UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

TESIS:

“Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios, Ayacucho, 2025”

Para optar el título profesional de:
INGENIERO DE SISTEMAS

PRESENTADO POR:
Bach. Julio Marti AYALA BELLIDO

ASESORA:
Mg. Ing. Elinar CARRILLO RIVEROS

AYACUCHO - PERÚ

2026

DEDICATORIA

Dedico esta tesis a mis padres y a mi familia, quienes con su esfuerzo, apoyo y confianza me brindaron las oportunidades necesarias para alcanzar esta meta. Su ejemplo de trabajo, responsabilidad y perseverancia ha sido una fuente constante de inspiración en cada etapa de mi formación.

También dedico este logro a todas las personas que creyeron en mis capacidades, incluso en los momentos de mayor incertidumbre, motivándome a seguir adelante y a no rendirme frente a los desafíos.

Finalmente, dedico este trabajo a mí mismo, como resultado de años de aprendizaje, sacrificio y compromiso con mis objetivos. Cada dificultad superada y cada conocimiento adquirido forman parte del camino que hoy culmina con la presentación de esta tesis.

AGRADECIMIENTO

A mis padres y familiares, por su apoyo incondicional, sus consejos y la confianza depositada en mí a lo largo de todos estos años. Su respaldo ha sido fundamental para mantener la motivación y la determinación necesarias para alcanzar esta meta.

A mis docentes y asesores, por compartir sus conocimientos, experiencia y orientación durante mi formación académica y el desarrollo de esta investigación. Sus aportes contribuyeron significativamente a mi crecimiento profesional y personal.

Asimismo, agradezco a mis compañeros, amigos y a todas aquellas personas que, de una u otra manera, colaboraron durante este proceso, brindándome apoyo, sugerencias y motivación para continuar avanzando.

Finalmente, agradezco a la tecnología y al constante aprendizaje que esta exige, por permitirme desarrollar habilidades que hoy forman parte de mi vocación profesional. Esta tesis representa no solo la culminación de una etapa académica, sino también el inicio de nuevos desafíos y oportunidades para seguir creciendo, innovando y aportando valor a través de mi trabajo.

INDICE

LISTA DE TABLAS vii
LISTA DE FIGURAS viii
RESUMEN xi
INTRODUCCIÓN xii

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

1.1. DIAGNÓSTICO Y ENUNCIADO DEL PROBLEMA 14
1.1.1. JUSTIFICACIÓN 18
1.1.2. DELIMITACIÓN 18
1.2. FORMULACIÓN DEL PROBLEMA 19
1.2.1. PROBLEMA GENERAL 19
1.2.2. PROBLEMAS ESPECÍFICOS 19
1.3. OBJETIVOS 19
1.3.1. OBJETIVO GENERAL 19
1.3.2. OBJETIVOS ESPECÍFICOS 19
1.4. HIPÓTESIS 20

CAPÍTULO II

MARCO TEÓRICO

2.1. ANTECEDENTES DE LA INVESTIGACIÓN 21
2.2. BASE TEÓRICA 25
2.2.1. MIGRACIÓN DE SOFTWARE 25
2.2.2. PATRÓN STRANGLER FIG 27
2.3. BASE REFERENCIAL 34
2.3.1. ARQUITECTURA DE SOFTWARE 34
2.3.2. ARQUITECTURA MONOLÍTICA 35
2.3.3. ARQUITECTURA DE MICROSERVICIOS 36

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1. ENFOQUE DE INVESTIGACIÓN 40
3.2. TIPO DE INVESTIGACIÓN 40
3.3. NIVEL DE INVESTIGACIÓN 41
3.4. DISEÑO DE LA INVESTIGACIÓN 42
3.5. UNIDAD DE ANÁLISIS Y UNIDADES DE OBSERVACIÓN 43
3.6. VARIABLES Y DIMENSIONES 44
3.7. TÉCNICAS E INSTRUMENTOS PARA RECOLECTAR INFORMACIÓN .. 46
3.8. TÉCNICAS DE PROCESAMIENTO Y ANÁLISIS DE DATOS 47
3.9. IDENTIFICACIÓN DE BOUNDED CONTEXTS 48
3.10. CRITERIOS Y MÉTODO DE PRIORIZACIÓN 48

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1. ANÁLISIS DEL SISTEMA MONOLÍTICO 52
4.1.2. ARQUITECTURA DEL MONOLITO 54
4.1.3. INVENTARIO DE MÓDULOS DEL SISTEMA 55
4.1.4. DEPENDENCIAS ENTRE MÓDULOS DEL SISTEMA 57
4.1.5. LIMITACIONES DEL SISTEMA 59
4.2. ARQUITECTURA PROPUESTA 60
4.2.1. MAPEO DE MÓDULOS A MICROSERVICIOS 61
4.2.2. PRINCIPIOS DE DISEÑO ADOPTADOS 62
4.2.3. DIAGRAMA DE ARQUITECTURA PROPUESTA 63
4.2.4. STACK TECNOLÓGICO 65
4.3. PLAN DE MIGRACIÓN 65
4.3.1. ESTRATEGIA DE MIGRACIÓN 65
4.3.2. PRIORIZACIÓN Y ORDEN DE MIGRACIÓN 66

4.3.3. CRONOGRAMA DE SPRINTS 69
4.4. APLICACIÓN DEL PATRÓN STRANGLER FIG 70
4.4.1. PRE MIGRACIÓN 70
4.4.2. SPRINT 1 – FILE SERVICE 72
4.4.3. SPRINT 2 – DOCUMENT SERVICE 81
4.4.4. SPRINT 3 – PERSON SERVICE 91
4.4.5. SPRINT 4 – SESSION SERVICE 99
4.4.6. SPRINT 5 – ACADEMIC SERVICE 106
4.4.7. SPRINT 6 – USERS SERVICE 113
4.4.8. SPRINT 7 – DILIGENCESERVICE Y PARAMETERSSERVICE 119
4.4.9. SPRINT 8 – BASE DE DATOS Y COMUNICACIÓN ENTRE SERVICIOS 124
4.4.10. DOCKERIZACIÓN 127
4.5. RESULTADO FINAL DE LA MIGRACIÓN 131
4.5.1. ESTADO FINAL DEL SISTEMA 131
4.5.2. TRAZABILIDAD DEL PATRÓN STRANGLER FIG 133
4.5.3. COMPARATIVA GLOBAL DEL SISTEMA 134
4.5.4. REGISTRO DE TIEMPOS DE RESPUESTA 135
4.6. DISCUSIÓN 137

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES 143
5.2. RECOMENDACIONES 145
5.3. LIMITACIONES DEL ESTUDIO 147
REFERENCIAS BIBLIOGRÁFICAS 149
ANEXOS 154
ANEXO A. INSTRUMENTO DE RECOLECCIÓN DE DATOS 154

ANEXO B. MATRIZ DE CONSISTENCIA 156
ANEXO C. MATRIZ DE OPERACIONALIZACIÓN DE VARIABLES 158
ANEXO D. PROCESO SCRUM ADOPTADO 160
ANEXO E. FICHA DE ANÁLISIS DOCUMENTAL EN FILE SERVICE 162
ANEXO F. GUIA DE SCORING 164
ANEXO G. PROYECTO MONOLITO Y MICROSERVICIOS 165
ANEXO H. PROYECTO MICROSERVICIOS 171
ANEXO I. VISTAS DEL SISTEMA 175
ANEXO J. VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS 179
ANEXO K. TABLA DE CÁLCULO V DE AIKEN 187
ANEXO L. CAPTURAS DE TIEMPOS DE RESPUESTA 188

LISTA DE TABLAS

Tabla 1 Criterios de priorización y pesos asignados *49*
Tabla 2 Tabla de Componentes Base para la migración *50*
Tabla 3 Características del Sistema Monolítico *53*
Tabla 4 Lista de Módulos del Monolito *55*
Tabla 5 Matriz de dependencias entre módulos del sistema monolítico *57*
Tabla 6 Tabla de Limitaciones del Sistema *59*
Tabla 7 Identificación de Bounded Contexts y su correspondencia con módulos del monolito *60*
Tabla 8 Mapeo de bounded contexts a microservicios *61*
Tabla 9 Stack tecnológico *65*
Tabla 10 Puntuación de priorización para cada microservicio *66*
Tabla 11 Configuración original del monolito *70*
Tabla 12 Endpoints implementados — FileService *73*
Tabla 13 Criterios de aceptación — FileService *73*
Tabla 14 Endpoints implementados — DocumentService *82*
Tabla 15 Criterios de aceptación — DocumentService *82*
Tabla 16 Endpoints implementados — PersonService *91*
Tabla 17 Criterios de aceptación — PersonService *92*
Tabla 18 Endpoints implementados — SesionService *99*
Tabla 19 Criterios de aceptación — SesionService *100*
Tabla 20 Endpoints implementados — AcademicService *106*
Tabla 21 Criterios de aceptación — AcademicService *108*
Tabla 22 Endpoints implementados — UserService *113*
Tabla 23 Criterios de aceptación — UserService *114*
Tabla 24 Endpoints implementados — DiligenceService *119*
Tabla 25 Endpoints implementados — ParametersService *120*
Tabla 26 Criterios de aceptación — DiligenceService y ParametersService *120*
Tabla 27 Motor de Base de Datos por Servicio *124*
Tabla 28 Mapa completo de dependencias HTTP *125*
Tabla 29 Resumen de microservicios implementados al finalizar la migración *131*
Tabla 30 Trazabilidad del patrón Strangler Fig *133*
Tabla 31 Comparativa entre el sistema monolítico original y el sistema de microservicios resultante *134*
Tabla 32 Tiempos de Respuesta *136*

LISTA DE FIGURAS

Figura 1 Tamaño del mercado de microservicios en la nube *15*
Figura 2 Mercado de microservicios en la nube: tasa de crecimiento por región *16*
Figura 3 Fase de Transformación - Configuración Inicial *30*
Figura 4 Fase de Coexistencia – Convivencia Monolito y Microservicio *31*
Figura 5 Fase de Eliminación – Esquema del sistema después de la eliminación *32*
Figura 6 Fases del Proceso de Migración con el Patrón Strangler Fig según AWS - Ejemplo *33*
Figura 7 Ejemplo de Arquitectura Monolítica y Microservicio *37*
Figura 8 Configuración de Eureka Server *50*
Figura 9 Configuración de Config Server *51*
Figura 10 Configuración de API Gateway *51*
Figura 11 Arquitectura Actual del Monolito *54*
Figura 12 Arquitectura Propuesta *64*
Figura 13 Cronograma de migración *69*
Figura 14 Request URL Original del Monolito *70*
Figura 15 Request URL del monolito con el API Gateway *71*
Figura 16 Arquitectura del sistema pre-migración *71*
Figura 17 Registro de Eureka Server *74*
Figura 18 Endpoint Health de File Service *75*
Figura 19 Endpoint de Registro de File Service *75*
Figura 20 Endpoint de Listado de File Service *76*
Figura 21 Lista de Expedientes de Bachiller con Endpoint de FileService *76*
Figura 22 Lista de Expedientes de Bachiller con Endpoint de Monolito *77*
Figura 23 Configuración de rutas del Sprint 1 ..... *77*
Figura 24 Arquitectura del sistema en el Sprint 1 ..... *78*
Figura 25 Request de Listado de Expedientes del Monolito *79*
Figura 26 Request de Lista de Expedientes de FileService *79*
Figura 27 Bloqueo del módulo del Sprint 1 ..... *80*
Figura 28 Rutas Bloqueadas del Monolito - Sprint 1 ..... *80*
Figura 29 Registro de Eureka Server *83*
Figura 30 Endpoint de Generación de ListaCU de Document Service *83*
Figura 31 Documento Lista Oficial Generado *84*
Figura 32 Endpoint de Generación de Resoluciones de Document Service *85*
Figura 33 Documento Resolución Generado *85*
Figura 34 Documento Diploma Generado *86*
Figura 35 Lista Oficial generada *86*

Figura 36 Configuración de rutas del Sprint 2 ..... *87*
Figura 37 Arquitectura del sistema en el Sprint 2 ..... *88*
Figura 38 Bloqueo del módulo del Sprint 2 ..... *89*
Figura 39 Rutas Bloqueadas del Monolito - Sprint 2 ..... *90*
Figura 40 Registro de Eureka Server *92*
Figura 41 Endpoint de Consulta de DNI *93*
Figura 42 Validación de respuesta de la API UNSCH *93*
Figura 43 Lógica de consumo de API UNSCH *94*
Figura 44 Endpoint de registro de Persona *94*
Figura 45 Comparación de response de consulta de Personas *95*
Figura 46 Arquitectura del sistema en el Sprint 3 ..... *96*
Figura 47 Configuración de rutas del Sprint 3 ..... *96*
Figura 48 Request de Consulta de DNI en el monolito *97*
Figura 49 Request de Consulta de DNI por PersonService *97*
Figura 50 Bloqueo del módulo del Sprint 3 ..... *98*
Figura 51 Rutas Bloqueadas del Monolito - Sprint 3 ..... *98*
Figura 52 Registro de Eureka Server *100*
Figura 53 Sesión Activa *101*
Figura 54 Activación de otra Sesión *101*
Figura 55 Sesión Anterior Desactivada *102*
Figura 56 Datos de Sesión Activa obtenido desde el monolito *102*
Figura 57 Datos de Sesión Activa desde SesionService *103*
Figura 58 Configuración de rutas del Sprint 4 ..... *103*
Figura 59 Arquitectura del sistema en el Sprint 4 ..... *104*
Figura 60 Bloqueo del módulo del Sprint 4 ..... *105*
Figura 61 Rutas Bloqueadas del Monolito - Sprint 4 ..... *105*
Figura 62 Registro de Eureka Server *108*
Figura 63 Facultades Registradas *109*
Figura 64 Endpoint de consulta por código de escuela *109*
Figura 65 Endpoint de listado de denominaciones *110*
Figura 66 Configuración de rutas del Sprint 5 ..... *110*
Figura 67 Arquitectura del sistema en el Sprint 5 ..... *111*
Figura 68 Bloqueo del módulo del Sprint 5 ..... *112*
Figura 69 Rutas Bloqueadas del Monolito - Sprint 5 ..... *112*
Figura 70 Registro de Eureka Server *114*
Figura 71 Servicio Login *115*

Figura 72 Validación de Usuario Incorrecto *115*
Figura 73 Encriptación de contraseña *116*
Figura 74 Validación del Token *116*
Figura 75 Arquitectura del sistema en el Sprint 6 ..... *117*
Figura 76 Configuración de rutas del Sprint 6 ..... *118*
Figura 77 Bloqueo del módulo del Sprint 6 ..... *118*
Figura 78 Registro de Eureka Server *120*
Figura 79 Servicio Obtener Diligencia por Sesión *121*
Figura 80 Obtener listado de Niveles *121*
Figura 81 Arquitectura del sistema en el Sprint 7 ..... *122*
Figura 82 Configuración de rutas del Sprint 7 ..... *123*
Figura 83 Bloqueo del módulo del Sprint 7 ..... *123*
Figura 84 Configuración de base de datos propia - FileService *126*
Figura 85 Configuración de rutas para conexión HTTP entre Servicios *126*
Figura 86 Eliminación del monolito *127*
Figura 87 Configuración de application-docker.yml del API Gateway *128*
Figura 88 Configuración de Docker-compose.yml *129*
Figura 89 Deploy de microservicios *129*
Figura 90 Registro final en Eureka *130*
Figura 91 Arquitectura del sistema post – migración *132*

RESUMEN

Actualmente, la creciente demanda de escalabilidad y flexibilidad en los sistemas web ha evidenciado las limitaciones de las arquitecturas monolíticas, caracterizadas por su rigidez, alto acoplamiento entre módulos y dificultad para adaptarse a cambios constantes. Ante este problema, la presente investigación tuvo como objetivo describir la aplicación del Patrón Strangler Fig en la migración de un sistema web monolítico a microservicios. Se toma como caso de estudio el Sistema de Grados y Títulos de la Universidad Nacional de San Cristóbal de Huamanga (UNSCH), y se detalla el proceso a través de sus tres fases: Transformación, Coexistencia y Eliminación.

La investigación fue de tipo aplicada, de enfoque mixto de predominancia cualitativa, nivel descriptivo y diseño de estudio de caso único con implementación, de corte longitudinal. El proceso se desarrolló en ocho sprints bajo el marco de trabajo ágil Scrum e incluyó una infraestructura basada en Eureka Server, API Gateway y Config Server. Como instrumento se empleó una Ficha de Análisis Documental estructurada según las tres fases del patrón.

Los resultados evidencian la aplicación exitosa del patrón a lo largo de las fases de Transformación, Coexistencia y Eliminación. Los módulos de expedientes académicos fueron consolidados en un microservicio según su cohesión de dominio, mientras que la coexistencia entre el sistema monolítico y los microservicios se logró mediante autenticación híbrida con sesiones PHP y JWT. Finalmente, el código PHP fue retirado progresivamente hasta dejar el sistema operando íntegramente sobre microservicios independientes gestionados por el API Gateway. Además, se identificó que la priorización por puntuación numérica debía complementarse con un análisis de dependencias para resolver restricciones de orden entre servicios.

Los resultados demuestran que el Patrón Strangler Fig permitió modernizar el sistema monolítico de manera incremental sin interrumpir su operación continua, demostrando su viabilidad. El estudio aporta evidencia práctica sobre la aplicación de sus tres fases y ofrece recomendaciones para futuros procesos de migración de sistemas legacy.

PALABRAS CLAVE: Migración a Microservicios, Patrón Strangler Fig, Sistema Monolítico, Proceso de Migración.

INTRODUCCIÓN

En el contexto actual del desarrollo de software, los sistemas web enfrentan una presión creciente para adaptarse con rapidez a nuevos requerimientos funcionales, escalar ante el aumento de la demanda y mantenerse disponibles de forma continua. Sin embargo, muchas organizaciones, especialmente instituciones públicas y universidades, operan sobre sistemas heredados construidos bajo una arquitectura monolítica, en la que todas las funcionalidades conviven en una única base de código desplegada como un solo bloque. Si bien este tipo de arquitectura fue apropiada en el momento de su desarrollo, con el tiempo se convierte en un obstáculo para la evolución tecnológica, generando altos costos de mantenimiento, ciclos de despliegue lentos y un riesgo elevado ante cualquier modificación del sistema (Li et al., 2020).

El Sistema de Grados y Títulos UNSCH, desarrollado en PHP 7.4 nativo sin framework bajo el patrón Modelo-Vista-Controlador (MVC), gestionaba los procesos de generación de grados académicos y títulos profesionales a través de dieciséis módulos fuertemente acoplados en una única aplicación. Este sistema presentaba limitaciones concretas que afectaban su operación: lentitud, ausencia de roles diferenciados de acceso y dificultades para incorporar nuevas funcionalidades sin afectar las existentes.

Es por ello que, la migración hacia una arquitectura de microservicios se presenta como una solución viable y ampliamente respaldada por la literatura técnica y científica. Los microservicios permiten descomponer una aplicación en componentes independientes, cada uno desplegable, escalable y actualizable de forma autónoma, lo que reduce el acoplamiento, mejora la tolerancia a fallos y acelera los ciclos de desarrollo (RedHat, 2023). Sin embargo, la transición desde un sistema monolítico en producción hacia microservicios no está exenta de riesgos. Realizarla de forma abrupta lo que se conoce como migración *big bang* implica detener el sistema completo, asumir una reescritura total del código y exponer a la organización a un período prolongado sin servicio, lo cual es inaceptable en un sistema de uso institucional continuo (Hasan et al., 2023).

Para abordar este desafío, el Patrón Strangler Fig, introducido por Martin Fowler en 2004 e incorporado por Amazon Web Services (AWS) como uno de sus patrones prescriptivos de migración en la nube, propone una estrategia incremental que permite reemplazar gradualmente los módulos del sistema antiguo por nuevos microservicios, manteniendo ambos sistemas en operación simultánea durante la transición. El patrón se aplica en tres

fases secuenciales: Transformación, en la que se desarrolla el nuevo microservicio en paralelo al monolito; Coexistencia, en la que ambos sistemas operan simultáneamente y el tráfico es redirigido progresivamente hacia el nuevo componente mediante un *proxy* o API Gateway; y Eliminación, en la que el módulo equivalente del monolito es retirado una vez validado el microservicio (AWS, 2023). Este enfoque reduce significativamente el riesgo de la migración al permitir la validación incremental de cada componente antes de eliminar su equivalente en el sistema original.

A pesar de la relevancia del tema, la literatura existente sobre la aplicación práctica del Patrón Strangler Fig en sistemas institucionales universitarios de América Latina es escasa. Entre las investigaciones de referencia más directas, Ma et al. (2022) y Li et al. (2020) aplicaron el patrón en sistemas monolíticos de otros rubros, y Thamma et al. (2025) lo emplearon en la migración de un sistema de gestión de carnés para personas con discapacidad. En el contexto peruano, los trabajos de Noriega Yarlequé (2022) y Mendoza Vallejos & Alva Caffo (2024) abordan migraciones a microservicios, pero sin centrarse específicamente en el Patrón Strangler Fig ni en el contexto universitario regional. Esta brecha justifica la pertinencia de la presente investigación.

En consecuencia, este trabajo tuvo como objetivo describir la aplicación del Patrón Strangler Fig en la migración del Sistema de Grados y Títulos de la UNSCH hacia una arquitectura de microservicios, detallando el proceso a través de sus tres fases. La investigación es de tipo aplicada, de enfoque mixto de predominancia cualitativa, nivel descriptivo y diseño de estudio de caso único con implementación, de corte longitudinal, y se organizó mediante el marco de trabajo ágil Scrum en ocho sprints de desarrollo, en los que los dieciséis módulos del monolito fueron migrados progresivamente hacia ocho microservicios implementados en Java con Spring Boot, coordinados por un API Gateway con Spring Cloud Gateway y un servidor de descubrimiento Eureka.

El presente documento se organiza de la siguiente manera. El Capítulo I presenta el plan de investigación, que incluye los antecedentes, el planteamiento del problema y los objetivos. El Capítulo II contiene marco teórico. El Capítulo III describe el marco metodológico adoptado. El Capítulo IV expone los resultados de la investigación, detallando el análisis del sistema monolítico, la arquitectura propuesta, el plan de migración y la aplicación del patrón en cada sprint, y presenta la discusión de los resultados en contraste con los antecedentes. Finalmente, el Capítulo V recoge las conclusiones y recomendaciones derivadas del estudio.

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

1.1. DIAGNÓSTICO Y ENUNCIADO DEL PROBLEMA

En el pasado, las aplicaciones heredadas se desarrollaban utilizando una arquitectura monolítica. Con el tiempo, factores como actualizaciones técnicas, cambios de personal, modificaciones en los requisitos y la incorporación de nuevas características pueden transformar estas aplicaciones en sistemas complejos y difíciles de manejar lo que limita significativamente su escalabilidad y capacidad de mantenimiento (Li et al., 2020). La tecnología utilizada puede volverse obsoleta, lo que eleva los costos de mantenimiento. Las aplicaciones antiguas y monolíticas se construyeron siguiendo las mejores prácticas de su época, pero no fueron diseñadas para ser distribuidas eficientemente (AWS, 2025). Es por ello que, muchas organizaciones enfrentan el reto de migrar sus aplicaciones monolíticas a una arquitectura más flexible. Este proceso es complicado, especialmente para los sistemas que ya están en funcionamiento y son esenciales para las operaciones diarias (Ren et al., 2018). Adicionalmente, en los últimos años, el desarrollo de software ha experimentado un cambio significativo hacia la construcción de proyectos mediante la combinación de pequeños componentes, lo que se conoce como aplicaciones basadas en microservicios (Freitas et al., 2023).

Mordor Intelligence Research & Advisory (2023) realizó un informe en el que se analiza el crecimiento del mercado global de microservicios en la nube, abarcando diferentes regiones geográficas y dividiéndolo en segmentos según el tipo de implementación, el tamaño de la empresa y la industria. Según el informe, se espera que el mercado de microservicios crezca a una tasa compuesta anual del 22.88 % entre 2024 y 2029.

La figura 1 presenta un análisis del mercado de microservicios en la nube, destacando su crecimiento proyectado, las regiones con mayor dinamismo y los actores clave en la industria.

Figura 1

Tamaño del mercado de microservicios en la nube



Nota: Tomado de Mordor Intelligence Research & Advisory (2023).

Los microservicios han ganado popularidad como un patrón arquitectónico debido a la necesidad creciente de escalar, adaptarse y hacer más flexibles los proyectos de software, impulsados por el auge de la computación en la nube y las prácticas de DevOps (Brito et al., 2021).

Actualmente, muchas empresas eligen sustituir su arquitectura monolítica por una de microservicios con el objetivo de mejorar la calidad de software, y como resultado, la migración hacia microservicios se ha vuelto un tema clave en la investigación práctica. Sin embargo, todavía no existe un proceso de migración completamente definido y sistemático (Ma et al., 2022).

La migración de un sistema web monolítico presenta desafíos significativos que pueden afectar la continuidad operativa y la calidad del servicio. Este proceso requiere un enfoque metódico para garantizar que la transformación, coexistencia y eliminación

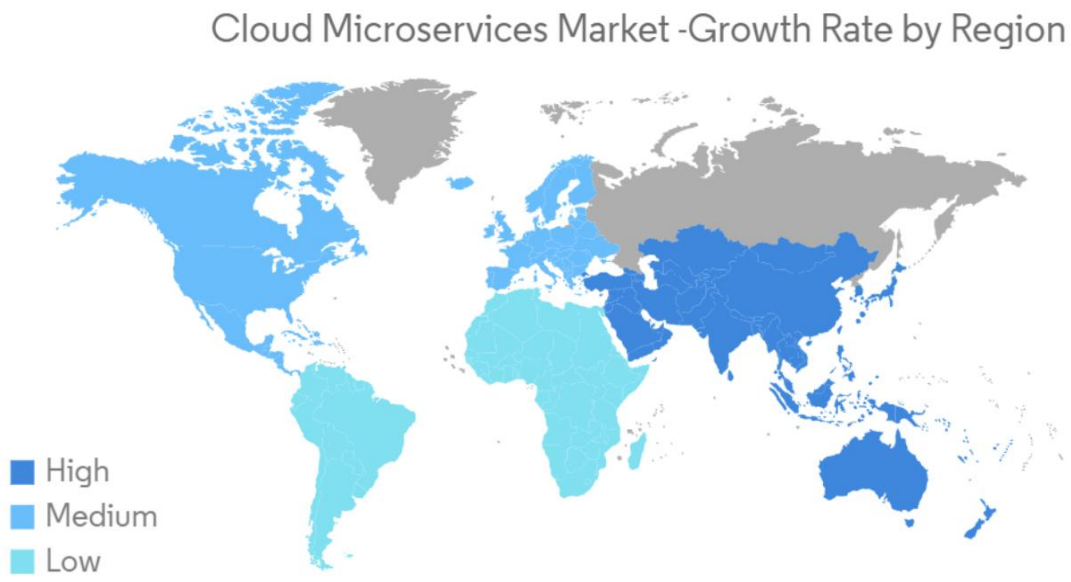
de componentes se realice de manera efectiva.

Una solución viable es la aplicación del Strangler Fig Pattern, un enfoque que permite una migración gradual de los sistemas monolíticos. A través de la implementación progresiva de microservicios, este patrón reduce riesgos al mantener el sistema original en funcionamiento mientras se reemplazan módulos de manera controlada, lo que facilita una transición más segura y eficiente.

América Latina es una de las regiones con menor tasa de crecimiento en el mercado de microservicios, lo que indica que muchas empresas en la región enfrentan dificultades en la modernización de sus sistemas, tienen una alta dependencia de infraestructuras heredadas y enfrentan desafíos relacionados con la inversión y los recursos (Mordor Intelligence Research & Advisory, 2023).

Figura 2

Mercado de microservicios en la nube: tasa de crecimiento por región



Nota: Tomado de Mordor Intelligence Research & Advisory, (2023).

En el Perú, muchas organizaciones, especialmente en regiones como Ayacucho, utilizan sistemas monolíticos que han estado en funcionamiento durante años para

gestionar sus operaciones. Un ejemplo de esto es el Sistema de Grados y Títulos de la Universidad Nacional de San Cristóbal de Huamanga (UNSCH). Este sistema, desarrollado bajo una arquitectura monolítica, ha enfrentado serias dificultades para adaptarse a los constantes cambios en los requisitos funcionales y la incorporación de nuevas funcionalidades.

En el análisis realizado al Sistema de Grados y Títulos de la UNSCH se identificaron diversos problemas propios del sistema que sustentan esta investigación. Entre ellos se encontró la dificultad para incorporar nuevas funcionalidades, validaciones o cambios sin afectar otros módulos del sistema, lo que limita su evolución. Asimismo, se evidenció un código altamente acoplado y sin una adecuada separación de capas, lo que dificulta el mantenimiento y la corrección de errores. También se observó que cada actualización requiere desplegar nuevamente toda la aplicación, incrementando el tiempo y el riesgo durante las implementaciones. Otro problema identificado fue la falta de roles y permisos diferenciados para los usuarios, lo que representa una limitación en términos de seguridad. Del mismo modo, la ausencia de mecanismos de registro y monitoreo dificulta la detección y el seguimiento de errores. Se comprobó además que un fallo en cualquiera de los módulos puede afectar el funcionamiento de toda la aplicación, debido a la naturaleza monolítica del sistema. A ello se suma la duplicidad de lógica de negocio entre distintos controladores, lo que incrementa el riesgo de inconsistencias en los datos, así como un crecimiento desordenado del sistema producto de mejoras realizadas sin una planificación adecuada ni documentación suficiente. Estos hallazgos, obtenidos a partir del análisis directo del sistema, complementan las tendencias observadas en el mercado global sobre la adopción de microservicios y permiten sustentar el problema de investigación con evidencia específica del caso de estudio.

1.1.1. JUSTIFICACIÓN

En la presente investigación, se describe el proceso de migración de un sistema monolítico a una arquitectura de microservicios utilizando el Patrón Strangler Fig, que ofrece un enfoque estructurado para minimizar los riesgos asociados con este tipo de migración.

Esta investigación es relevante porque detalla y proporciona información sobre cada fase del proceso, lo que permite a desarrolladores y arquitectos de software implementar estrategias más efectivas en sus proyectos de migración. Además, el estudio de caso del Sistema de Grados y Títulos de la UNSCH permite describir la aplicabilidad del patrón en un contexto real y ofrecer recomendaciones basadas en evidencia.

Asimismo, esta investigación contribuye a la generación de nuevas ideas respecto a la aplicación del Patrón Strangler Fig en la migración de sistemas monolíticos y sirve como guía para futuras investigaciones en este campo.

1.1.2. DELIMITACIÓN

Esta investigación se centró en la aplicación del Patrón Strangler Fig en la migración de un sistema web monolítico a microservicios, específicamente en el contexto del Sistema de Grados y Títulos de la Universidad Nacional de San Cristóbal de Huamanga (UNSCH), en Ayacucho, Perú, durante el año 2025. El estudio abarcó las fases de transformación, coexistencia y eliminación del proceso de migración, describiendo las técnicas, herramientas y resultados obtenidos. No se abordaron aspectos relacionados con la implementación de otros patrones arquitectónicos ni se realizó un análisis comparativo con otros sistemas que no sean el mencionado. Al tratarse de un estudio de caso único, los hallazgos descritos corresponden a las condiciones particulares del sistema estudiado y no son directamente generalizables a otros procesos de migración sin la validación correspondiente.

1.2. FORMULACIÓN DEL PROBLEMA

1.2.1. PROBLEMA GENERAL

¿Cómo se aplica el patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios?

1.2.2. PROBLEMAS ESPECÍFICOS

a. ¿Cómo se desarrolla la transformación de componentes de un sistema web monolítico a microservicios con el Patrón Strangler Fig?
b. ¿Cómo se gestiona la coexistencia entre los componentes del sistema monolítico y los microservicios durante el proceso de migración con el Patrón Strangler Fig?
c. ¿Cómo se realiza la eliminación del monolito en el proceso de migración con el Patrón Strangler Fig?

1.3. OBJETIVOS

1.3.1. OBJETIVO GENERAL

Describir el proceso de aplicación del Patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios, en el caso de estudio del Sistema de Grados y Títulos de la Universidad Nacional de San Cristóbal de Huamanga, Ayacucho, 2025.

1.3.2. OBJETIVOS ESPECÍFICOS

d. Describir cómo se desarrolla la fase de Transformación del Patrón Strangler Fig en la migración de componentes de un sistema monolítico a microservicios.
e. Describir cómo se gestiona la fase de Coexistencia del Patrón Strangler Fig durante el proceso de migración.
f. Describir la ejecución de la fase de Eliminación del Patrón Strangler Fig en el proceso de migración.

1.4. HIPÓTESIS

"El hecho de que formulemos o no hipótesis depende de un factor esencial: el alcance inicial del estudio. Las investigaciones cuantitativas que formulan hipótesis son aquellas cuyo planteamiento define que su alcance será correlacional o explicativo, o las que tienen un alcance descriptivo, pero que intentan pronosticar una cifra o un hecho" (Hernández Sampieri et al., 2014).

Dado que esta investigación es de nivel descriptivo y no tiene como objetivo predecir un valor, cifra o evento, se consideró adecuado no formular ninguna hipótesis. Esta decisión es igualmente válida bajo el enfoque mixto adoptado: el componente cualitativo dominante, orientado a la interpretación del proceso de migración, tampoco requiere formulación de hipótesis por su naturaleza no predictiva.

CAPÍTULO II

MARCO TEÓRICO

2.1. ANTECEDENTES DE LA INVESTIGACIÓN

Li et al. (2020), en su investigación “Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System”, proponen un enfoque de migración basado en el patrón Strangler Fig y en Domain Driven Design (DDD) para proporcionar un proceso de migración de arquitectura completo, además se detalla el paso a paso de la aplicación de este enfoque en un sistema monolítico. Resultado de esta investigación se identifican y establecen con éxito los microservicios apropiados y se realizan experimentos cuantitativos para evaluar el rendimiento del sistema post migración.

Ma et al. (2022), en su investigación “Microservice Migration Using Strangler Fig Pattern and Domain-Driven Design”, desarrollan el enfoque propuesto en la investigación anterior, luego aplican el enfoque propuesto en dos sistemas existentes: DataCustodian de Green Button y BBDP. Además, se evalúa la eficacia del esquema propuesto y concluyen que la migración de la arquitectura mejora la confiabilidad y eficiencia del sistema.

Thamma et al. (2025), propusieron una estrategia para migrar el Disabled Person Card System desde una arquitectura monolítica hacia una arquitectura basada en microservicios mediante la aplicación del patrón Strangler Fig. La investigación surgió debido a las limitaciones del sistema monolítico en términos de escalabilidad, mantenibilidad y flexibilidad, las cuales dificultaban la evolución del software conforme aumentaba su complejidad. Para afrontar esta problemática, los autores diseñaron una arquitectura compuesta por un API Gateway, una Strangler Facade y microservicios que coexistían temporalmente con el sistema legado, permitiendo reemplazar gradualmente las funcionalidades del monolito sin interrumpir su operación. Asimismo, definieron un

proceso de migración basado en la identificación de dominios de negocio mediante Domain-Driven Design (DDD), el establecimiento de una hoja de ruta incremental y estrategias para la gestión de datos, pruebas y despliegue. Los resultados demostraron que este enfoque redujo los riesgos asociados a la transformación arquitectónica, mejoró el rendimiento de las funcionalidades evaluadas y favoreció la escalabilidad, mantenibilidad y continuidad operativa del sistema. Este antecedente constituye una referencia directa para la presente investigación, ya que valida la aplicación del patrón Strangler Fig como una estrategia efectiva para realizar una migración incremental desde una arquitectura monolítica hacia microservicios, aspecto que también se desarrolla en el presente estudio.

En el contexto latinoamericano, Arteaga et al. (2019), desarrollaron una propuesta de migración de un monolito a una arquitectura de microservicios para un sistema de compensación de pasajeros en una aerolínea salvadoreña, el cual presentaba serios problemas de escalabilidad y tiempos de respuesta de hasta 30 minutos. Utilizaron Domain-Driven Design (DDD) para descomponer la lógica de negocio en subdominios independientes y diseñaron una infraestructura distribuida con contenedores Docker, un API Gateway y patrones de resiliencia como Circuit Breaker y colas de mensajes. Este trabajo aporta a nuestra investigación al evidenciar que DDD es clave para definir los límites de cada microservicio y al demostrar cómo la comunicación asíncrona y la tolerancia a fallos resuelven los cuellos de botella del monolito, sirviendo como base estructural antes de aplicar nuestra estrategia de migración progresiva.

Por otro lado, Geovany & Averos (2025) abordaron el rediseño arquitectónico del sistema de administración de informes de una empresa de telefonía en Ecuador, motivado por las limitaciones de su arquitectura monolítica, las cuales generaban cuellos de botella, rigidez operativa y dificultades para implementar cambios continuos. Para solucionar esta problemática, el autor desarrolló una migración hacia una arquitectura de

microservicios mediante la reestructuración del sistema bajo el Principio de Responsabilidad Única e incorporando un API Gateway y un servidor de descubrimiento para la comunicación entre servicios. Como resultado, se obtuvo una mejora en la disponibilidad del sistema, la velocidad de procesamiento y la confiabilidad percibida por los usuarios, evidenciando que la adopción de microservicios constituye una estrategia efectiva para incrementar la escalabilidad y mantenibilidad de aplicaciones monolíticas. Este antecedente respalda la presente investigación al demostrar la viabilidad de la migración hacia microservicios, diferenciándose de la propuesta desarrollada en este estudio, la cual incorpora el patrón Strangler Fig como estrategia para realizar una migración gradual con menor impacto sobre la operación del sistema.

En el sector público, Cruz et al. (2024), analizaron la reestructuración del sistema catastral ServerBox del Municipio de Puebla, cuya arquitectura monolítica presentaba obsolescencia tecnológica, dificultades de mantenimiento y la necesidad de desplegar toda la aplicación ante cualquier modificación, incrementando el riesgo de fallos en producción. Para superar estas limitaciones, los autores implementaron una migración hacia una arquitectura de microservicios mediante una estrategia de descomposición vertical utilizando el framework .NET Core. Los resultados evidenciaron una mejora en la mantenibilidad del sistema, la optimización de los procesos de despliegue, la simplificación de las pruebas de software y la reducción de errores operativos. Este estudio constituye un antecedente relevante para la presente investigación, ya que demuestra los beneficios de migrar aplicaciones monolíticas hacia microservicios; sin embargo, la propuesta desarrollada en este trabajo se diferencia al emplear el patrón Strangler Fig como mecanismo para ejecutar la migración de manera incremental, minimizando los riesgos asociados a la sustitución del sistema legado.

En el contexto peruano, Noriega Yarlequé (2022), en su trabajo de suficiencia

profesional, describe el proceso de migración de un sistema monolítico a una arquitectura cloud basada en microservicios y sin servidor bajo el marco de trabajo Scrum, con el objetivo de mejorar el rendimiento y disponibilidad del sistema. Por otro lado, Mendoza Vallejos & Alva Caffo (2024) realizaron un trabajo enfocado en migrar los sistemas de cobranza del Aeropuerto Jorge Chávez a una arquitectura de microservicios usando tecnologías de la nube, logrando mejorar significativamente la escalabilidad y eficiencia de los procesos de gestión de datos.

De la revisión de los antecedentes expuestos, se evidencia que, si bien el Patrón Strangler Fig es referenciado como una estrategia reconocida para la migración incremental de sistemas monolíticos hacia arquitecturas de microservicios, el patrón sí ha sido abordado como objeto central de estudio en la literatura internacional: Li et al. (2020), Ma et al. (2022) y Thamma et al. (2025) documentan su aplicación completa, articulándola con Domain-Driven Design y evaluando los resultados posteriores a la migración. En cambio, los trabajos latinoamericanos y peruanos revisados como Arteaga et al. (2019), Geovany & Averos (2025), Cruz et al. (2024), Noriega Yarlequé (2022) y Mendoza Vallejos & Alva Caffo (2024) presentan migraciones de sistemas monolíticos hacia microservicios mediante descomposición vertical, rediseño arquitectónico o despliegue en la nube, sin adoptar el patrón Strangler Fig como estrategia de extracción incremental. Asimismo, no se identificaron investigaciones en el Perú que estudien de manera específica la aplicación del patrón Strangler Fig ni trabajos, nacionales o internacionales, desarrollados sobre sistemas de gestión académica de universidades. Por ello, el vacío que aborda esta investigación radica en la ausencia de documentación de la aplicación del patrón en el contexto de los sistemas institucionales universitarios, donde las restricciones tecnológicas, un monolito PHP nativo que no expone APIs REST consumibles, condicionan la estrategia de coexistencia adoptada.

2.2. BASE TEÓRICA

2.2.1. MIGRACIÓN DE SOFTWARE

Según Malton (2001), la migración de código fuente es un proceso de reingeniería enfocado en adaptar el software existente a un nuevo entorno, lo que implica modificaciones sustanciales en el código para hacerlo compatible y eficiente en dicho entorno. Los principales objetivos de esta migración son optimizar el uso de la nueva plataforma, preservar la funcionalidad original y asegurar que el software sea mantenible a largo plazo. Este proceso es de carácter adaptativo y generalmente se realiza cuando la plataforma actual se vuelve obsoleta o cuando se considera que una nueva plataforma ofrece mejores perspectivas para el futuro del software.

Por otro lado, Bragagnolo et al. (2021), definen la migración de software como el proceso completo de modernización que implica el traslado de un sistema de un entorno tecnológico a otro, los cuales están en relación de exclusión mutua por razones tecnológicas o estratégicas. Estos proyectos suelen ser largos y riesgosos, ya que generalmente tratan con sistemas heredados que sufren de obsolescencia y deterioro en varios de sus componentes. La migración puede incluir la reingeniería, traducción o modernización de aplicaciones para alinearse con nuevas tecnologías o arquitecturas, y se lleva a cabo en fases que incluyen planificación, evaluación, adaptación y validación, siendo un proceso incremental o iterativo.

Brito et al. (2021) indican que la migración es un proceso complejo, altamente dependiente del dominio del proyecto de software, que puede necesitar varias iteraciones antes de completarse.

Por otro lado, IBM (2024), señala la migración de software como el proceso de trasladar aplicaciones o sistemas de un entorno a otro, ya sea por motivos de actualización, modernización o cambio de arquitectura. Este proceso implica no solo la

reubicación del código y los datos, sino también la adaptación de la infraestructura y la gestión del cambio organizacional. La migración puede adoptar diferentes enfoques, como la reingeniería, la refactorización o la reescritura, y busca mejorar el rendimiento, la escalabilidad y la mantenibilidad del software en su nuevo entorno.

El líder mundial en código abierto, RedHat (2024), indica que la migración de software implica mover aplicaciones de un entorno a otro con el fin de optimizar las cargas de trabajo, y ofrece beneficios como mayor eficiencia operativa, reducción de tiempos de implementación y menores costos, pero enfrenta desafíos. Entre ellos, están los costos inesperados, capacitación y operaciones; tiempo de inactividad imprevisto; complejidad técnica debido a dependencias de aplicaciones; y la gestión del cambio, que puede ralentizar el proceso.

Asimismo, indica que existen varias estrategias para la migración de aplicaciones:

A. **Retiro o reemplazo**: Consiste en eliminar una aplicación o sustituirla por una nueva, especialmente si su valor ha disminuido o sus funciones están disponibles en otras soluciones, como SaaS.
B. **Realojamiento**: También llamado "lift and shift", implica trasladar una aplicación sin cambios significativos. Aunque puede ser rápido y económico, puede resultar costoso a largo plazo si no se adapta al entorno de la nube.
C. **Cambio de plataforma**: Incluye pequeñas modificaciones, como ajustar sistemas operativos o adaptar aplicaciones para bases de datos en la nube.
D. **Reescritura**: Consiste en desarrollar una aplicación desde cero, específicamente para el entorno de la nube.
E. **Rediseño**: Implica cambios significativos en la arquitectura de la aplicación para optimizarla para la nube.

Según Microsoft (2024), el rediseño es el proceso en el que las organizaciones ajustan y amplían la funcionalidad y el código de sus aplicaciones para mejorar su escalabilidad.

