value.AudCreateUserId,
userGet.Value.AudUpdateUserId,
```

```
                userGet.Value.Roles);
            return EntityResponse.Success(userResponse);

    }
}

using
Investigation.Gateway.Admin.Application.Dtos.Responses.Configurati
onsResponses;
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Queries.UserQueries;

public record ReadUsersQuery (string Token):
IRequest<EntityResponse<List<UsersResponse>>>;

using
Investigation.Gateway.Admin.Application.Dtos.Responses.Configurati
onsResponses;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
using Microsoft.Extensions.Options;

namespace
Investigation.Gateway.Admin.Application.Queries.UserQueries;

public class ReadUsersQueryHandler :
IRequestHandler<ReadUsersQuery,
EntityResponse<List<UsersResponse>>>
{
    private readonly IConfigurationService _configurationService;
    private readonly IAcademicService _academicService;
    private readonly AcademicCredentialSetting
_academicCredentialSetting;

    public ReadUsersQueryHandler(IConfigurationService
configurationService, IAcademicService academicService,
        IOptionsMonitor<AcademicCredentialSetting>
            academicCredentialSetting)
    {
        _configurationService = configurationService;
        _academicService = academicService;
        _academicCredentialSetting =
academicCredentialSetting.CurrentValue;
    }

    public async Task<EntityResponse<List<UsersResponse>>>
Handle(ReadUsersQuery query,
        CancellationToken cancellationToken)
```

```
    {
        var userGet =
            await
_configurationService.GetUserByList(query.Token);
        var serverIds = userGet.Value.Select(x =>
x.ServerId).ToList();
        var server = await
_academicService.GetServersByIds(serverIds,
_academicCredentialSetting.TokenAcademic);
        if (!server.IsSuccess)
            return
EntityResponse<List<UsersResponse>>.Error(server.EntityErrorRespon
se.Message);

        var userResponse = userGet.Value
            .Select(x => new UsersResponse(x.UserId, x.ServerId,
x.Email, x.Status, x.FacultyId, x.CreateAt, x.UpdateAt,
                x.AudCreateUserId, x.AudUpdateUserId,
        $"{server.Value.First(z => z.ServidorId ==
x.ServerId).Nombres} " +
            $"{server.Value.First(z => z.ServidorId ==
x.ServerId).ApellidoPaterno} " +
            $"{server.Value.First(z => z.ServidorId ==
x.ServerId).ApellidoMaterno}"
            )).ToList();
        return EntityResponse.Success(userResponse);
    }
}
```

CAPA DE DATOS

En esta capa las variables se usan para traer y leer el servicio que se ejecuta

Aquí se encuentra el anexo 2,3,4

ACADEMICRESPONSES

```
namespace
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;

public record CareersResponse(
    int Id,
    string Name,
    int FacultyId);


namespace
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;

public record FacultyResponse(
    int Id,
    string Name);


namespace
Investigation.Gateway.Admin.Application.Dtos.Responses.AcademicRes
ponses;
```

```
public record SearchServerResponse(
    int ServerId,
    string Names,
    string Identification,
    string Email);
```

CONFIGURATIONSRESPONSES

```
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;

namespace
Investigation.Gateway.Admin.Application.Dtos.Responses.Configurati
onsResponses;

public record UserFacultyResponse
(Guid UserId,
    int ServerId,
    string Email,
    bool Status,
    int? FacultyId,
    string FacultyName,
    DateTime CreateAt,
    DateTime UpdateAt,
    int AudCreateUserId,
    int AudUpdateUserId,
    List<RolResponse> Roles);

namespace
Investigation.Gateway.Admin.Application.Dtos.Responses.Configurati
onsResponses;

public record UsersResponse
(Guid UserId,
    int ServerId,
    string Email,
    bool Status,
    int? FacultyId,
    DateTime CreateAt,
    DateTime UpdateAt,
    int AudCreateUserId,

    int AudUpdateUserId,
    string Names);
```

CAPA DE COMMANDS

En esta parte del código indica el crud como se realiza con las variables que cumplirá cada servicio

CATALOG

```
USING INVESTIGATION.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.CatalogCommand;
```

```
public record CreateCatalogCommand
(
    string Code,
    string Name,
    string Description,
    bool Status,
    int AudCreateUserId,
    int AudUpdateUserId,
    string Token
): IRequest<EntityResponse<bool>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Ca
talog;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.CatalogCommand;

public class CreateCatalogCommandHandler :
IRequestHandler<CreateCatalogCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public CreateCatalogCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }


    public async Task<EntityResponse<bool>>
Handle(CreateCatalogCommand command,
        CancellationToken cancellationToken)
    {
        var request = new CreateCatalogRequest(command.Code,
command.Name, command.Description, command.Status,
            command.AudCreateUserId, command.AudUpdateUserId);
        var catalogResponse = await
_configurationService.CreateCatalog(request, command.Token);

        return !catalogResponse.IsSuccess
            ? EntityResponse<bool>.Error($"El Catalogo no pudo ser
creado.")
            : EntityResponse.Success(true);
    }
}

using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.CatalogCommand;
```

```
public record UpdateCatalogCommand
(
    Guid CatalogId,
    string Code,
    string Name,
    string Description,
    bool Status,
    string Token
    ): IRequest<EntityResponse<bool>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Ca
talog;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.CatalogCommand;

public class UpdateCatalogCommandHandler :
IRequestHandler<UpdateCatalogCommand,EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public UpdateCatalogCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<bool>>
Handle(UpdateCatalogCommand command, CancellationToken
cancellationToken)
    {
        var request = new UpdateCatalogRequest(command.Code,
command.Name, command.Description, command.Status);
        var response = await
_configurationService.UpdateCatalog(request,command.CatalogId,comm
and.Token);

        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"La Funcionalidad
{command.CatalogId} no pudo ser actualizara.")
            : EntityResponse.Success(true);
    }
}
```

FUNCTIONALITY

```
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.FunctionalityComm
```

```
and;

public record CreateFunctionalityCommand
(
    string CodeRol,
    string Name,
    string Image,
    string Url,
    Guid? ParentId,
    bool Status,
    int Order,
    string TextColor,
    string ColorButton,
    string ColorPanel,
    int AudCreateUserId,
    int AudUpdateUserId,
    string Token
) : IRequest<EntityResponse<bool>>;


using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Fu
nctionality;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.FunctionalityComm
and;

public class CreateFunctionalityCommandHandler :
IRequestHandler<CreateFunctionalityCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public CreateFunctionalityCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }


    public async Task<EntityResponse<bool>>
Handle(CreateFunctionalityCommand command,
        CancellationToken cancellationToken)
    {
        var request = new
CreateFunctionalityRequest(command.CodeRol,command.Name,
command.Image, command.Url, command.ParentId,
            command.Status, command.Order, command.TextColor,
        command.ColorButton, command.ColorPanel,
command.AudCreateUserId, command.AudUpdateUserId);
        var response = await _configurationService.Create(request,
command.Token);
```

```
        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"La Funcionalidad
{command.ParentId} no pudo ser creada.")
            : EntityResponse.Success(true);
    }
}

using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.FunctionalityComm
and;

public record DeleteFunctionalityCommand
(
    Guid FunctionalityId,
    string Token
) : IRequest<EntityResponse<bool>>;


using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.FunctionalityComm
and;

public class DeleteFunctionalityCommandHandler:
IRequestHandler<DeleteFunctionalityCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public DeleteFunctionalityCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<bool>>
Handle(DeleteFunctionalityCommand command, CancellationToken
cancellationToken)
    {
        var functionalityId = await
_configurationService.DeleteFunctionality(command.FunctionalityId
,command.Token);
        return EntityResponse.Success(functionalityId.Value);
    }
}
```