Por otro lado, IBM (2024), indica que el rediseño consiste en hacer modificaciones sustanciales a una aplicación para mejorar su escalabilidad o rendimiento en un entorno de nube. Esto puede incluir la recodificación de gran parte del código para aprovechar mejor las características nativas de la nube, como transformar una aplicación monolítica en microservicios o modernizar una base de datos.

Stuurman et al. (2016) mencionan que el rediseño de una aplicación busca optimizar aspectos no funcionales, como la extensibilidad, sin alterar su funcionalidad. Es un proceso complejo que requiere experiencia en diseño de software y el uso de patrones de diseño.

2.2.2. PATRÓN STRANGLER FIG

En 2004, Martin Fowler introdujo una estrategia para la modernización de software conocida como Strangler Fig Application. El Patrón Strangler Fig es un enfoque gradual para la modernización de sistemas de software heredados, inspirado en el crecimiento de los higos estranguladores, que crece en las selvas tropicales. La planta se desarrolla alrededor de un árbol para acceder a más luz solar, lo que provoca que el árbol huésped muera gradualmente, dejando solo la enredadera con forma de árbol. En este proceso, en lugar de reemplazar todo el sistema de una sola vez, se agregan nuevas funcionalidades sobre el sistema existente. Poco a poco, se van transfiriendo comportamientos del sistema antiguo a la nueva base de código, lo que permite que el software evolucione de manera más segura y eficiente. Este enfoque tiene como objetivo reducir el riesgo y mejorar los resultados al descomponer el sistema en partes pequeñas que pueden ser reemplazadas de forma independiente, generando un retorno de inversión

temprano. Además, facilita la coexistencia del sistema antiguo y el nuevo durante el proceso de modernización, a medida que se desarrollan las nuevas funcionalidades. Sin embargo, este proceso requiere un cambio organizacional, ya que los sistemas heredados suelen estar basados en prácticas y estructuras rígidas. El éxito de esta estrategia depende de la claridad en los resultados deseados, la identificación de partes manejables del sistema, y la capacidad de entregar nuevas piezas de software de manera continua.

Según Li et al. (2020), el patrón Strangler es una estrategia que permite una transición gradual de una arquitectura monolítica a una más moderna, facilitando un cambio tecnológico sin necesidad de realizar una revisión completa del sistema. Este enfoque implica reemplazar de manera incremental funcionalidades del sistema antiguo con nuevas implementaciones que operan de forma independiente, lo que resulta en beneficios como una mayor escalabilidad y resiliencia, ya que cada componente puede adaptarse a la demanda sin afectar al resto del sistema. Sin embargo, esta transición presenta desafíos, como la complejidad en la integración y comunicación entre los nuevos componentes, así como la necesidad de realizar pruebas exhaustivas para asegurar que las nuevas funciones mantengan la operatividad del sistema original.

Según AWS (2025), el patrón de higo estrangulador ayuda a realizar la migración de un sistema monolítico a una arquitectura de microservicios gradualmente, reduciendo el riesgo de transformación y con una disrupción empresarial menor. Este patrón es especialmente útil cuando no es posible realizar una migración masiva debido a limitaciones de recursos o la complejidad del sistema. Además, facilita la transición hacia microservicios, permitiendo que la funcionalidad se divida en componentes más pequeños y escalables que pueden ser actualizados de manera independiente. Este enfoque reduce los riesgos asociados con una reescritura completa del sistema y mejora la flexibilidad, resiliencia y eficiencia de la infraestructura. El proceso de transición se

realiza en tres fases:

- Transformar: identificar y crear componentes modernizados reescribiéndolos o portándolos en paralelo con la aplicación monolítica.
- Coexistir: Gestionar las llamadas externas incorporando un proxy HTTP en el perímetro del sistema monolítico y redirigir el tráfico a la nueva versión. Conservar el monolito en caso de requerir reversión.
- Eliminar: Retirar la funcionalidad antigua a medida que el tráfico es redirigido al nuevo servicio.

Según Microsoft Learn (2024), el Patrón Strangler Fig es un enfoque de migración incremental para sistemas heredados, en el que las funcionalidades del sistema antiguo se reemplazan gradualmente por nuevos servicios y aplicaciones. A medida que se lleva a cabo la sustitución, el sistema nuevo va absorbiendo las características del anterior hasta que, eventualmente, el sistema antiguo es completamente desmantelado.

Este proceso se realiza a través de una fachada que intercepta las solicitudes dirigidas al sistema heredado y las redirige a las nuevas aplicaciones o al sistema antiguo, según corresponda. La fachada permite que las funciones existentes se migren de manera progresiva sin afectar la experiencia del usuario, quienes continúan interactuando con el sistema sin notar el cambio.

El patrón facilita una transición controlada y segura, minimizando el riesgo asociado con la migración y distribuyendo los esfuerzos de desarrollo a lo largo del tiempo. Con el tiempo, conforme las funcionalidades son migradas, el sistema heredado queda obsoleto y puede ser retirado sin riesgos, garantizando una modernización exitosa y sin interrupciones.

A. FASES DEL PATRÓN STRANGLER FIG

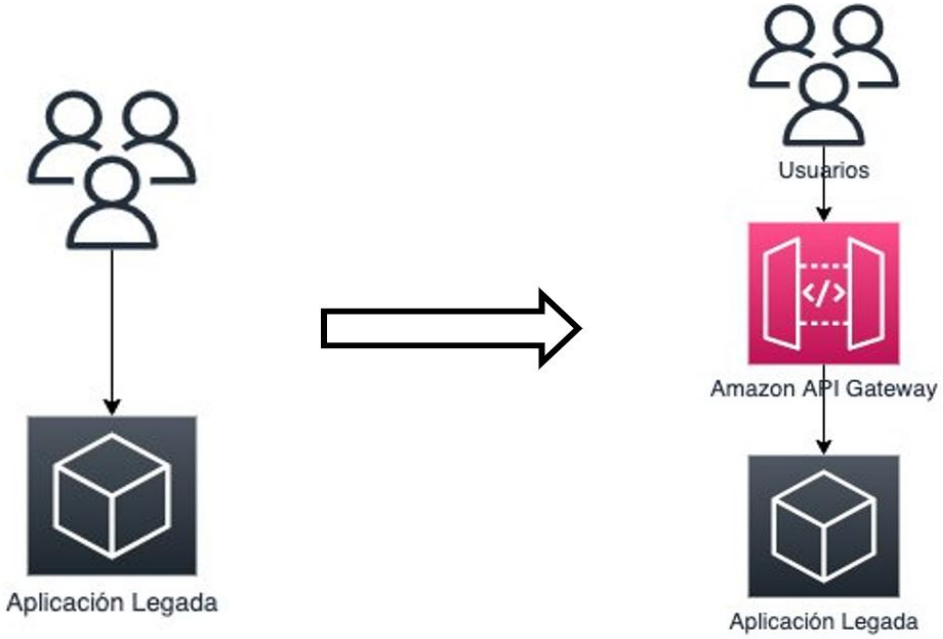
TRANSFORMACIÓN

Según AWS (2023), la transformación en el proceso de migración de software implica identificar los elementos o componentes clave de la aplicación original, definir estrategias para su descomposición y distribución, y desarrollar nuevas versiones que respeten los requerimientos y necesidades del sistema.

La Figura 3 muestra la arquitectura inicial del sistema y cómo los usuarios interactúan directamente con el monolito, así como la configuración inicial con la implementación de un proxy.

Figura 3

Fase de Transformación - Configuración Inicial



Nota: Tomado de (AWS, 2023)

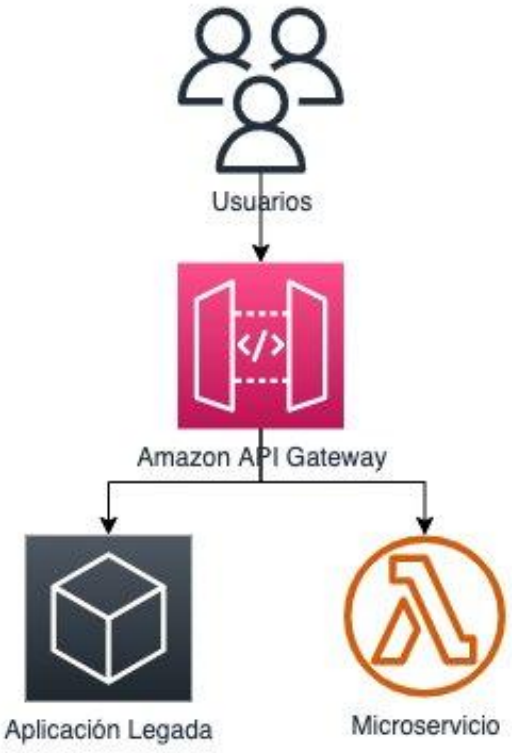
Según Kumar (2024), en un proceso de migración de software, transformar es la fase donde se identifica una funcionalidad del monolito para reemplazarla con un microservicio, desarrollándolo en paralelo y permitiendo una transición hacia una arquitectura moderna.

COEXISTENCIA

Según AWS (2023), la coexistencia permite una transición fluida para los usuarios al implementar nuevos modelos, además de ofrecer la posibilidad de revertir al sistema original en caso de fallos. Para facilitar la coexistencia se utiliza comúnmente un proxy que distribuye la carga de manera incremental entre los desarrollos, lo que permite migrar módulos de forma progresiva.

Figura 4

Fase de Coexistencia – Convivencia Monolito y Microservicio



Nota: Tomado de (AWS, 2023)

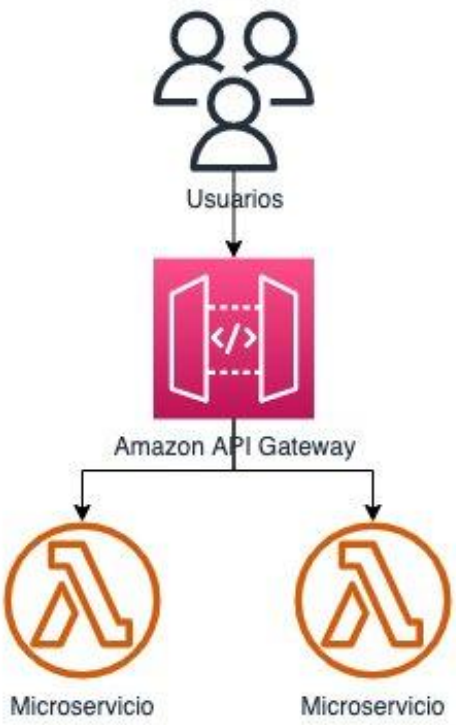
Según Kumar (2024), la coexistencia es la fase en la que los sistemas originales y los nuevos microservicios existen y funcionan en paralelo, permitiendo que interactúen sin interrupciones, generalmente se implementa un proxy o API Gateway para redirigir las solicitudes.

ELIMINACIÓN

Según AWS (2023), la eliminación consiste en, mediante el uso del proxy, el tráfico se redirige gradualmente a los nuevos desarrollos, evaluando su rendimiento, hasta que finalmente se detiene y elimina la aplicación original.

Figura 5

Fase de Eliminación – Esquema del sistema después de la eliminación

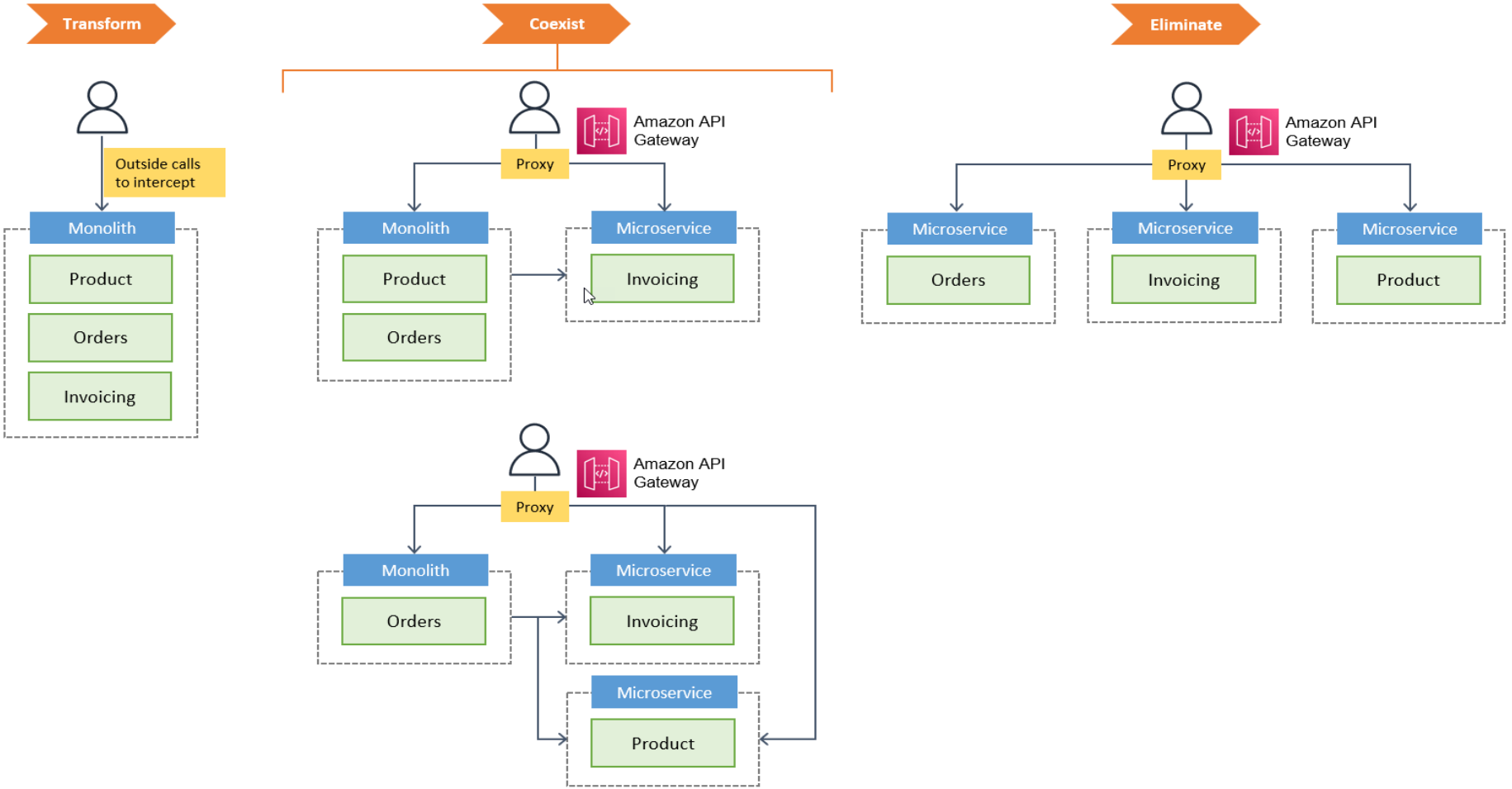


Nota: Tomado de (AWS, 2023)

Según Kumar (2024), la eliminación es la etapa final del proceso de migración, en la que, tras validar que las funcionalidades migradas operan correctamente, se redirige todo el tráfico hacia los nuevos microservicios y se retira el sistema monolítico original.

Figura 6

Fases del Proceso de Migración con el Patrón Strangler Fig según AWS - Ejemplo



Nota: Tomado de (AWS, 2023)

2.3. BASE REFERENCIAL

2.3.1. ARQUITECTURA DE SOFTWARE

Según Oracle (2019), la arquitectura de software se describe como la estructura organizativa de un sistema, que abarca sus componentes, las relaciones entre ellos y los principios que guían su diseño y evolución. La arquitectura es relevante para asegurar la escalabilidad, disponibilidad y rendimiento del sistema. Un ejemplo es la arquitectura de bases de datos, que se enfoca en cómo los componentes colaboran para gestionar eficientemente grandes volúmenes de datos y transacciones.

Según RedHat (2023), La arquitectura de software es el diseño estructural y conceptual que guía la construcción de aplicaciones, definiendo los patrones, técnicas y prácticas recomendadas para su desarrollo. Proporciona un plan para estructurar los componentes del sistema, como los servicios de frontend y backend, y promueve el uso de patrones de diseño reutilizables que facilitan soluciones eficientes y garantizan la interoperabilidad. Además, permite tomar decisiones estratégicas sobre tecnologías, lenguajes de programación y estilos arquitectónicos, como monolitos o microservicios, adaptándose a las necesidades del proyecto y las tendencias modernas, como el desarrollo en la nube.

Según Yépez et al. (2024), la arquitectura de software es un pilar esencial en la ingeniería de software, enfocada en las decisiones fundamentales que definen la estructura y el diseño de un sistema. Incluye la organización de componentes, las interacciones entre ellos y la asignación de responsabilidades, además de abordar atributos de calidad como rendimiento, escalabilidad, seguridad, mantenibilidad y facilidad de uso. Sirve como una abstracción clave para conceptualizar, diseñar y comunicar soluciones, fomentando la reutilización de componentes y marcos para mejorar la productividad y garantizar la calidad a lo largo del ciclo de vida del software.

2.3.2. ARQUITECTURA MONOLÍTICA

Según Al-Debagy & Martinek (2018), la arquitectura monolítica es un sistema con múltiples servicios y funcionalidades, pero con una base de código única. Los servicios se comunican son sistemas externos a través de diversas interfaces como páginas HTML, servicios web o APIs. Este tipo de sistemas requiere un escalado como una unidad completa, lo que puede generar dificultades para los equipos que trabajan en el mismo entorno. Es más adecuada para aplicaciones pequeñas o menos complejas debido a su estructura unificada.

Según Chandler (2024b), una arquitectura monolítica es un modelo tradicional para el desarrollo de software, donde el programa se compila como una única unidad autónoma, sin depender de otras aplicaciones. Este término evoca la imagen de algo grande y pesado, lo cual refleja la realidad de estas arquitecturas, que integran todos los aspectos del negocio en una única base de código. Realizar cambios en una aplicación monolítica implica actualizar toda la pila, lo que requiere acceder a la base de código y compilar e implementar una nueva versión del servicio, lo que puede hacer que las actualizaciones sean lentas y complicadas.

Según AWS (2024a), una arquitectura monolítica es un enfoque tradicional para el desarrollo de software en el que un único código base abarca múltiples funcionalidades empresariales. En este modelo, los componentes del sistema están estrechamente interconectados debido a la forma en que comparten datos internamente. Esto hace que las modificaciones sean complicadas y lentas, ya que incluso los ajustes menores pueden tener un impacto significativo en amplias secciones del código.

Según RedHat (2023), es un modelo en el que todas las funcionalidades de una aplicación están integradas en un único bloque o pila de código, característico de sistemas heredados. Los componentes están estrechamente conectados, lo que implica

que cualquier modificación, incluso en una pequeña parte de la aplicación, puede afectar tanto al sistema completo como a la infraestructura subyacente. Además, cualquier cambio en el código requiere volver a implementar toda la aplicación, lo que ralentiza el ciclo de actualizaciones, limitándolo a unas pocas veces al año y enfocándose principalmente en tareas de mantenimiento.

2.3.3. ARQUITECTURA DE MICROSERVICIOS

Según RedHat (2023), La arquitectura de microservicios es un enfoque de diseño de software en el que las aplicaciones se dividen en componentes pequeños, independientes y distribuidos, llamados microservicios. Cada microservicio funciona de forma autónoma, con bajo acoplamiento respecto a los demás, lo que permite escalabilidad dinámica y tolerancia a fallos, ya que los servicios pueden ampliarse o recuperarse sin afectar al resto. Este modelo acelera el desarrollo, ya que los microservicios pueden implementarse y actualizarse de manera independiente, permitiendo que varios equipos trabajen simultáneamente en diferentes componentes.

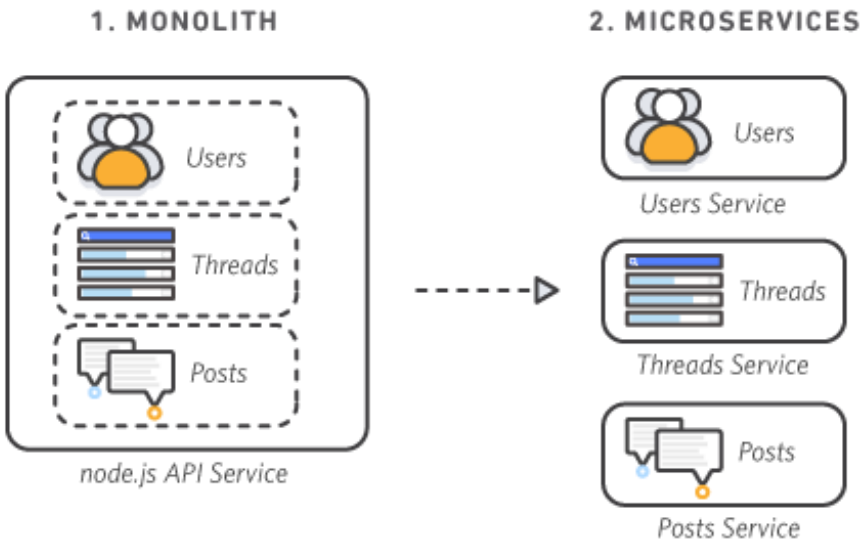
Según Chandler (2024a), la arquitectura de microservicios organiza una aplicación como un conjunto de servicios independientes, desarrollados, implementados y escalados de forma autónoma, que se comunican mediante API. Este enfoque agiliza el desarrollo de aplicaciones complejas al permitir la entrega frecuente de nuevas funciones, reduciendo la necesidad de modificar grandes partes del código existente. Los microservicios se caracterizan por su fácil mantenimiento, pruebas independientes y una organización orientada a capacidades empresariales, favoreciendo prácticas ágiles y de DevOps, además de beneficiarse de la automatización de infraestructuras como la integración y entrega continuas.

Según Oracle (2024), La arquitectura de microservicios organiza una aplicación como un conjunto de servicios pequeños, independientes y especializados, que

interactúan mediante API. Este enfoque permite escalar servicios específicos sin afectar al resto del sistema, adoptar diversas tecnologías para mayor flexibilidad, y limitar los impactos de fallos a servicios individuales. Además, mejora la velocidad de desarrollo y facilita el mantenimiento al permitir actualizaciones y despliegues independientes de cada servicio.

Figura 7

Ejemplo de Arquitectura Monolítica y Microservicio



Nota: Tomado de (AWS, 2023)

A. PATRONES DE MIGRACIÓN

Según Newman (2021b), en su libro "Monolith to Microservices", los patrones de migración son enfoques y estrategias que facilitan la transición de una arquitectura monolítica a microservicios. Estos patrones ayudan a minimizar riesgos y permiten una migración gradual, permitiendo a las organizaciones aprender y adaptarse durante el proceso. Estos patrones abordan cómo extraer funcionalidad, gestionar datos compartidos, y minimizar el riesgo durante la transición. Los patrones mencionados son los siguientes:

- Patrón UI Composition: Se refiere a la forma en que se puede componer la interfaz de usuario (UI) de una aplicación utilizando múltiples microservicios. En lugar de tener una única aplicación monolítica que maneje toda la UI, se pueden construir componentes de UI independientes que se integran dinámicamente, permitiendo un desarrollo más ágil y flexible.
- Patrón Parallel Run: Consiste en ejecutar tanto el sistema monolítico como los nuevos microservicios en paralelo durante un tiempo determinado. Esto permite validar que los nuevos microservicios funcionan correctamente antes de desactivar el monolito, reduciendo así el riesgo de fallos en producción.
- Patrón Decorating Collaborator: Permite agregar funcionalidades a un servicio existente sin modificar su código base. Se utiliza un decorador que envuelve al colaborador original, añadiendo nuevas capacidades mientras se preserva la funcionalidad existente, lo que facilita la transición hacia microservicios.
- Patrón Change Data Capture: Se utiliza para detectar y capturar cambios en los datos del sistema monolítico y replicarlos en los nuevos microservicios. Permite mantener la consistencia entre el monolito y los microservicios durante la migración, asegurando que ambos sistemas reflejen los mismos datos.
- Patrón Strangler Fig: Este patrón sugiere un enfoque gradual para reemplazar un sistema monolítico por microservicios. Se identifican partes del monolito que pueden ser reemplazadas y se desarrollan nuevas funcionalidades como microservicios, mientras que las funcionalidades antiguas se eliminan progresivamente.

Según AWS (2024b), identifica varios patrones de migración para ayudar a las organizaciones a transformar sus aplicaciones monolíticas en microservicios. Estos patrones son fundamentales para facilitar una transición efectiva y minimizar riesgos.

A continuación, se presentan los principales patrones de migración según AWS:

- Strangler Fig Pattern: Este patrón permite una migración gradual de un sistema monolítico a microservicios. Se identifican componentes monolíticos que se pueden reemplazar por microservicios, creando nuevos servicios que gradualmente sustituyen las funcionalidades antiguas. Esto minimiza el riesgo y la interrupción del negocio.
- Event-Driven Architecture (EDA): Se basa en la comunicación asíncrona entre microservicios mediante eventos. Los servicios reaccionan a eventos generados por otros servicios, lo que permite una mayor escalabilidad y flexibilidad en la arquitectura
- Patrón de Mensajería Desacoplada: Permite la comunicación entre microservicios sin que estos estén directamente acoplados entre sí, utilizando colas o sistemas de mensajería para enviar y recibir mensajes, lo que mejora la resiliencia del sistema.

CAPÍTULO III

MATERIALES Y MÉTODOS

3.1. ENFOQUE DE INVESTIGACIÓN

La presente investigación adopta un enfoque mixto de predominancia cualitativa. El componente cualitativo corresponde al análisis documental de los artefactos producidos en cada sprint y a la interpretación de las decisiones técnicas, incidencias y hallazgos del proceso de migración. El componente cuantitativo, de carácter descriptivo, corresponde al registro mediante frecuencias absolutas de indicadores como el número de endpoints REST implementados, los criterios de aceptación cumplidos y los módulos migrados por sprint, así como al registro descriptivo de los tiempos de respuesta de las operaciones que cuentan con funcionalidad equivalente en el monolito y en el sistema migrado. Ambos componentes se integran para describir de forma completa el proceso de aplicación del Patrón Strangler Fig, sin recurrir a estadística inferencial ni a la contrastación de hipótesis Hernández Sampieri et al. (2014). Siguiendo la tipología de Creswell & Plano Clark (2011), corresponde a un diseño anidado o incrustado concurrente CUAL(cuan), en tanto ambos tipos de datos se recolectan de manera simultánea a lo largo de los ocho sprints, sin fases secuenciales separadas.

3.2. TIPO DE INVESTIGACIÓN

Según Pimienta Prieto & la Orden Hoz (2017), la investigación aplicada se centra en desarrollar generalizaciones que permitan descubrir y entender el objeto o fenómeno que se está estudiando. Este tipo de conocimiento es de carácter científico y busca ampliar nuestra comprensión del mundo real. Su objetivo principal es recuperar y sintetizar el conocimiento existente, así como aplicar tanto el saber cultural como el científico, con el fin de generar tecnologías que sean útiles para la sociedad.

Según Cerda (1991), la investigación aplicada, también conocida como "investigación práctica o empírica", se distingue por su enfoque en la implementación y utilización de los conocimientos adquiridos. Este tipo de investigación no solo busca aplicar lo aprendido, sino que también facilita la adquisición de nuevos conocimientos a través de la sistematización y ejecución de prácticas basadas en la investigación. Se caracteriza por el uso riguroso, organizado y sistemático del conocimiento y los resultados obtenidos, lo que permite una comprensión más profunda de la realidad.

Por lo tanto, la presente investigación corresponde a una investigación de **tipo aplicada**.

3.3. NIVEL DE INVESTIGACIÓN

Según Hernández Sampieri et al. (2014), los estudios descriptivos tienen como objetivo identificar y detallar las propiedades, características y perfiles relevantes de personas, grupos, comunidades o cualquier otro fenómeno que se esté analizando. Una de las funciones clave de la investigación descriptiva es la capacidad de seleccionar y describir minuciosamente las características esenciales del objeto de estudio, así como sus diversas partes, categorías o clases. Este enfoque es uno de los métodos más comunes y utilizados, especialmente por aquellos que se inician en el campo de la investigación. Muchos trabajos de grado en programas de pregrado y en diversas maestrías son predominantemente descriptivos. En estos estudios, se presentan, narran y analizan hechos, situaciones y rasgos distintivos del objeto de estudio, así como el diseño de productos, modelos, prototipos y guías, entre otros.

Según Bernal (2010), La investigación descriptiva es un enfoque que tiene como objetivo especificar las propiedades y características de un fenómeno, grupo o situación particular. Se centra en la recolección y presentación de datos sobre diversas dimensiones del objeto de estudio, sin buscar establecer relaciones causales o explicaciones profundas.

Este tipo de investigación permite obtener una visión clara y detallada del tema analizado, facilitando así una comprensión más amplia de la realidad observada.

Es por esto que la presente investigación se considera en el **nivel descriptivo**.

3.4. DISEÑO DE LA INVESTIGACIÓN

Esta investigación adopta un **diseño de estudio de caso único con implementación, de corte longitudina**l.

La investigación no experimental se caracteriza por no manipular deliberadamente las variables, observando los fenómenos en su contexto natural sin alterar experimentalmente las condiciones existentes (Hernández Sampieri et al., 2014). El presente estudio comparte ese atributo esencial: la no manipulación de la variable de estudio, pero su diseño se define, ante todo, según la tradición metodológica que le es propia: el estudio de caso. De acuerdo con Yin (2014) y Stake (1995) , el estudio de caso es una estrategia de investigación empírica orientada al análisis en profundidad de un fenómeno contemporáneo dentro de su contexto real, especialmente pertinente cuando las preguntas de investigación son del tipo cómo y por qué. Las tres preguntas específicas de esta investigación son, en efecto, del tipo cómo, lo que confirma la pertinencia de la estrategia adoptada.

En el presente trabajo, la variable de estudio es la aplicación del Patrón Strangler Fig, cuya definición no fue manipulada por el investigador: el patrón se aplicó exactamente como está descrito en la literatura Fowler (2004) y AWS (2025), sin crear versiones alternativas del patrón, sin grupo de control y sin condiciones experimentales controladas. La construcción e implementación del software constituye una acción técnica inherente al objeto de estudio, no una manipulación experimental de una variable independiente, por lo que el estudio no corresponde a un diseño experimental. El estudio es longitudinal porque los datos se recolectaron en múltiples momentos a lo largo de ocho sprints del

proceso de migración.

Precisión sobre la clasificación del diseño. El presente estudio no es experimental: no existe manipulación de una variable independiente con fines de contraste causal, no se conformaron grupos de comparación ni de control, no hubo asignación aleatoria de unidades y no se persiguió establecer relaciones de causa-efecto entre variables. Tampoco corresponde estrictamente a un diseño no experimental en su acepción clásica, dado que el proceso descrito: la migración, no preexistía a la investigación, sino que fue ejecutado por el propio investigador como parte de ella. Por esta razón, el diseño se clasifica según la tradición del estudio de caso, ya expuesta, que es la estrategia metodológica que le corresponde.

3.5. UNIDAD DE ANÁLISIS Y UNIDADES DE OBSERVACIÓN

El presente estudio corresponde a un caso único: el Sistema de Grados y Títulos de la Universidad Nacional de San Cristóbal de Huamanga (UNSCH), que constituye la unidad de análisis. Según Yin (2014), en el diseño de estudio de caso la unidad de análisis es el caso mismo, esto es, aquello sobre lo que la investigación pretende formular conclusiones al término del estudio. En esta investigación, todas las conclusiones se predican del sistema en su conjunto y no de cada módulo por separado.

La unidad de observación es cada uno de los ocho (8) sprints de migración ejecutados, entendido cada sprint como el ciclo completo de aplicación de las tres fases del patrón: Transformación, Coexistencia y Eliminación, sobre un conjunto de módulos agrupados en un mismo bounded context. Se adopta esta definición porque el sprint constituye la unidad de registro efectiva del instrumento: la Ficha de Análisis Documental se aplicó una vez por sprint y por fase, documentando en cada aplicación las evidencias correspondientes. Mientras la unidad de análisis responde a la pregunta sobre qué se concluye, la unidad de observación responde a la pregunta sobre qué unidades se recogen

los datos; ambas no coinciden necesariamente y su distinción debe declararse de forma explícita.

Los dieciséis (16) módulos del sistema monolítico constituyen el alcance de la migración, es decir, el conjunto completo de componentes sobre los que se aplicó el patrón. El estudio los abarcó en su totalidad, sin excluir ninguno.

Dado que el estudio abarcó la totalidad de los módulos del sistema y de los sprints ejecutados, sin excluir ninguno, no se aplicó procedimiento de muestreo alguno, probabilístico ni no probabilístico. Se trata, por tanto, de un estudio de carácter censal: como señala Arias (2012), cuando la población resulta accesible en su totalidad no es necesario extraer una muestra, sino que se investiga a todas las unidades que la integran, sin que ello constituya estrictamente un muestreo.

Los criterios que delimitaron el alcance de la migración fueron:

- (1) que el módulo formara parte del Sistema de Grados y Títulos en producción al inicio del estudio.

- (2) que contara con un controlador propio identificable en el código fuente del monolito o una funcionalidad delimitable dentro de la arquitectura monolítica.

- (3) que tuviera tablas asociadas identificables en la base de datos del sistema. Los dieciséis módulos cumplieron estos criterios y fueron considerados en su totalidad.

3.6. VARIABLES Y DIMENSIONES

3.6.1. DEFINICIÓN CONCEPTUAL DE LAS VARIABLES

VARIABLE DE INTERÉS

Aplicación del Patrón Strangler Fig (proceso de migración): se refiere al proceso de aplicar la estrategia arquitectónica de modernización incremental descrita por (Fowler, 2004) al Sistema de Grados y Títulos de la UNSCH, observado a través de sus tres fases: Transformación, Coexistencia y Eliminación, en cada uno de los dieciséis módulos y ocho

sprints. La variable no corresponde al patrón como concepto teórico, sino a su ejecución concreta en el caso de estudio, que es lo que la investigación describe.

VARIABLES DESCRIPTIVAS (DIMENSIONES)

Transformación: Se refiere al proceso de identificar, diseñar y migrar funcionalidades del sistema monolítico a nuevos componentes independientes, típicamente en un entorno basado en microservicios (AWS, 2023).

Coexistencia: Describe la fase en la que el sistema monolítico y los nuevos componentes coexistentes operan en conjunto, gestionando el tráfico a través de un mecanismo como un proxy o un router, que redirige solicitudes según las funcionalidades migradas (AWS, 2023).

Eliminación: Consiste en el desmantelamiento progresivo de las partes obsoletas del sistema monolítico, una vez que las funcionalidades han sido completamente migradas y validadas en los nuevos componentes (AWS, 2023).

3.6.2. DEFINICIÓN OPERACIONAL DE LAS VARIABLES

VARIABLE DESCRIPTIVA

X: Aplicación del Patrón Strangler Fig (proceso de migración)

VARIABLES DESCRIPTIVAS (DIMENSIONES)

X1: Transformación

X2: Coexistencia

X3: Eliminación

3.7. TÉCNICAS E INSTRUMENTOS PARA RECOLECTAR INFORMACIÓN

A. TÉCNICAS

La técnica para la recolección de datos para esta investigación es el Análisis Documental. En el contexto de esta investigación, los "documentos" analizados son los artefactos producidos durante cada sprint del proceso de migración. Estos incluyen: el código fuente del sistema monolítico original (PHP 7.4), los archivos de configuración de cada microservicio (application.yml, docker-compose.yml), las respuestas JSON obtenidas mediante Postman durante la verificación de criterios de aceptación, los registros del servidor Eureka que evidencian el estado de cada microservicio desplegado, el Sprint Backlog de cada sprint (historias de usuario y tareas técnicas), y los documentos institucionales generados por el sistema (listas oficiales, resoluciones y diplomas).

Estos artefactos constituyen el corpus documental que registra, de forma verificable, el comportamiento del sistema en cada fase del proceso de migración.

El análisis de estos documentos se realizó de forma sistemática mediante la Ficha de Análisis Documental, diseñada específicamente para capturar las evidencias de las fases de Transformación, Coexistencia y Eliminación en cada sprint.

B. INSTRUMENTOS

El instrumento es la Ficha de Análisis Documental. La Ficha de Análisis Documental fue sometida a un proceso de validación de contenido mediante juicio de expertos. Se solicitó a profesionales del área de Ingeniería de Sistemas evaluar la pertinencia, claridad y suficiencia Escurra Mayaute (1988) y Corral (2009) de los campos de registro del instrumento respecto a las tres fases del patrón (Transformación, Coexistencia y Eliminación). Los expertos emitieron un veredicto de aplicabilidad y las observaciones formuladas fueron incorporadas en la versión final del instrumento. Las constancias de validación firmadas se presentan la sección de anexos.

El Sprint 1 (FileService) constituyó la primera aplicación práctica del instrumento; su uso sin necesidad de modificaciones en los sprints posteriores confirma la estabilidad y adecuación de los campos definidos desde el inicio.

Los cuatro expertos calificaron de manera unánime los criterios de pertinencia y suficiencia del instrumento, y coincidieron mayoritariamente en claridad. Uno de los jueces formuló una observación de forma sobre los campos de "Observaciones" (Secciones B y C), sugiriendo precisar que dichos campos registran información técnica del sprint; la observación fue incorporada en la versión final del instrumento. El coeficiente V de Aiken, propuesto por Aiken (1985), permite cuantificar la validez de contenido de un instrumento a partir de las valoraciones emitidas por un conjunto de jueces sobre cada ítem; sus valores oscilan entre 0 y 1, y los resultados más próximos a 1 expresan un mayor acuerdo entre los jueces respecto de la representatividad del ítem evaluado. El coeficiente V de Aiken obtenido fue de 0.98 en general (Pertinencia = 1.00, Claridad = 0.95, Suficiencia = 1.00), valores que evidencian un nivel adecuado de validez de contenido (Aiken, 1985). En consecuencia, el instrumento fue calificado como aplicable, incorporando dicha observación.

3.8. TÉCNICAS DE PROCESAMIENTO Y ANÁLISIS DE DATOS

El procesamiento y análisis de los datos recolectados se realizó de forma ordenada y sistemática, mediante análisis de contenido de los artefactos documentados en cada sprint. Las categorías cualitativas (decisiones de transformación, mecanismos de coexistencia, criterios de eliminación) se interpretaron a partir de la Ficha de Análisis Documental, mientras que los indicadores cuantitativos incrustados (endpoints implementados y criterios de aceptación cumplidos) se procesaron mediante frecuencias absolutas, empleadas como evidencia de apoyo y verificación de lo descrito cualitativamente, sin constituir un análisis estadístico independiente.

3.9. IDENTIFICACIÓN DE BOUNDED CONTEXTS

Para delimitar los microservicios se aplicó el concepto de Bounded Context proveniente de Domain-Driven Design (DDD) de manera pragmática: el análisis se basó en la estructura existente de las tablas de la base de datos y en las dependencias de código del monolito, agrupando los módulos por afinidad funcional y cohesión de dominio.

No se ejecutó un proceso formal completo de DDD ni talleres de *Event Storming*; por tanto, la identificación de los límites responde a las características particulares del sistema estudiado. Esta precisión es coherente con el carácter de estudio de caso único de la investigación y acota el alcance de los hallazgos.

El proceso de identificación consistió en analizar los dieciséis módulos del monolito, evaluando sus responsabilidades, dependencias funcionales y relaciones de datos, a fin de agrupar en un mismo microservicio aquellos módulos que compartían un mismo dominio de negocio.

Los bounded contexts resultantes de la aplicación de este procedimiento se presentan en el capítulo de resultados, por constituir un hallazgo de la investigación y no parte de su método.

3.10. CRITERIOS Y MÉTODO DE PRIORIZACIÓN

Para determinar el orden en que los módulos fueron migrados a microservicios, se aplicó la metodología de puntuación propuesta por (Ma et al., 2022) en la Tabla 1 de su artículo. Esta metodología asigna puntuaciones a cada criterio según su importancia relativa. A continuación, se presenta la adaptación de los criterios al contexto del Sistema de Grados y Títulos UNSCH.

Tabla 1

Criterios de priorización y pesos asignados

Criterio	Descripción	Peso máximo
FM — Función próxima a modificarse	El módulo requiere cambios o mejoras en el corto plazo	40
PP — Problemas de rendimiento	El módulo presenta cuellos de botella o alta carga	20
CF — Funcionalidad core	El módulo es central para el negocio y sus operaciones	20
RS — Relación con servicios ya migrados	El módulo depende de servicios ya convertidos en microservicio	10
SS — Servicio fácilmente separable	El módulo tiene bajo acoplamiento y puede extraerse con menor riesgo	10

Nota: Adaptado de (Ma et al., 2022).

El criterio FM (Función próxima a modificarse) tiene el mayor peso, 40 puntos, y es el factor principal en la priorización. Migrar primero los módulos que requieren cambios urgentes es más beneficioso ya que los cambios en el microservicio serán más seguros, aislados y pueden ser desplegados de forma independiente. Migrar primero módulos que no necesitan cambios urgentes no agrega valor inmediato.

La asignación de puntajes a cada criterio constituye una valoración técnica fundamentada en el análisis documental del sistema monolítico, revisión del código fuente de cada módulo, de sus consultas SQL y de sus dependencias de datos, y no una medición instrumental. En particular, el criterio PP (Problemas de rendimiento) se asignó a partir de la identificación, en el código del monolito, de consultas con múltiples operaciones JOIN sobre la tabla expediente, de búsquedas sin índice adecuado en la consulta de personas por documento y de procesos de generación documental ejecutados de forma individual por registro. Esta valoración se declara expresamente como decisión técnica del investigador, sujeta al sesgo señalado en las limitaciones del estudio, y resulta coherente con la naturaleza descriptiva de la investigación, cuyo objeto es la aplicación

del patrón Strangler Fig y no la evaluación del desempeño del sistema.

Finalmente, y como condición previa a la ejecución de los sprints, se implementó la infraestructura base que hizo posible la coexistencia y el enrutamiento controlado entre el monolito y los microservicios. Su configuración se detalla a continuación por corresponder al procedimiento metodológico de la investigación y no a un resultado de esta.

A) IMPLEMENTACIÓN DE LA INFRAESTRUCTURA BASE

Para realizar el proceso de migración se desplegaron tres componentes:

Tabla 2

Tabla de Componentes Base para la migración

Componente	Tecnología	Puerto	Función
Eureka Server	Netflix Eureka	8081	Registro y descubrimiento de microservicios
Config Server	Spring Cloud Config	8888	Configuración centralizada
API Gateway	Spring Cloud Gateway	8087	Enrutamiento de peticiones

Nota: Elaboración propia.

- Configuración de Eureka Server y Config Server

Figura 8

Configuración de Eureka Server

```yaml
server:
  port: 8081

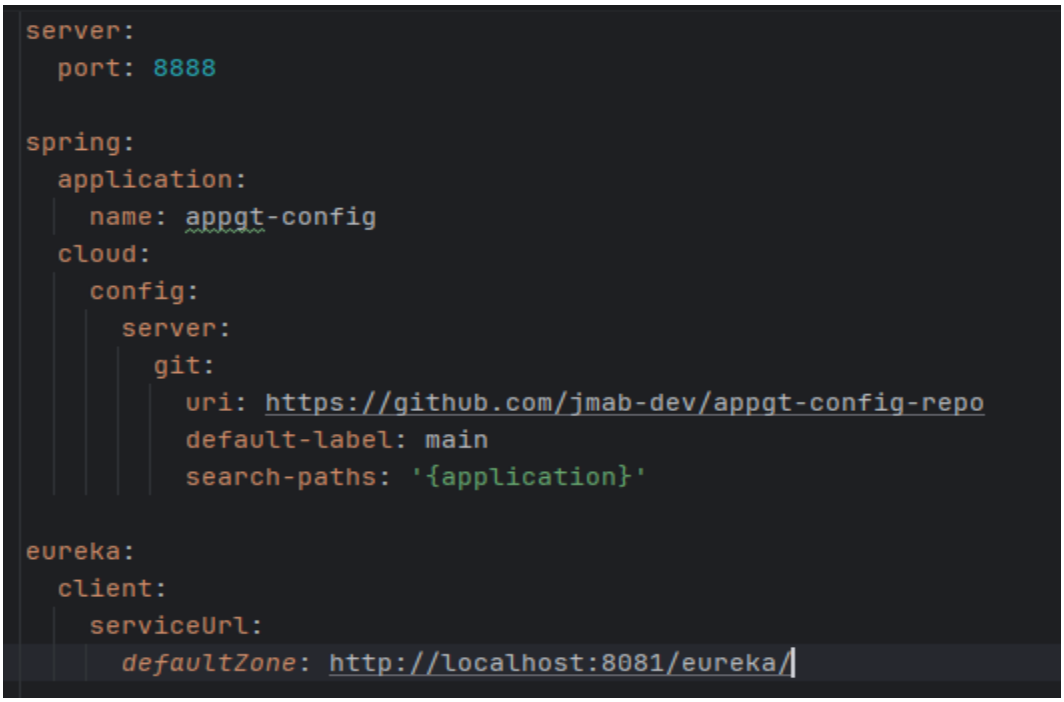
spring:
  application:
    name: ms-eureka
  config:
    import: optional:configserver:http://localhost:8888

eureka:
  server:
    enable-self-preservation: false
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
```

Nota: Se muestra el contenido del archivo application.yml del proyecto appgt-eureka.