```
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.FunctionalityComm
and;

public record UpdateFunctionalityCommand
(
    Guid FunctionalityId,
    string Name,
    string Image,
    string Url,
    bool Status,
    Guid? ParentId,
    int Order,
    string TextColor,
    string ColorButton,
    string ColorPanel,
    DateTime UpdateAt,
    int AudUpdateUserId,
    string Token
): IRequest<EntityResponse<bool>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Fu
nctionality;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.FunctionalityComm
and;

public class UpdateFunctionalityCommandHandler :
IRequestHandler<UpdateFunctionalityCommand, EntityResponse<bool>>
{

    private readonly IConfigurationService _configurationService;

    public UpdateFunctionalityCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<bool>>
Handle(UpdateFunctionalityCommand command, CancellationToken
cancellationToken)
    {
        var request = new UpdateFunctionalityRequest(command.Name,
```

```
command.Image, command.Url, command.Status,
            command.ParentId, command.Order, command.TextColor,
            command.ColorButton, command.ColorPanel,
command.UpdateAt, command.AudUpdateUserId);
        var response = await
_configurationService.UpdateFunctionality(request,command.Function
alityId,command.Token);

        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"La Funcionalidad
{command.FunctionalityId} no pudo ser actualizara.")
            : EntityResponse.Success(true);
    }
}
```

ITEM CATALOG

```
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.ItemCatalogComman
d;

public record CreateItemCatalogCommand
(
    string Code,
    string Description,
    string Value,
    int Order,
    bool Status,
    Guid? ItemParentId,
    bool IsDelete,
    Guid CatalogId,
    int AudCreateUserId,
    int AudUpdateUserId,
    string Token
) : IRequest<EntityResponse<bool>>;

using
Investigation.Gateway.Admin.Application.Commands.ItemCatalogComman
d;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.It
emCatalog;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.ItemCalatogComman
d;

public class CreateItemCatalogCommandHandler :
IRequestHandler<CreateItemCatalogCommand, EntityResponse<bool>>
{
```

```
    private readonly IConfigurationService _configurationService;

    public CreateItemCatalogCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }


    public async Task<EntityResponse<bool>>
Handle(CreateItemCatalogCommand command,
        CancellationToken cancellationToken)
    {
        var request = new CreateItemCatalogRequest(command.Code,
command.Description, command.Value, command.Order,

command.Status,command.ItemParentId,command.IsDelete,command.Catal
ogId, command.AudCreateUserId, command.AudUpdateUserId);
        var catalogResponse = await
_configurationService.CreateItemCatalog(request, command.Token);

        return !catalogResponse.IsSuccess
            ? EntityResponse<bool>.Error($"El item del catalogo no
pudo ser creado.")
            : EntityResponse.Success(true);
    }
}

using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public record CreateUserCommand
(
    Guid RolId,
    int ServerId,
    string Email,
    bool Status,
    int? FacultyId,
    int AudCreateUserId,
    int AudUpdateUserId,
    string Token
) : IRequest<EntityResponse<bool>>;


using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
```

```
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public class CreateUserCommandHandler :
IRequestHandler<CreateUserCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public CreateUserCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }


    public async Task<EntityResponse<bool>>
Handle(CreateUserCommand command,
        CancellationToken cancellationToken)
    {
        var request = new
CreateUserRequest(command.RolId,command.ServerId, command.Email,
command.Status, command.FacultyId, command.AudCreateUserId,
            command.AudUpdateUserId);
        var response = await
_configurationService.CreateUser(request, command.Token);

        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"El Rolid
{command.RolId} no Existe.")
            : EntityResponse.Success(true);
    }
}

using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public record UpdateUserCommand
(
    Guid UserId,
    int ServerId,
    string Email,
    int? FacultyId,
    bool Status,
    string Token
    ): IRequest<EntityResponse<bool>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;
```

```
namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public class UpdateUserCommandHandler:
IRequestHandler<UpdateUserCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public UpdateUserCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }

    public async Task<EntityResponse<bool>>
Handle(UpdateUserCommand command, CancellationToken
cancellationToken)
    {
        var request = new UpdateUserRequest(command.ServerId,
command.Email,command.FacultyId, command.Status);
        var response = await
_configurationService.UpdateUser(request,command.UserId,
command.Token);

        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"El usuario
{command.UserId} no existe.")
            : EntityResponse.Success(true);
    }
}

using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;

public record UpdateUserRolCommand
(
    Guid UserId,
    List<Guid>RolesId,
    int AudServerId,
    string Token
): IRequest<EntityResponse<bool>>;

using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Shared.Api.Domain.Models;
using MediatR;

namespace
Investigation.Gateway.Admin.Application.Commands.UserCommand;
```

```
public class UpdateUserRolCommandHandler :
IRequestHandler<UpdateUserRolCommand, EntityResponse<bool>>
{
    private readonly IConfigurationService _configurationService;

    public UpdateUserRolCommandHandler(IConfigurationService
configurationService)
    {
        _configurationService = configurationService;
    }
    public async Task<EntityResponse<bool>>
Handle(UpdateUserRolCommand command, CancellationToken
cancellationToken)
    {
        var request = new
UpdateUserRolRequest(command.RolesId,command.AudServerId);
        var response = await
_configurationService.UpdateUserRol(request,command.UserId,
command.Token);

        return !response.IsSuccess
            ? EntityResponse<bool>.Error($"El usuario
{command.UserId} no existe.")
            : EntityResponse.Success(true);
    }

}
```

MIDDLEWARES

```
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Microsoft.IO;
using System.IO;
using System.Threading.Tasks;

namespace Investigation.Gateway.Admin.Infrastructure.Middlewares
{
   public class RequestResponseLoggingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger _logger;
        private readonly RecyclableMemoryStreamManager
_recyclableMemoryStreamManager;

        public RequestResponseLoggingMiddleware(RequestDelegate
next, ILogger<RequestResponseLoggingMiddleware> logger)
        {
            _next = next;
            _logger = logger;
            _recyclableMemoryStreamManager = new
RecyclableMemoryStreamManager();
        }

        public async Task Invoke(HttpContext context)
        {
```

```
            if (_logger.IsEnabled(LogLevel.Information))
            {
                await LogRequest(context);
                await LogResponse(context);
            }
            else
            {
                await _next(context);
            }
        }

        private async Task LogRequest(HttpContext context)
        {
            context.Request.EnableBuffering();

            await using MemoryStream requestStream =
_recyclableMemoryStreamManager.GetStream();
            await context.Request.Body.CopyToAsync(requestStream);

            _logger.LogInformation("Request, {Schema}, {Host},
{RequestPath}, {QueryString}, {Body}",
                                   context.Request.Scheme,
                                   context.Request.Host,
                                   context.Request.Path,
                                   context.Request.QueryString,

ReadStreamInChunks(requestStream));

            context.Request.Body.Position = 0;
        }

        /// <summary>
        /// Convert to string information received on stream
        /// </summary>
        private static string ReadStreamInChunks(Stream stream)
        {
            const int readChunkBufferLength = 4096;

            stream.Seek(0, SeekOrigin.Begin);

            using var textWriter = new StringWriter();
            using var reader = new StreamReader(stream);

            var readChunk = new char[readChunkBufferLength];
            int readChunkLength;

            do
            {
                readChunkLength = reader.ReadBlock(readChunk,
                                                   0,

readChunkBufferLength);
                textWriter.Write(readChunk, 0, readChunkLength);
            } while (readChunkLength > 0);

            return textWriter.ToString();
```

```
        }

        private async Task LogResponse(HttpContext context)
        {
            var originalBodyStream = context.Response.Body;

            await using MemoryStream responseBody =
_recyclableMemoryStreamManager.GetStream();
            context.Response.Body = responseBody;

            await _next(context);

            context.Response.Body.Seek(0, SeekOrigin.Begin);
            var text = await new
StreamReader(context.Response.Body).ReadToEndAsync();
            context.Response.Body.Seek(0, SeekOrigin.Begin);