Figura 9

Configuración de Config Server



```yaml
server:
  port: 8888

spring:
  application:
    name: appgt-config
  cloud:
    config:
      server:
        git:
          uri: https://github.com/jmab-dev/appgt-config-repo
          default-label: main
          search-paths: '{application}'

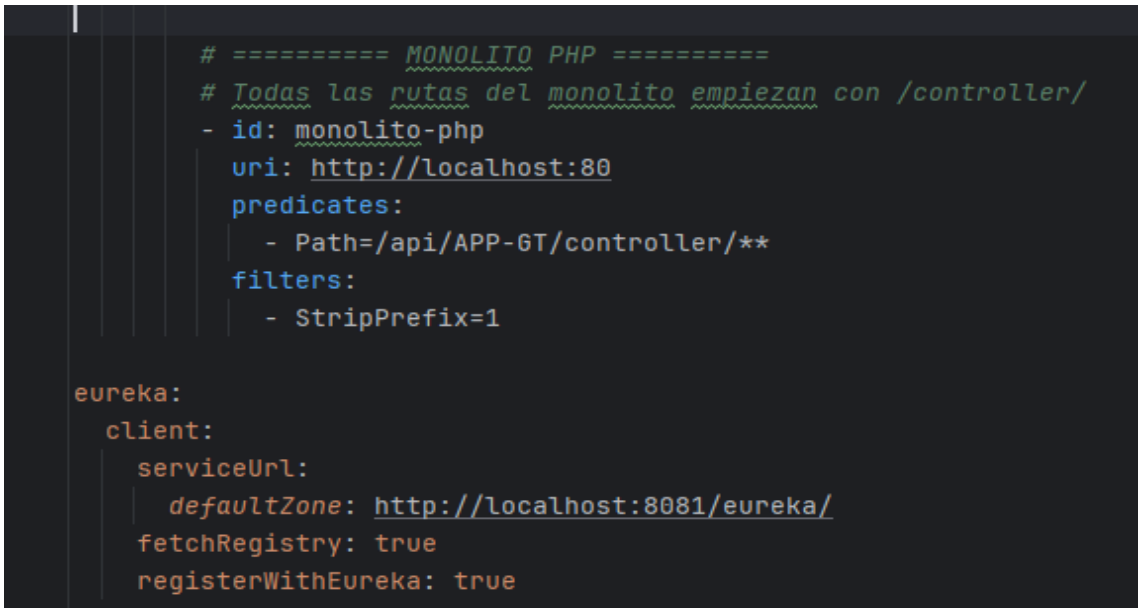
eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8081/eureka/
```

Nota: La imagen muestra el contenido del archivo application.yml del proyecto appgt-config.

- Configuración del APIGateway:

Figura 10

Configuración de API Gateway



```yaml
        # ========== MONOLITO PHP ==========
        # Todas las rutas del monolito empiezan con /controller/
        - id: monolito-php
          uri: http://localhost:80
          predicates:
            - Path=/api/APP-GT/controller/**
          filters:
            - StripPrefix=1

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8081/eureka/
    fetchRegistry: true
    registerWithEureka: true
```

Nota: Muestra el contenido del archivo application.yml del proyecto appgt-gateway.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

El detalle del método aplicado en esta investigación, criterios de priorización, identificación de bounded contexts y configuración de la infraestructura base se describe de forma ampliada en el capítulo anterior. El presente capítulo expone los resultados obtenidos al aplicar dicho método, sprint a sprint, documentando las tres fases del patrón (Transformación, Coexistencia y Eliminación) para cada microservicio.

4.1. ANÁLISIS DEL SISTEMA MONOLÍTICO

4.1.1. DESCRIPCIÓN

En concordancia con (Ma et al., 2022), quienes señalan que el primer paso de toda migración es el análisis del sistema legado para identificar sus dominios, dependencias y limitaciones, esta sección describe el estado del Sistema de Grados y Títulos de la UNSCH previo al proceso de migración.

El Sistema de Grados y Títulos de la UNSCH fue desarrollado como una aplicación web monolítica utilizando PHP 7.4 sin el uso de frameworks, ejecutándose sobre un servidor Apache y con MySQL 8.2 como motor de base de datos. Su arquitectura se basa en el patrón Modelo-Vista-Controlador (MVC), implementado de manera manual. Todo el sistema se encuentra desplegado en un único servidor, compartiendo el mismo proceso, la misma memoria y la misma base de datos.

Si bien esta configuración permitió un desarrollo inicial directo, actualmente presenta limitaciones importantes. Entre las principales se encuentran el despliegue completo obligatorio, que hacía que cualquier corrección requería el redespliegue completo de la aplicación; la dificultad de mantenimiento, asociada a la falta de estandarización y al crecimiento progresivo del código; y la escasa modularidad, resultado del fuerte acoplamiento entre componentes y de la centralización de la lógica y

los datos. A ello se suma la dependencia directa de la base de datos mediante consultas SQL complejas, lo que incrementa la dificultad para actualizar el sistema.

Asimismo, se identifican debilidades en la gestión de usuarios y seguridad, como la ausencia de roles y permisos diferenciados, y el uso de mecanismos básicos de autenticación. Estas condiciones, la integración implícita entre módulos y la centralización de la lógica de negocio, dificultan la evolución del sistema y limitan su capacidad de adaptación a nuevos requerimientos.

Es importante señalar que este sistema forma parte de un proceso crítico dentro de la universidad: la gestión, aprobación y emisión de grados y títulos. Por ello, está sujeto a cambios frecuentes derivados de actualizaciones en los reglamentos internos, disposiciones de la SUNEDU y modificaciones en los requisitos de las distintas modalidades de titulación. En este contexto, las limitaciones identificadas dificultan la incorporación oportuna de dichos cambios, generando retrasos y posibles errores en el proceso, lo que refuerza la necesidad de migrar hacia una arquitectura más flexible, escalable y mantenible.

Tabla 3

Características del Sistema Monolítico

Elemento	Descripción
Arquitectura	Monolítica
Lenguaje	PHP 7.4 (sin framework)
Servidor web	Apache
Base de datos	MySQL 8.2
Patrón	MVC
Despliegue	Servidor único
Modelo	Acceso a datos (SQL, CRUD)
Vista	HTML + PHP + Bootstrap
Controlador	Gestión de solicitudes
Autenticación	Sesiones nativas PHP
Usuarios	Creación manual
Roles	No implementados

Nivel de Acceso	Total
Comunicación interna	Consultas SQL con múltiples JOIN’s
Acoplamiento	Alto (a nivel de BD)

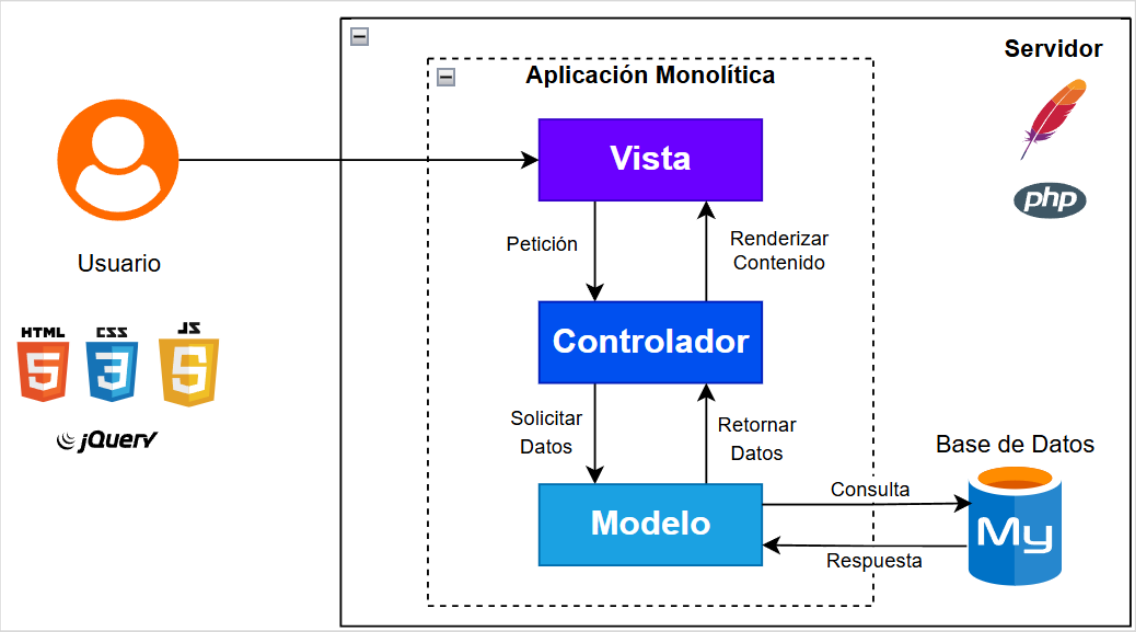
Nota: Elaboración propia.

4.1.2. ARQUITECTURA DEL MONOLITO

Se presenta la arquitectura actual del Sistema de Grados y Títulos:

Figura 11

Arquitectura Actual del Monolito



Nota: El Monolito esta desarrollado con PHP Nativo, JavaScript, HTML, CSS y Bootstrap, bajo el patrón Modelo Vista Controlador (MVC) con una fuente de datos única en MySQL.

4.1.3. INVENTARIO DE MÓDULOS DEL SISTEMA

El sistema estaba compuesto por dieciséis módulos que clasificaremos en tres categorías: Core (funcionalidad central del proceso académico), Documental (generación de documentos y reportes) y Maestro (datos de configuración institucional). La tabla siguiente presenta el inventario completo.

Tabla 4

Lista de Módulos del Monolito

N°	Módulo	Tablas BD	Descripción funcional	Operaciones	Tipo	Complejidad
01	Sesión Universitaria	sesión , ses_tipo	Gestiona la información de sesiones de consejo o asamblea universitaria. Permite solo una sesión activa y no admite dos sesiones con la misma fecha.	CRUD + reglas de negocio	Core	Media
02	Diligencia	diligencia	Gestiona la información de la diligencia con la que se procesan los expedientes de una sesión. Solo se crea una diligencia por sesión.	CRUD + reglas de negocio	Core	Baja
03	Persona	persona, documento_tipo	Gestiona la información del interesado. Al registrar, consulta la base de datos; si no hay resultado, permite registro manual.	CRUD	Core	Media
04	Bachiller	expediente, sesión y resolución	Gestiona expedientes de grado de bachiller. Valida que no exista más de un expediente del mismo nivel, generación y escuela para la misma persona.	CRUD + validación	Core	Alta
05	Título	expediente, sesión y resolución	Gestiona expedientes de título profesional con las mismas validaciones de unicidad del módulo Bachiller.	CRUD + validación	Core	Alta
06	Maestría	expediente, sesión y resolución	Gestiona expedientes de grado de maestro con las mismas validaciones de unicidad del módulo Bachiller.	CRUD + validación	Core	Alta

07	Doctorado	expediente, sesión y resolución	Gestiona expedientes de grado de doctor con las mismas validaciones de unicidad del módulo Bachiller.	CRUD + validación	Core	Alta
08	Segunda Especialidad	expediente, sesión y resolución	Gestiona expedientes de segunda especialidad con las mismas validaciones de unicidad del módulo Bachiller.	CRUD + validación	Core	Alta
09	Lista Oficial	expediente, sesión, nivel, generación_copia	Visualiza la lista de expedientes a aprobarse en la sesión activa y genera el documento correspondiente con plantilla Word institucional.	Generación de documento	Documental	Alta
10	Resoluciones	diligencia, nivel, expediente, sesion	Genera la resolución en formato Word asociada a cada expediente registrado en la sesión universitaria activa.	Generación de documento	Documental	Alta
11	Diplomas	diligencia, nivel, expediente, sesion	Genera la plantilla del diploma en formato Word asociada a un expediente.	Generación de documento	Documental	Alta
12	Reportes	expediente, nivel escuela, facultad	Visualiza listados de expedientes filtrados por facultad, por escuela y por nivel académico.	Consulta con filtros	Documental	Baja
13	Escuela	escuela, facultad	Gestiona la información de las Escuelas Profesionales y su pertenencia a una Facultad.	CRUD básico	Maestro	Baja
14	Facultad	facultad	Gestiona la información de las Facultades de la universidad.	CRUD básico	Maestro	Baja
15	Denominación	denominacion, subdenominacion	Gestiona las denominaciones de grados y títulos y sus subdenominaciones asociadas a cada escuela.	CRUD básico	Maestro	Baja
16	Usuarios	usuarios	Gestiona los usuarios del sistema con un único rol de acceso, creados de forma manual en la base de datos.	CRUD básico	Maestro	Baja

Nota: *La clasificación por tipo responde a la naturaleza funcional de cada módulo dentro del proceso de tramitación de grados y títulos.*

4.1.4. DEPENDENCIAS ENTRE MÓDULOS DEL SISTEMA

En el Sistema de Grados y Títulos, las dependencias entre módulos no se manifestaban como llamadas directas entre componentes de software, sino como referencias de clave foránea en la base de datos y como consultas SQL con múltiples JOIN en el código de cada módulo.

Tabla 5

Matriz de dependencias entre módulos del sistema monolítico

Nro	Módulo	Depende de	Descripción de la Dependencia	Impacto Estimado
1	Sesión Universitaria	Sesion_tipo	Requiere clasificar la sesión como ordinaria o extraordinaria.	Alto
2	Diligencia	Sesión Universitaria	No puede existir sin una sesión específica asociada.	Alto
3	Persona	documento_tipo	Requiere el tipo de documento de identidad del interesado.	Medio
4	Bachiller	Persona, Sesión, Diligencia, Denominación, Escuela, Acto Académico, Órgano, Generación, Nivel, Resolución	Registro de expediente que consolida datos del egresado, sesión, aprobación y escuela (múltiples JOINs).	Muy Alto
5	Título	Igual que Bachiller	Mismo comportamiento y dependencias que Bachiller, diferenciado solo por el nivel.	Muy Alto
6	Maestría	Igual que Bachiller	Mismo comportamiento y dependencias que Bachiller, diferenciado solo por el nivel.	Muy Alto
7	Doctorado	Igual que Bachiller	Mismo comportamiento y dependencias que Bachiller, diferenciado solo por el nivel.	Muy Alto
8	Segunda Especialidad	Igual que Bachiller	Mismo comportamiento y dependencias que Bachiller, diferenciado solo por el nivel.	Muy Alto
9	Lista Oficial	Expediente, Sesión, Persona, Denominación, Diligencia, Órgano	Agrupa expedientes activos mediante JOINs para presentarlos ante el órgano correspondiente.	Medio

10	Resoluciones	Expediente, Sesión, Persona, Denominación, Diligencia, Resolución	Emite el documento resolutivo de un expediente cruzando datos del expediente (JOINs múltiples).	Medio
11	Diplomas	Expediente, Persona, Denominación	Genera la plantilla del diploma de un expediente cruzando datos del expediente (JOINs múltiples).	Medio
12	Reportes	Expediente, Escuela, Facultad, Nivel	Consulta dinámicamente información de expedientes por facultad, escuela y nivel.	Alto
13	Escuela	Facultad	Depende jerárquicamente de una facultad para su existencia.	Bajo
14	Facultad	Ninguna	Unidad académica base; no posee dependencias salientes hacia otros módulos.	Bajo
15	Denominación	Escuela	Vincula formalmente cada nombre de grado o título a una escuela específica.	Bajo
16	Usuarios	Ninguna	Módulo independiente utilizado para la autenticación.	Bajo

Nota: Elaboración propia basada en el análisis de consultas SQL del código fuente.

La tabla de dependencias muestra que el núcleo crítico del sistema está en los módulos de expedientes. Estos módulos tienen un impacto estimado como “Muy Alto” ya que dependen de múltiples JOINs simultáneos hacia tablas clave. Al ser el centro de la lógica del negocio, su migración a microservicios representa el mayor desafío técnico.

En cambio, el resto del sistema muestra un acoplamiento menor: el módulo Persona actúa como un catálogo de referencia “Medio” para los expedientes; los módulos de salida (Lista Oficial, Resoluciones, Diplomas y Reportes) dependen de la consolidación de estos expedientes; y finalmente, Facultad y Usuarios no tienen dependencias salientes.

4.1.5. LIMITACIONES DEL SISTEMA

Durante el análisis del Sistema de Grados y Títulos de la UNSCH se identificaron diversas limitaciones que afectaban su rendimiento, mantenimiento y evolución tecnológica. Estas problemáticas justifican la migración hacia una arquitectura de microservicios mediante la aplicación del patrón Strangler Fig Pattern.

De acuerdo con (AWS, 2025) y (Li et al., 2020), los sistemas legados se caracterizan como un *big ball of mud*: sistemas altamente acoplados, difíciles de mantener, escalar y modificar. En el sistema se identificaron las siguientes limitaciones:

Tabla 6

Tabla de Limitaciones del Sistema

Limitación	Descripción	Impacto en el sistema	Impacto en la organización
Evolución funcional	Dificultad para agregar funcionalidades, validaciones y cambios sin afectar otros módulos del sistema.	Alto: incremento de errores, mayor tiempo de desarrollo y dificultad para adaptar el sistema a nuevos requerimientos.	Alto: retrasos en la atención de cambios normativos y académicos y menor capacidad de respuesta institucional.
Mantenibilidad del código	Código altamente acoplado y desarrollos sin separación clara de capas.	Alto: dificultad para corregir errores o implementar cambios.	Medio-Alto: aumento del tiempo de adecuación al reglamento.
Ciclo de despliegue	Cada actualización requería desplegar toda la aplicación.	Medio-Alto: interrupciones y riesgo de fallas en producción.	Medio: afectación temporal a los usuarios y operaciones.
Control de acceso y seguridad	Ausencia de roles y permisos diferenciados entre usuarios.	Alto: riesgo de modificaciones no autorizadas.	Alto: posible pérdida de integridad y control de información.
Observabilidad	Ausencia de logging y monitoreo del sistema.	Medio: dificultad para detectar errores y cuellos de botella.	Medio: tiempos mayores de diagnóstico y soporte.
Impacto global de fallos	Un fallo en cualquier módulo podía hacer que la aplicación completa quedara inoperativa.	Alto: Un error crítico en un módulo secundario puede provocar la caída total de la aplicación	Alto: Interrupción total de la operación administrativa, afectando la percepción de fiabilidad tecnológica ante el usuario final.

Duplicidad Lógica	Repetición de operaciones en múltiples controladores sin abstracción compartida.	Alto: Inconsistencia de datos; actualizar una lógica en un punto y olvidar otro genera respuestas erróneas del sistema.	Medio: Riesgo administrativo por falta de integridad en la información de personas y expedientes.
Crecimiento Desordenado	El monolito creció sin una arquitectura formal. Se identificaron rutas no documentadas resultado de mejoras no planificadas.	Medio: Complejidad para revisar y mantener los flujos del sistema.	Dificultada para auditar el sistema, se vuelve una "caja negra" que no se entiende en su totalidad.

Nota: Elaboración Propia.

4.2. ARQUITECTURA PROPUESTA

Con base en el análisis del sistema monolítico presentado en la sección anterior, se diseñó una arquitectura de microservicios. Esta arquitectura busca superar las limitaciones identificadas y es compatible con la aplicación del patrón Strangler Fig. Es decir, puede coexistir con el monolito durante el proceso de transición sin interrumpir las operaciones del sistema. Esto reduce el riesgo de transformación y provoca menos interrupciones en el negocio. La arquitectura propuesta fue diseñada precisamente para facilitar este proceso gradual.

A partir del análisis de dominio cuyo procedimiento se describe en el capítulo anterior, se definieron los siguientes bounded contexts:

Tabla 7

Identificación de Bounded Contexts y su correspondencia con módulos del monolito

Bounded Context	**Módulos del monolito incluidos**	**Responsabilidad Funcional**
Estructura Académica	Facultad, Escuela, Denominación	Datos maestros de la organización académica institucional
Seguridad y Acceso	Usuarios	Autenticación, autorización y gestión de roles
Gestión de Personas	Persona	Información del interesado o solicitante del trámite

Gestión de Sesiones	Sesión Universitaria	Ciclo de vida de las sesiones de consejo universitario y asamblea universitaria
Gestión de Diligencias	Diligencia	Tramitación de expedientes dentro de una sesión
Gestión de Expedientes	Bachiller, Título, Maestría, Doctorado, 2da Especialidad	Proceso académico central por nivel
Gestión Documental	Lista CU, Resoluciones, Diplomas	Generación de documentos oficiales del proceso
Parámetros del Sistema	Acto Académico, Nivel, Tipo de Sesión, Órgano, Generación	Datos de configuración transversales consultados por múltiples contextos
Reportes y Estadísticas	Reportes	Consultas de solo lectura para análisis y reportes

Nota: La tabla muestra los bounded contexts identificados a partir del análisis funcional del monolito y su correspondencia con los módulos agrupados dentro de cada dominio.

4.2.1. MAPEO DE MÓDULOS A MICROSERVICIOS

Cada bounded context identificado se materializó en un microservicio independiente. La siguiente tabla presenta el mapeo final con la justificación de cada agrupación.

Tabla 8

Mapeo de bounded contexts a microservicios

Bounded Context	Microservicio	Justificación de Agrupación
Estructura Académica	AcademicService	Agrupa datos académicos compartidos y consultados por otros módulos.
Seguridad y Acceso	UserService	Centraliza autenticación, roles y control de acceso.
Gestión de Personas	PersonService	Dominio propio con integración al API Personas de la UNSCH. Es referenciado por todos los expedientes
Gestión de Sesiones	SesionService	Dominio con ciclo de vida propio y reglas de negocio específicas.
Gestión de Diligencias	DiligenceService	Depende de sesiones y posee lógica específica.
Gestión de Expedientes	FileService	Comparten la misma tabla base (expediente), la misma estructura de datos y las mismas reglas

		de validación de unicidad. Separarlos generaría duplicación innecesaria de lógica de negocio.
Gestión Documental	DocumentService	Comparten el mismo dominio: transformar datos de expedientes y sesiones en documentos con formato institucional. Su agrupación centraliza la lógica de generación documental.
Parámetros del Sistema	ParametersService	Centraliza configuraciones usadas por varios servicios. Su centralización evita la duplicación de datos y garantiza la consistencia en todo el sistema.
Reportes y Estadísticas	FileService	Las funcionalidades de consulta de este módulo se incorporaron a FileService mediante tres endpoints de reporte, por ser dicho servicio el propietario del dominio de expedientes.

Nota: La columna justificación explica el criterio de dominio que sustenta la agrupación de cada conjunto de módulos en un único microservicio.

4.2.2. PRINCIPIOS DE DISEÑO ADOPTADOS

El diseño de la arquitectura se fundamentó en los principios que (Ma et al., 2022) identifican como necesarios para una migración exitosa con el patrón Strangler Fig:

- **Responsabilidad única**: Cada microservicio gestiona un único bounded context sin conocimiento de los demás, reduciendo el acoplamiento implícito del monolito.
- **Punto único de entrada**: El API Gateway actúa como fachada central que, durante la coexistencia, enruta peticiones tanto al monolito como a los microservicios, constituyendo el mecanismo técnico del patrón Strangler Fig según (AWS, 2024b).
- **Autonomía de despliegue**: Cada microservicio puede desplegarse, actualizarse y escalarse de forma independiente, resolviendo el problema del ciclo de despliegue.
- **Descubrimiento dinámico**: Los microservicios se registran automáticamente en Eureka al iniciar, eliminando configuración manual de rutas.

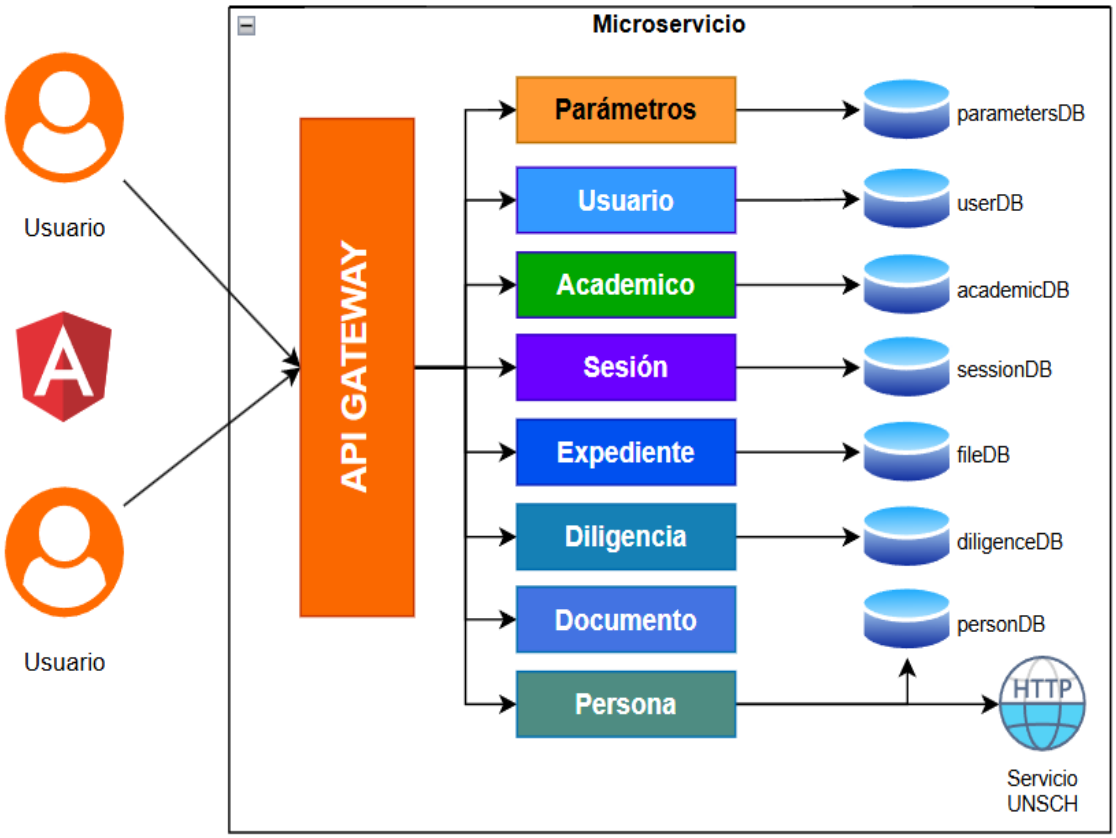
- **Desacoplamiento**: Cada microservicio tiene su propio contexto de Spring Boot, su propio archivo de configuración y su propio ciclo de vida de despliegue independiente. Los cambios en un servicio no afectan la compilación ni el despliegue de los demás servicios.
- **Separación por dominio**: La estructura de código de cada microservicio sigue el patrón uniforme Controller → Service → Repository → Entity, con Data Transfer Objects (DTOs) para la comunicación externa.
- **Migración incremental**: La migración no se realizó mediante una reescritura total simultánea (enfoque *big bang*), sino de forma incremental, extrayendo un dominio a la vez del monolito y construyendo el microservicio equivalente. Este principio es la esencia del patrón Strangler Fig.
- **Base de datos compartida**: Mantener temporalmente la base de datos compartida para garantizar la consistencia de datos durante la coexistencia.
- **Base de datos por servicio:** Cada servicio es dueño exclusivo de su propia base de datos. Ningún servicio accede directamente a la base de datos de otro.

4.2.3. DIAGRAMA DE ARQUITECTURA PROPUESTA

La Figura siguiente presenta el diagrama de la arquitectura de microservicios propuesta, que representa el estado objetivo al que se busca llegar al finalizar el proceso de migración. En este diagrama se identifican los distintos componentes del sistema y sus interacciones.

Figura 12

Arquitectura Propuesta



Nota: La figura muestra el modelo de arquitectura propuesta resultado de la migración.

Todas las solicitudes son enviadas al API Gateway, encargado de recibir las peticiones y redirigirlas al microservicio correspondiente. Cada microservicio se despliega de forma independiente.

Durante la migración, el API Gateway mantuvo además rutas hacia el monolito PHP para las funcionalidades que aún no habían sido migradas, permitiendo que ambos sistemas funcionaran en paralelo sin interrumpir las operaciones.

Respecto a los datos, un principio fundamental de la arquitectura de microservicios es que cada servicio es dueño exclusivo de sus propios datos. Esta propiedad garantiza que ningún servicio acceda directamente a la base de datos de otro. Todas las interacciones de datos entre servicios se realizan únicamente a través de contratos de API.

4.2.4. STACK TECNOLÓGICO

La siguiente tabla presenta el detalle de cada tecnología por capa de la arquitectura, indicando su rol específico dentro del sistema.

Tabla 9

Stack tecnológico

Capa	Tecnología	Rol en la arquitectura
Lenguaje de Programación	Java	Runtime de todos los microservicios
Framework	Spring Boot	Base de cada microservicio.
API Gateway	Spring Cloud Gateway	Punto único de entrada; enrutamiento hacia monolito o microservicios; validación y API Key
Service Registry	Netflix Eureka	Registro y descubrimiento dinámico de microservicios
Configuración Centralizada	Spring Cloud Config	Gestión de propiedades por entorno
ORM	Spring Data JPA + Hibernate	Mapeo Objeto – Relacional. Acceso a datos ORM
Base de datos	MySQL	Persistencia relacional
Seguridad	Spring Security + JWT	Autenticación de usuarios y BCrypt
Generación Documental	Apache POI	Generación de archivos .docx
Contenedores	Docker + Docker Compose	Despliegue local reproducible
Pruebas	Postman	Pruebas funcionales de endpoints REST

Nota: Elaboración propia.

4.3. PLAN DE MIGRACIÓN

4.3.1. ESTRATEGIA DE MIGRACIÓN

La estrategia seleccionada es el patrón Strangler Fig, basado en una migración incremental. Esta estrategia permite:

- Coexistencia temporal: El monolito PHP y los microservicios operan simultáneamente durante la migración, garantizando la continuidad de los procesos académicos y administrativos sin interrupciones para los usuarios.
- Extracción progresiva: Los módulos del monolito se migran de forma secuencial y priorizada, convirtiéndose en microservicios independientes.

- Proxy centralizado El API Gateway administra y redirige las solicitudes entre el sistema monolítico y los nuevos microservicios, facilitando una transición gradual y controlada de los servicios académicos.
- Reducción de riesgos: La migración incremental permite aislar problemas por módulo, minimizando el impacto sobre los procesos institucionales.
- Transición controlada: El tráfico hacia el monolito se reduce progresivamente hasta su retiro completo, permitiendo un seguimiento ordenado de la migración.

4.3.2. PRIORIZACIÓN Y ORDEN DE MIGRACIÓN

En base al método de scoring se definió el orden de migración, el servicio candidato con la puntuación más alta representa los mayores beneficios esperados y debe extraerse primero.

Tabla 10

Puntuación de priorización para cada microservicio

Microservicio	FM	PP	CF	RS	SS	Total	Orden
FileService	40	20	20	10	0	90	Sprint 1
DocumentService	30	20	20	10	0	80	Sprint 2
PersonService	10	20	20	0	10	60	Sprint 3
SesionService	0	0	20	0	10	30	Sprint 4
AcademicService	0	0	10	0	10	20	Sprint 5
UserService	5	0	0	0	10	15	Sprint 6
DiligenceService	0	0	0	0	10	10	Sprint 7
ParametersService	0	0	0	0	10	10	Sprint 7

Nota: Elaboración Propia.

FileService (90 puntos) obtuvo la puntuación máxima en FM (40), PP (20) y CF (20). El módulo de expedientes era el que más cambios urgentes requería: la lógica PHP del monolito tenía validaciones incompletas, presentaba problemas de rendimiento en las consultas y es el módulo más central del negocio. El puntaje RS = 10 y SS=0 refleja que este módulo tiene múltiples dependencias de datos y requiere datos de otros módulos.

En segundo lugar, se ubicó DocumentService (80), FM=30 porque era urgente su mejora: la implementación PHP genera documentos de forma individual y añadir campos

nuevos causaba placeholders sin reemplazar y errores en el documento final, constituyendo un defecto crítico. PP=20 y CF=20 confirman que es un módulo crítico. RS = 10 y SS=0 porque el servicio necesita datos de múltiples módulos.

PersonService (60 puntos): FM=10 (cambios menores pero necesarios), PP=20 (búsquedas por DNI con consultas lentas en el monolito) y CF=20 (los egresados son fundamentales para el proceso). SS=10 refleja que es relativamente fácil de separar por ser una tabla independiente.

SesionService (30 puntos): FM=0 indica que el módulo no requería cambios urgentes. Su prioridad viene de CF=20 (las sesiones son el punto de inicio de todo el flujo operativo).

AcademicService (20 puntos): FM=0 y PP=0 indican que el módulo de catálogo académico no tenía problemas urgentes. Su priorización se basa en SS=10 (es el módulo más fácil de separar: principalmente lectura de tablas de catálogo).

UserService (15 puntos): FM=5 indica cambios menores relacionados con autenticación y seguridad, manteniendo una funcionalidad estable durante la migración.

DiligenceService y ParametersService (10 puntos): Ambos módulos sin urgencia de cambio, sin problemas de rendimiento y sin ser funcionalidad core crítica. Se migran juntos por afinidad de puntuación.

PROBLEMA EN EL ORDEN DE MIGRACIÓN Y COEXISTENCIA

FileService y DocumentService fueron priorizados en la estrategia de migración debido a que ambos módulos presentaban defectos activos que requerían atención inmediata. No obstante, estos servicios dependen de información relacionada con personas, sesiones, denominaciones y otras entidades que aún residían en el monolito. Desde una perspectiva teórica, los microservicios con dependencias funcionales hacia otros dominios no deberían migrarse antes que los servicios de los cuales dependen.

Asimismo, el patrón Strangler Fig establece que, durante la fase de coexistencia, el monolito y los microservicios pueden intercambiar información mediante comunicación HTTP. Sin embargo, esta estrategia no resultaba viable en el sistema de Grados y Títulos, debido a que los controladores PHP del monolito no exponen APIs REST con respuestas JSON estructuradas y predecibles. Implementar integración HTTP habría requerido desarrollar parsers específicos para cada módulo PHP, introduciendo una complejidad temporal que no aportaba valor a la arquitectura objetivo y que sería descartada una vez finalizada la migración.

Ante este escenario, se definió una estrategia de migración en dos fases. En la primera fase se mantuvo una base de datos compartida, permitiendo que los microservicios accedan a las mismas tablas utilizadas por el monolito. Esto simplificó el proceso de migración y garantizó la consistencia de la información durante el periodo de coexistencia. Posteriormente, en la segunda fase, se realizó el desacoplamiento de la base de datos mediante la separación por microservicio y la habilitación de comunicación HTTP entre servicios para el intercambio de información.

Este enfoque resolvió el problema de priorización de manera efectiva. Gracias al uso de una base de datos compartida, durante el Sprint 1 FileService pudo acceder directamente a las tablas persona, sesion y denominacion, replicando el mismo mecanismo utilizado previamente por el monolito PHP y evitando cualquier dependencia de comunicación HTTP. De esta manera, el módulo de expedientes pudo ser extraído con máxima prioridad sin que sus dependencias de datos representaran un impedimento.

El mismo criterio se aplicó en el Sprint 2 con DocumentService. Las consultas SQL que involucraban múltiples operaciones JOIN entre las tablas expediente, persona, sesion, escuela, facultad y denominacion continuaron funcionando sin modificaciones, ya

que todas las entidades permanecían temporalmente dentro de la misma base de datos compartida.

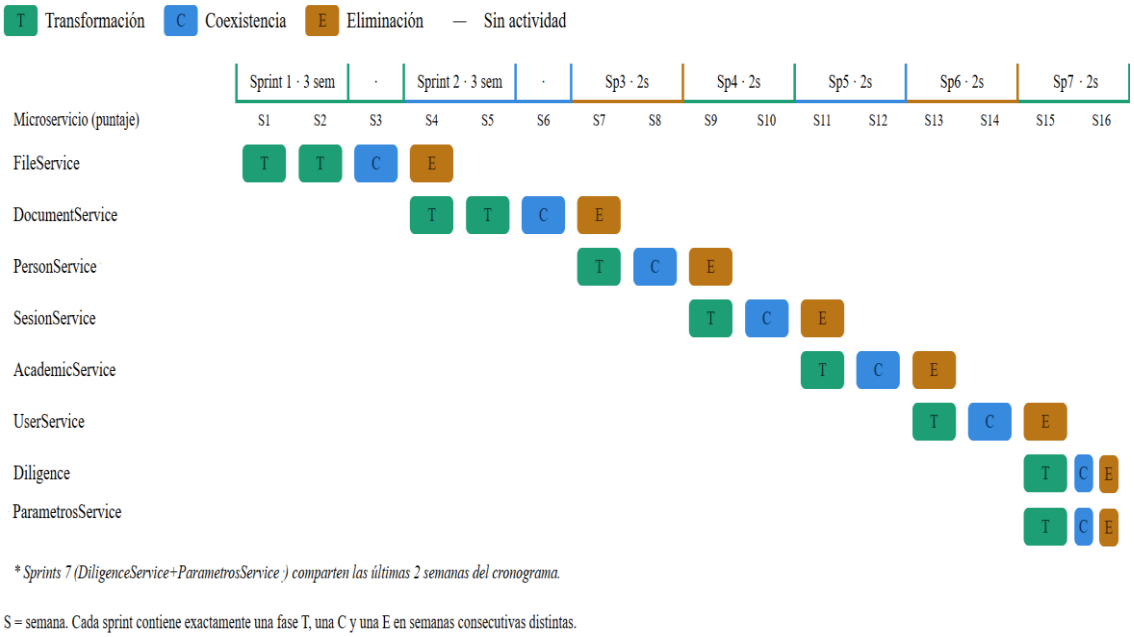
Finalmente, una vez completada la extracción de los módulos del monolito hacia microservicios independientes, se ejecutó la separación definitiva de las bases de datos. En esta etapa el monolito PHP ya no se encontraba operativo, por lo que el desacoplamiento no introdujo riesgos para la continuidad operativa del sistema.

4.3.3. CRONOGRAMA DE SPRINTS

Con base en la priorización, se definió un plan de migración compuesto por 8 sprints, siguiendo el marco de trabajo ágil Scrum. Scrum no constituye una metodología, sino un marco de trabajo ligero para la gestión del trabajo complejo: define un conjunto mínimo de roles, eventos y artefactos, y deja deliberadamente sin prescribir las técnicas de ingeniería empleadas dentro de cada iteración (Schwaber & Sutherland, 2020).

Figura 13

Cronograma de migración



Nota: Cronograma de migración por sprints – cada sprint tiene las 3 fases del patrón.

4.4. APLICACIÓN DEL PATRÓN STRANGLER FIG

Esta sección describe la aplicación de las tres fases del patrón Strangler Fig, Transformación, Coexistencia y Eliminación, en cada uno de los ocho sprints ejecutados.

4.4.1. PRE MIGRACIÓN

Antes de iniciar los sprints y las fases del patrón Strangler Fig se dispuso de la infraestructura base que permitió la coexistencia y el enrutamiento controlado. Consta de tres componentes fundamentales: el Service Registry (Eureka Server), el API Gateway (Spring Cloud Gateway) y el Config Server para la gestión centralizada de configuraciones. Asimismo, en este paso previo se encapsuló el monolito en el API Gateway para gestionar las peticiones a medida que se ejecutaba cada sprint.

B) CONFIGURACIÓN ORIGINAL DEL MONOLITO:

La tabla e imagen siguiente muestra la configuración inicial del monolito.

Tabla 11

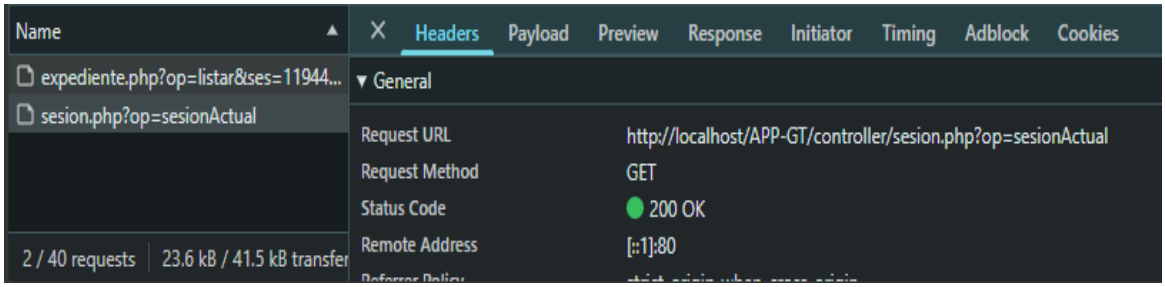
Configuración original del monolito

Componente	Especificación
URL de acceso	http://localhost/APP-GT
Servidor web	Apache HTTP Server, puerto 80
Autenticación	Sesiones nativas de PHP
Gestión de Rutas	Enrutamiento basado en archivos PHP

Nota: Elaboración Propia

Figura 14

Request URL Original del Monolito



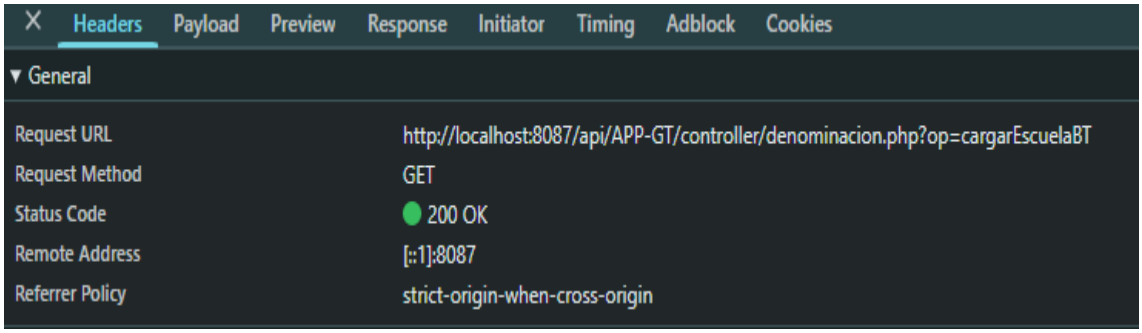
Nota: La imagen muestra un ejemplo de los Request originales del Monolito apuntando a localhost:80 puerto de Apache.

C) MONOLITO CON APIGATEWAY

El usuario continúa utilizando la misma URL para acceder al sistema. Internamente, las peticiones realizadas por el frontend son actualizadas para dirigirse al API Gateway, el cual reenvía las solicitudes hacia el monolito PHP. Este proceso es no genera interrupciones ni errores, ya que el Gateway realiza un enrutamiento invisible de las peticiones, por lo que el sistema funciona igual desde la perspectiva del usuario.