            _logger.LogInformation("Response, {Schema}, {Host},
{RequestPath}, {QueryString}, {Body}",
                                   context.Request.Scheme,
                                   context.Request.Host,
                                   context.Request.Path,
                                   context.Request.QueryString,
                                   text);

            await responseBody.CopyToAsync(originalBodyStream);
        }

    }
}
```

SERVICIOS

```
using Investigation.Gateway.Admin.Domain.Dtos.Response.Academics;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class AcademicService : IAcademicService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly AcademicSettings _academicSettings;

    public AcademicService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<AcademicSettings> academicSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _serviceClient = new ServiceClient();
        _academicSettings = academicSettings.CurrentValue;
    }
```

```
    public async
Task<EntityResponse<List<ServerPersonalInformationResponse>>>
SearchServers(string filter,
        string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Academic}{_academicSettings.Inve
stigations.First()!.SearchServer}",
            filter);
        return await
_serviceClient.GetAcademic<List<ServerPersonalInformationResponse>
>(url, token);
    }
    public async Task<EntityResponse<List<CareerResponse>>>
GetCareers(string token)
    {
        var url =

$"{_baseUrlSettings.Academic}{_academicSettings.Investigations!.Fi
rst().GetCareers}";
        return await
_serviceClient.GetAcademic<List<CareerResponse>>(url, token);

    }
    public async
Task<EntityResponse<List<AcademicDegreesResponse>>>
AcademicDegrees(List<int> serverIds, string token)
    {
        var url =

$"{_baseUrlSettings.Academic}{_academicSettings.Investigations!.Fi
rst().AcademicDegrees}";
        return await
_serviceClient.PostAcademic<List<AcademicDegreesResponse>>(serverI
ds, url, token);
    }

    public async
Task<EntityResponse<List<ServerPersonalInformationResponse>>>
GetServersByIds(List<int> serverIds, string token)
    {
        var url =

$"{_baseUrlSettings.Academic}{_academicSettings.Investigations!.Fi
rst().GetServersByIds}";
        return await
_serviceClient.PostAcademic<List<ServerPersonalInformationResponse
>>(serverIds, url, token);
    }

    public async Task<EntityResponse<List<FacultyResponse>>>
GetFaculties (string token)
    {
        var url =
```

```
$"{_baseUrlSettings.Academic}{_academicSettings.Investigations!.Fi
rst().GetFaculties }";
        return await
_serviceClient.GetAcademic<List<FacultyResponse>>(url, token);
    }

    public async Task<EntityResponse<int>> GetPeriod(string token)
    {
        var url =
$"{_baseUrlSettings.Academic}{_academicSettings.Investigations.Fir
st()!.GetPeriod}";
        return await _serviceClient.GetAcademic<int>(url, token);
    }

    public async Task<EntityResponse<int>>
GetPeriodLanguage(string token)
    {
        var url =
$"{_baseUrlSettings.Academic}{_academicSettings.Investigations.Fir
st()!.GetPeriodLanguage}";
        return await _serviceClient.GetAcademic<int>(url, token);

    }

    public async Task<EntityResponse<CareerHigherLoad>>
GetCareerHigherLoad(int serverId, int periodId, string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Academic}{_academicSettings.Inve
stigations.First()!.GetCareerHigherLoad}",
            serverId, periodId);
        return await
_serviceClient.GetAcademic<CareerHigherLoad>(url, token);
    }
}

using Investigation.Gateway.Admin.Domain.Dtos.Requests.Centers;
using Investigation.Gateway.Admin.Domain.Dtos.Response.Centers;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class CenterService : ICenterService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly CentersSettings _centersSettings;

    public CenterService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<CentersSettings> centersSettings)
```

```
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _serviceClient = new ServiceClient();
        _centersSettings= centersSettings.CurrentValue;
    }


    public  async Task<EntityResponse<bool>>
Create(CreateInvestigationCenterRequest request, string token)
    {
        var url =