Figura 15

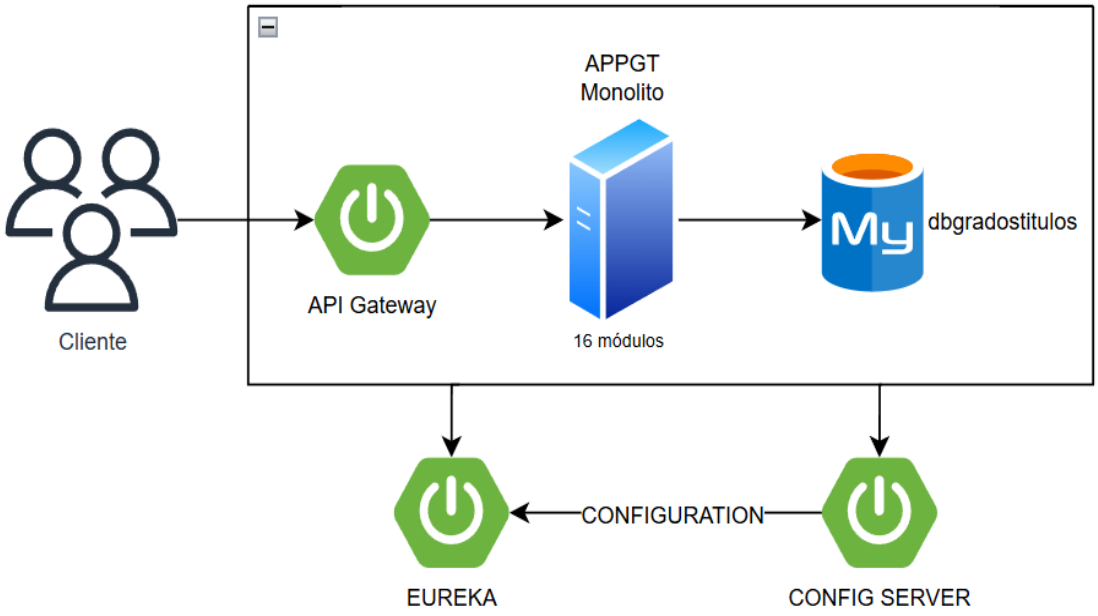
Request URL del monolito con el API Gateway



Nota: La imagen muestra un ejemplo de request configurado pre migración, ahora apunta a localhost:8087 puerto del API Gateway.

Figura 16

Arquitectura del sistema pre-migración



Nota: La imagen la arquitectura del sistema con la configuración base.

4.4.2. SPRINT 1 – FILE SERVICE

Módulos migrados: Bachiller (04), Título (05), Maestría (06), Doctorado (07), Segunda Especialidad (08), Reportes (12).

Puntuación de priorización: 90/100 **Puerto:** 8082

FileService fue seleccionado como el primer microservicio a migrarse por tratarse de un módulo crítico dentro del proceso académico y a la necesidad de modernizar la gestión de expedientes. Su migración permitió establecer la base inicial de la arquitectura de microservicios, validando la integración con el API Gateway, la comunicación con la base de datos compartida y el despliegue independiente del servicio.

a) TRANSFORMACIÓN

La decisión de agrupar cinco módulos en un único microservicio se basó en el análisis del bounded context realizado previamente. Bachiller, Título, Maestría, Doctorado y Segunda Especialidad comparten la misma tabla expediente, la misma estructura de datos base y la misma regla de validación de unicidad, diferenciándose únicamente por el valor del campo nivel y por requerir algunos campos adicionales específicos por cada nivel académico. Crear cinco microservicios separados habría generado duplicación de código y lógica sin ningún beneficio de dominio.

FileService se implementó con la arquitectura en capas de Spring Boot. La capa de entidades mapea la tabla expediente con todos sus campos y relaciones de clave foránea. La capa de repositorios utiliza Spring Data JPA para el acceso a datos. La capa de servicios contiene la lógica de negocio, incluyendo la validación de unicidad. La capa de controladores expone los endpoints REST.

Tabla 12

Endpoints implementados — FileService

Método	Endpoint	Descripción	Código de respuesta
GET	/api/file	Lista todos los expedientes	200
GET	/api/file/id/{id}	Obtiene un expediente por ID	200 / 404
GET	/api/file/bySession	Filtra por sesión, nivel y generación	200 / 400
POST	/api/file/create	Registra expediente	200 / 400
PATCH	/api/file/update	Actualiza un expediente	200 / 404
DELETE	/api/file/{id}	Elimina un expediente	200 / 404
GET	/api/file/health	Verifica la salud del microservicio	200 / 500
GET	/api/file/expediente/count/{sessionId}	Conteo de expedientes de una sesión	200 / 404
GET	/api/file/report/by-sesion/{sesId}	Datos de reporte por sesión	200 / 404
GET	/api/file/report/by-escuela/{escCode}	Datos de reporte por escuela	200 / 404
GET	/api/file/report/by-nivel/{nivelId}	Datos de reporte por nivel académico	200 / 404

Nota: Los endpoints de reporte se implementan directamente en FileService, ya que FileService es el propietario del dominio de expedientes. Esta decisión elimina la necesidad de un microservicio adicional exclusivo de reportes.

Pruebas y criterios de aceptación

Se utilizó el software Postman para realizar pruebas y validar las respuestas de los endpoints y la estructura de los JSON resultantes, también se usó el dashboard de Eureka Server para validar el registro y estado de los microservicios.

Tabla 13

Criterios de aceptación — FileService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka como FILESERVICE	Cumple

2	El endpoint POST valida que no exista expediente duplicado por persona/nivel/generación/escuela	Retorna 409 Conflict con mensaje descriptivo ante duplicado	Cumple
3	El endpoint GET /bySession retorna solo expedientes de la sesión, nivel y generación solicitado	Filtra correctamente por valor del campo sesión, nivel, generación.	Cumple
4	Los endpoints REST retornan JSON con la estructura esperada	Validado con colección Postman	Cumple

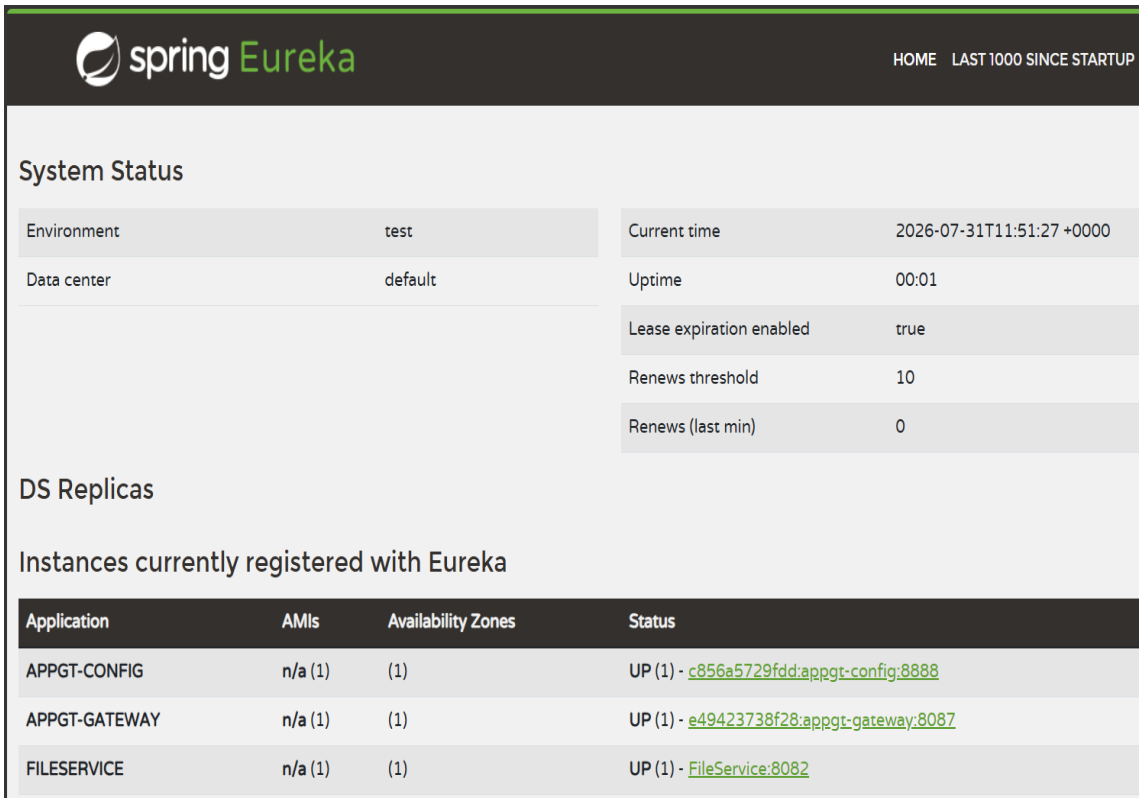
Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 17

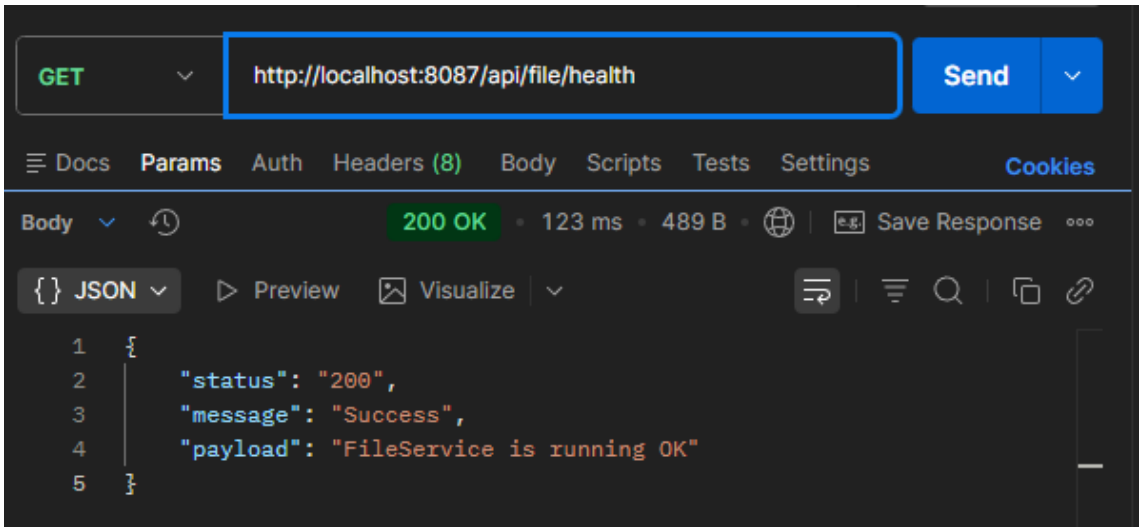
Registro de Eureka Server



Nota: Se visualiza los servicios desplegados y registrados en eureka server: Appgt-config, appgt-gateway y fileservice.

Figura 18

Endpoint Health de File Service

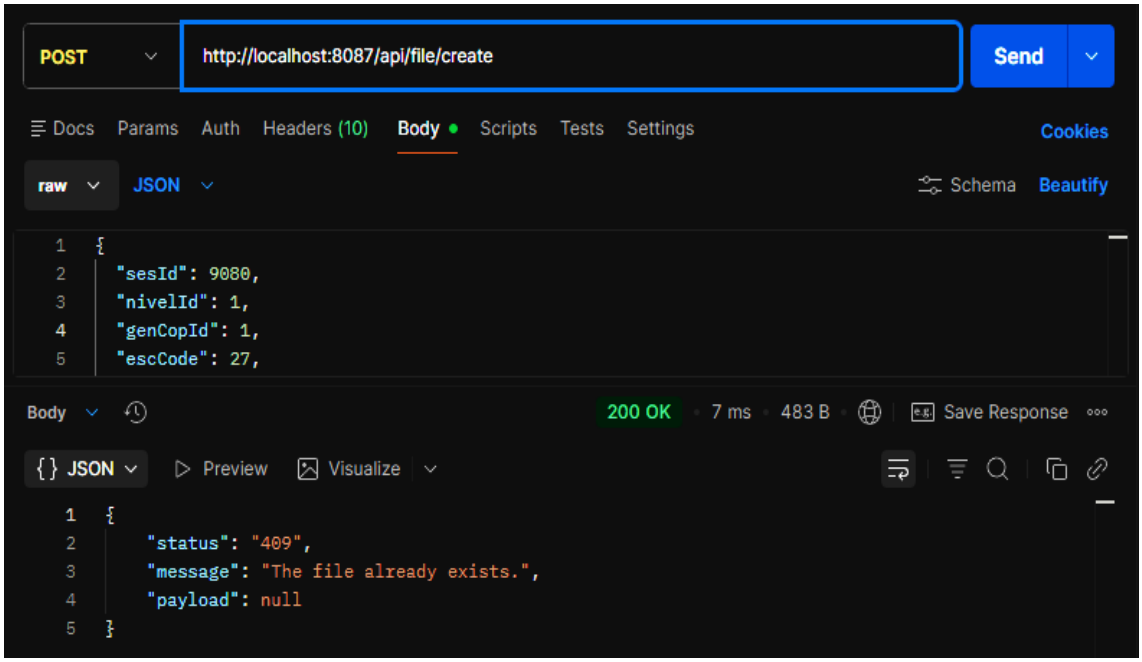


Nota: Se ejecutó el servicio health de FileService para validar que se está ejecutando correctamente en el puerto 8087 del API Gateway.

Criterio de Aceptación Nº 2

Figura 19

Endpoint de Registro de File Service

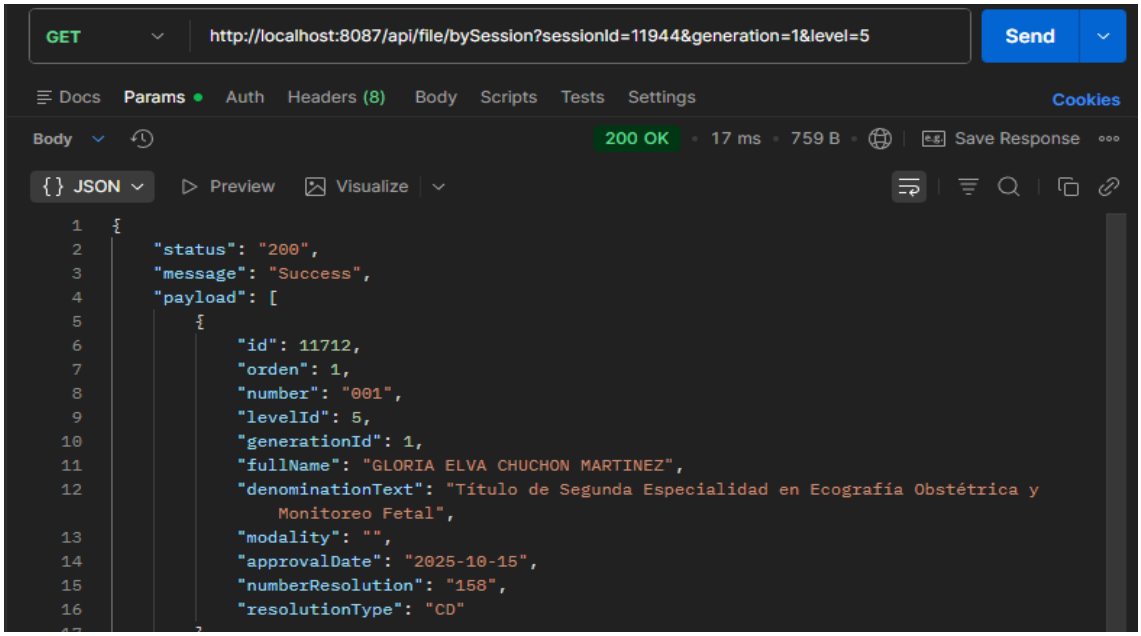


Nota: La imagen muestra la validación del criterio de aceptación Nº 1 con el endpoint create del servicio FileService en el puerto 8087 del API Gateway.

Criterio de Aceptación Nº 3

Figura 20

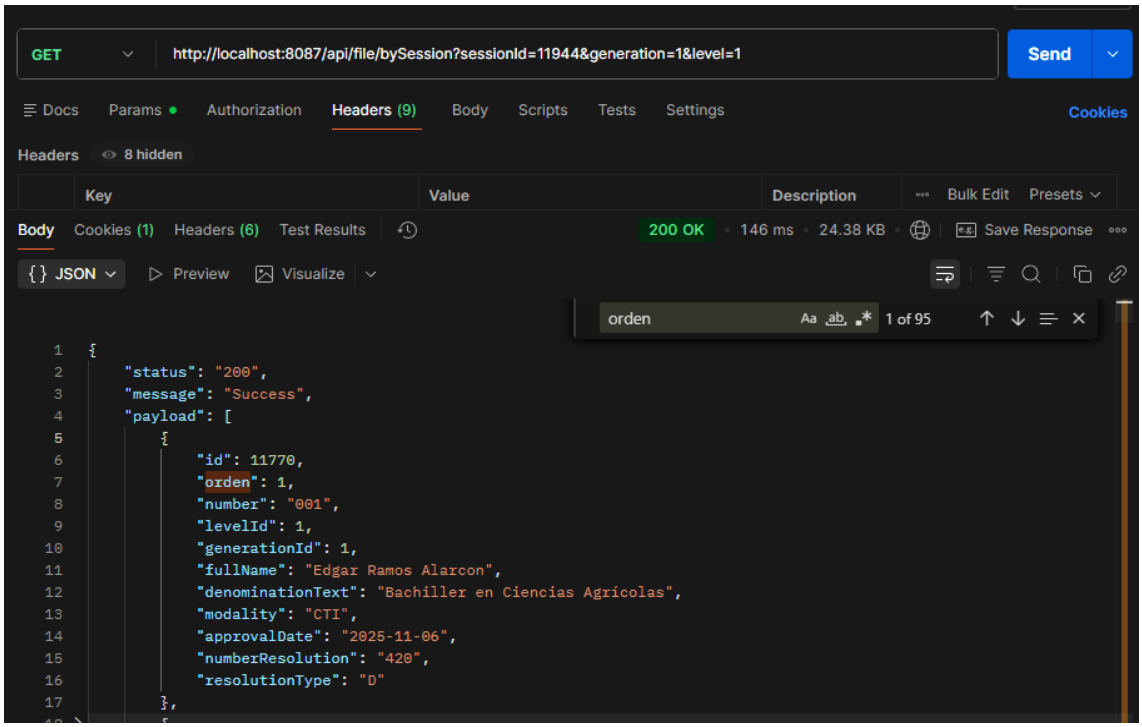
Endpoint de Listado de File Service



Nota: La imagen muestra una respuesta exitosa en el puerto 8087 del API Gateway.

Figura 21

Lista de Expedientes de Bachiller con Endpoint de FileService

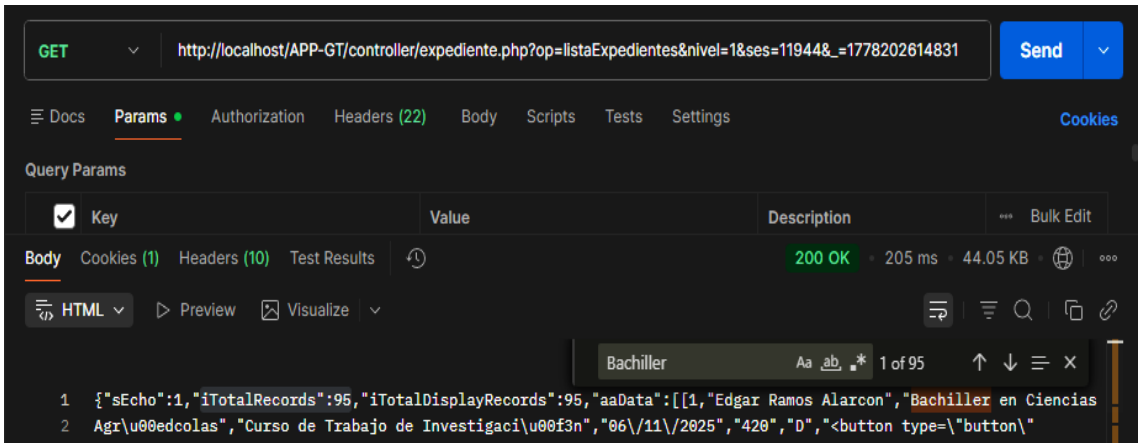


Nota: Se visualiza que FileService retorna 95 registros de bachiller con los parámetros de prueba Sesión = 11944, nivel = 1 y generación = 1.

Criterio de Aceptación Nº 4:

Figura 22

Lista de Expedientes de Bachiller con Endpoint de Monolito



Nota: Se visualiza que el monolito retorna 95 registros de bachiller con los parámetros de prueba Sesión = 11944, nivel = 1 y generación = 1.

b) COEXISTENCIA

Una vez que FileService superó todas las pruebas de aceptación, se configuró Spring Cloud Gateway para enrutar las peticiones de gestión de expedientes hacia el nuevo microservicio. El resto de rutas continuaron apuntando al monolito PHP.

Figura 23

Configuración de rutas del Sprint 1

```
routes:
  # ========== MICROSERVICIOS ==========

  # FileService - Microservicio
  - id: file-service
    uri: http://localhost:8082
    predicates:
      - Path=/api/file/**
    filters:
      - StripPrefix=0
      - AddRequestHeader=X-Forwarded-Prefix, /api

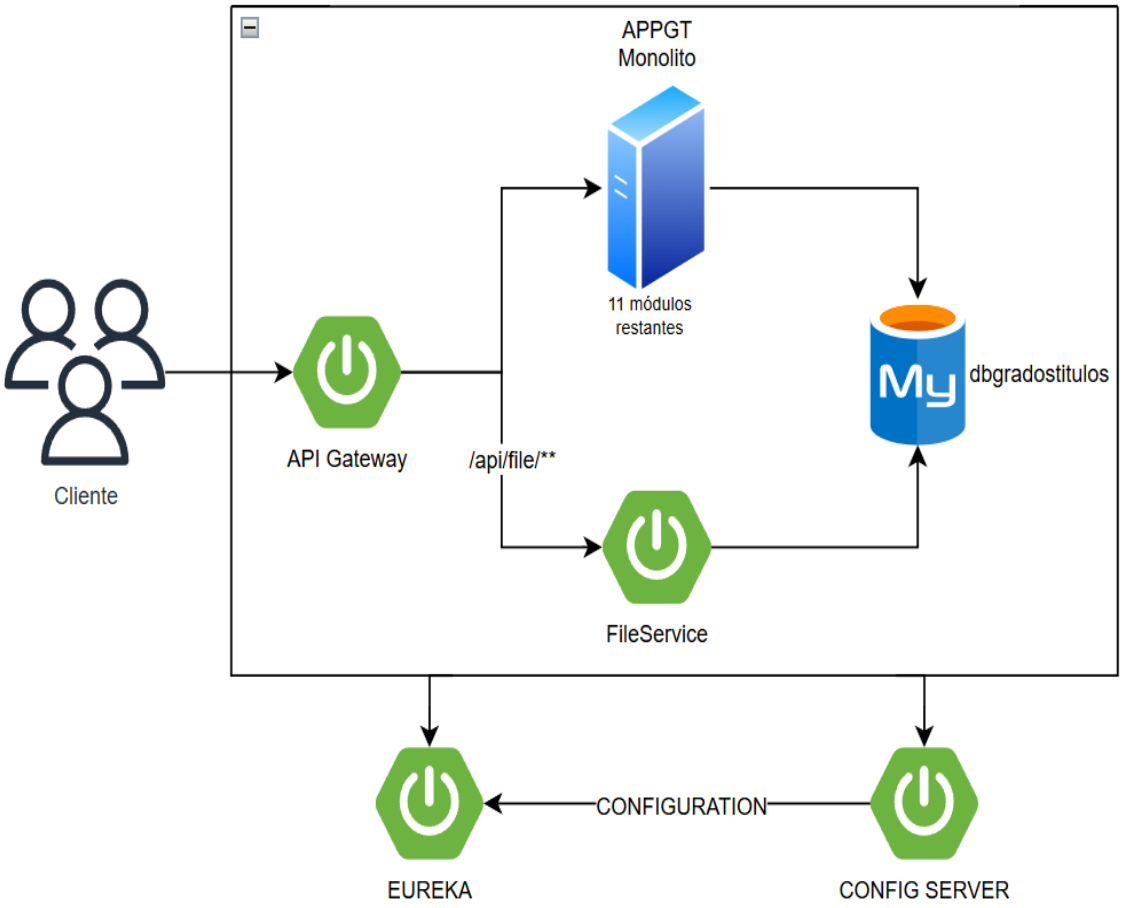
  # ========== MONOLITO PHP ==========
  # Todas las rutas del monolito empiezan con /controller/
  - id: monolito-php
    uri: http://localhost:8080
    predicates:
      - Path=/api/controller/**
    filters:
      - StripPrefix=1
```

Nota: Se muestra la configuración de rutas para la coexistencia con FileService.

- Arquitectura en el Sprint 1:

Figura 24

Arquitectura del sistema en el Sprint 1

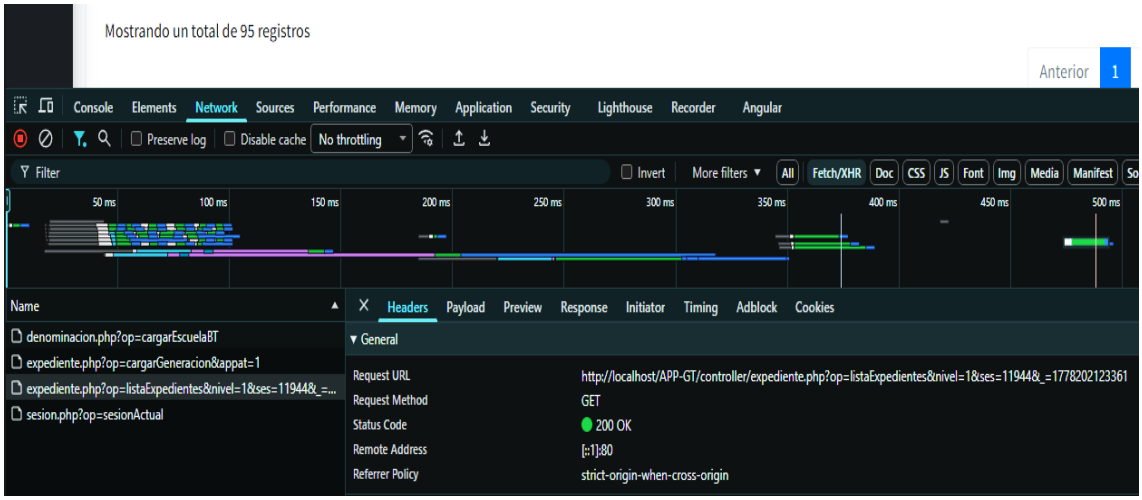


Nota: Se muestra la separación del módulo de Expedientes del Monolito al microservicio FileService.

Durante este período se verificó que FileService accedía correctamente a las mismas tablas del monolito. Los usuarios del sistema continuaron operando sin interrupciones, sin percibir ningún cambio en el comportamiento del sistema. Se monitorearon los logs del microservicio y se compararon las respuestas de FileService con las del monolito y se confirma que no hay errores.

Figura 25

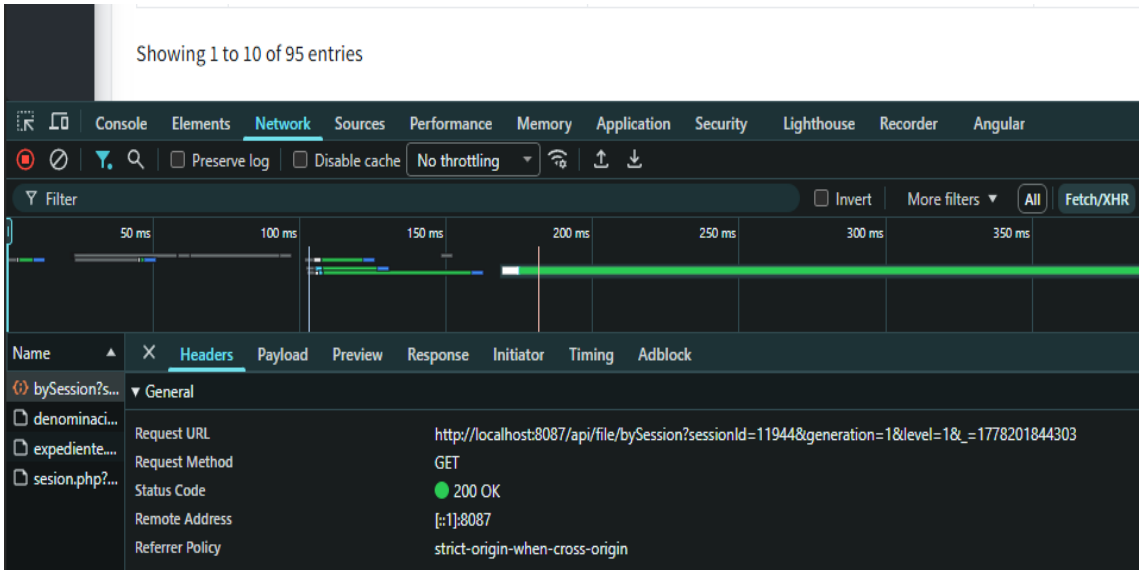
Request de Listado de Expedientes del Monolito



Nota: Se visualiza el request original del monolito para listado de expedientes con una respuesta de 95 registros.

Figura 26

Request de Lista de Expedientes de FileService



Nota: Se visualiza el request actualizado del microservicio para listado de expedientes con una respuesta de 95 registros para los mismos parámetros de la figura anterior.

Se realizó la validación del endpoint para listar expedientes utilizando el siguiente caso de prueba: sesion_id = 11944, generacion = 1, nivel_id = 1. El microservicio retornó una respuesta con estructura y contenido equivalentes a los producidos por el monolito (95 expedientes en el mismo orden y condiciones) para los mismos parámetros de entrada, confirmando la correcta migración de la funcionalidad de listado de expedientes.

c) ELIMINACIÓN

Una vez verificado que FileService funciona sin errores durante el período de coexistencia y que todos los criterios de aceptación se cumplen, se procedió a desactivar los controladores PHP de Bachiller, Título, Maestría, Doctorado y Segunda Especialidad del monolito. Las reglas de enrutamiento del Gateway fueron actualizadas para eliminar cualquier referencia residual al monolito para esas rutas y se le da un status 410 Gone.

Figura 27

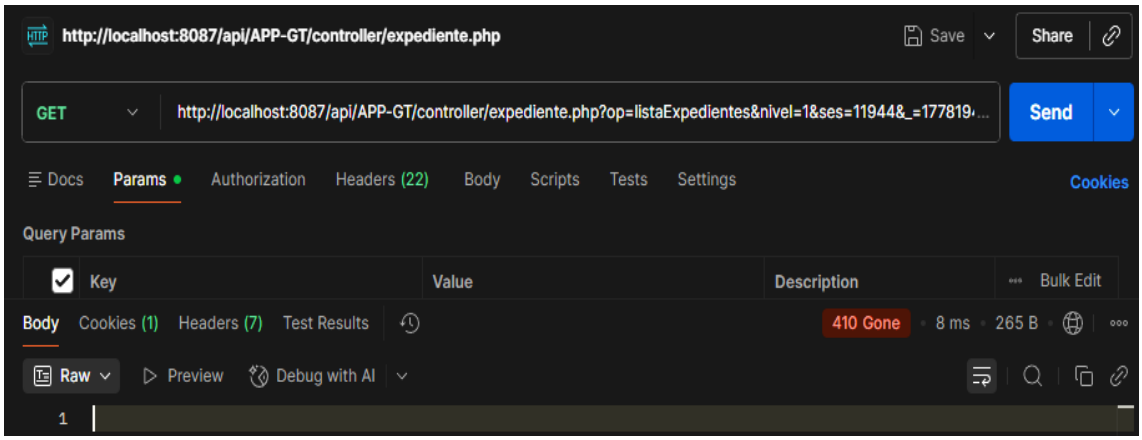
Bloqueo del módulo del Sprint 1

```
10        gateway:
30          routes:
46            # Ruta legacy bloqueada
47            - id: expediente-php-migrated
48              uri: no://op
49              predicates:
50                - Path=/api/APP-GT/controller/expediente.php
51              filters:
52                - SetStatus=410
53                - AddResponseHeader=X-Deprecated-Endpoint, migrated-to-microservice
```

Nota: Se muestra la configuración para el bloqueo de los módulos de expedientes del monolito.

Figura 28

Rutas Bloqueadas del Monolito - Sprint 1



Nota: Se verifique que los endpoints asociados a los módulos de expedientes responden con el estado 410 Gone, confirmando el bloqueo y el retiro de dichas funcionalidades, quedando 10 módulos a migrar.

4.4.3. SPRINT 2 – DOCUMENT SERVICE

Módulos migrados: Lista de Consejo Universitario (09), Resoluciones (10), Diplomas (11).

Puntuación de priorización: 80/100 **Puerto:** 8083

DocumentService fue incorporado en el segundo sprint con el objetivo de modernizar el proceso de generación documental del sistema. Este módulo era uno de los que más cambios requería, debido a que los formatos de resoluciones, listas oficiales y diplomas debían actualizarse constantemente conforme a disposiciones universitarias y lineamientos de la Superintendencia Nacional de Educación Superior Universitaria. Además, las modificaciones realizadas previamente para adaptarse a nuevos requerimientos funcionales ocasionaron que la generación de documentos, además de ejecutarse de forma individual por expediente, presentara errores e inconsistencias en los documentos generados. Esto produjo lentitud en el procesamiento y afectó significativamente la operatividad del módulo.

a) TRANSFORMACIÓN

Los tres módulos PHP de generación documental fueron consolidados en un único microservicio Java utilizando Apache POI XWPFDocument y retorna el documento como descarga HTTP directa.

La principal complejidad técnica de este sprint fue la generación de documentos Word con plantillas institucionales de forma masiva. DocumentService implementó la lógica de combinación de datos del expediente con plantillas .docx, replicando el comportamiento que tenía el monolito, pero con la particularidad de que la generación del documento pueda también realizarse de forma masiva y no solo individualmente como en la funcionalidad original.

Tabla 14

Endpoints implementados — DocumentService

Método	Endpoint	Descripción	Código de respuesta
GET	/api/document/ listCU/ {sesionId}	Genera lista oficial de la sesión	200 + archivo word
GET	/api/document /resolution/ {expId}	Genera resolución de un expediente	200 + archivo word
GET	/api/document /resolutionsCU/{sesionId}	Genera las resoluciones de una sesión	200 + archivo word
GET	/api/document/ diploma/ {expId}	Genera plantilla de diploma de un expediente	200 + archivo word
GET	/api/document/ diplomasCU/ {sesionId}	Genera plantillas de diploma de una sesión	200 + archivo word
GET	/api/document/health	Verifica la salud del microservicio	200 / 500

Nota: Elaboración Propia.

Pruebas y criterios de aceptación

Se utilizó el software Postman para realizar pruebas y validar las respuestas de los endpoints y la estructura de los JSON resultantes, así como se verificó la creación de los archivos Word de cada documento, también se usó el dashboard de Eureka Server para validar el registro y estado del microservicio.

Tabla 15

Criterios de aceptación — DocumentService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka como DOCUMENTSERVICE	Cumple
2	El documento Word generado tiene el formato institucional de la UNSCH	El archivo .docx generado coincide con la plantilla	Cumple
3	La lista de consejo incluye todos los expedientes de la sesión activa	Se verificó con datos de prueba	Cumple

4	La lista de consejo se genera en el orden y con los campos del formato institucional	El contenido del archivo .docx cumple con el formato.	Cumple

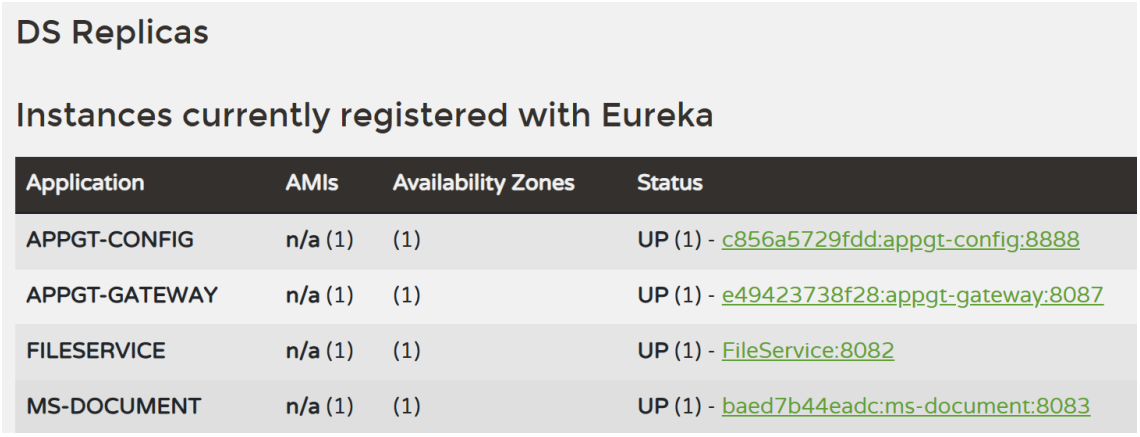
Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 29

Registro de Eureka Server

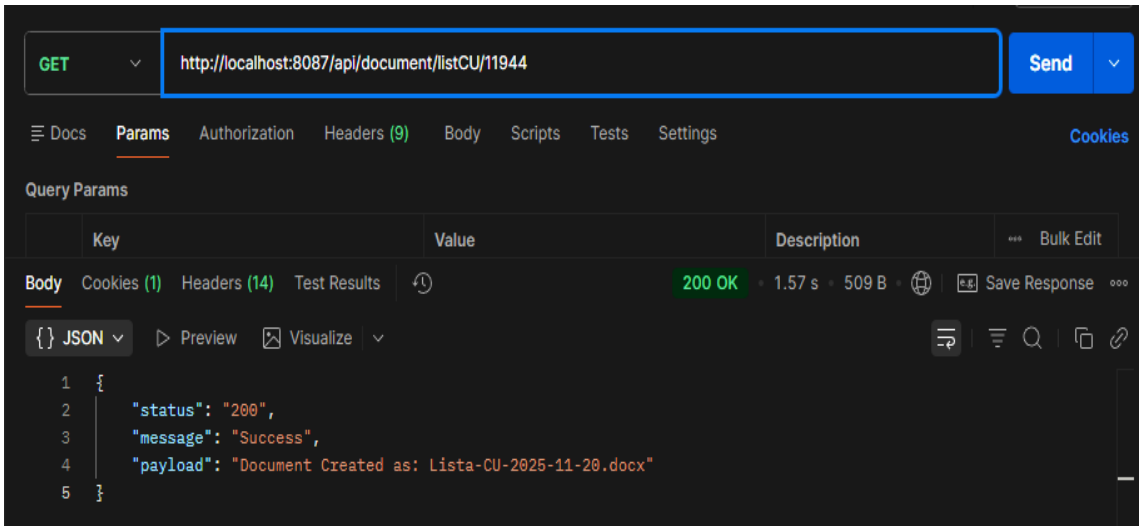


Nota: Se visualiza los servicios desplegados y registrados en eureka server: Appgt-config, appgt-gateway y documentservice.

Criterio de Aceptación Nº 2 y 3

Figura 30

Endpoint de Generación de ListaCU de Document Service



Nota: La imagen muestra la validación del criterio de aceptación Nº 2 con el endpoint listCU del servicio DocumentService en el puerto 8087 del API Gateway.

Figura 31

Documento Lista Oficial Generado



UNIVERSIDAD NACIONAL DE SAN
CRISTÓBAL DE HUAMANGA
Unidad de Certificación, Grados y Títulos
Portal Constitución Nº 57
AYACUCHO

RELACIÓN DE GRADOS Y TÍTULOS APROBADOS EN EL CONSEJO UNIVERSITARIO DE FECHA 20 DE NOVIEMBRE DEL 2025

13

NRO	NOMBRES Y APELLIDOS	FECHA APROBACIÓN CONSEJO FACULTAD
	GRADO DE DOCTOR	
001	BONIFACIO LEZANO CHICLLA, Doctor en Educación (TESIS)	15/10/2025
002	FELIX OCTAVIO HUAMANI SUAREZ, Doctor en Educación (TESIS)	15/10/2025
003	MARITZA GODOY QUISPE, Doctora en Educación (TESIS)	29/10/2025
	TÍTULO DE SEGUNDA ESPECIALIZACION	
001	GLORIA ELVA CHUCHON MARTINEZ, Título de Segunda Especialidad en Ecografía Obstétrica y Monitoreo Fetal (null)	15/10/2025
	DUPLICADO	
001	ZENOBIA JONISLLA VALLEJO, Licenciada en Educación Física	18/09/2025
002	DAYTON CHOQUE HUAMANI, Abogado	11/11/2025

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

GRADO DE BACHILLER	: 95
TITULO PROFESIONAL	: 63
GRADO DE MAESTRO	: 16
GRADO DE DOCTOR	: 3
TÍTULO DE SEGUNDA ESPECIALIZACION	: 1
DUPLICADO	: 2
TOTAL	**: 180**

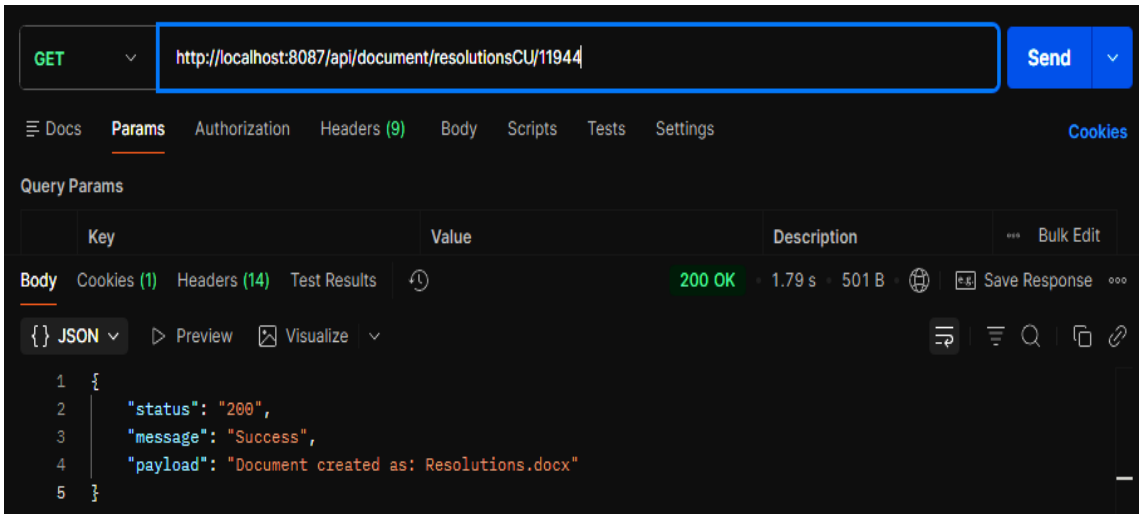
Ayacucho, 20 de noviembre del 2025

CAZP/SPH/llt.

Nota: Se generó el documento Word de lista oficial según el formato de la universidad y se verifica que generó 95 registros de bachiller que corresponden con la información obtenida del endpoint de FileService.

Figura 32

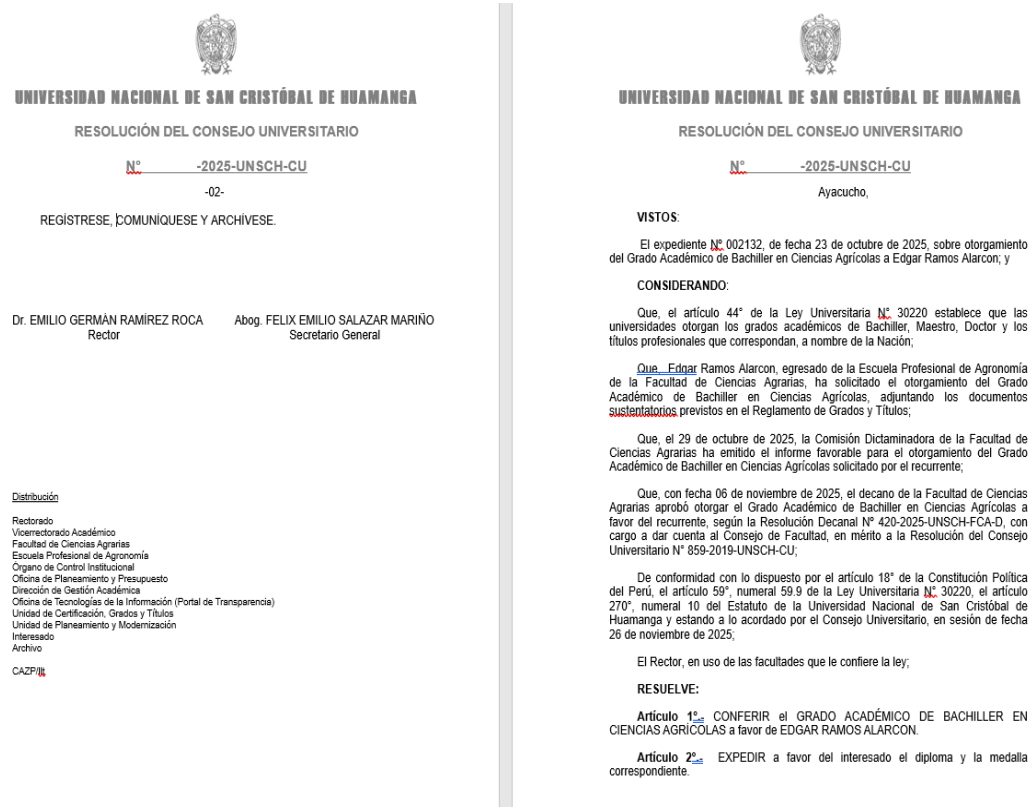
Endpoint de Generación de Resoluciones de Document Service



Nota: La imagen muestra la validación del criterio de aceptación Nº 2 con el endpoint resolutionsCU del servicio DocumentService en el puerto 8087 del API Gateway.

Figura 33

Documento Resolución Generado



UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

RESOLUCIÓN DEL CONSEJO UNIVERSITARIO

N° -2025-UNSCH-CU

-02-

REGÍSTRESE, COMUNÍQUESE Y ARCHÍVESE.

Dr. EMILIO GERMÁN RAMÍREZ ROCA
Rector

Abog. FELIX EMILIO SALAZAR MARIÑO
Secretario General

Distribución

Rectorado
Vicerrectorado Académico
Facultad de Ciencias Agrarias
Escuela Profesional de Agronomía
Órgano de Control Institucional
Oficina de Planeamiento y Presupuesto
Dirección de Gestión Académica
Oficina de Tecnologías de la Información (Portal de Transparencia)
Unidad de Certificación, Grados y Títulos
Unidad de Planeamiento y Modernización
Interesado
Archivo

CAZP/llk

UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA

RESOLUCIÓN DEL CONSEJO UNIVERSITARIO

N° -2025-UNSCH-CU

Ayacucho,

VISTOS:

El expediente Nº 002132, de fecha 23 de octubre de 2025, sobre otorgamiento del Grado Académico de Bachiller en Ciencias Agrícolas a Edgar Ramos Alarcon; y

CONSIDERANDO:

Que, el artículo 44° de la Ley Universitaria N° 30220 establece que las universidades otorgan los grados académicos de Bachiller, Maestro, Doctor y los títulos profesionales que correspondan, a nombre de la Nación;

Que, Edgar Ramos Alarcon, egresado de la Escuela Profesional de Agronomía de la Facultad de Ciencias Agrarias, ha solicitado el otorgamiento del Grado Académico de Bachiller en Ciencias Agrícolas, adjuntando los documentos sustentatorios previstos en el Reglamento de Grados y Títulos;

Que, el 29 de octubre de 2025, la Comisión Dictaminadora de la Facultad de Ciencias Agrarias ha emitido el informe favorable para el otorgamiento del Grado Académico de Bachiller en Ciencias Agrícolas solicitado por el recurrente;

Que, con fecha 06 de noviembre de 2025, el decano de la Facultad de Ciencias Agrarias aprobó otorgar el Grado Académico de Bachiller en Ciencias Agrícolas a favor del recurrente, según la Resolución Decanal Nº 420-2025-UNSCH-FCA-D, con cargo a dar cuenta al Consejo de Facultad, en mérito a la Resolución del Consejo Universitario N° 859-2019-UNSCH-CU;

De conformidad con lo dispuesto por el artículo 18° de la Constitución Política del Perú, el artículo 59°, numeral 59.9 de la Ley Universitaria Nº 30220, el artículo 270°, numeral 10 del Estatuto de la Universidad Nacional de San Cristóbal de Huamanga y estando a lo acordado por el Consejo Universitario, en sesión de fecha 26 de noviembre de 2025;

El Rector, en uso de las facultades que le confiere la ley;

RESUELVE:

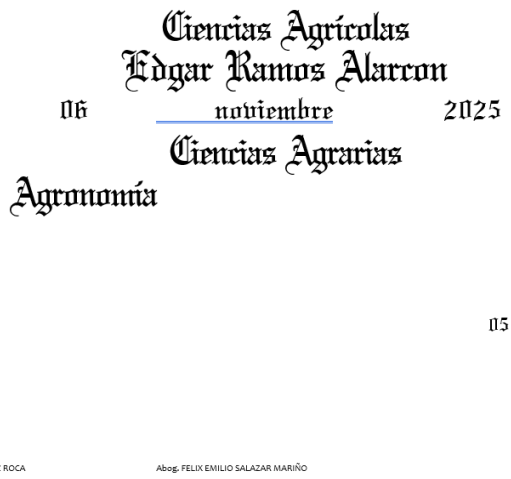
Artículo 1°.- CONFERIR el GRADO ACADÉMICO DE BACHILLER EN CIENCIAS AGRÍCOLAS a favor de EDGAR RAMOS ALARCON.

Artículo 2°.- EXPEDIR a favor del interesado el diploma y la medalla correspondiente.

Nota: Se generó el documento de resoluciones, se visualiza que la primera resolución corresponde al primer registro de la lista oficial.

Figura 34

Documento Diploma Generado

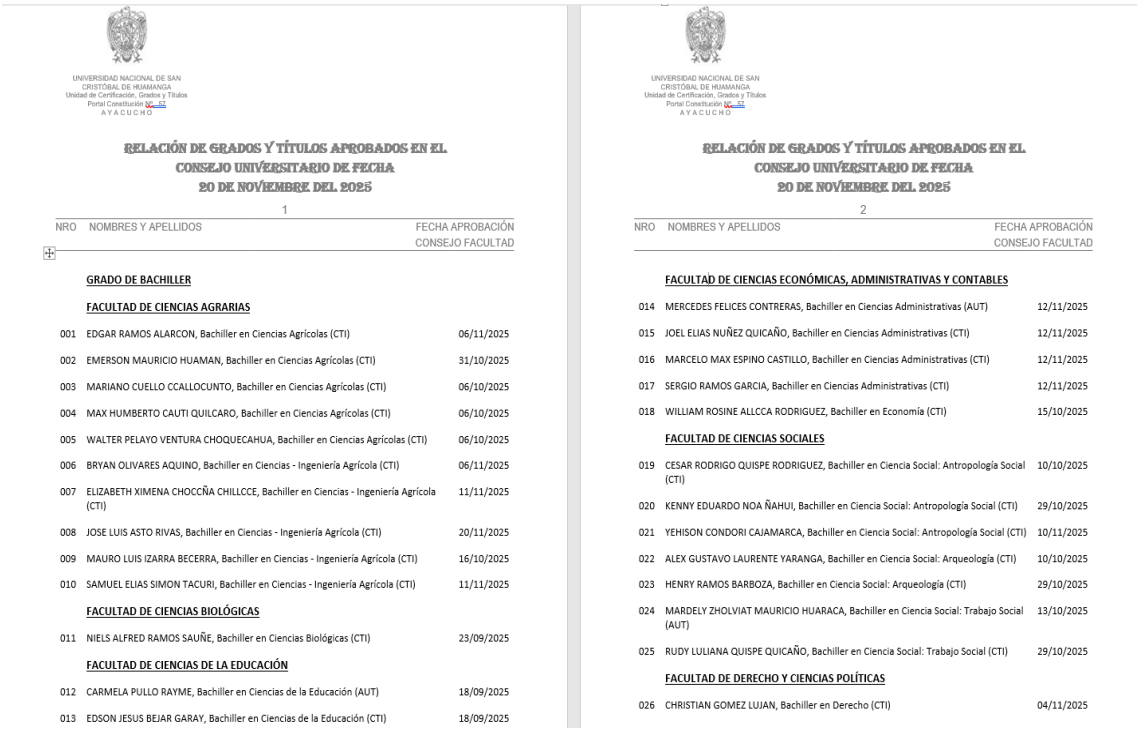


Nota: Se generó el documento de plantilla de diplomas, se visualiza que la primera hoja corresponde al primer registro de la lista oficial.

Criterio de Aceptación Nº 4

Figura 35

Lista Oficial generada



UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA
Unidad de Certificación, Grados y Títulos
Portal Constitución N° 57
AYACUCHO

RELACIÓN DE GRADOS Y TÍTULOS APROBADOS EN EL CONSEJO UNIVERSITARIO DE FECHA 20 DE NOVIEMBRE DEL 2025

1

NRO	NOMBRES Y APELLIDOS	FECHA APROBACIÓN CONSEJO FACULTAD
	GRADO DE BACHILLER	
	FACULTAD DE CIENCIAS AGRARIAS	
001	EDGAR RAMOS ALARCON, Bachiller en Ciencias Agrícolas (CTI)	06/11/2025
002	EMERSON MAURICIO HUAMAN, Bachiller en Ciencias Agrícolas (CTI)	31/10/2025
003	MARIANO CUELLO CCALLOCUNTO, Bachiller en Ciencias Agrícolas (CTI)	06/10/2025
004	MAX HUMBERTO CAUTI QUILCARO, Bachiller en Ciencias Agrícolas (CTI)	06/10/2025
005	WALTER PELAYO VENTURA CHOQUECAHUA, Bachiller en Ciencias Agrícolas (CTI)	06/10/2025
006	BRYAN OLIVARES AQUINO, Bachiller en Ciencias - Ingeniería Agrícola (CTI)	06/11/2025
007	ELIZABETH XIMENA CHOCCÑA CHILLCCE, Bachiller en Ciencias - Ingeniería Agrícola (CTI)	11/11/2025
008	JOSE LUIS ASTO RIVAS, Bachiller en Ciencias - Ingeniería Agrícola (CTI)	20/11/2025
009	MAURO LUIS IZARRA BECERRA, Bachiller en Ciencias - Ingeniería Agrícola (CTI)	16/10/2025
010	SAMUEL ELIAS SIMON TACURI, Bachiller en Ciencias - Ingeniería Agrícola (CTI)	11/11/2025
	FACULTAD DE CIENCIAS BIOLÓGICAS	
011	NIELS ALFRED RAMOS SAUÑE, Bachiller en Ciencias Biológicas (CTI)	23/09/2025
	FACULTAD DE CIENCIAS DE LA EDUCACIÓN	
012	CARMELA PULLO RAYME, Bachiller en Ciencias de la Educación (AUT)	18/09/2025
013	EDSON JESUS BEJAR GARAY, Bachiller en Ciencias de la Educación (CTI)	18/09/2025

UNIVERSIDAD NACIONAL DE SAN CRISTÓBAL DE HUAMANGA
Unidad de Certificación, Grados y Títulos
Portal Constitución N° 57
AYACUCHO

RELACIÓN DE GRADOS Y TÍTULOS APROBADOS EN EL CONSEJO UNIVERSITARIO DE FECHA 20 DE NOVIEMBRE DEL 2025

2

NRO	NOMBRES Y APELLIDOS	FECHA APROBACIÓN CONSEJO FACULTAD
	FACULTAD DE CIENCIAS ECONÓMICAS, ADMINISTRATIVAS Y CONTABLES	
014	MERCEDES FELICES CONTRERAS, Bachiller en Ciencias Administrativas (AUT)	12/11/2025
015	JOEL ELIAS NUÑEZ QUICAÑO, Bachiller en Ciencias Administrativas (CTI)	12/11/2025
016	MARCELO MAX ESPINO CASTILLO, Bachiller en Ciencias Administrativas (CTI)	12/11/2025
017	SERGIO RAMOS GARCIA, Bachiller en Ciencias Administrativas (CTI)	12/11/2025
018	WILLIAM ROSINE ALLCCA RODRIGUEZ, Bachiller en Economía (CTI)	15/10/2025
	FACULTAD DE CIENCIAS SOCIALES	
019	CESAR RODRIGO QUISPE RODRIGUEZ, Bachiller en Ciencia Social: Antropología Social (CTI)	10/10/2025
020	KENNY EDUARDO NOA ÑAHUI, Bachiller en Ciencia Social: Antropología Social (CTI)	29/10/2025
021	YEHISON CONDORI CAJAMARCA, Bachiller en Ciencia Social: Antropología Social (CTI)	10/11/2025
022	ALEX GUSTAVO LAURENTE YARANGA, Bachiller en Ciencia Social: Arqueología (CTI)	10/10/2025
023	HENRY RAMOS BARBOZA, Bachiller en Ciencia Social: Arqueología (CTI)	29/10/2025
024	MARDELY ZHOLVIAT MAURICIO HUARACA, Bachiller en Ciencia Social: Trabajo Social (AUT)	13/10/2025
025	RUDY LULIANA QUISPE QUICAÑO, Bachiller en Ciencia Social: Trabajo Social (CTI)	29/10/2025
	FACULTAD DE DERECHO Y CIENCIAS POLÍTICAS	
026	CHRISTIAN GOMEZ LUJAN, Bachiller en Derecho (CTI)	04/11/2025

Nota: La lista generada cumple con las condiciones de orden por generación, nivel, facultad, escuela, modalidad y orden alfabético de nombres.

DocumentService accede directamente a la base de datos compartida mediante queries nativas con JOIN para obtener todos los datos necesarios para los documentos. Esta estrategia es especialmente adecuada para DocumentService porque los documentos requieren joins complejos entre múltiples tablas que serían costosos de implementar como llamadas HTTP entre microservicios.

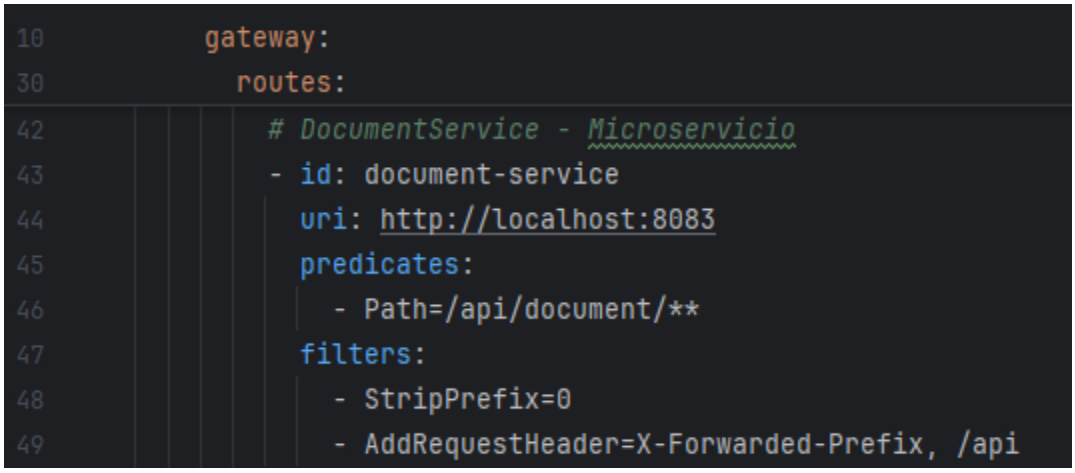
b) COEXISTENCIA

Una vez que DocumentService superó todas las pruebas de aceptación, se configuró Spring Cloud Gateway para enrutar las peticiones de gestión de documentos hacia el nuevo microservicio. El resto de rutas continuaron apuntando al monolito.

- Configuración de Ruta para DocumentService en el APIGateway

Figura 36

Configuración de rutas del Sprint 2

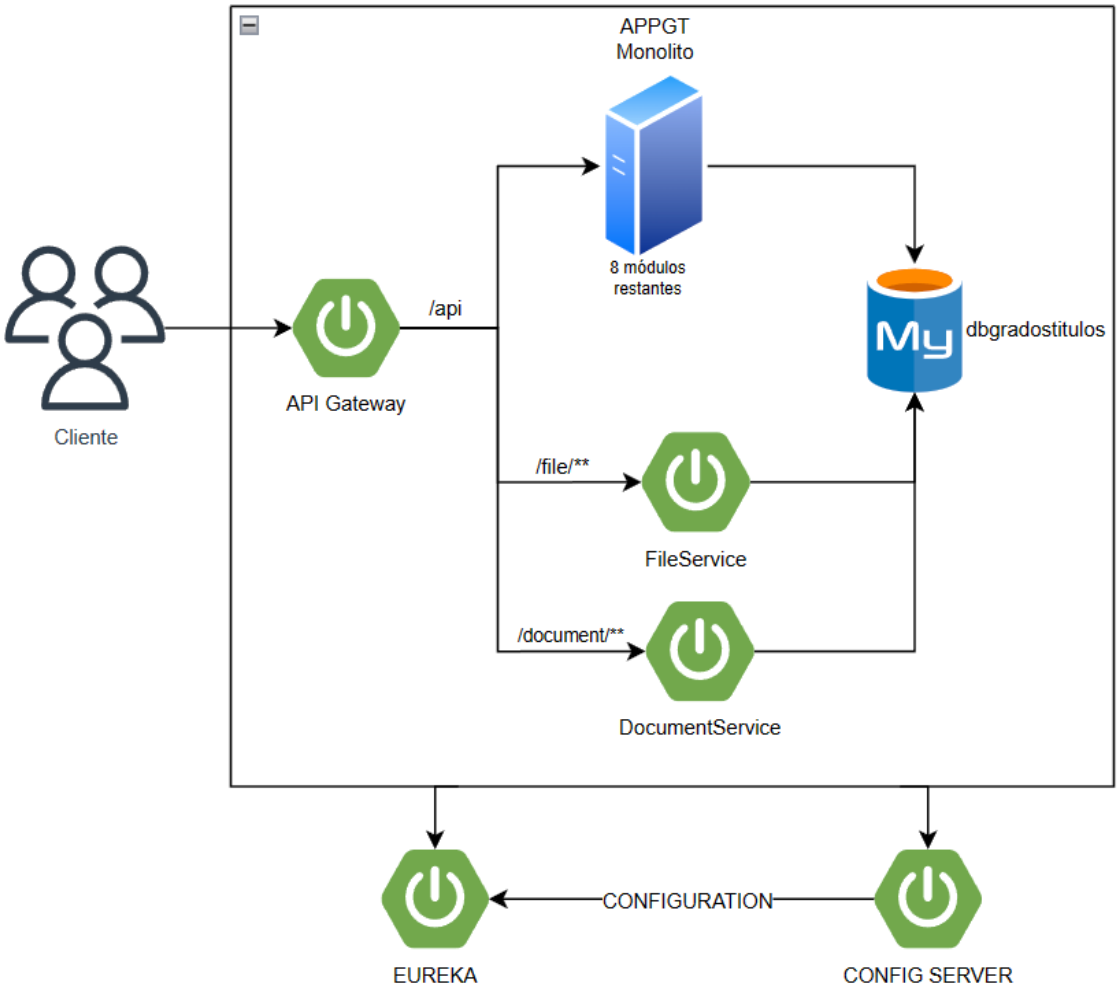


Nota: Se muestra la configuración de rutas para la coexistencia del monolito con el microservicio DocumentService.

- Arquitectura en el Sprint 2:

Figura 37

Arquitectura del sistema en el Sprint 2



Nota: Se muestra la separación de los módulos de Diploma, Resoluciones y Lista Oficial del Monolito al microservicio DocumentService.

Durante este período se verificó que DocumentService accedía correctamente a la información de los expedientes. Los usuarios del sistema generaron documentos sin inconvenientes. Se monitoreo los resultados obtenidos en los documentos generados y se compararon con la información que generaba el monolito inicialmente y se verificó que la información es equivalente.

c) ELIMINACIÓN

Validado el funcionamiento de DocumentService, se retiraron del monolito los controladores PHP de Lista de Consejo Universitario, Resoluciones y Diplomas.

Los módulos PHP de generación documental (generacion_lista.php, generacion_resolucion.php, generacion_diploma.php) dejan de recibir tráfico del frontend.

Los documentos institucionales son generados íntegramente por DocumentService con Apache POI, con mejor fidelidad de formato y datos correctamente insertados. Las reglas de enrutamiento del Gateway fueron actualizadas para eliminar las referencias a la ruta de documentos y se le da un status 410 Gone.

Figura 38

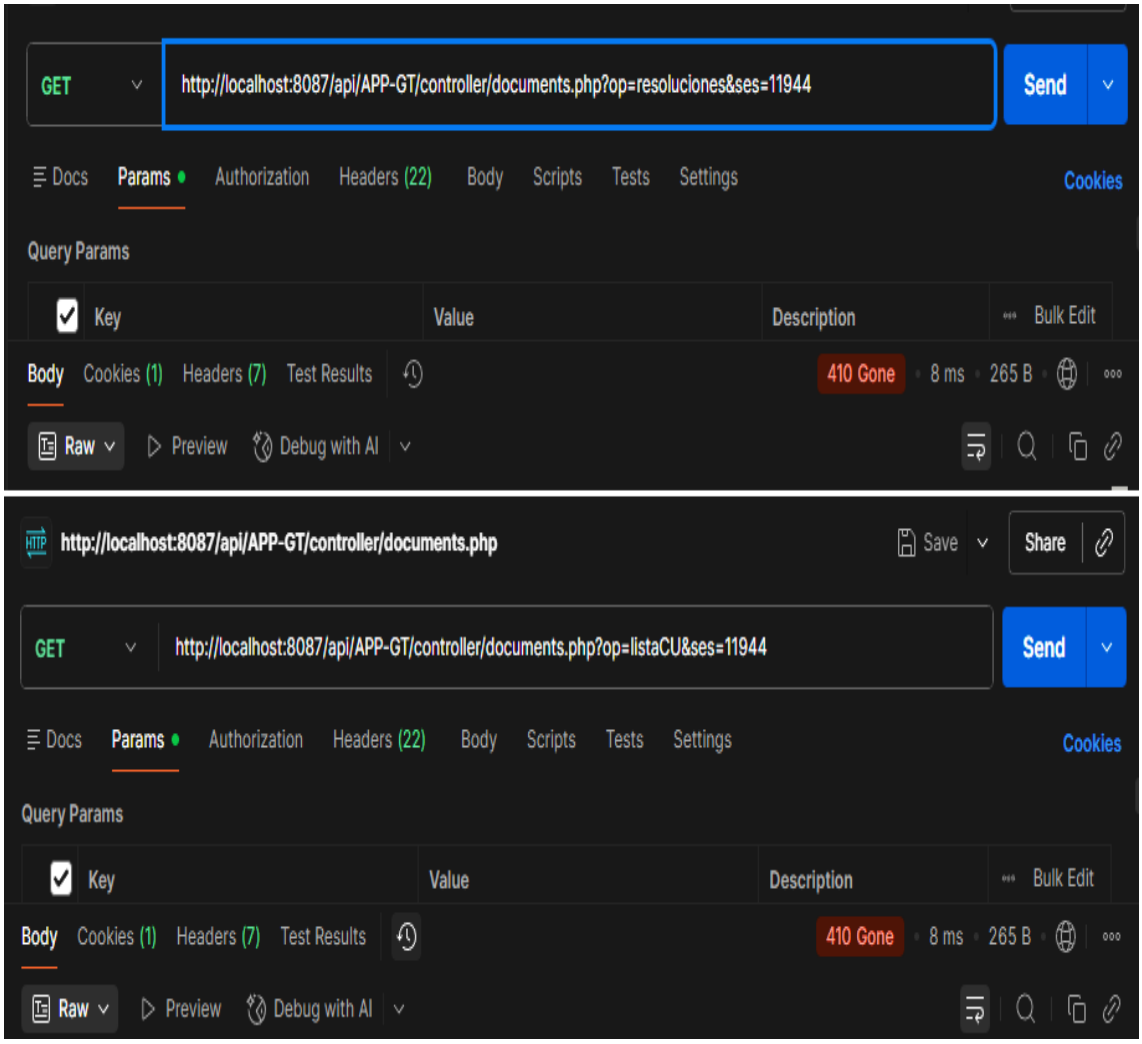
Bloqueo del módulo del Sprint 2

```
spring:
  cloud:
    gateway:
      routes:
        # Documentos
        - id: documento-php-migrated
          uri: no://op
          predicates:
            - Path=/api/APP-GT/controller/documents.php
          filters:
            - SetStatus=410
            - AddResponseHeader=X-Deprecated-Endpoint, migrated-to-microservice
```

Nota: Se muestra la configuración para el bloqueo de los módulos de documentos del monolito.

Figura 39

Rutas Bloqueadas del Monolito - Sprint 2



Nota: Se verifique que los endpoints asociados a los módulos de documentos responden con el estado 410 Gone, confirmando el bloqueo y el retiro de dichas funcionalidades.

4.4.4. SPRINT 3 – PERSON SERVICE

Módulos migrados: Persona (03).

Puntuación de priorización: 60/100 **Puerto:** 8084

a) TRANSFORMACIÓN

PersonService encapsula el dominio del interesado o solicitante del trámite académico. Su principal particularidad técnica se relaciona con la integración al API institucional de personas de la UNSCH. En el monolito, este módulo no se encontraba integrado con dicho servicio: al registrar una persona, el sistema consultaba únicamente la base de datos propia y, si no existía coincidencia, requería el registro manual de los datos. Durante este Sprint se aprovechó la migración del módulo para incorporar la funcionalidad de consumo del API externo, permitiendo consultar el DNI y recuperar automáticamente los datos del interesado. Esta lógica, que no existía en el monolito, fue encapsulada en la capa de servicio del PersonService mediante un cliente HTTP independiente.

Tabla 16

Endpoints implementados — PersonService

Método	Endpoint	Descripción	Código de respuesta
GET	/api/personas	Lista todas las personas registradas	200
GET	/api/personas/{id}	Obtiene una persona por ID	200 / 404
GET	/api/personas/dni/{dni}	Busca persona por DNI, realiza la búsqueda en la base de datos, si no obtiene coincidencias, busca en el API de la UNSCH.	200 / 400
POST	/api/personas	Registra una persona manualmente	200 / 400
PATCH	/api/personas/{id}	Actualiza datos de una persona	200 / 404
DELETE	/api/personas/{id}	Elimina una persona	200 / 404
GET	/api/personas/health	Verifica la salud del microservicio	200 / 500

Nota: Elaboración Propia.

Pruebas y criterios de aceptación

Se utilizaron Postman y Eureka Server para validar los endpoints y el registro/estado de los microservicios respectivamente.

Tabla 17

Criterios de aceptación — PersonService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka como PERSONSERVICE	Cumple
2	El endpoint GET /api/personas/{dni} retorna los datos del API UNSCH si el DNI no existe en base de datos	Retorna datos correctos del API institucional	Cumple
3	Cuando el API UNSCH no retorna resultado, el sistema permite registro manual	El endpoint POST /personas registra correctamente.	Cumple
4	Los endpoints REST retornan JSON con la estructura esperada	Validado con colección Postman	Cumple

Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 40

Registro de Eureka Server

Instances currently registered with Eureka

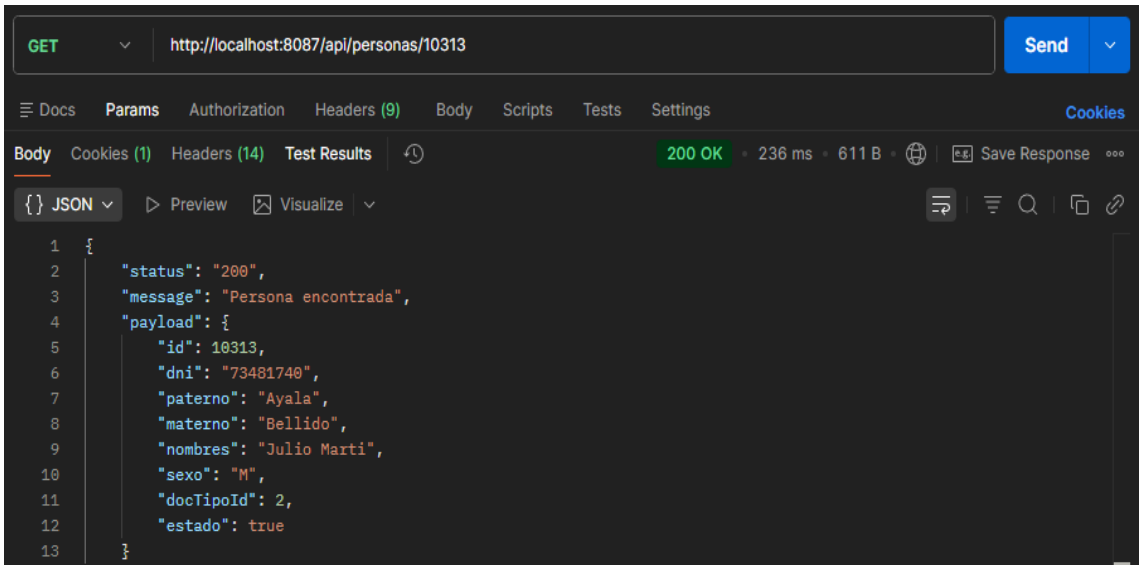
Application	AMIs	Availability Zones	Status
APPGT-CONFIG	n/a (1)	(1)	UP (1) - c856a5729fdd:appgt-config:8888
APPGT-GATEWAY	n/a (1)	(1)	UP (1) - e49423738f28:appgt-gateway:8087
FILESERVICE	n/a (1)	(1)	UP (1) - FileService:8082
DOCUMENTSERVICE	n/a (1)	(1)	UP (1) - DocumentService:8083

Nota: Se visualiza los servicios desplegados y registrados en eureka server: Appgt-config, appgt-gateway y personservice.

Criterio de Aceptación Nº 2

Figura 41

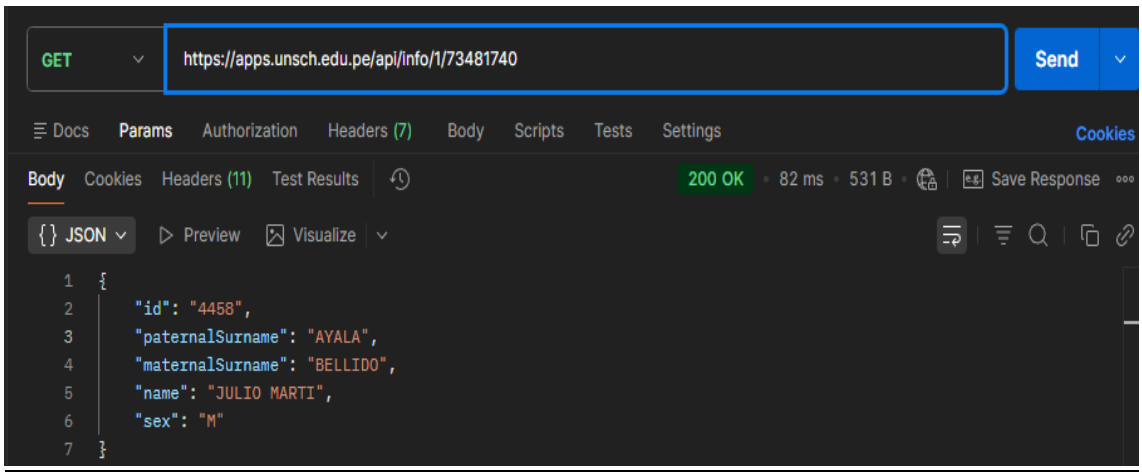
Endpoint de Consulta de DNI



Nota: La imagen muestra la validación del criterio de aceptación Nº 2 con el endpoint de consulta por DNI del servicio PersonService en el puerto 8087 del API Gateway.

Figura 42

Validación de respuesta de la API UNSCH



Nota: La imagen muestra la validación de que la API UNSCH responde correctamente a las consultas realizadas.

La lógica de consulta de la API UNSCH se implementa en la capa de lógica de negocio: primero se realiza la búsqueda en la base de datos y, si no hay coincidencias, hace un llamado al cliente que consume la API UNSCH.

Figura 43

Lógica de consumo de API UNSCH

```java
public class PersonServiceImpl implements PersonService {

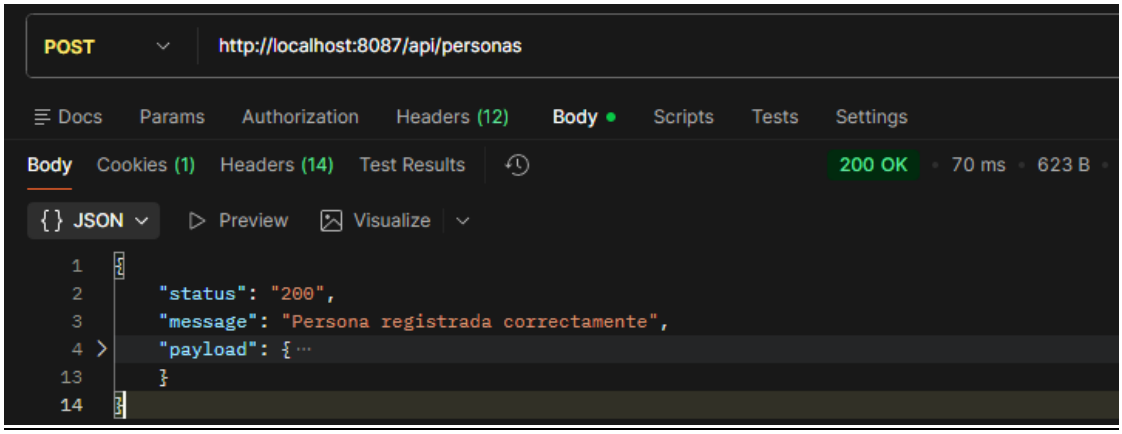
    @Loggable  1 usage
    @Override
    @Transactional
    public Response findByDni(String dni) {
        // 1. Buscar en base de datos
        Optional<PersonModel> personOpt = personRepository.findByDni(dni);
        if (personOpt.isPresent()) {
            log.info("Persona encontrada en BD con DNI: {}", dni);
            return new Response(
                    status: "200",
                    message: "Persona encontrada",
                    PersonMapper.toDto(personOpt.get())
            );
        }
        // 2. No existe en BD, consultar a API externa
        log.info("Persona no encontrada en BD, consultando API UNMSCH para DNI: {}", dni);
        Optional<PersonApiResponse> apiResponseOpt = personClient.findPersonByDni(dni);
        if (apiResponseOpt.isPresent()) {
            PersonApiResponse apiResponse = apiResponseOpt.get();
            // 3. Mapear y guardar en BD
            PersonModel newPerson = PersonMapper.toEntityFromApi(apiResponse, dni);
            PersonModel saved = personRepository.save(newPerson);
            log.info("Persona guardada en BD desde API externa con DNI: {}", dni);
            return new Response(
                    status: "200",
                    message: "Persona encontrada y registrada correctamente",
                    PersonMapper.toDto(saved)
            );
        }
        // 4. No existe en BD ni en API externa
        log.error("Persona no encontrada en BD ni en API UNMSCH para DNI: {}", dni);
        throw new PersonException("Persona con DNI " + dni + " no encontrada");
    }
```

Nota: La imagen muestra la lógica de la capa de servicio donde se codifica el flujo de consulta de DNI y la lógica para consumir la API UNSCH.

Criterio de Aceptación Nº 3

Figura 44

Endpoint de registro de Persona

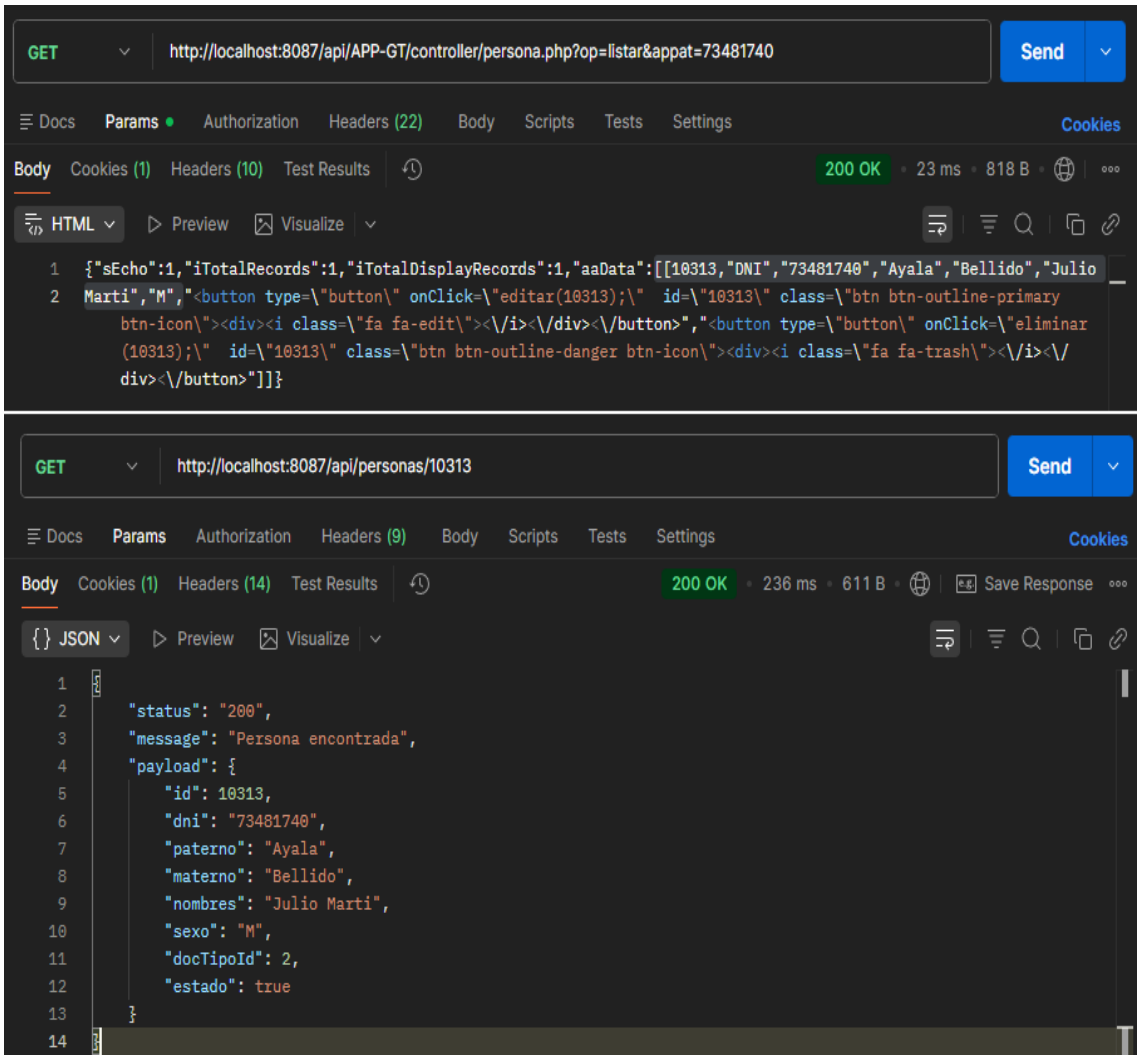


Nota: La imagen muestra la validación del criterio de aceptación Nº 3.

Criterio de Aceptación Nº 4:

Figura 45

Comparación de response de consulta de Personas



Nota: Se visualiza que el endpoint de consulta de personas por id o dni retornan los mismos datos tanto en el monolito como en el microservicio Personas.

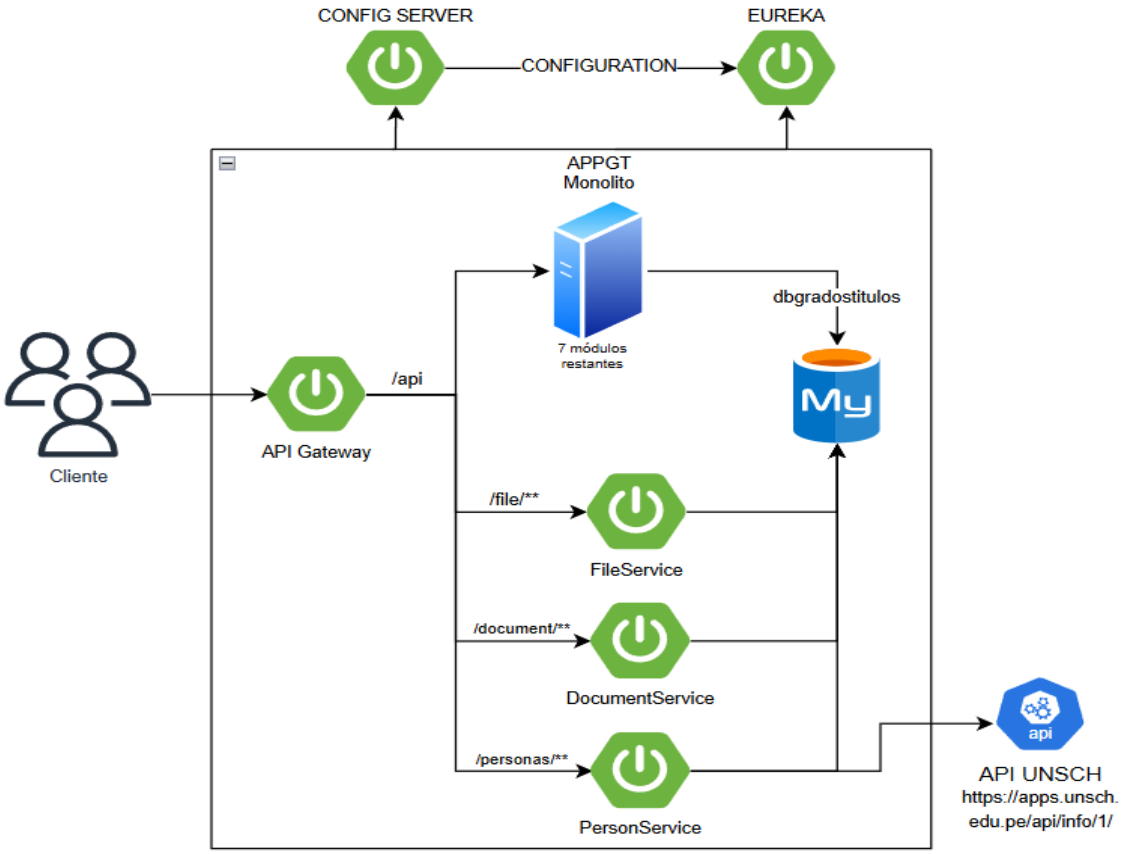
d) COEXISTENCIA

Una vez que PersonService superó todas las pruebas de aceptación, se configuró Spring Cloud Gateway para enrutar las peticiones de gestión de personas hacia el nuevo microservicio. El resto de rutas continuaron apuntando al monolito PHP.

- Arquitectura en el Sprint 3:

Figura 46

Arquitectura del sistema en el Sprint 3



Nota: Se muestra la separación del módulo de Personas del Monolito al microservicio PersonService, así como la integración el API UNSCH.

- Configuración de Ruta para PersonService en el APIGateway

Figura 47

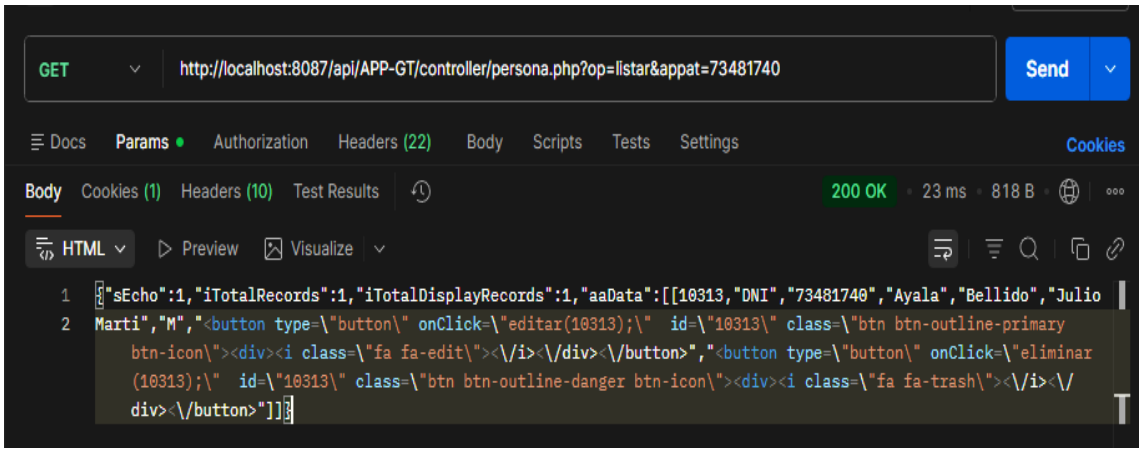
Configuración de rutas del Sprint 3

```
51          # PersonService - Microservicio
52          - id: person-service
53            uri: http://localhost:8084
54            predicates:
55              - Path=/api/personas/**
56            filters:
57              - StripPrefix=0
58              - AddRequestHeader=X-Forwarded-Prefix, /api
```

Nota: Se muestra la configuración de rutas para la coexistencia del monolito con el microservicio PersonService.