$"{_baseUrlSettings.Center}{_centersSettings.Centers!.First().Crea
teCenter}";
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async
Task<EntityResponse<List<InvestigationCenterResponse>>>
GetCenterByServerId(int serverId, string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Center}{_centersSettings.Centers
!.First().GetCenterByServerId}",
            serverId);
        return await
_serviceClient.Get<List<InvestigationCenterResponse>>(url, token);
    }
}



using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Ca
talog;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Fu
nctionality;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.It
emCatalog;
using
Investigation.Gateway.Admin.Domain.Dtos.Requests.Configurations.Us
er;
using
Investigation.Gateway.Admin.Domain.Dtos.Response.Configurations;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;
```

```
public class ConfigurationService : IConfigurationService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly ConfigurationsSettings
_configurationsSettings;

    public ConfigurationService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<ConfigurationsSettings>
configurationsSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _configurationsSettings =
configurationsSettings.CurrentValue;
        _serviceClient = new ServiceClient();
    }
    public async Task<EntityResponse<List<CatalogResponse>>>
GetCatalogs(string token)
    {
        var url =

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Catalog
s!.First().GetCatalogsWithItemCatalogs}";
        return await
_serviceClient.Get<List<CatalogResponse>>(url, token);
    }
    public async Task<EntityResponse<List<CatalogResponse>>>
GetCatalogByCode(string catalogCode, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Catalog
s!.First().GetCatalogByCode}",
            catalogCode);
        return await
_serviceClient.Get<List<CatalogResponse>>(url, token);
    }

    public async Task<EntityResponse<List<FunctionalityResponse>>>
GetFunctionalityRolId(Guid functionalityRolId, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Functio
nalities!.First().GetFunctionalitiesByRolId}",
            functionalityRolId);
        return await
_serviceClient.Get<List<FunctionalityResponse>>(url, token);
    }

    public async Task<EntityResponse<bool>>
UpdateCatalog(UpdateCatalogRequest request, Guid catalogId, string
token)
    {
```

```
        var url =string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Catalog
s!.First().UpdateCatalogId}",catalogId);
        return await _serviceClient.Patch<bool>(request,url,
token);
    }

    public async Task<EntityResponse<List<CatalogResponse>>>
GetItemsCatalogs(string token)
    {
        var url =

$"{_baseUrlSettings.Configuration}{_configurationsSettings.ItemCat
alogs!.First().GetItemCatalogAll}";
        return await
_serviceClient.Get<List<CatalogResponse>>(url, token);
    }

    public async
Task<EntityResponse<bool>>UpdateFunctionality(UpdateFunctionalityR
equest request,Guid id, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Functio
nalities!.First().UpdateFunctionality}",
            id);
        return await _serviceClient.Patch<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
Create(CreateFunctionalityRequest request, string token)
    {
        var url =string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Functio
nalities!.First().CreateFunctionality}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
CreateCatalog(CreateCatalogRequest request, string token)
    {
        var url =string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Catalog
s!.First().CreateCatalog}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
CreateItemCatalog(CreateItemCatalogRequest request, string token)
```

```
    {
        var url =string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.ItemCat
alogs!.First().CreateItemCatalog}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<ItemCatalogResponse>>
GetItemCatalogByCode(Guid code, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.ItemCat
alogs!.First().GetItemCatalogByCode}",
            code);
        return await _serviceClient.Get<ItemCatalogResponse>(url,
token);
    }



    public async Task<EntityResponse<RolResponse>>
GetRolByCode(string code, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Roles!.
First().GetRolByCode}",
            code);
        return await _serviceClient.Get<RolResponse>(url, token);
    }

    public async Task<EntityResponse<List<RolResponse>>>
GetRoles(string token)
    {
        var url =

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Roles!.
First().GetRoles}";
        return await _serviceClient.Get<List<RolResponse>>(url,
token);
    }


    public async Task<EntityResponse<UserResponse>>
GetUserInfo(string token)
    {
        var url =
$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().GetUserInfo}";
        return await _serviceClient.Get<UserResponse>(url, token);
    }
```