Figura 48

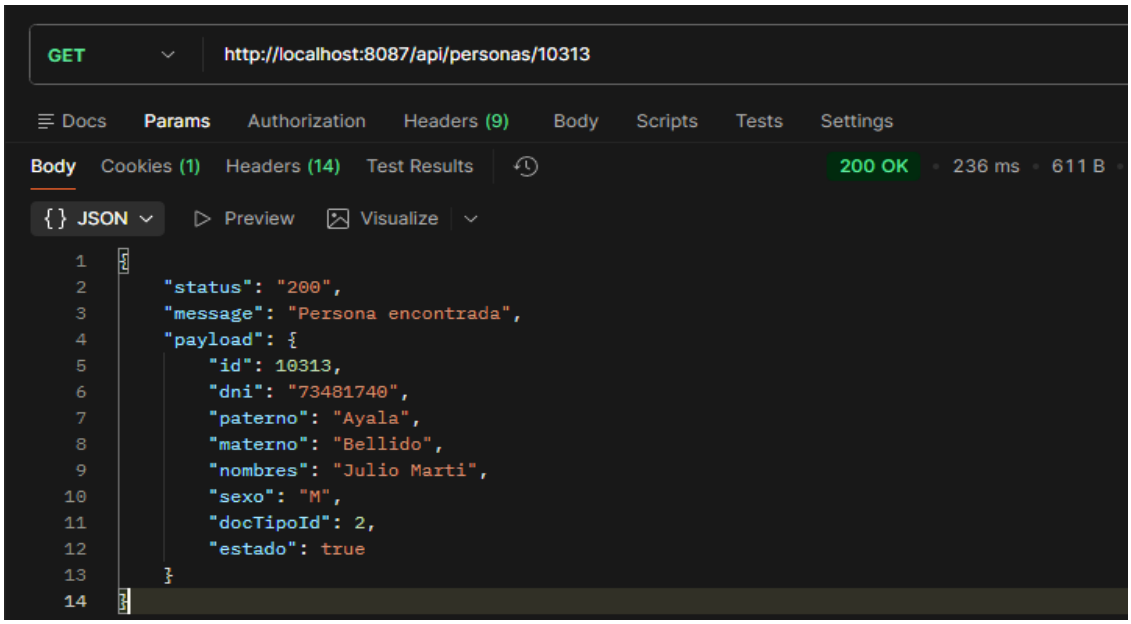
Request de Consulta de DNI en el monolito



Nota: Se visualiza el request original del monolito para consulta por DNI responde correctamente pasando por el API GATEWAY.

Figura 49

Request de Consulta de DNI por PersonService



Nota: Se visualiza el request actualizado del microservicio para consulta porDNI responde correctamente pasando por el API GATEWAY.

e) ELIMINACIÓN

Se retiró el módulo Persona (persona.php) del monolito PHP. La gestión de registros de personas y la integración con la API UNSCH se realizan íntegramente a

través del microservicio. Al igual que en los servicios anteriores, las reglas de enrutamiento del Gateway fueron actualizadas para eliminar cualquier referencia residual al monolito para la ruta /persona y se le da un status 410 Gone.

Figura 50

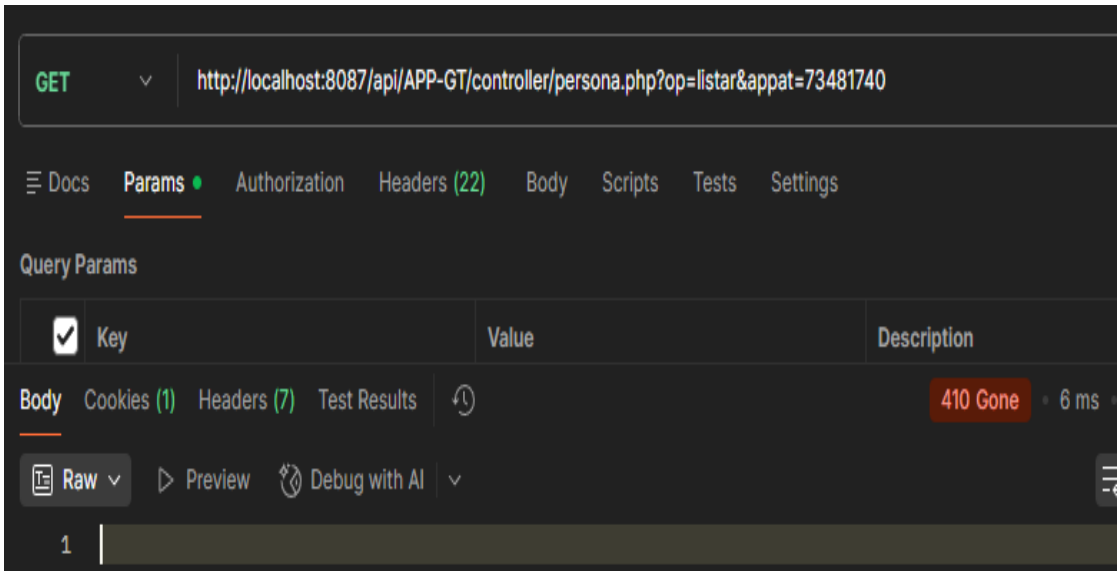
Bloqueo del módulo del Sprint 3



Nota: Se muestra la configuración para el bloqueo de los módulos de personas del monolito.

Figura 51

Rutas Bloqueadas del Monolito - Sprint 3



Nota: Se verifique que los endpoints asociados a los módulos de personas responden con el estado 410 Gone, confirmando el bloqueo y el retiro de dichas funcionalidades, quedando 6 módulos a migrar.

4.4.5. SPRINT 4 – SESSION SERVICE

Módulos migrados: Sesión Universitaria (01).

Puntuación de priorización: 30/100 **Puerto:** 8085

a) TRANSFORMACIÓN

Una de las reglas de negocio más estrictas es la que se contempla en este microservicio: solo puede existir una sesión activa por consejo universitario. Esta restricción, que en el monolito se consultaba por varios módulos con métodos para cada uno, queda centralizada y bien encapsulada en la capa de servicio del microservicio.

Tabla 18

Endpoints implementados — SesionService

Método	Endpoint	Descripción	Código de respuesta
GET	/api/session/all	Lista todas las sesiones	200
GET	/api/session/ current	Obtiene la sesión actualmente activa	200 / 404
GET	/api/session/{id}	Obtiene una sesión por ID	200 / 400
GET	/api/session/cu	Sesiones de consejo universitario	200 / 400
POST	/api/session/ createSession	Registra una nueva sesión	200 / 400
PATCH	/api/session/ updateSession	Actualiza una sesión	200 / 404
DELETE	/api/session/{id}	Elimina una sesión	200 / 400
GET	/api/ session /health	Verifica la salud del microservicio	200 / 500

Nota: Elaboración Propia.

Pruebas y criterios de aceptación

Se utilizó el software Postman para realizar pruebas y validar las respuestas de los endpoints y la estructura de los JSON resultantes, también se usó el dashboard de Eureka Server para validar el registro y estado de los microservicios.

Tabla 19

Criterios de aceptación — SesionService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka como SESIONSERVICE	Cumple
2	Al activar una sesión, la sesión anteriormente activa se desactiva automáticamente	Verificado con la actualización o registro de una sesión con estado true.	Cumple
3	El endpoint GET /current retorna solo la sesión con estado activo	Retorna la sesión correcta o 404 si no hay ninguna activa	Cumple
4	Los endpoints REST retornan JSON con la estructura esperada	Validado con colección Postman	Cumple

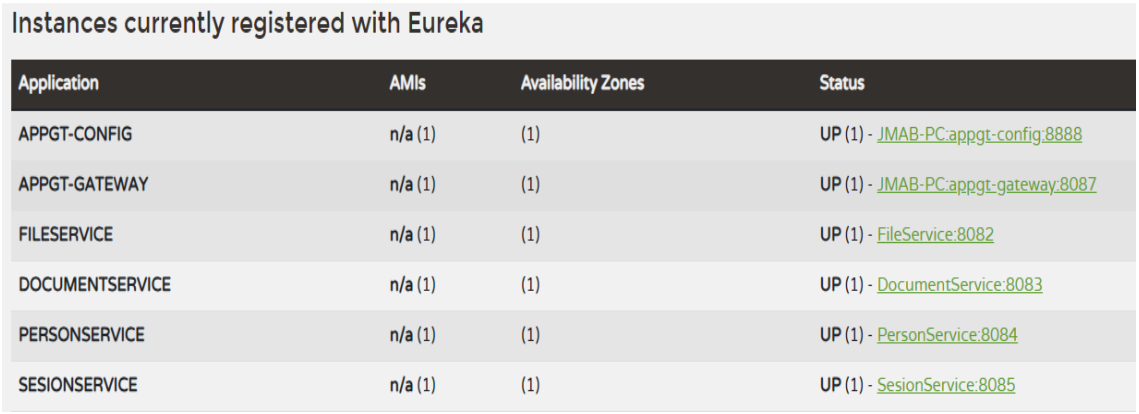
Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 52

Registro de Eureka Server

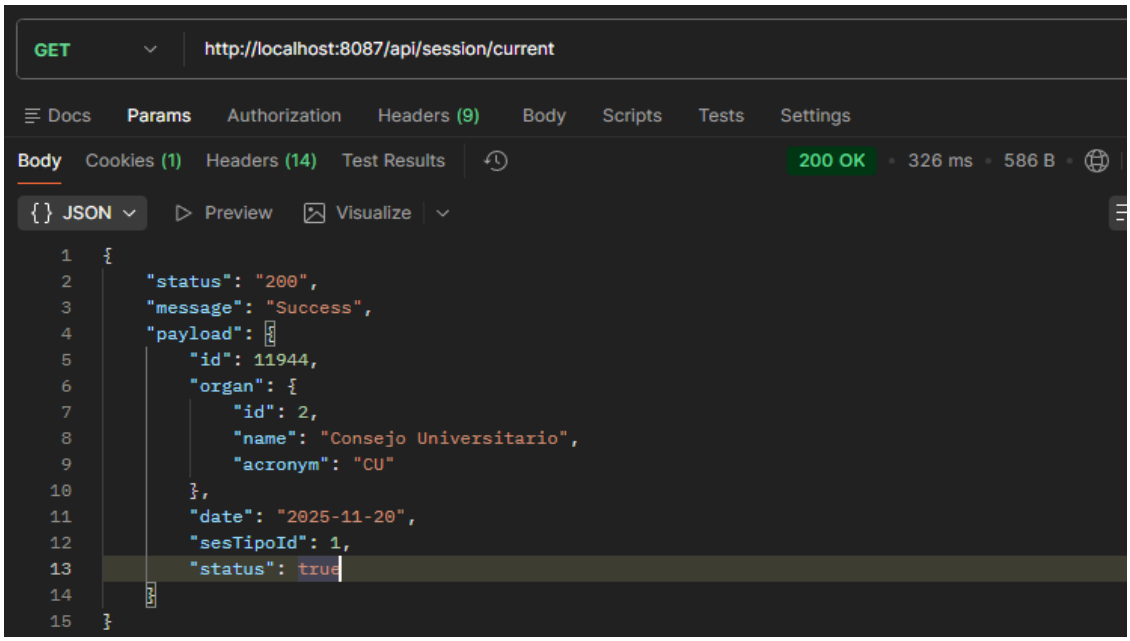


Nota: Se visualiza sesion service en la lista de microservicios desplegados.

Criterio de Aceptación Nº 2 y 3

Figura 53

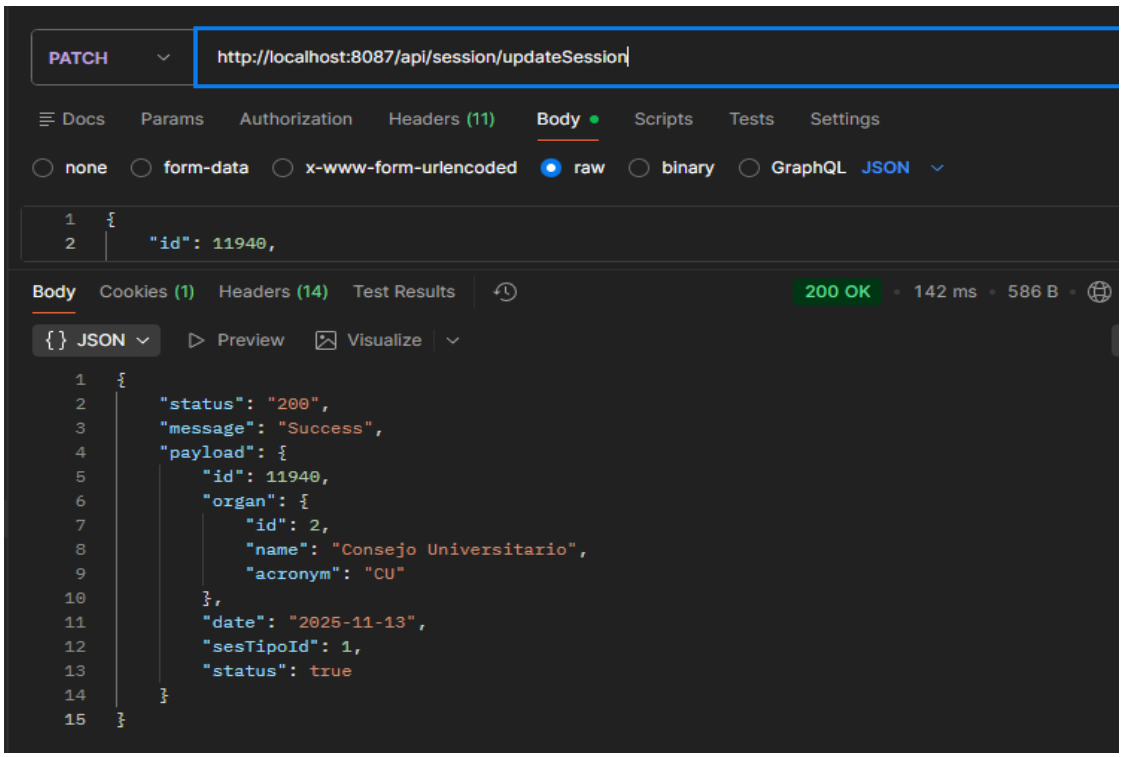
Sesión Activa



Nota: La imagen muestra la sesión activa.

Figura 54

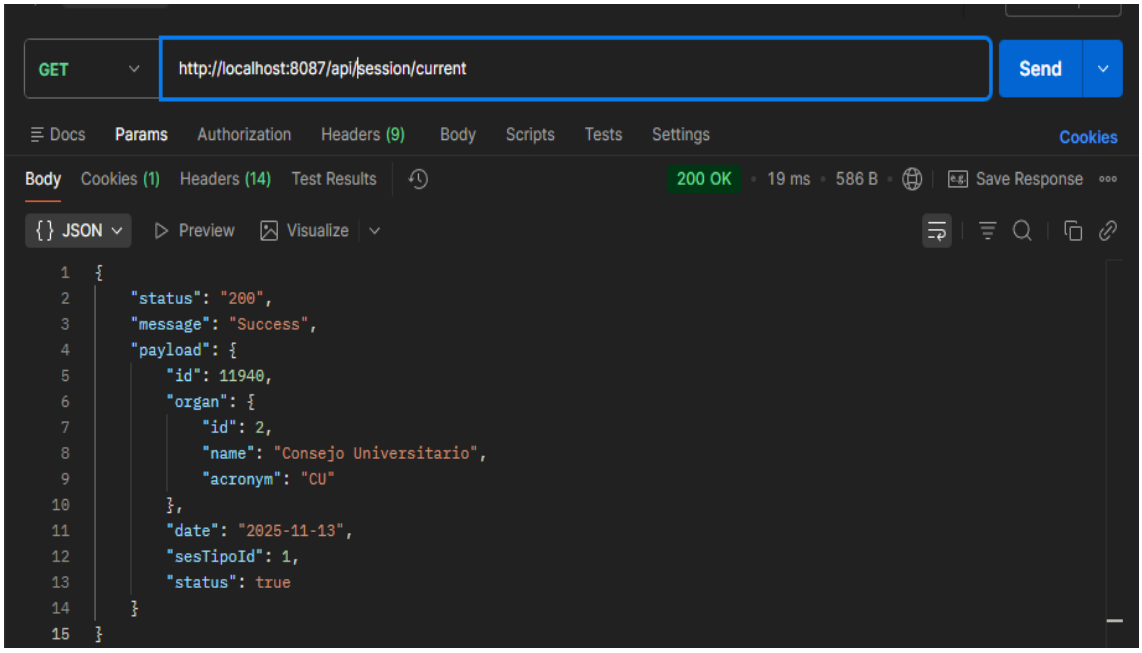
Activación de otra Sesión



Nota: La imagen muestra la activación de otra sesión y la respuesta indica un status true.

Figura 55

Sesión Anterior Desactivada

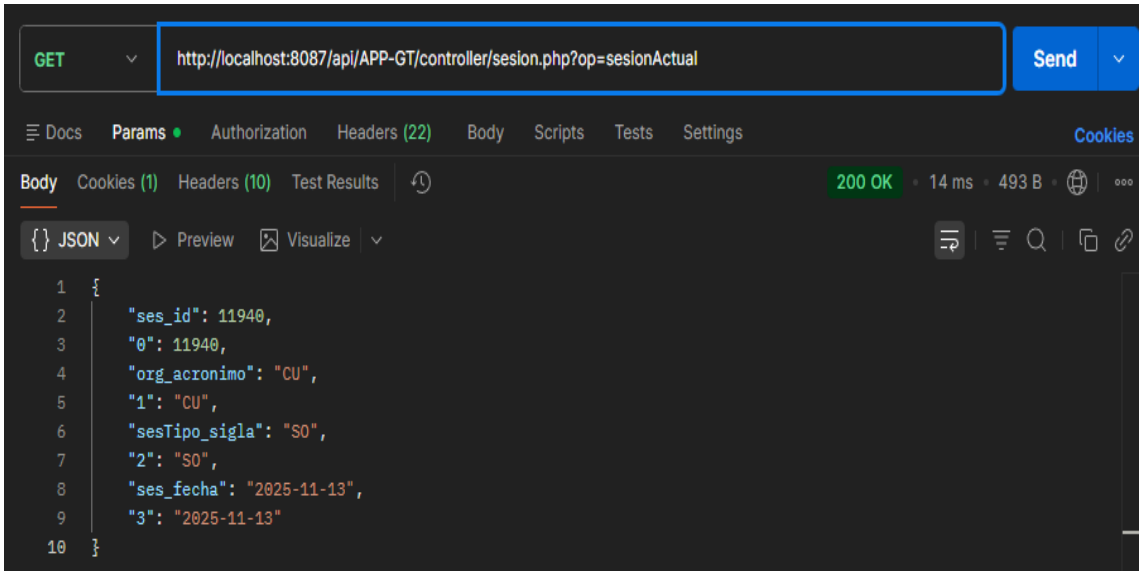


Nota: La imagen muestra la sesión activa es la que se acaba de editar y ya no la anterior, así como solo retorna 1 registro de sesión lo que cumple el criterio de aceptación Nº 3.

Criterio de Aceptación Nº 4:

Figura 56

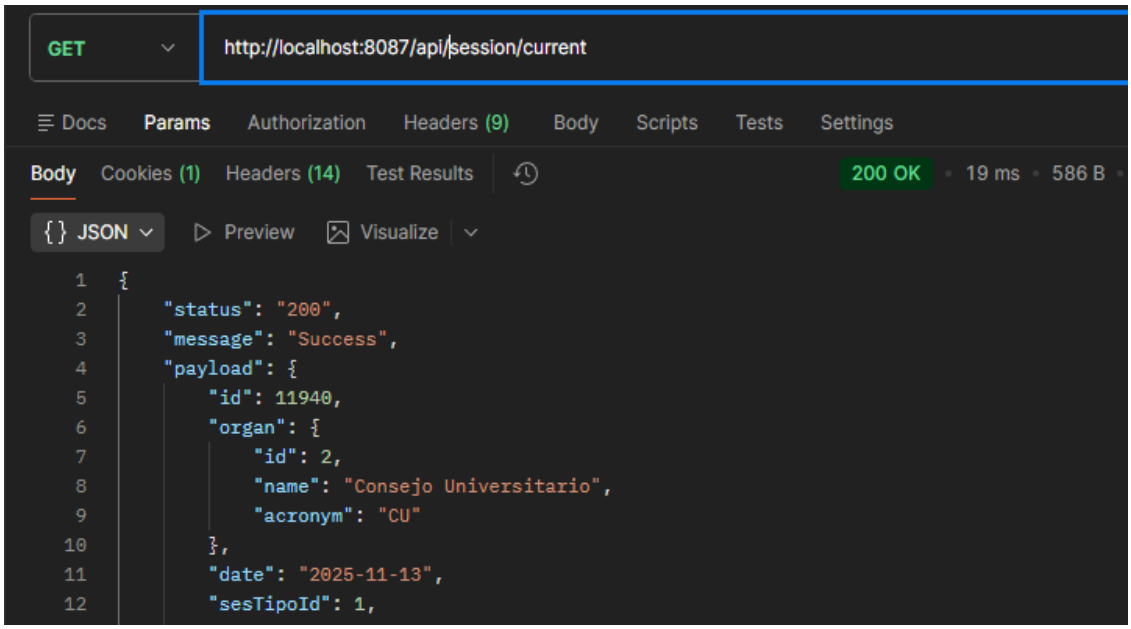
Datos de Sesión Activa obtenido desde el monolito



Nota: Se visualiza la información de la sesión actual obtenida.

Figura 57

Datos de Sesión Activa desde SesionService



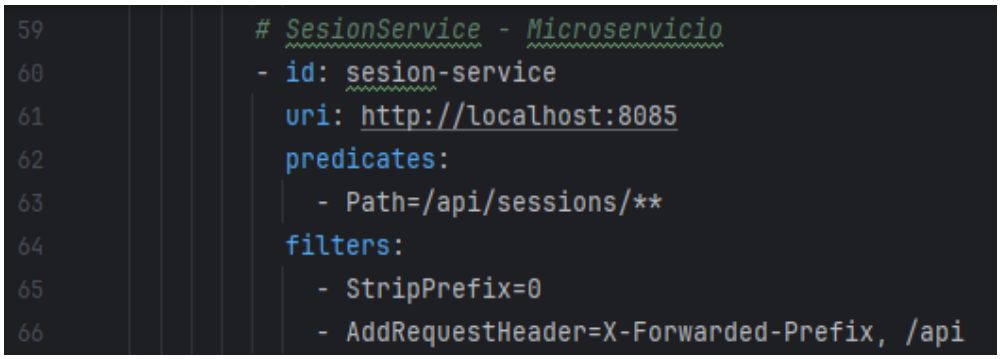
Nota: Se visualiza que los datos obtenidos de la sesión activa son iguales a los obtenidos por el monolito en la figura anterior.

b) COEXISTENCIA

El Gateway redirigió /api/sesiones/** a SesionService. Se verificó que se cumple la restricción de una sola sesión activa y que se puede obtener correctamente la información desde SesionService a través del Gateway. Los datos de sesiones creados por el microservicio son leídos directamente de la tabla sesion por FileService y DocumentService, sin necesidad de llamadas HTTP al servicio.

Figura 58

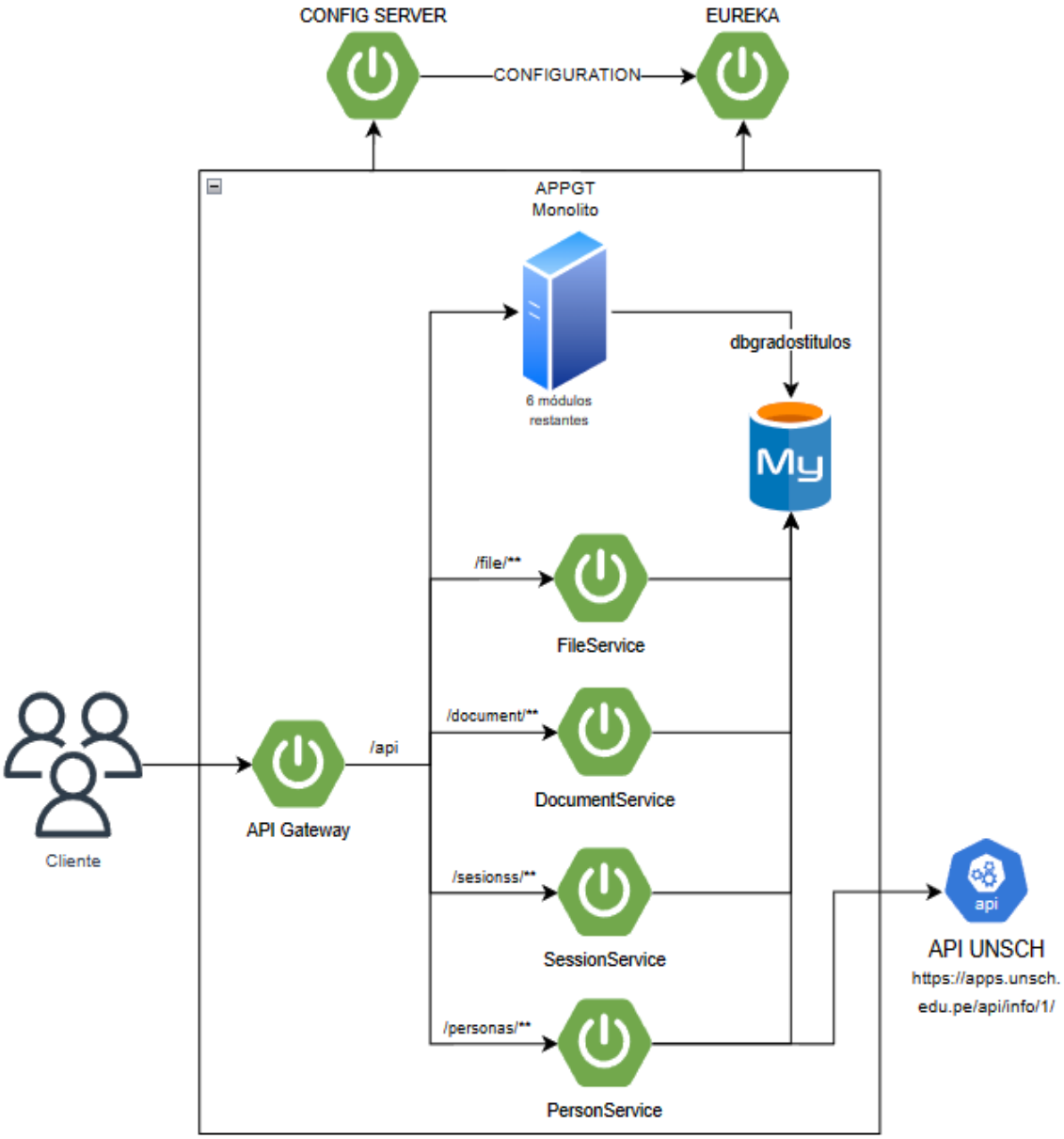
Configuración de rutas del Sprint 4



Nota: Se muestra la configuración de rutas para la coexistencia del monolito con el microservicio SesionService.

Figura 59

Arquitectura del sistema en el Sprint 4



Nota: Se muestra la separación del módulo de Sesiones del Monolito al microservicio SesionService.

c) ELIMINACIÓN

Se retiró el módulo Sesión Universitaria (sesión.php) del monolito. La gestión de sesiones se realiza íntegramente a través del microservicio con reglas de negocio formalizadas. Las reglas de enrutamiento del Gateway fueron actualizadas para eliminar cualquier referencia residual al monolito para esas rutas y se le da un status 410 Gone.

Figura 60

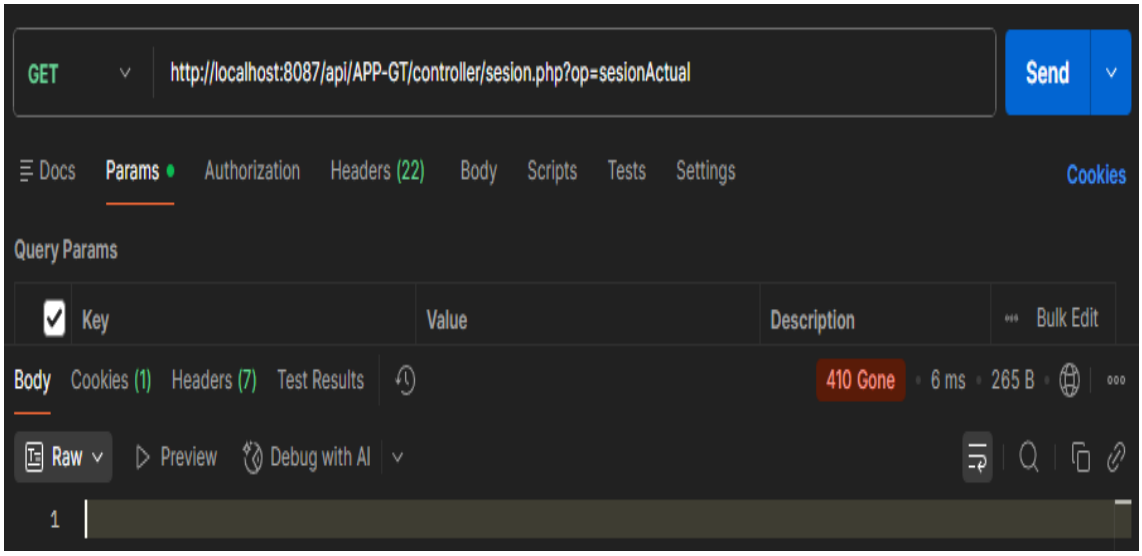
Bloqueo del módulo del Sprint 4



Nota: Se muestra la configuración para el bloqueo de los módulos de sesiones del monolito.

Figura 61

Rutas Bloqueadas del Monolito - Sprint 4



Nota: Se verifique que los endpoints asociados a los módulos de sesiones responden con el estado 410 Gone, confirmando el bloqueo y el retiro de dichas funcionalidades, quedando 5 módulos a migrar.

4.4.6. SPRINT 5 – ACADEMIC SERVICE

Módulos migrados: Facultad (14), Escuela (13), Denominación (15).

Puntuación de priorización: 20/100 **Puerto:** 8086

a) TRANSFORMACIÓN

AcademicService gestiona los datos maestros de la estructura académica institucional: facultades, escuelas profesionales y denominaciones de grados y títulos. Esta consolidación se justifica porque todos pertenecen al mismo Bounded Context (Estructura Académica).

AcademicService agrupa tres controladores REST independientes dentro de un mismo proyecto Spring Boot, cada uno responsable de un subdominio del contexto académico: FacultyController para las facultades, CollegeController para las escuelas profesionales y DenominationController para las denominaciones y subdenominaciones de grados y títulos. Esta separación por controladores dentro de un único microservicio respeta el principio de cohesión sin crear servicios innecesariamente pequeños para dominios que comparten el mismo contexto de negocio.

Se valida que estos datos cambian muy poco, por lo que no representan un cuello de botella ni requieren cambios frecuentes.

Tabla 20

Endpoints implementados — AcademicService

Controlador	Método	Endpoint	Descripción	Respuesta
Faculty	GET	/api/academic/faculty/all	Lista todas las facultades	200
Faculty	GET	/api/academic/faculty/id/{id}	Obtiene facultad por ID	200 / 404
Faculty	POST	/api/academic/faculty/create	Crea una facultad	200
Faculty	PATCH	/api/academic/faculty/update	Actualiza una facultad	200
Faculty	DELETE	/api/academic/faculty/{id}	Elimina una facultad	200

College	GET	/api/academic/college/all	Lista todas las escuelas	200
College	GET	/api/academic/college/id/{id}	Obtiene escuela por ID	200 / 404
College	GET	/api/academic/college/code/{code}	Obtiene escuela por código	200 / 404
College	POST	/api/academic/college/create	Crea una escuela	200
College	PATCH	/api/academic/college/update	Actualiza una escuela	200
College	DELETE	/api/academic/college/{id}	Elimina una escuela	200
Denomination	GET	/api/academic/denomination/all	Lista todas las denominaciones	200
Denomination	GET	/api/academic/denomination/id/{id}	Obtiene denominación por ID	200 / 404
Denomination	POST	/api/academic/denomination/create	Crea una denominación	200
Denomination	PATCH	/api/academic/denomination/update	Actualiza una denominación	200
Denomination	DELETE	/api/academic/denomination/{id}	Elimina una denominación	200
Denomination	GET	/api/academic/denomination/subden/all	Lista todas las subdenominaciones	200
Denomination	GET	/api/academic/denomination/subden/id/{id}	Obtiene subdenominación por ID	200 / 404
Denomination	POST	/api/academic/denomination/subden/create	Crea una subdenominación	200
Denomination	PATCH	/api/academic/denomination/subden/update	Actualiza una subdenominación	200
Denomination	DELETE	/api/academic/denomination/subden/{id}	Elimina una subdenominación	200

Nota: Elaboración Propia.

Pruebas y criterios de aceptación

Las pruebas se realizaron con Postman verificando los endpoints con casos válidos e inválidos.

Tabla 21

Criterios de aceptación — AcademicService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka como ACADEMICSERVICE	Cumple
2	GET /faculty/all retorna la lista completa de facultades en formato JSON	Retorna correctamente todas las facultades registradas	Cumple
3	GET /college/code/{code} permite buscar una escuela por su código institucional	Filtrado por código correcto en todos los casos probados	Cumple
4	GET /denomination/all retorna todas las denominaciones	Retorna la lista con la relación correcta	Cumple
5	Los endpoints REST retornan JSON con la estructura esperada	Validado con colección Postman	Cumple

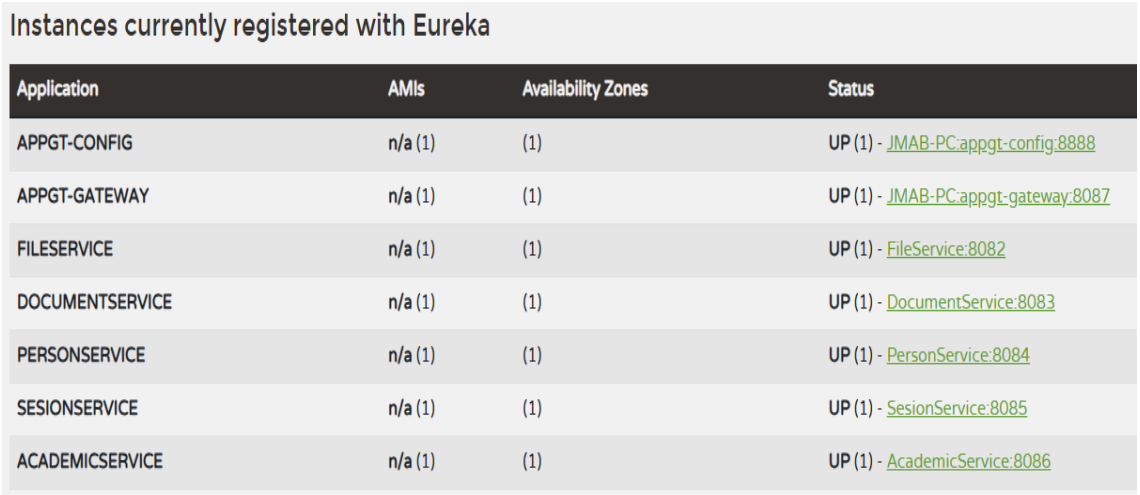
Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 62

Registro de Eureka Server

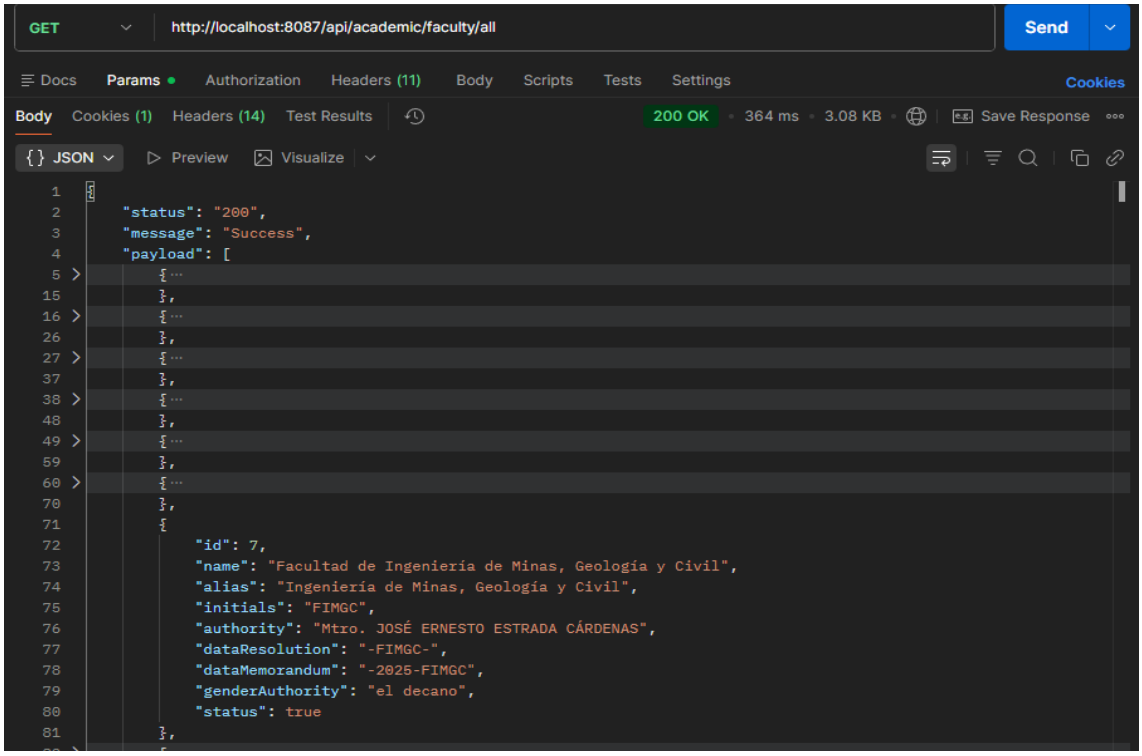


Nota: Se visualiza academic service en la lista de microservicios desplegados.

Criterio de Aceptación Nº 2

Figura 63

Facultades Registradas

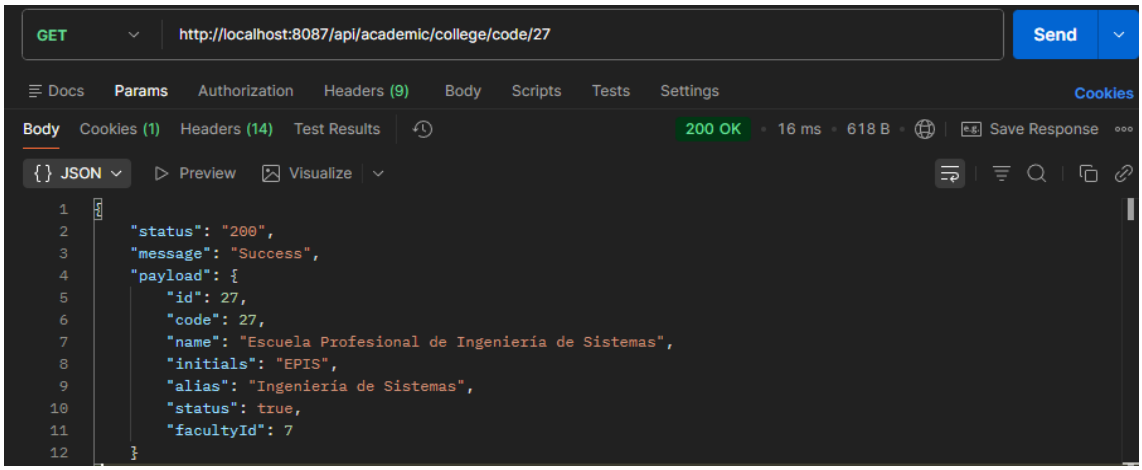


Nota: La imagen muestra las facultades registradas en base de datos.

Criterio de Aceptación Nº 3

Figura 64

Endpoint de consulta por código de escuela

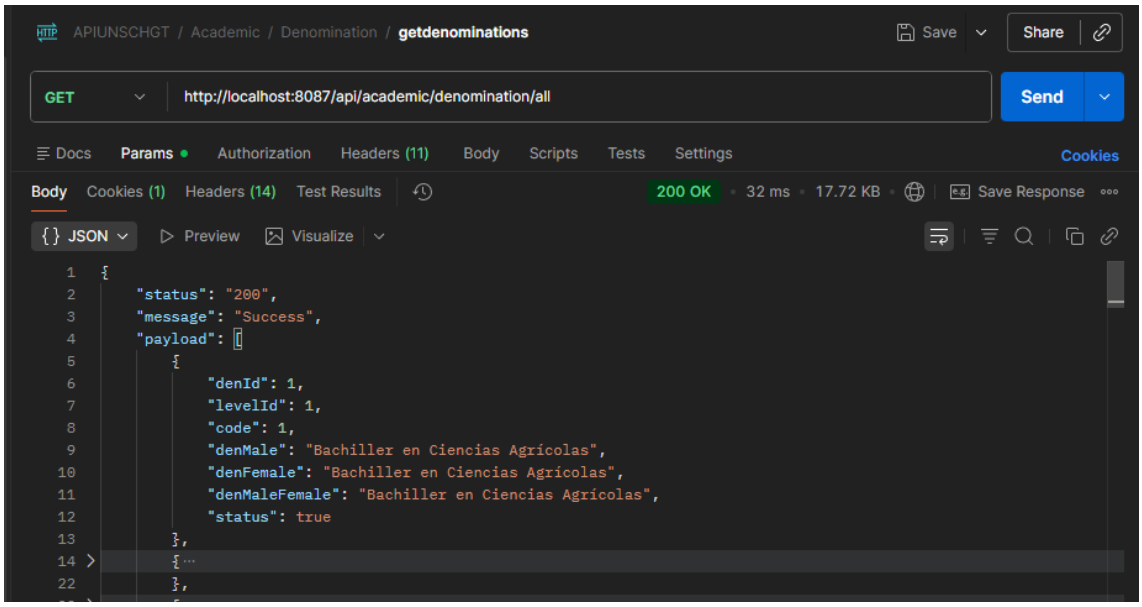


Nota: La imagen muestra la petición de consulta por escuela con parámetro 27 correspondiente a la Escuela Profesional de Ingeniería de Sistemas.

Criterio de Aceptación Nº 4

Figura 65

Endpoint de listado de denominaciones



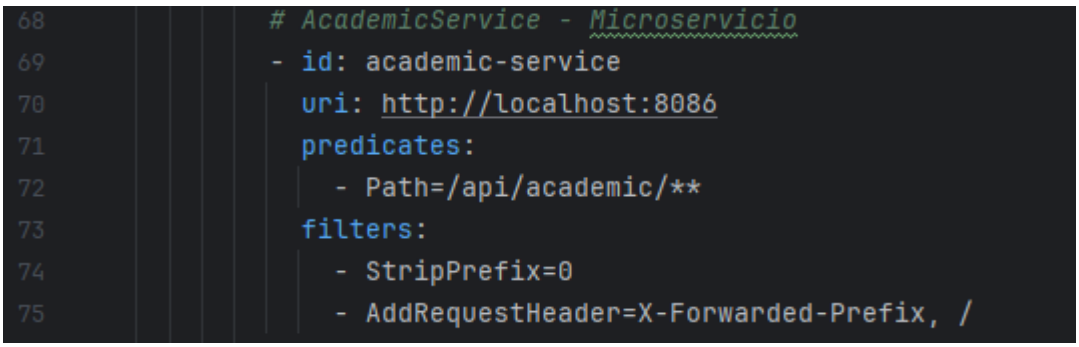
Nota: La imagen muestra la petición de listado de todas las denominaciones asociadas a las escuelas profesionales y sus niveles.

b) **COEXISTENCIA**

Con AcademicService desplegado y validado, se configuró el API Gateway para redirigir todas las peticiones asociadas a los modulos de facultades, escuelas y denominaciones hacia el nuevo microservicio.

Figura 66

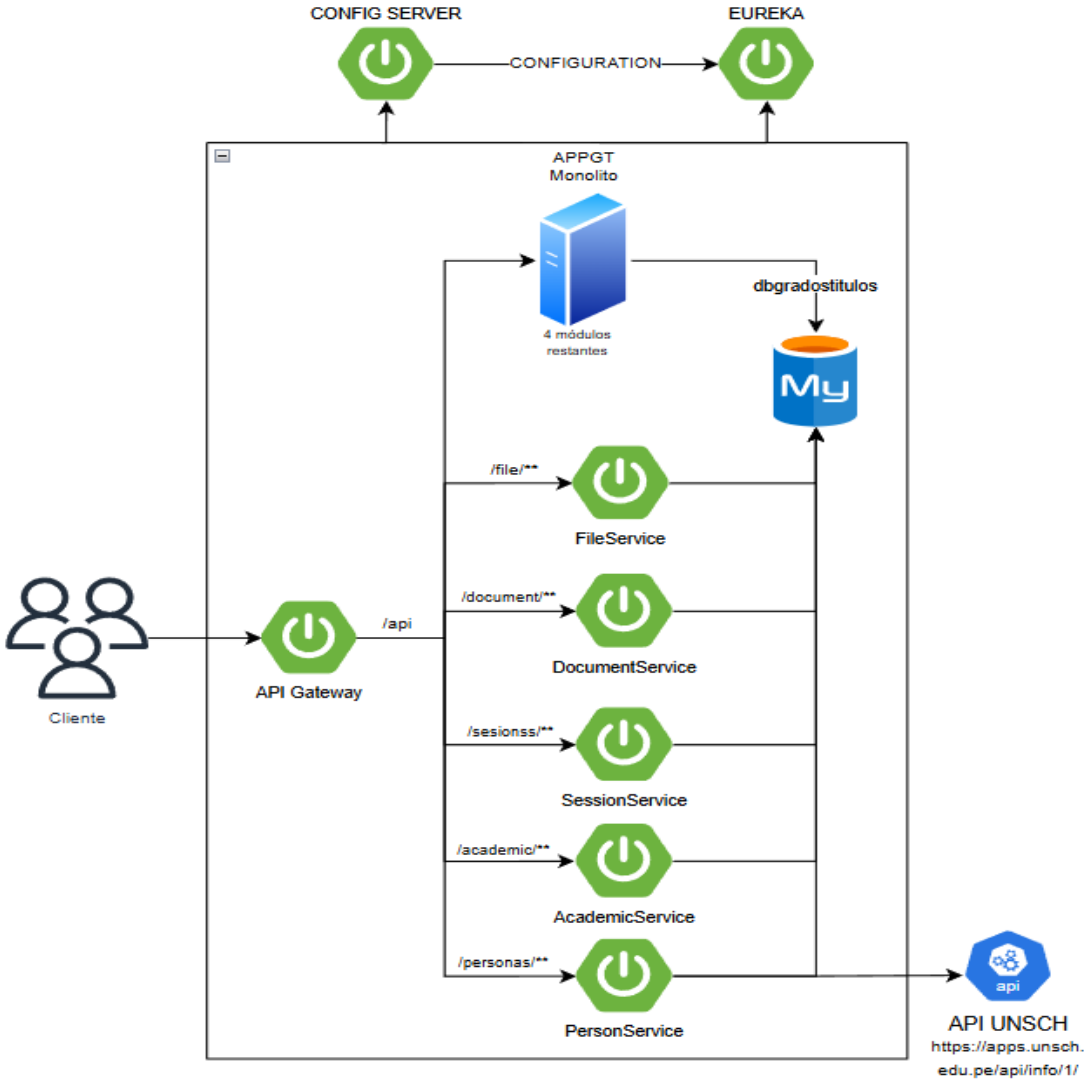
Configuración de rutas del Sprint 5



Nota: Se muestra la configuración de rutas para la coexistencia del monolito con el microservicio AcademicService.

Figura 67

Arquitectura del sistema en el Sprint 5



Nota: Se muestra la separación del módulo de Facultad, Escuela y Denominación del Monolito al microservicio AcademicService.

c) **ELIMINACIÓN**

Verificado el correcto funcionamiento de AcademicService durante el período de coexistencia, se desactivaron los controladores PHP correspondientes a Facultad, Escuela y Denominación del monolito. Las reglas de enrutamiento del Gateway fueron actualizadas para eliminar cualquier referencia residual al monolito para esas rutas y se le da un status 410 Gone.

Figura 68

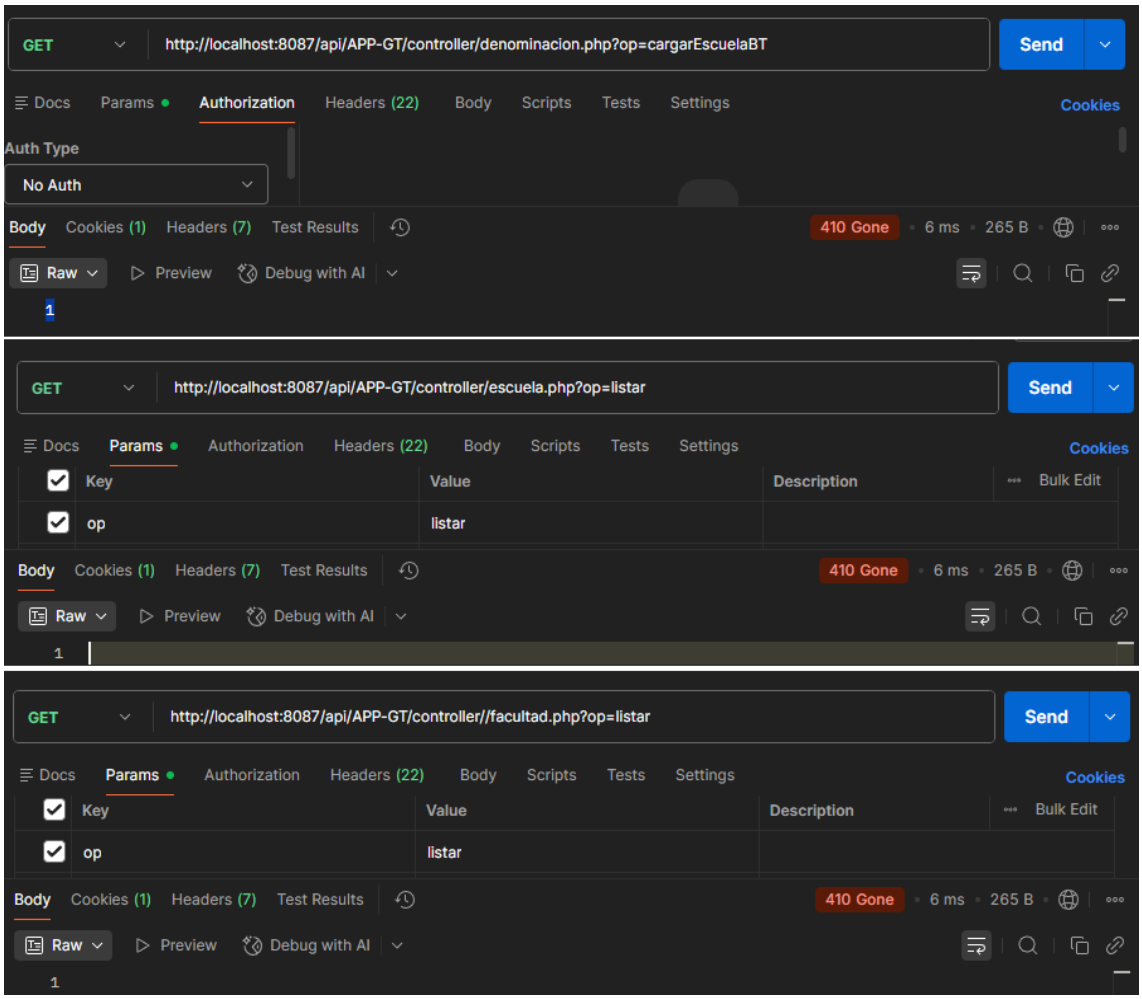
Bloqueo del módulo del Sprint 5



Nota: Se muestra la configuración para el bloqueo de los módulos de facultad, escuela, denominación y subdenominación del monolito.

Figura 69

Rutas Bloqueadas del Monolito - Sprint 5



Nota: Se verifique que los endpoints asociados a los módulos de facultad, escuela y denominaciones responden con el estado 410 Gone, confirmando el bloqueo y el retiro de dichas funcionalidades.

4.4.7. SPRINT 6 – USERS SERVICE

Funcionalidad migrada: Usuarios (16).

Puntuación de priorización: 15/100 **Puerto:** 8089

a) TRANSFORMACIÓN

UserService se incorporó como un nuevo módulo independiente para sustituir el mecanismo tradicional de manejo de sesiones del sistema PHP heredado. Esta transición permitió centralizar el control de acceso y fortalecer la seguridad dentro de la nueva arquitectura mediante un esquema de autenticación y autorización basado en JWT.

Entre sus funcionalidades principales, el servicio implementa el método register para la creación de nuevos usuarios utilizando encriptación. Esto resuelve una vulnerabilidad crítica del monolito anterior, donde los usuarios se registraban manualmente en la base de datos sin un proceso controlado ni seguro.

Tabla 22

Endpoints implementados — UserService

Método	Endpoint	Descripción	Código de respuesta
POST	/api/users/login	Autentica al usuario y genera token JWT con claims (id, nombre, rol)	200 + token
POST	/api/users/loginV2	Autenticación alternativa sin BCrypt para compatibilidad temporal	200 + token
POST	/api/users/register	Registra un nuevo usuario con contraseña encriptada en BCrypt	200 / 404
POST	/api/users/validate	Valida el token JWT del header Authorization y retorna los datos del usuario	200 / 404
GET	/api/users /health	Verifica la salud del microservicio	200 / 500
DELETE	/api/users/{user}	Realiza el logout del usuario	200 / 500

Nota: El endpoint loginV2 fue implementado como mecanismo de compatibilidad durante la fase de coexistencia con el monolito, que no utilizaba BCrypt. Una vez completada la migración completa, este endpoint quedó en desuso.

Pruebas y criterios de aceptación

Tabla 23

Criterios de aceptación — UserService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka como USERSERVICE	Cumple
2	POST /login retorna un token JWT válido con credenciales correctas	Token generado y verificado con las claims correctas (id, fullname, usuario, rol)	Cumple
3	POST /login retorna 404 si el usuario no existe	Respuesta 404 con mensaje "Usuario no existe"	Cumple
4	POST /register encripta la contraseña con BCrypt antes de persistir	Contraseña almacenada en BD con hash BCrypt verificado	Cumple
5	GET /validate retorna los claims del usuario cuando el token es válido	UsuarioDto reconstruido correctamente desde claims JWT	Cumple

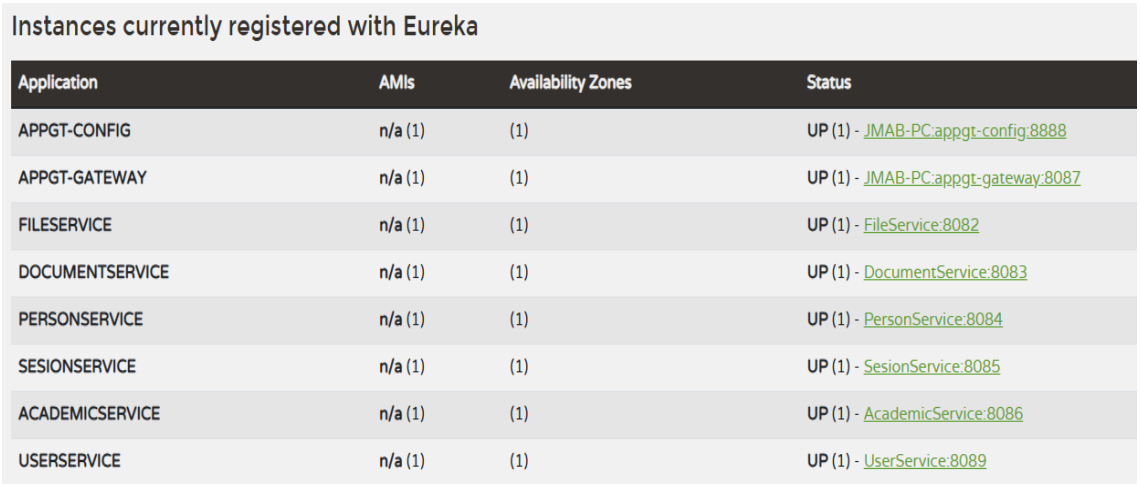
Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 70

Registro de Eureka Server

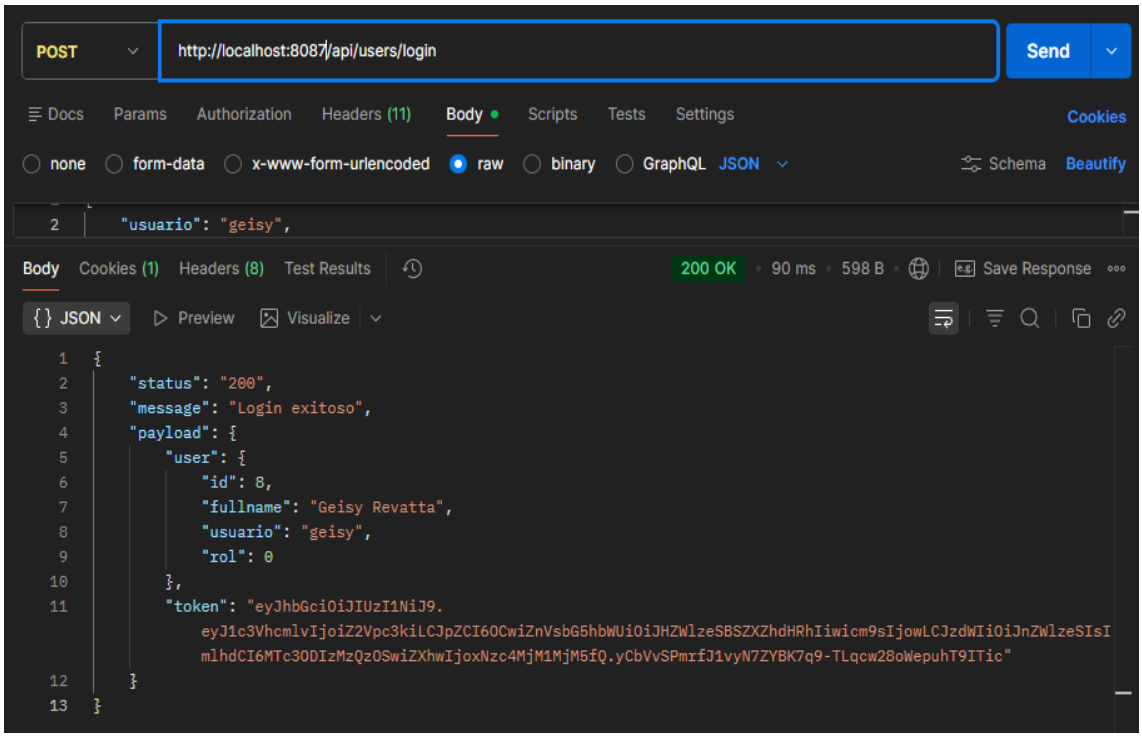


Nota: Se visualiza user service en la lista de microservicios desplegados.

Criterio de Aceptación Nº 2

Figura 71

Servicio Login

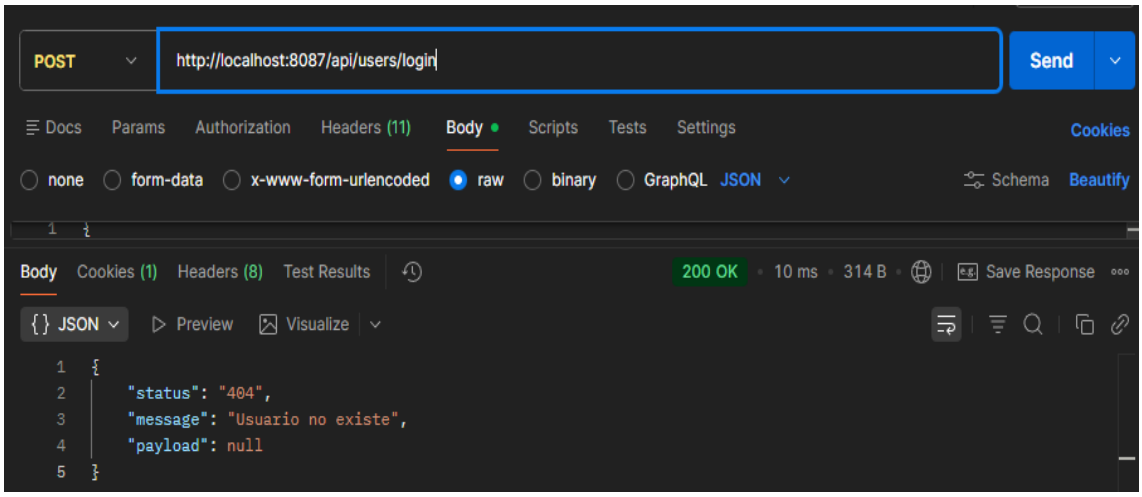


Nota: El servicio login retorna correctamente el JWT y datos del usuario.

Criterio de Aceptación Nº 3

Figura 72

Validación de Usuario Incorrecto

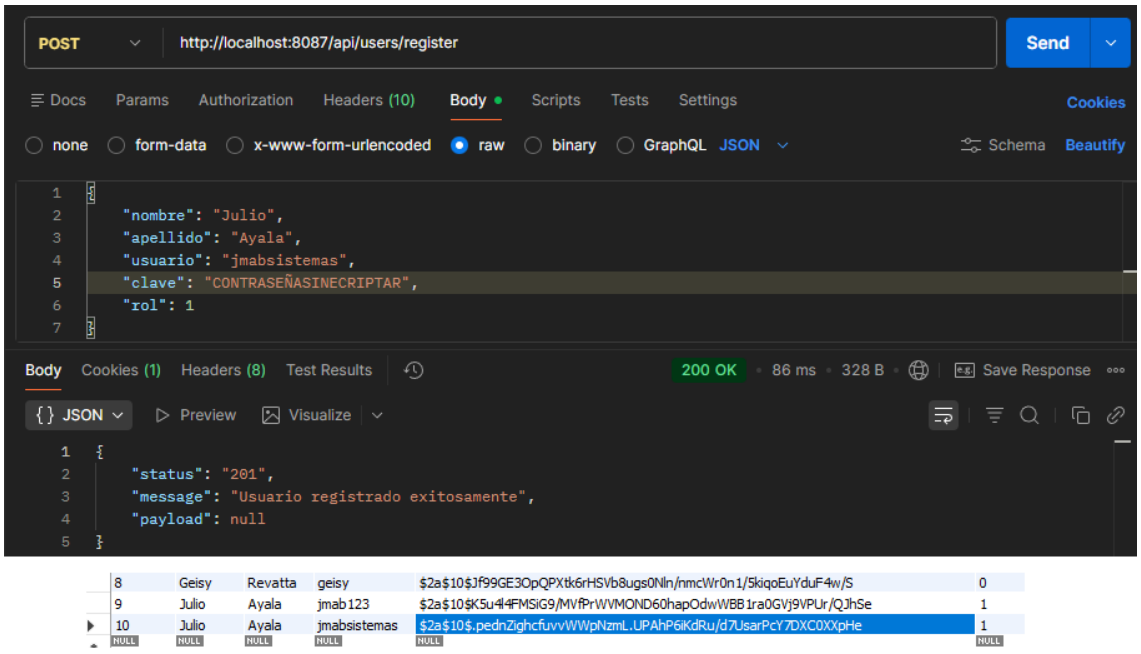


Nota: El endpoint retorna un mensaje de Usuario no existe y status 404.

Criterio de Aceptación Nº 4

Figura 73

Encriptación de contraseña

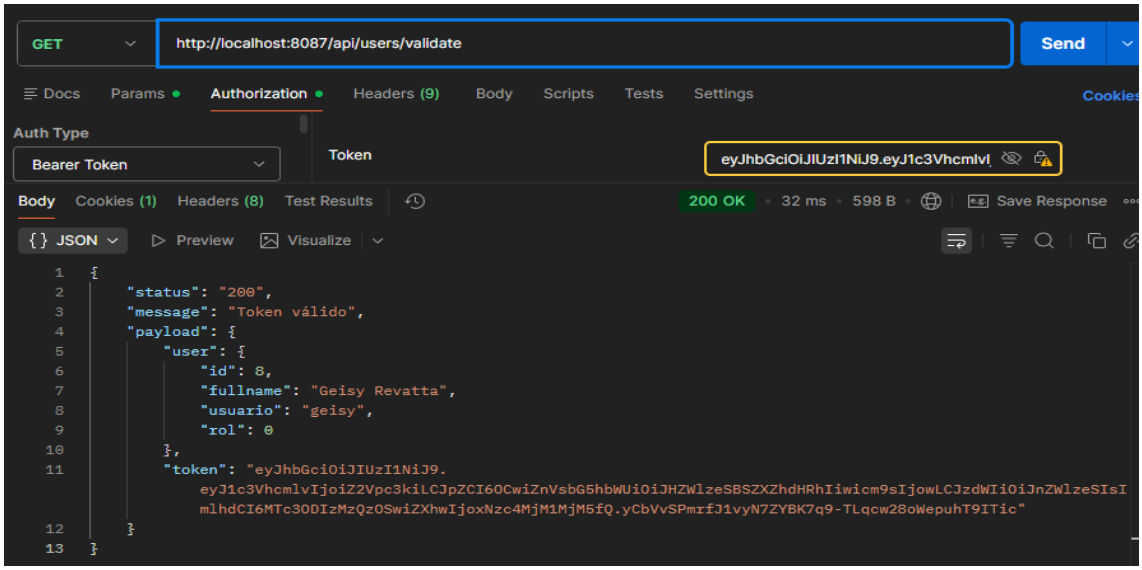


Nota: Se muestra el envio de contraseña sin encriptar pero en base de datos figura encriptado.

Criterio de Aceptación Nº 5:

Figura 74

Validación del Token



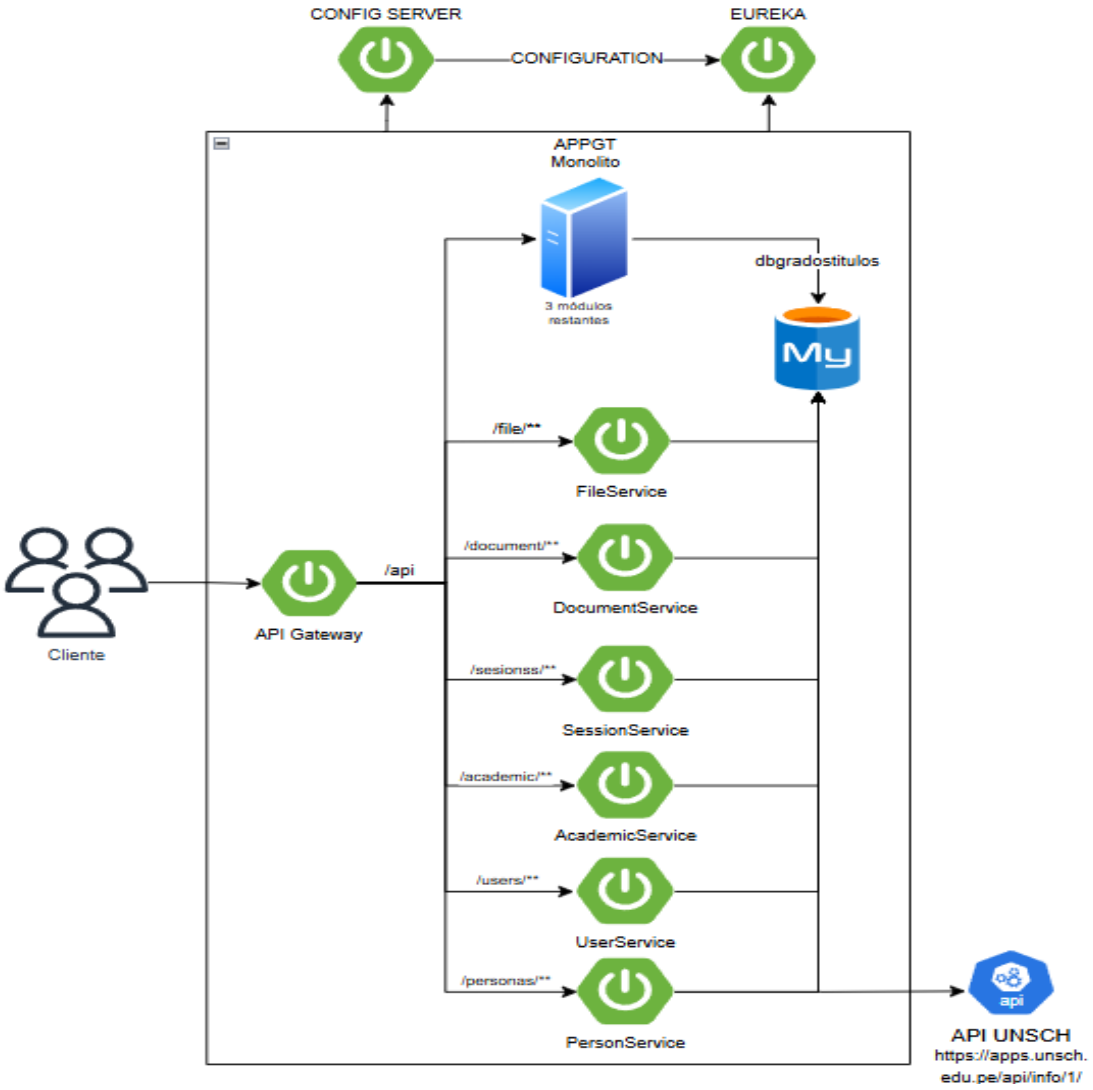
Nota: Se visualiza que el endpoint validate retorna la información del usuario cuando el token es válido.

b) COEXISTENCIA

En este caso, la coexistencia se da a nivel funcional, ya que el microservicio de usuarios no reemplaza un módulo completo del monolito, sino únicamente la funcionalidad de autenticación originalmente implementada en PHP. Durante el período de coexistencia se verificó que la autenticación JWT funcionó correctamente, y que el endpoint loginV2 permitió la compatibilidad temporal con usuarios del sistema que aún no habían migrado sus contraseñas a BCrypt.

Figura 75

Arquitectura del sistema en el Sprint 6

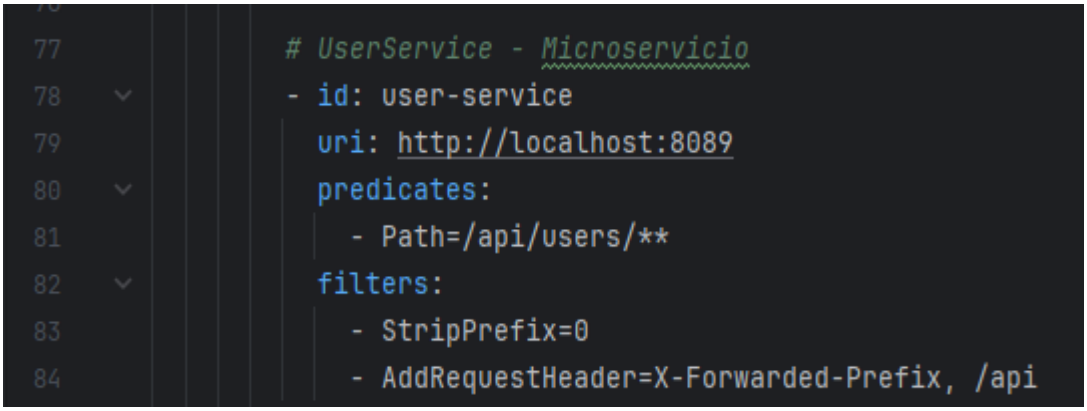


Nota: Se muestra la separación de la funcionalidad de autenticación por sesiones del Monolito al microservicio UserService.

- Configuración de Ruta para UserService en el APIGateway

Figura 76

Configuración de rutas del Sprint 6



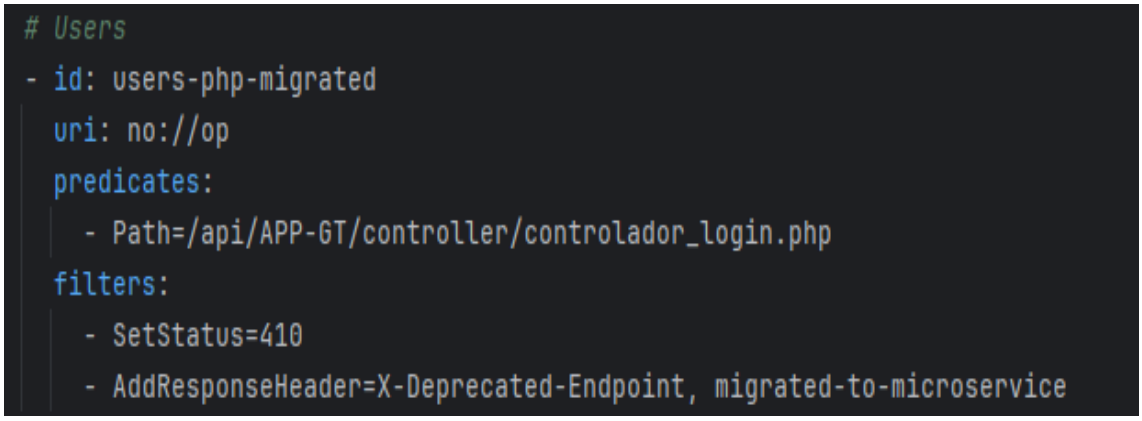
Nota: Se muestra la configuración de ruta para user service en el api gateway.

c) ELIMINACIÓN

Con todos los criterios de aceptación verificados, la funcionalidad de login con sesiones PHP (controlador_login.php) fue retirado del monolito. Se bloqueo la ruta del controlador de login antiguo.

Figura 77

Bloqueo del módulo del Sprint 6



Nota: Se muestra la configuración para el bloqueo del controlador de login original.

4.4.8. SPRINT 7 – DILIGENCESERVICE Y PARAMETERSSERVICE

Funcionalidad migrada: Diligencia (2) y Parámetros.

Puntuación de priorización: 10/100 **Puerto:** 8090 y 8092

d) TRANSFORMACIÓN

DiligenceService implementó la restricción de una sola diligencia por sesión y desarrolló su CRUD básico. Este microservicio fue desarrollado antes que ParametersService dentro del mismo Sprint, ya que ambos obtuvieron el mismo puntaje en la priorización; sin embargo, debido a su dependencia con SesionService, se eligió como el primer desarrollo del Sprint.

ParametersService centralizó los datos de configuración transversales del sistema: tipos de acto académico, niveles, tipos de sesión, órganos universitarios y tipos de generación, los cuales anteriormente se encontraban dispersos en el monolito y eran accedidos mediante consultas JOIN directas desde distintos módulos PHP.

Tabla 24

Endpoints implementados — DiligenceService

Método	Endpoint	Descripción	Código de respuesta
GET	/api/diligence	Obtiene todas las diligencias	200 / 404
GET	/api/diligence/{id}	Obtiene diligencia por ID	200 / 404
GET	/api/diligence/bySession /{sessionId}	Obtiene diligencia de una sesión específica	200 / 404
POST	/api/diligence	Registro (valida máxima 1 por sesión)	200 / 500
PATCH	/api/diligence	Actualiza los datos de una diligencia	200 / 500
DELETE	/api/diligence/{id}	Elimina una diligencia	200 / 500

Nota: Elaboración propia.

Tabla 25

Endpoints implementados — ParametersService

Entidad	Endpoints	Descripción
Nivel	GET, POST, PATCH, DELETE /api/parameters/nivel + /{id}	Gestiona los Niveles de Grados Académicos
Órgano	GET, POST, PATCH, DELETE /api/parameters/organo + /{id}	Gestiona los Órganos Universitarios
Acto Académico	GET, POST, PATCH, DELETE /api/parameters/acto-academico + /{id}	Gestiona los Actos Académicos de Grados y Títulos
Generación Copia	GET, POST, PATCH, DELETE /api/parameters/generacion-copia + /{id}	Gestiona las Generaciones de un Diploma
Tipo Sesión	GET, POST, PATCH, DELETE /api/parameters/sesion-tipo + /{id}	Gestiona los tipos de Sesión disponible

Nota: Elaboración propia.

Pruebas y criterios de aceptación

Tabla 26

Criterios de aceptación — DiligenceService y ParametersService

Nº	Criterio de aceptación	Resultado obtenido	Estado
1	El servicio se registra en el servidor Eureka al iniciar	Visible en el dashboard de Eureka	Cumple
2	GET /api/diligence/bySession/{sessionId}	Retorna la diligencia de la sesión indicada	Cumple
3	GET /api/parameters/nivel /api/parameters/organo	Retorna la lista de niveles y órganos académicos del sistema	Cumple

Nota: Elaboración Propia.

A continuación, se muestran evidencias de los criterios de aceptación:

Criterio de Aceptación Nº 1:

Figura 78

Registro de Eureka Server

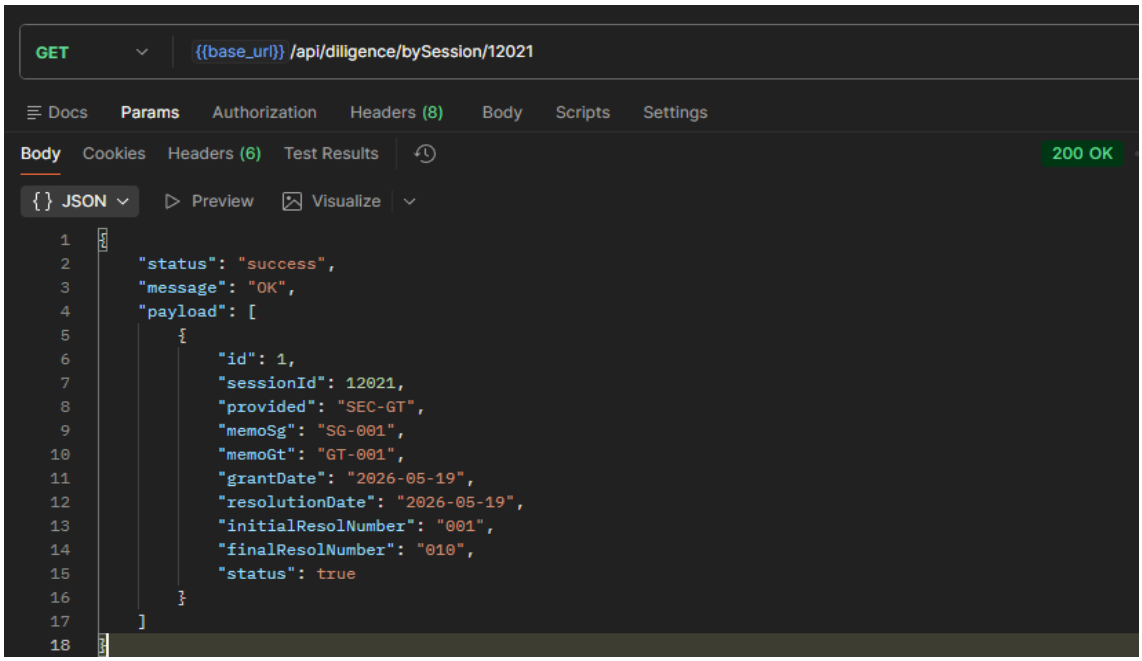


Nota: Muestra DiligenceService y ParametersService *en la lista de servicios desplegados.*

Criterio de Aceptación Nº 2

Figura 79

Servicio Obtener Diligencia por Sesión

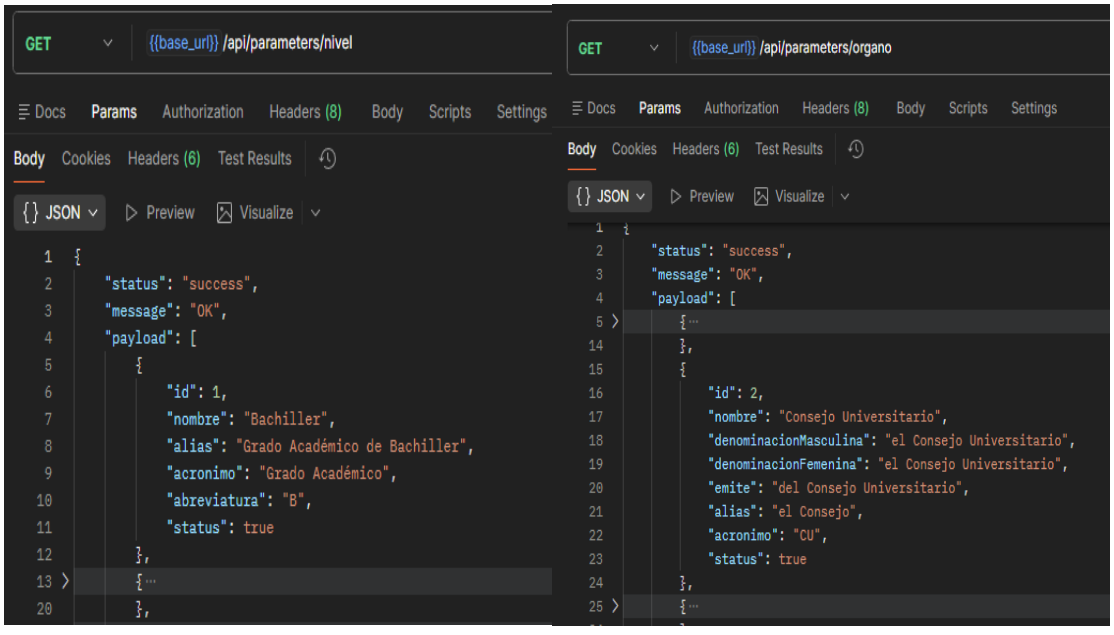


Nota: El servicio retorna correctamente la diligencia de una Sesión Específica.

Criterio de Aceptación Nº 3

Figura 80

Obtener listado de Niveles



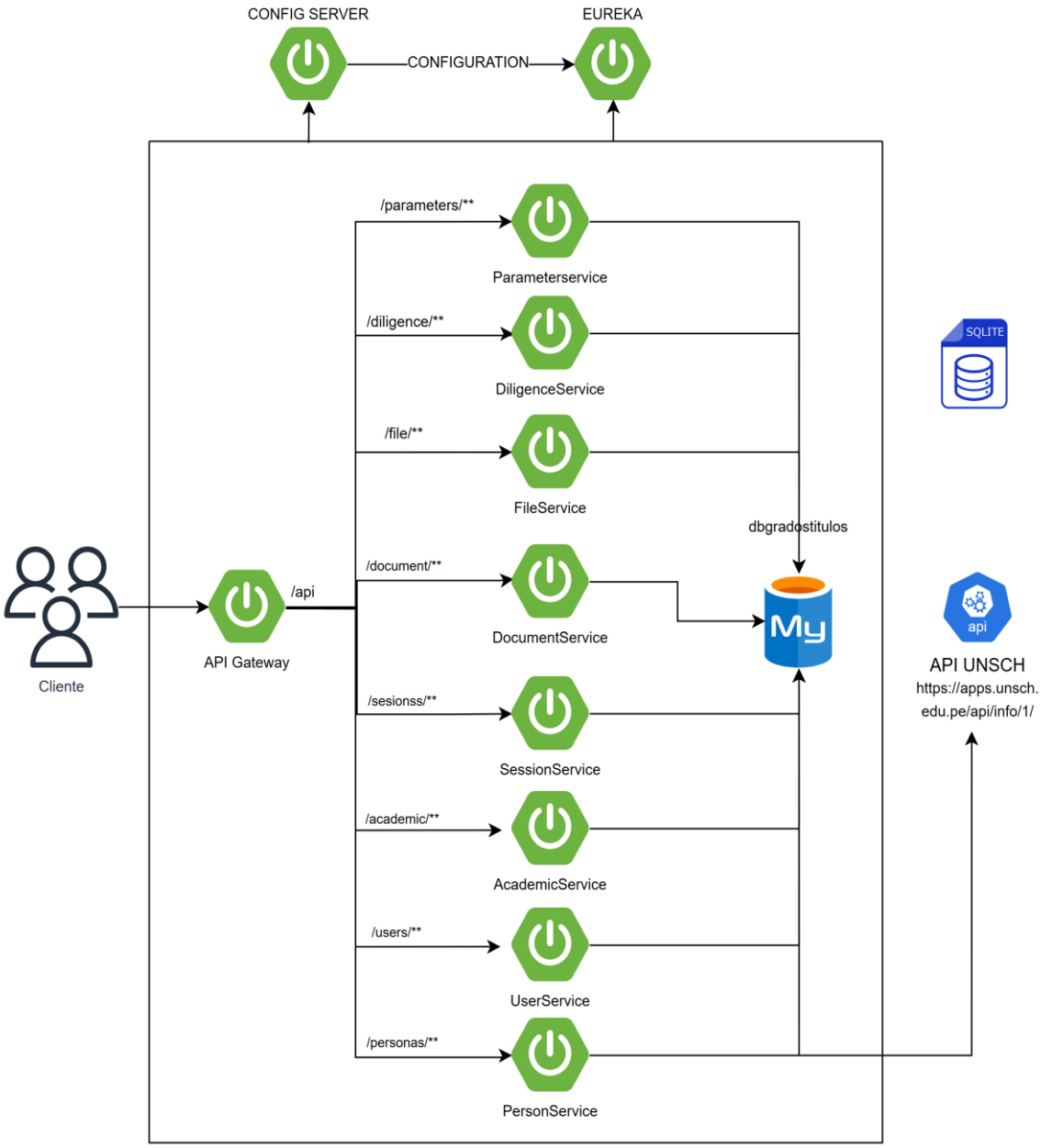
Nota: El endpoint retorna la lista de niveles y órganos disponibles.

e) COEXISTENCIA

DiligenceService y ParametersService son los últimos módulos de negocio del monolito en ser extraídos. A partir de este sprint, todos los módulos funcionales del monolito PHP han sido desplazados como destinos activos del Gateway.

Figura 81

Arquitectura del sistema en el Sprint 7



Nota: Se muestra la inclusión de los microservicios de ParametersService y DiligenceService en la arquitectura.

- Configuración de Ruta para DiligenceService y ParametersService en el APIGateway

Figura 82

Configuración de rutas del Sprint 7

```
# DiligenceService - Microservicio
- id: diligence-service
  uri: http://localhost:8090
  predicates:
    - Path=/api/diligence/**
  filters:
    - StripPrefix=0
    - AddRequestHeader=X-Forwarded-Prefix, /api

# ParametersService - Microservicio
- id: parameters-service
  uri: http://localhost:8091
  predicates:
    - Path=/api/parameters/**
  filters:
    - StripPrefix=0
    - AddRequestHeader=X-Forwarded-Prefix, /api
```

Nota: Se muestra la configuración de ruta en el Api Gateway.

f) ELIMINACIÓN

Con la extracción del Módulo de Diligencia (diligencia.php) el monolito deja de recibir tráfico del frontend, y con este sprint concluye la Fase 1: todos los módulos del monolito han sido extraídos y el monolito PHP ha sido completamente estrangulado como sistema activo.

Figura 83

Bloqueo del módulo del Sprint 7

```
# DiligenceService
- id: diligence-php-migrated
  uri: no://op
  predicates:
    - Path=/api/APP-GT/controller/diligencia.php
  filters:
    - SetStatus=410
    - AddResponseHeader=X-Deprecated-Endpoint, migrated-to-microservice
```

Nota: Se muestra la configuración para el bloqueo del controlador de diligencia.

4.4.9. SPRINT 8 – BASE DE DATOS Y COMUNICACIÓN ENTRE SERVICIOS

Con el sistema funcionando correctamente y sin dependencia del monolito, se ejecutó la Fase 2: la separación de la base de datos y la activación de comunicación HTTP inter servicios. Este sprint transforma la arquitectura transitoria (código separado, base de datos compartida) en la arquitectura objetivo (código separado, BD-por-servicio, comunicación HTTP).

a) TRANSFORMACIÓN

El principio Base de Datos por Servicio nos da la posibilidad de seleccionar el motor de base de datos más adecuado para las características de cada servicio. Esta decisión se toma en esta fase cuando cada servicio recibe su base de datos propia.