```
    public async
Task<EntityResponse<bool>>DeleteFunctionality(Guid
functionalityId, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Functio
nalities!.First().DeleteFunctionality}",functionalityId);
        return await _serviceClient.Delete<bool>(url, token);
    }



    public async Task<EntityResponse<bool>>
CreateUser(CreateUserRequest request, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().CreateUser}");
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<bool>>
UpdateUser(UpdateUserRequest request,Guid id, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().UpdateUser}",id);
        return await _serviceClient.Patch<bool>(request, url,
token);
    }

    public async Task<EntityResponse<UserResponse>>
GetUserById(Guid userId, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().GetUserById}",
            userId);
        return await _serviceClient.Get<UserResponse>(url, token);
    }

    public async Task<EntityResponse<List<UserResponse>>>
GetUserByList(string token)
    {
         var url
            =
$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().GetUserByList}";
        return await _serviceClient.Get<List<UserResponse>>(url,
token);
```

```
    }

    public async Task<EntityResponse<bool>>
UpdateUserRol(UpdateUserRolRequest request,Guid id, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Configuration}{_configurationsSettings.Users!.
First().UpdateUserRol}",id);
        return await _serviceClient.Patch<bool>(request, url,
token);


    }

}

using Investigation.Gateway.Admin.Domain.Dtos.Requests.Groups;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Dtos.Response.Groups;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class GroupService : IGroupService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly GroupsSettings _groupsSettings;

    public GroupService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<GroupsSettings> groupsSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _serviceClient = new ServiceClient();
        _groupsSettings = groupsSettings.CurrentValue;
    }
    public async Task<EntityResponse<bool>>
Create(CreateInvestigationGroupRequest request, string token)
    {
        var url =

$"{_baseUrlSettings.Group}{_groupsSettings.Groups!.First().CreateG
roup}";
        return await _serviceClient.Create<bool>(request, url,
token);
    }
    public async
Task<EntityResponse<List<InvestigationGroupResponse>>>
GetByIds(List<Guid> investigationGroupIds,
        string token)
```

```
    {
        var url =

$"{_baseUrlSettings.Group}{_groupsSettings.Groups!.First().GetById
s}";
        return await
_serviceClient.Post<List<InvestigationGroupResponse>>(investigatio
nGroupIds, url, token);
    }
    public async
Task<EntityResponse<List<InvestigationGroupResponse>>>
GetByServerId(int serverId, string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Group}{_groupsSettings.Groups!.F
irst().GetByServerId}",
            serverId);
        return await
_serviceClient.Get<List<InvestigationGroupResponse>>(url, token);
    }
    public async Task<EntityResponse<InvestigationGroupResponse>>
GetByInvestigationGroupId(Guid investigationGroupId,
        string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Group}{_groupsSettings.Groups!.F
irst().GetByInvestigationGroupId}",
            investigationGroupId);
        return await
_serviceClient.Get<InvestigationGroupResponse>(url, token);
    }
    public async Task<EntityResponse<bool>>
Create(CreateInternalPersonalInvestigatorRequest request, string
token)
    {
        var url =
$"{_baseUrlSettings.Group}{_groupsSettings.GroupPersonalInvestigat
ors!.First()
            .CreateInternalPersonal}";
        return await _serviceClient.Create<bool>(request, url,
token);
    }

}

using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Domain.Dtos.Response.Observatories;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class ObservatoryService : IObservatoryService
{
```

```
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly ObservatoriesSettings _observatoriesSettings;

    public ObservatoryService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<ObservatoriesSettings>
observatoriesSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _serviceClient = new ServiceClient();
        _observatoriesSettings =
observatoriesSettings.CurrentValue;
    }

    public async
Task<EntityResponse<IntellectualPropertyResponse>>
GetIntellectualPropertyByFilters(string identification,
        string lastName, string secondLastName, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.InvestigationDirection}{_observatoriesSettings
.Observatories!.First().GetIntellectualPropertyByFilters}",
            identification, lastName, secondLastName);
        return await
_serviceClient.Get<IntellectualPropertyResponse>(url, token);
    }
}

using Investigation.Gateway.Admin.Domain.Dtos.Requests.Projects;
using Investigation.Gateway.Admin.Domain.Dtos.Response.Projects;
using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Dtos.Response.Projects;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class ProjectService : IProjectService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ProjectsSettings _projectsSettings;
    private readonly ServiceClient _serviceClient;

    public ProjectService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<ProjectsSettings> projectsSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _projectsSettings = projectsSettings.CurrentValue;
        _serviceClient = new ServiceClient();
    }
```

```
    public async Task<EntityResponse<bool>>
Create(CreateProjectInvestigationRequest request, string token)
    {
        var url =

$"{_baseUrlSettings.Project}{_projectsSettings.Projects!.First().C
reateProject}";
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async
Task<EntityResponse<List<ProjectInvestigationResponse>>>
GetByServerId(int serverId, string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Project}{_projectsSettings.Proje
cts!.First().GetProjectByServerId}",
            serverId);
        return await
_serviceClient.Get<List<ProjectInvestigationResponse>>(url,
token);
    }

    public async Task<EntityResponse<bool>>
CreateCoverage(CreateExecutionCoverageRequest request, string
token)
    {
        var url =
$"{_baseUrlSettings.Project}{_projectsSettings.ExecutionCoverages!
.First().CreateExecutionCoverage}";
        return await _serviceClient.Create<bool>(request, url,
token);
    }

    public async Task<EntityResponse<ExecutionCoverageResponse>>
GetExecutionByProjectInvestigationId(
        Guid projectInvestigationId, string token)
    {
        var url = string.Format(

$"{_baseUrlSettings.Project}{_projectsSettings.ExecutionCoverages!
.First().GetExecutionByProjectInvestigationId}",
            projectInvestigationId);
        return await
_serviceClient.Get<ExecutionCoverageResponse>(url, token);
    }

    public async
Task<EntityResponse<ObjectiveResponse>>GetObjectiveByProjectInvest
igationId(Guid projectInvestigationId,
        string token)
    {
        var url = string.Format(
```

```
$"{_baseUrlSettings.Project}{_projectsSettings.Objectives!.First()
.GetObjectiveByProjectInvestigationId}",
            projectInvestigationId);
        return await _serviceClient.Get<ObjectiveResponse>(url,
token);
    }
}

using Investigation.Gateway.Admin.Domain.Services;
using Investigation.Gateway.Admin.Domain.Settings;
using Investigation.Gateway.Domain.Dtos.Response.Seedbeds;
using Investigation.Gateway.Domain.Settings;
using Investigation.Shared.Api.Domain.Models;
using Microsoft.Extensions.Options;

namespace Investigation.Gateway.Admin.Infrastructure.Services;

public class SeedbedService : ISeedbedService
{
    private readonly BaseUrlSettings _baseUrlSettings;
    private readonly ServiceClient _serviceClient;
    private readonly SeedbedsSettings _seedbedsSettings;

    public SeedbedService(IOptionsMonitor<BaseUrlSettings>
baseUrlSettings,
        IOptionsMonitor<SeedbedsSettings> seedbedsSettings)
    {
        _baseUrlSettings = baseUrlSettings.CurrentValue;
        _serviceClient = new ServiceClient();
        _seedbedsSettings = seedbedsSettings.CurrentValue;
    }

    public async Task<EntityResponse<List<SeedbedResponse>>>
GetSeedbedByServerId(int serverId, string token)
    {
        var url =
string.Format($"{_baseUrlSettings.Seedbed}{_seedbedsSettings.Seedb
eds!.First().GetSeedbedByServerId}",
            serverId);
        return await
_serviceClient.Get<List<SeedbedResponse>>(url, token);
    }
}
```