SELECCIÓN DE MOTOR DE BASE DE DATOS

Tabla 27

Motor de Base de Datos por Servicio

MySQL - Para servicios transaccionales:		
Servicio	**Base de datos**	**Justificación**
FileService	filedb	Mayor volumen transaccional — expedientes con escrituras concurrentes de múltiples usuarios
DocumentService	filedb (compartida, lectura)	Consultas SQL complejas de generación documental sobre datos de FileService
PersonService	persondb	Conjunto de datos en crecimiento continuo (egresados), integración con API externa UNSCH
SesionService	sessiondb	Escrituras concurrentes durante períodos de registro de expedientes
SQLite - Para servicios de catálogo y baja concurrencia:		
Servicio	**Archivo SQLite**	**Justificación**
AcademicService	academicdb.db	Catálogo de referencia (facultades, escuelas, denominaciones), datos estables, principalmente lectura
UserService	userdb.db	Usuarios del sistema administrativo base pequeña y fija
DiligenceService	diligencedb.db	Una diligencia por sesión tasa de escritura mínima
ParametersService	parametersdb.db	Parámetros del sistema (niveles, órganos, actos académicos), datos casi estáticos

Nota: Se muestra la selección de motor de base de datos por servicio.

Los datos de cada dominio son migrados desde la base de datos compartida hacia las bases de datos individuales. Los servicios con SQLite reciben sus datos mediante archivos data.sql.

DocumentService representó el mayor problema en la separación de bases de datos, debido a que sus consultas SQL nativas realizan múltiples JOINs entre entidades distribuidas en diferentes bases de datos en la arquitectura de la Fase 2. Para resolver este problema, se implementaron VIEWs MySQL en filedb que referencian tablas ubicadas en otras bases de datos, como persondb y sessiondb. Asimismo, las tablas de catálogo, como facultad, escuela, nivel y acto_academico, fueron replicadas físicamente en filedb mediante scripts de sincronización. Esta estrategia permitió mantener las consultas nativas de DocumentService sin modificar el código existente, siempre que los servicios permanezcan en el mismo servidor MySQL.

COMUNICACIÓN HTTP ENTRE SERVICIOS

Tabla 28

Mapa completo de dependencias HTTP

Servicio	Llama a	Endpoint	Propósito
FileService	PersonService	GET /api/personas/{id}	Datos del egresado al consultar expedientes
FileService	SesionService	GET /api/session/{id}	Datos de la sesión al consultar expedientes
FileService	ParametersService	GET /api/parameters/ acto-academico/{id}	Datos de la modalidad del acto académico
FileService	AcademicService	GET /api/academic/ denomination/id/{id}	Denominación del grado o título
FileService	AcademicService	GET /api/academic/ denomination/subden /id/{id}	Subdenominación del grado o título
DiligenceService	SesionService	GET /api/session/{id}	Verificar existencia de la sesión al registrar una diligencia
SesionService	ParametersService	GET /api/parameters/ organo/{id}	Nombre del órgano que convocó la sesión
SesionService	FileService	GET /api/file/expediente/ count/{sessionId}	Verificar expedientes antes de eliminar una sesión

Nota: Se muestra el mapa de dependencias entre servicios.

En la arquitectura en esta Fase 2, las dependencias de datos entre servicios se resuelven mediante llamadas HTTP REST.

b) COEXISTENCIA

Los microservicios operan en un estado transitorio: algunos ya apuntan a sus bases de datos individuales mientras otros aún leen la base de datos compartida. Este período de coexistencia entre la arquitectura transitoria y la arquitectura objetivo se gestiona servicio a servicio, preservando la continuidad operativa en cada paso.

Figura 84

Configuración de base de datos propia - FileService

```yaml
datasource:
  url: jdbc:mysql://localhost:3306/filedb
  username:
  password:
  driver-class-name: com.mysql.cj.jdbc.Driver
jpa:
  hibernate:
    ddl-auto: none
  database: mysql
  database-platform: org.hibernate.dialect.MySQL8Dialect
```

Nota: Se muestra la configuración de base de datos propia en FileService.

Figura 85

Configuración de rutas para conexión HTTP entre Servicios

```yaml
services:
  person:
    url: http://localhost:8084
  sesion:
    url: http://localhost:8085
  parameters:
    url: http://localhost:8091
  academic:
    url: http://localhost:8086
```

Nota: Se muestra la configuración de rutas para la comunicación entre Servicios en application.yml de FileService.

c) ELIMINACIÓN

La base de datos compartida deja de ser la fuente centralizada del sistema. Cada microservicio opera sobre su propia base de datos aislada y la comunicación entre servicios se realiza exclusivamente mediante HTTP REST. Con este sprint concluye la Fase 2 del proceso de migración. Se procede a eliminar el monolito definitivamente.

Figura 86

Eliminación del monolito

```
# Monolito
- id: monolito-php-deleted
  uri: no://op
  predicates:
    - Path=/api/APP-GT/controller/**
  filters:
    - SetStatus=410
    - AddResponseHeader=X-Deprecated-Endpoint, migrated-to-microservice
```

Nota: Se configura el bloqueo y desconexión del monolito definitivamente.

4.4.10. DOCKERIZACIÓN

Con la arquitectura objetivo completamente operativa (código desacoplado, base de datos por servicio y comunicación HTTP entre servicios), se ejecutó la dockerización del sistema como etapa de despliegue del entorno de producción. Esta etapa no forma parte del proceso de extracción definido por el patrón Strangler Fig, sino que corresponde a la implementación del entorno de ejecución del sistema migrado. El objetivo de este paso es garantizar que el ecosistema de microservicios pueda desplegarse de forma reproducible y que todos sus componentes funcionen correctamente de manera integrada.

Se crearon Dockerfiles para los once componentes y se configuró docker-compose.yml para orquestar la red completa:

- **Red interna:** appgt-net (bridge) con comunicación entre contenedores por nombre de servicio.

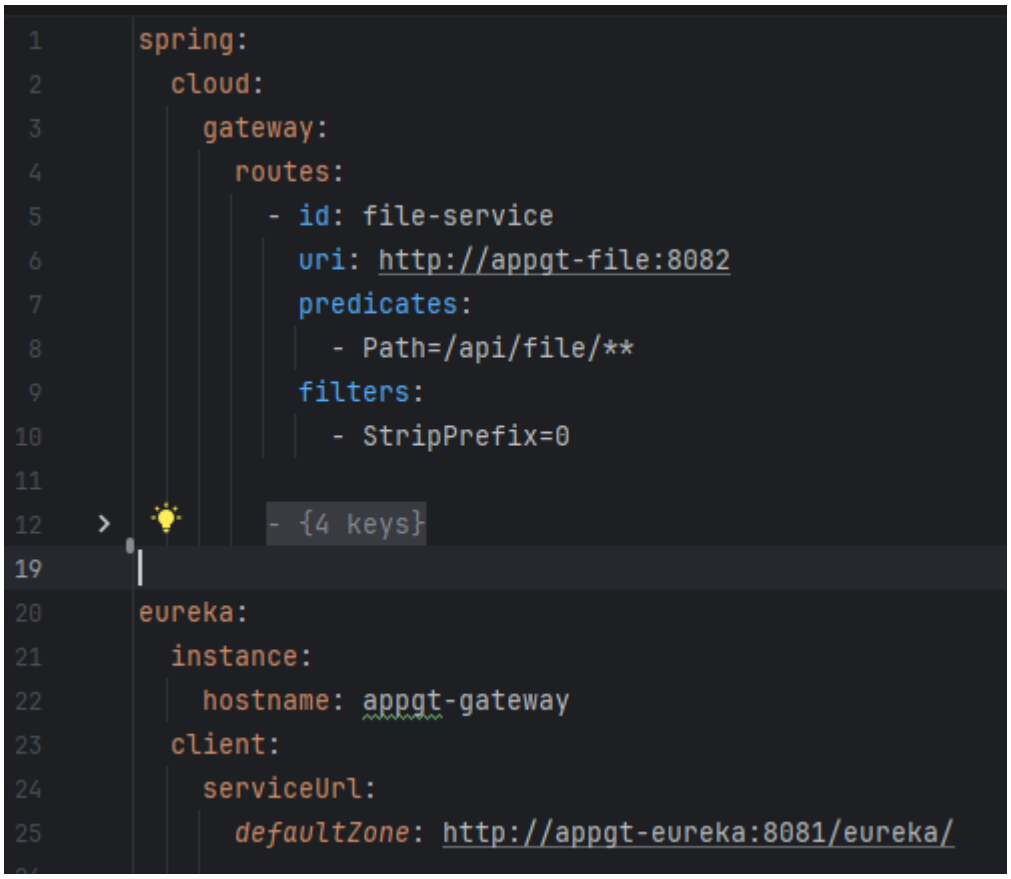
- **Healthcheck en Eureka** para garantizar el orden de inicio correcto.
- **Perfil docker del Gateway** (application-docker.yml) para URIs de enrutamiento con nombres de contenedores.
- **Volúmenes Docker para SQLite**: Los cuatro servicios con SQLite montan volúmenes para garantizar la persistencia de los archivos .db entre reinicios del contenedor.

Número de contenedores: 11 en total: 3 de infraestructura (Eureka, Config, Gateway) y 8 microservicios de negocio.

La coexistencia ya no es entre monolito y microservicios, sino entre los once contenedores del ecosistema nuevo.

Figura 87

Configuración de application-docker.yml del API Gateway



Nota: Se muestra la configuración enrutamiento del API Gateway.

Figura 88

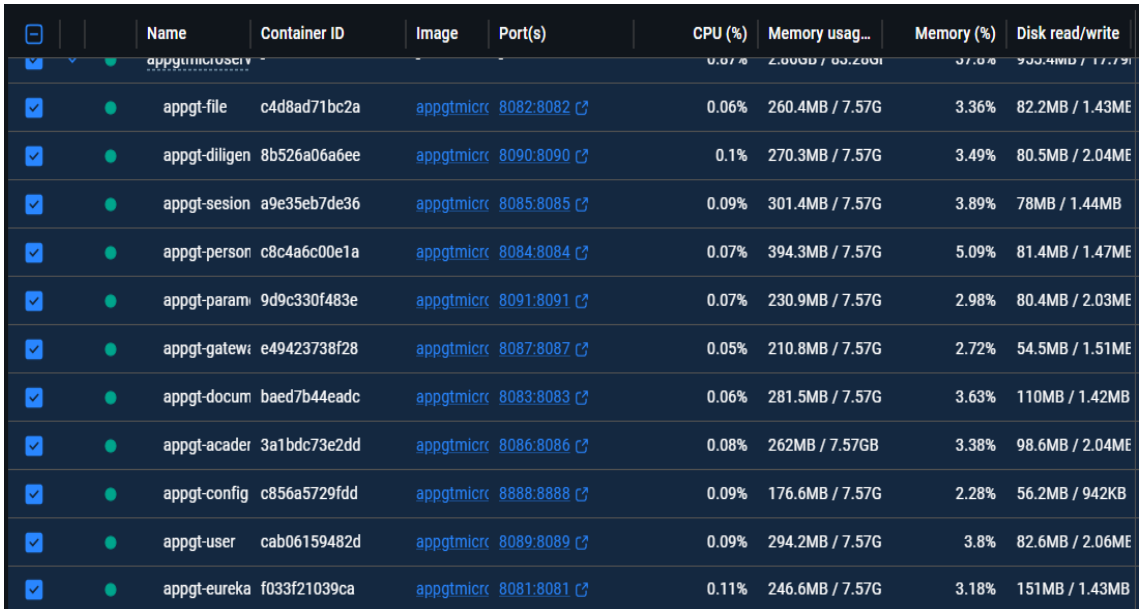
Configuración de Docker-compose.yml



Nota: Se muestra el contenido de docker-compose.yml creado para deploy en Docker.

Figura 89

Deploy de microservicios



Nota: Se muestra los microservicios ejecutándose a través de Docker Desktop.

Figura 90

Registro final en Eureka

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APPGT-CONFIG	n/a (1)	(1)	UP (1) - c856a5729fdd:appgt-config:8888
APPGT-GATEWAY	n/a (1)	(1)	UP (1) - e49423738f28:appgt-gateway:8087
FILESERVICE	n/a (1)	(1)	UP (1) - FileService:8082
ACADEMICSERVICE	n/a (1)	(1)	UP (1) - 3a1bdc73e2dd:academicservice:8086
DILIGENCESERVICE	n/a (1)	(1)	UP (1) - 8b526a06a6ee:diligenceservice:8090
DOCUMENTSERVICE	n/a (1)	(1)	UP (1) - baed7b44eadc:documentservice:8083
PARAMETERSSERVICE	n/a (1)	(1)	UP (1) - 9d9c330f483e:parametersservice:8091
PERSONSERVICE	n/a (1)	(1)	UP (1) - c8c4a6c00e1a:personservice:8084
SESIONSERVICE	n/a (1)	(1)	UP (1) - a9e35eb7de36:sesionservice:8085
USERSERVICE	n/a (1)	(1)	UP (1) - cab06159482d:userservice:8089

Nota: Se muestra el registro final de Eureka Server.

Una vez finalizado el proceso de transformación, coexistencia y eliminación de los módulos del Monolito y desplegados los microservicios se concluye el proceso de migración y despliegue del sistema.

4.5. RESULTADO FINAL DE LA MIGRACIÓN

4.5.1. ESTADO FINAL DEL SISTEMA

Al concluir el proceso de migración, el sistema AppGT opera íntegramente sobre la nueva arquitectura de microservicios. La tabla siguiente resume el estado final de cada componente:

Tabla 29

Resumen de microservicios implementados al finalizar la migración

Microservicio	Módulos reemplazados	Puerto	Endpoint	Base de Datos
Appgt-eureka	-	8081	Dashboard	-
Appgt-config	-	8888	Config API	-
Appgt-gateway	-	8087	Router + JWT + CORS	-
FileService	Bachiller, Título, Maestría, Doctorado, 2da Especialidad, Reportes	8082	11	MySQL: filedb
DocumentService	Lista CU, Resoluciones, Diplomas	8083	6	MySQL: filedb (Lectura)
PersonService	Persona	8084	7	MySQL: persondb
SesionService	Sesión Universitaria	8085	8	MySQL: sessiondb
AcademicService	Facultad, Escuela, Denominación	8086	21	SQLite: academicdb.db
UserService	Funcionalidad Login y Logout	8089	6	SQLite: userdb.db
DiligenceService	Diligencia	8090	6	SQLite: diligencedb.db
ParametersService	Módulo Nuevo	8092	20	SQLite: parametersdb.db

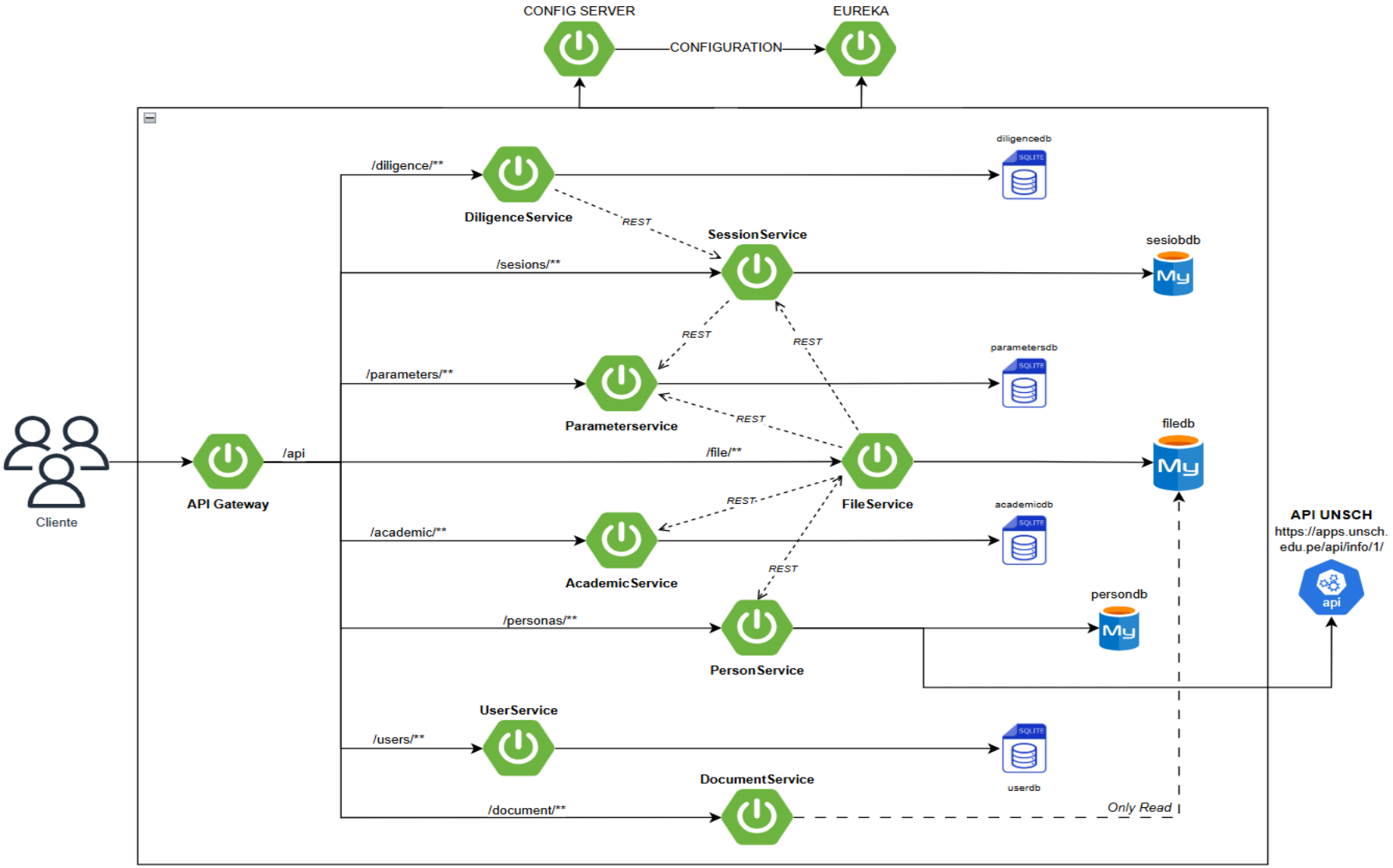
Nota: Elaboración Propia.

Arquitectura final: Todo el tráfico del frontend entra por el Gateway en el puerto 8087. El Gateway valida el JWT, inyecta las cabeceras necesarias y enruta cada solicitud al microservicio correspondiente. La comunicación entre servicios se resuelve mediante llamadas HTTP con API REST. Cada microservicio se conecta a su base de datos propia.

A continuación, la figura siguiente presenta la arquitectura del sistema, incluyendo las comunicaciones REST y las bases de datos:

Figura 91

Arquitectura del sistema post – migración



Nota: Se muestra la arquitectura del sistema al finalizar la migración y las fases del patrón strangler fig.

4.5.2. TRAZABILIDAD DEL PATRÓN STRANGLER FIG

La trazabilidad completa del proceso permite verificar que las tres fases del patrón Strangler Fig definidas por (AWS, 2023) fueron aplicadas de forma consistente en cada uno de los sprints ejecutados.

Tabla 30

Trazabilidad del patrón Strangler Fig

Sprint	Módulo extraído (X1)	Coexistencia Fase 1 (X2)	Impacto (X3)	Acción en Fase 2
Pre-migración	—	100% del tráfico en monolito	Gateway instalado	—
Sprint 1	FileService	Comparte base de datos con monolito	expediente.php inactivo	filedb creada, HTTP clients PersonService/ SesionService/ AcademicService activados
Sprint 2	DocumentService	Comparte base de datos; JOINs SQL íntegros	Generación.php inactivo	VIEWs MySQL creadas en filedb para persona y sesion
Sprint 3	PersonService	Comparte tabla persona	persona.php inactivo	persondb creada, datos migrados
Sprint 4	SesionService	Comparte tabla sesion	sesion.php inactivo	sessiondb creada, datos migrados
Sprint 5	AcademicService	Comparte tablas de catálogo	4 módulos PHP de catálogo inactivos	academicdb.db SQLite con semilla institucional
Sprint 6	UserService	Comparte tabla usuarios	controlador_login.php inactivo	userdb.db SQLite con BCrypt
Sprint 7	DiligenceService y ParametersService	Últimos módulos	Monolito completamente estrangulado	diligencedb.db, parametersdb.db SQLite
Sprint 8	Fase 2: Separación de BDs	BD compartida a BD individual por servicio; HTTP clients activados	Base de datos compartida eliminada.	—
Docker	—	11 contenedores en appgt-net	Monolito PHP solo como referencia histórica	—

Nota: La matriz muestra la trazabilidad del patrón Strangler Fig en sus 3 variables X1: Transformación, X2: Coexistencia y X3: Eliminación, y en las 2 fases de la migración.

4.5.3. COMPARATIVA GLOBAL DEL SISTEMA

Tabla 31

Comparativa entre el sistema monolítico original y el sistema de microservicios resultante

Dimensión	Sistema Monolítico	Sistema de Microservicios
Tecnología backend	PHP 7.4 sin framework	Java 17 + Spring Boot 3.3.4
Tecnología frontend	Vistas PHP renderizadas en servidor	Angular SPA desacoplada.
Patrón arquitectónico	MVC monolítico	Microservicios con API Gateway
Despliegue	Todo el sistema en conjunto	Independiente por microservicio
Escalabilidad	Global: toda la aplicación	Por servicio según demanda
Mantenimiento	Cambio en un módulo puede afectar a otros.	Cambios aislados por dominio
Acoplamiento	Alto – SQL Joins directos entre dominios	Bajo – BD propia; comunicación por HTTP REST
Motor de BD	MySQL único compartido	MySQL para transaccional; SQLite para catálogos
API	Rutas no REST	API REST con verbos HTTP
Autenticación	Sesiones PHP	JSON Web Token (JWT)
Tolerancia a fallos	Fallo total ante error crítico	Fallo parcial, otros servicios continúan
Dockerización	No aplica	Docker Compose con 11 contenedores

Nota: Elaboración propia.

4.5.4. REGISTRO DE TIEMPOS DE RESPUESTA

Con carácter complementario, se registraron los tiempos de respuesta de las operaciones con funcionalidad equivalente en ambos sistemas. Este registro no constituye un objetivo de la investigación sino una caracterización descriptiva del comportamiento observado.

La selección respondió a un único criterio: que la operación existiera en ambos sistemas, fuera ejecutable con los mismos parámetros y devolviera la misma salida. Quedaron excluidas la generación masiva de documentos y la búsqueda por DNI vía API institucional, por ser mejoras sin equivalente en el monolito, y las operaciones de UserService y ParametersService, por no contar con módulo equivalente o por sustituirse el mecanismo de autenticación.

Dado que al término de la migración el monolito ya no estaba operativo, sus tiempos se obtuvieron en un entorno de reproducción de su estado previo, sobre el mismo equipo, con el mismo código fuente y volumen de datos. Las mediciones se realizaron con Postman en condiciones equivalentes de red y carga. Las capturas se presentan en el Anexo L.

El comportamiento no es uniforme. En operaciones sobre expedientes y generación documental, donde el monolito ejecutaba consultas con múltiples JOIN o procesaba documentos individualmente, el sistema migrado registró tiempos menores. En consultas simples sobre una única tabla, los tiempos fueron iguales o ligeramente superiores, resultado consistente con la arquitectura adoptada: el enrutamiento por el API Gateway y la serialización de respuesta introducen un sobrecosto que no se compensa cuando el costo de consulta es bajo. Los registros corresponden a un entorno local sin concurrencia y no permiten afirmar que la migración mejore el desempeño de forma generalizada; su efecto depende de la naturaleza de cada operación.

Tabla 32

Comparación de Tiempos de Respuesta

Nro	Servicio	Funcionalidad	M1	M2	M3	M4	M5	Mediana	S1	S2	S3	S4	S5	Mediana	Variación
1	FileService	Listar Expedientes	1198	1224	1240	1263	1298	**1240**	386	393	397	403	412	**397**	-68,0%
2	FileService	Registrar Expediente	1556	1589	1610	1639	1685	**1610**	406	414	418	424	434	**418**	-74,0%
3	FileService	Listar Expedientes por Nivel	1812	1850	1875	1909	1963	**1875**	497	506	512	520	533	**512**	-72,7%
4	PersonService	Listar Personas	192	196	198	201	206	**198**	218	222	224	227	232	**224**	13,1%
5	PersonService	Buscar Persona por Id	126	130	132	135	140	**132**	181	184	186	188	193	**186**	40,9%
6	SesionService	Sesión Actual	136	140	142	145	150	**142**	147	150	151	153	156	**151**	6,3%
7	DocumentService	Generar Documento — Lista CU	3304	3374	3420	3483	3582	**3420**	899	916	926	940	963	**926**	-72,9%
8	DocumentService	Generar Documento — Resolución	2589	2644	2680	2730	2807	**2680**	720	734	742	754	772	**742**	-72,3%
9	DocumentService	Generar Documento — Diploma	2333	2383	2415	2460	2529	**2415**	668	680	688	699	716	**688**	-71,5%

Nota: La tabla compara los tiempos de respuesta del monolito y el microservicio, evidenciando mejoras en la mayoría de los endpoints, excepto en Listar Personas, Buscar Persona por Id y Sesión Actual, cuyos tiempos fueron menores o equivalentes en el monolito por tratarse de consultas simples sobre una sola tabla.

4.6. DISCUSIÓN

Sobre la efectividad del patrón Strangler Fig en el caso de estudio

Los resultados obtenidos permiten afirmar que el patrón Strangler Fig fue efectivo para la migración del sistema de Grados y Títulos de la UNSCH. La migración de dieciséis módulos del monolito, junto con la incorporación de nuevos servicios como ParametersService, se realizó de manera incremental en ocho sprints, sin requerir una reescritura total simultánea ni generar interrupciones registradas su funcionamiento.

Estos resultados coinciden con los planteamientos de Ma et al. (2022) y Li et al. (2020), quienes sostienen que las estrategias de migración incremental reducen el riesgo operativo frente a enfoques de reemplazo total. En esta investigación, la efectividad del patrón se evidenció principalmente en la reducción del riesgo durante el proceso de migración. Al dividir el proceso en sprints independientes, los errores detectados pudieron corregirse de forma aislada, sin afectar a los microservicios ya estabilizados.

Asimismo, la validación incremental permitió que cada sprint entregara un microservicio funcional y completamente probado antes de continuar con el siguiente módulo. Los criterios de aceptación definidos aseguraron que cada componente cumpliera sus requisitos funcionales antes de desactivar el módulo equivalente del monolito.

Este hallazgo es consistente con el reportado por Thamma et al. (2025), quienes aplicaron el patrón Strangler Fig al Disabled Person Card System empleando una arquitectura equivalente: API Gateway, Strangler Facade y microservicios en coexistencia temporal con el sistema legado y concluyeron que el enfoque redujo los riesgos asociados a la transformación arquitectónica y preservó la continuidad operativa. La diferencia sustantiva entre ambos estudios reside en el procedimiento de delimitación de servicios: mientras Thamma et al. (2025) definieron los dominios mediante un proceso

formal de Domain-Driven Design, en la presente investigación los bounded contexts se identificaron de manera pragmática, a partir de la estructura de las tablas de la base de datos y de las dependencias de código del monolito.

En el ámbito latinoamericano, los resultados obtenidos difieren de los reportados por Cruz et al. (2024) en la migración del sistema catastral ServerBox del Municipio de Puebla y por Geovany & Averos (2025) en el rediseño del sistema de administración de informes de una empresa de telefonía en Ecuador. Ambos trabajos alcanzaron la arquitectura de microservicios mediante descomposición vertical y rediseño arquitectónico respectivamente, sin emplear una estrategia de extracción incremental con coexistencia controlada del sistema legado. La comparación resulta pertinente porque evidencia que la elección del patrón Strangler Fig en el presente caso no respondió únicamente a una preferencia metodológica, sino a una restricción operativa concreta: el Sistema de Grados y Títulos soporta un proceso institucional crítico que no admitía interrupciones prolongadas, condición que desaconsejaba tanto la reescritura total como el rediseño simultáneo del sistema.

Sobre el rol del API Gateway y la coexistencia durante la migración

El API Gateway fue el componente que permitió materializar técnicamente el patrón Strangler Fig. Su incorporación desde la etapa de pre-migración hizo posible la coexistencia entre el monolito PHP y los nuevos microservicios Java, evitando agregar configuraciones adicionales en el frontend para identificar la ubicación de cada servicio.

Además del enrutamiento de solicitudes, el Gateway centralizó funcionalidades de seguridad y comunicación, como la validación de JWT, la gestión de CORS y la inyección de API Key para autenticación interna entre servicios. Esto permitió desacoplar progresivamente responsabilidades que en el monolito se encontraban distribuidas o implementadas de manera inconsistente.

La coexistencia entre ambos entornos también mostró dificultades propias de integrar tecnologías diferentes, especialmente durante la transición entre autenticación basada en sesiones PHP y autenticación JWT. Para resolver esta compatibilidad temporal se implementaron mecanismos de adaptación similares al patrón *Anti-Corruption Layer* (ACL) descrito por Evans (2003), cuyo objetivo es aislar los nuevos servicios de las limitaciones del sistema legado.

Sobre la estrategia de migración en dos fases y el principio *Database per Service*

La estrategia de migración en dos fases se eligió por una limitación técnica del sistema original: la interfaz PHP del monolito no podía ser consumida directamente mediante HTTP por los microservicios Java. Debido a esta restricción, durante la primera fase se utilizó una estrategia de base de datos compartida como mecanismo transitorio de coexistencia.

Esta decisión se encuentra respaldada en la literatura especializada. Newman (2021a, cap. 4), en Building Microservices, clasifica la base de datos compartida como un estado transitorio válido durante procesos de migración, siempre que exista una separación posterior planificada y ejecutada. Este patrón es llamado *Shared Database* y es aceptable en escenarios donde la separación inmediata de datos genera una complejidad excesiva en comparación con el valor añadido en esa etapa.

En esta investigación, la Fase 2 representó la ejecución de esa separación planificada. El resultado final, una vez completada la migración, cumple con el principio Database per Service al eliminar el acoplamiento estructural heredado del monolito.

Del mismo modo, Richardson (2019), en *Microservices Patterns*, clasifica la base de datos compartida como un antipatrón que debe evitarse o utilizarse únicamente como estado transitorio. Desde esta perspectiva, mantener de forma permanente una arquitectura con base de datos compartida habría significado reproducir el antipatrón

descrito por Richardson, en lugar de solucionar el problema de acoplamiento arquitectónico.

La adopción posterior de bases de datos independientes permitió que la separación entre servicios fuera real y no únicamente lógica. Cada microservicio pasó a administrar su propio esquema de datos, y el uso de distintos motores: (MySQL para servicios transaccionales y SQLite para servicios de catálogo) ayudó a optimizar cada dominio según sus necesidades específicas de concurrencia, volumen y consistencia.

La única excepción documentada fue DocumentService, que mantiene acceso directo a la base de datos de FileService mediante VIEWs para optimizar la obtención de datos durante la generación de documentos. Esta excepción fue aprobada por razones de rendimiento y se mantiene dentro de un único servicio. Como deuda técnica futura, se planea reemplazar esta dependencia mediante endpoints HTTP especializados que permitan eliminar completamente el acceso directo entre bases de datos.

Por su parte, Arteaga et al. (2019) incorporaron patrones de resiliencia como Circuit Breaker y comunicación asíncrona mediante colas de mensajes para resolver los cuellos de botella de su monolito. En la presente investigación no se adoptaron dichos mecanismos, dado que la comunicación entre servicios se resolvió mediante llamadas HTTP REST síncronas y el volumen transaccional del sistema no evidenció condiciones de saturación que los justificaran. Esta decisión constituye una diferencia de alcance respecto de dicho antecedente y se traslada como línea de trabajo futuro.

Sobre el orden de priorización de los módulos

Respecto al método de scoring para la priorización, los resultados mostraron una limitación no señalada explícitamente por Ma et al. (2022): la puntuación numérica no asegura por sí sola un orden de migración libre de conflictos de dependencia y sin ambigüedades cuando dos o más módulos obtienen el mismo puntaje. En el caso del

Sprint 1, FileService fue priorizado principalmente por el criterio de frecuencia y urgencia de modificación (FM=40), ya que era uno de los módulos que requería intervención inmediata. Gracias a la estrategia temporal de base de datos compartida, este servicio pudo operar desde los primeros sprints accediendo directamente a la base de datos del monolito, sin depender de la migración previa de otros servicios.

Por otro lado, DiligenceService obtuvo la misma puntuación que ParametersService. Sin embargo, se decidió iniciar primero con DiligenceService debido a su dependencia funcional con SesionService y porque representaba una funcionalidad con mayor impacto operativo. Esto permitió evidenciar que el método de scoring debía complementarse con un análisis de dependencias técnicas y funcionales antes de definir el orden definitivo de migración.

Estos resultados muestran que el patrón Strangler Fig no impone una secuencia arquitectónica rígida, sino que permite ajustar el orden de extracción según el valor operativo y las restricciones técnicas del sistema.

Sobre las mejoras funcionales obtenidas durante la migración

La migración no se limitó únicamente a replicar funcionalidades existentes del monolito. Durante el proceso también se realizaron correcciones y se incorporaron mejoras funcionales que habrían sido difíciles de implementar en la arquitectura original.

Entre los principales cambios se encuentra la integración de consultas con la API UNSCH institucional desde PersonService, así como la modernización del proceso de generación de documentos. Mientras el monolito permitía generar documentos únicamente de forma individual por expediente, la nueva arquitectura permitió implementar generación masiva de resoluciones y plantillas de diploma, reduciendo tiempos operativos y mejorando la experiencia de uso.

Estos resultados coinciden con lo señalado en la literatura sobre modernización incremental, donde la migración hacia microservicios no solo permite reemplazar componentes heredados, sino también incorporar mejoras evolutivas durante el proceso de extracción, generando valor funcional progresivamente. Al tratarse de un estudio de caso único, estos hallazgos describen la aplicación del patrón en las condiciones específicas del sistema estudiado y no son directamente extrapolables a otras migraciones sin la validación correspondiente; la delimitación de límites de servicios en otros contextos podría requerir un proceso formal de DDD y Event Storming.

Sobre el registro complementario de tiempos de respuesta

Respecto del desempeño, Li et al. (2020) y Ma et al. (2022) realizaron experimentos cuantitativos para evaluar el rendimiento posterior a la migración. El registro de tiempos de respuesta presentado en este estudio aporta evidencia parcialmente convergente, con un comportamiento que varía según la naturaleza de la operación, aunque con un alcance deliberadamente más acotado: se trata de un registro descriptivo obtenido en entorno local y sin concurrencia de usuarios, cuyo propósito es caracterizar el comportamiento de las operaciones equivalentes antes y después de la migración, y no evaluar el impacto del patrón sobre el desempeño del sistema, objetivo que excede el alcance descriptivo de la presente investigación.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

La aplicación del patrón Strangler Fig en la migración del Sistema de Grados y Títulos de la UNSCH permitió describir de forma detallada y reproducible el proceso de transición de una arquitectura monolítica hacia una arquitectura de microservicios con Spring Boot, ejecutando las tres fases del patrón: Transformación, Coexistencia y Eliminación, en ocho sprints consecutivos, sin interrumpir las operaciones del sistema. Los microservicios implementados reemplazaron los dieciséis módulos del monolito de forma gradual y controlada, validando la viabilidad del patrón en un entorno real y con alta dependencia operativa.

La fase de Transformación se desarrolló mediante la identificación de bounded contexts, la implementación de microservicios independientes con Spring Boot en capas (Controller, Service, Repository, Entity) y la verificación de criterios de aceptación con Postman. La priorización se realizó utilizando una matriz de criterios ponderados (FM, PP, CF, RS y SS), permitiendo iniciar con los módulos que requerían mayor urgencia de modificación. El proceso de transformación no consistió únicamente en trasladar el mismo código hacia otra tecnología, sino también en reorganizar responsabilidades, formalizar contratos API REST e incorporar mejoras funcionales durante cada sprint. La agrupación de módulos por dominio especialmente la consolidación de cinco tipos de expedientes en un único FileService demostró que una correcta definición de los límites del servicio reduce la duplicación de lógica y simplifica el mantenimiento. Además, cada sprint produjo un microservicio funcional, validado mediante criterios de aceptación antes de continuar con el siguiente módulo, lo que permitió controlar progresivamente el riesgo técnico de la migración.

La fase de Coexistencia fue gestionada mediante Spring Cloud Gateway como proxy central que, durante cada sprint, enrutó simultáneamente peticiones hacia el nuevo microservicio y hacia los módulos restantes del monolito PHP. Durante la primera fase de la migración se utilizó una estrategia transitoria de base de datos compartida, debido a que el monolito PHP no podía ser consumido directamente mediante HTTP por los microservicios Java. Esta decisión permitió mantener la continuidad operativa y respetar el orden de priorización definido para los sprints. El desafío de esta fase se presentó en el Sprint 6, donde la coexistencia de dos mecanismos de autenticación: sesiones PHP y JWT, requirió la creación de un endpoint de compatibilidad temporal, evidenciando que la gestión de la seguridad durante la coexistencia es un aspecto técnico a considerar.

La fase de Eliminación se ejecutó de forma ordenada y verificada en cada sprint, retirando los controladores PHP correspondientes del monolito únicamente después de confirmar que el microservicio superaba todos los criterios de aceptación durante el período de coexistencia. Este proceso no requirió modificaciones directas en el código fuente del monolito, ya que la desactivación se realizó mediante la redirección de rutas en el API Gateway. Al finalizar el Sprint 7, el monolito PHP dejó de funcionar como sistema activo para las solicitudes del frontend. Posteriormente, en el Sprint 8, se completó la separación de bases de datos, eliminando el último punto de acoplamiento estructural heredado del sistema monolítico.

En consecuencia, la eliminación en este caso de estudio tuvo dos dimensiones: la eliminación progresiva del tráfico hacia el monolito y la eliminación posterior del acoplamiento de datos mediante la adopción del principio Database per Service.

El API Gateway fue el componente central de la migración, ya que permitió redirigir progresivamente el tráfico entre el monolito y los microservicios durante las fases de Transformación, Coexistencia y Eliminación. Además, centralizó funciones de

seguridad y comunicación como validación de JWT, gestión de CORS y autenticación interna mediante API Key.

La adopción del principio Database per Service permitió que cada microservicio administrara sus propios datos de manera independiente, garantizando autonomía de despliegue y mantenimiento. Asimismo, el uso de diferentes motores de base de datos evidenció la flexibilidad de la arquitectura de microservicios para adaptar cada servicio según sus necesidades. La única excepción fue DocumentService, cuya dependencia directa de FileService quedó identificada como deuda técnica futura.

La migración incremental permitió reducir el riesgo técnico al aislar los problemas encontrados en cada sprint, evitando que afectaran a todo el sistema. Esto confirmó una de las principales ventajas del patrón Strangler Fig frente a estrategias de migración total o *big bang*.

La dockerización del ecosistema se realizó como una etapa posterior al proceso de migración, una vez completada la separación de bases de datos y eliminados los acoplamientos con el monolito. Esta decisión permitió evitar complejidad operativa durante la coexistencia entre el monolito PHP y los microservicios Java, diferenciando claramente el proceso de extracción incremental del patrón Strangler Fig de las estrategias posteriores de despliegue e infraestructura. Al tratarse de un estudio de caso único, estos hallazgos describen la aplicación del patrón en las condiciones específicas del Sistema de Grados y Títulos de la UNSCH y no son directamente extrapolables a otras migraciones sin la validación correspondiente.

5.2. RECOMENDACIONES

Para futuros procesos de migración con el patrón Strangler Fig, se recomienda complementar el método de scoring de Ma et al. (2022) con un análisis explícito de dependencias entre módulos antes de definir el orden final de migración.

Se recomienda establecer la infraestructura base antes de iniciar la extracción del primer módulo. Componentes como Eureka, Config Server y API Gateway deben encontrarse configurados y validados previamente, ya que permiten gestionar la coexistencia entre el monolito y los microservicios sin requerir modificaciones constantes en el frontend.

Asimismo, se recomienda evaluar previamente las capacidades de integración del sistema monolítico antes de definir la estrategia de coexistencia. En sistemas donde el monolito no expone APIs REST consumibles de forma directa, la estrategia transitoria de base de datos compartida puede resultar más práctica y reducir complejidades innecesarias durante las primeras etapas de la migración.

También se recomienda planificar desde el inicio la separación de bases de datos correspondiente a la segunda fase de la migración. Definir tempranamente la propiedad de tablas y los límites de cada servicio facilita posteriormente la implementación del principio Database per Service y reduce el tiempo de análisis en etapas avanzadas del proceso.

Del mismo modo, se recomienda documentar previamente las reglas de negocio implícitas presentes en el monolito antes de migrar cada módulo. Esta actividad reduce el riesgo de omitir comportamientos importantes durante la reimplementación de funcionalidades.

Se recomienda también, resolver la deuda técnica identificada en DocumentService, cuyo acceso directo a la base de datos de FileService mediante VIEWs MySQL constituye una excepción al principio Database per Service. La resolución propuesta consiste en reemplazar dicho acceso por endpoints HTTP especializados en FileService que devuelvan los datos consolidados necesarios para la generación documental, eliminando así el último punto de acoplamiento estructural entre bases de

datos del sistema migrado.

Esta investigación describió el proceso de aplicación del patrón Strangler Fig desde una perspectiva descriptiva e incorporó un registro complementario de tiempos de respuesta en condiciones controladas. Se recomienda a futuras investigaciones ampliar dicha evidencia mediante pruebas de carga con concurrencia de usuarios y mediciones de uso de CPU, memoria y tasa de errores antes y después de cada sprint, a fin de evaluar el impacto del patrón en el desempeño del sistema.

Finalmente, se recomienda que futuras investigaciones exploren la posibilidad de formalizar una metodología de migración utilizando el patrón de Strangler Fig. Para ello, dicha propuesta debería definir etapas reproducibles, validarse en múltiples casos de estudio y contar con métricas que permitan evaluar su efectividad en distintos contextos de modernización de sistemas legacy. Ello podría representar un aporte significativo para la modernización de los sistemas legacy en Ayacucho y en el Perú en general.

5.3. LIMITACIONES DEL ESTUDIO

El presente estudio reconoce las siguientes limitaciones metodológicas:

1. Verificación por el propio investigador. Los criterios de aceptación de cada sprint fueron definidos, ejecutados y verificados por el propio investigador, lo que podría introducir sesgo de confirmación. Para mitigarlo, los criterios se formularon de manera binaria, objetiva y verificable (códigos de respuesta HTTP, registro en Eureka, equivalencia del número y contenido de registros retornados por el microservicio frente al monolito para los mismos parámetros), y la evidencia quedó documentada en capturas reproducibles.

2. Validación del instrumento. La Ficha de Análisis Documental es un instrumento de registro elaborado ad hoc; su validez de contenido se cuantificó mediante juicio de cuatro expertos con el coeficiente V de Aiken (V = 0.98), sin calcular coeficientes de

consistencia interna, como el alfa de Cronbach, por tratarse de un instrumento de registro documental y no de una escala de medición con ítems politómicos.

3. Alcance cuantitativo. El registro de tiempos de respuesta presentado se obtuvo en un entorno local y sin concurrencia de usuarios, con carácter descriptivo y complementario. No se realizaron pruebas de carga ni mediciones de uso de CPU y memoria bajo concurrencia, lo que se declara como limitación y se traslada como línea de trabajo futuro.

4. Estudio de caso único. Los resultados corresponden a un caso único y no son directamente generalizables sin validación adicional en otros sistemas.

5. Delimitación pragmática de los bounded contexts. La identificación de los límites de cada microservicio se realizó a partir de la estructura de las tablas de la base de datos y de las dependencias de código del monolito, sin ejecutar un proceso formal de Domain-Driven Design ni talleres de Event Storming. Esta decisión resultó adecuada para las características del sistema estudiado, pero acota la transferibilidad del procedimiento de descomposición a sistemas de mayor complejidad de dominio.

El estudio reconoce cinco limitaciones. Los criterios de aceptación fueron verificados por el propio investigador, mitigado mediante criterios binarios y evidencia reproducible; y el instrumento fue validado por juicio de cuatro expertos (V de Aiken = 0.98). Las restantes acotan el alcance: el registro de tiempos de respuesta es descriptivo y sin concurrencia; los resultados corresponden a un caso único y no son generalizables; y los bounded contexts se delimitaron de forma pragmática sin proceso formal de DDD, lo que restringe la transferibilidad del procedimiento a sistemas de mayor complejidad. En conjunto, estas limitaciones delimitan las condiciones de interpretación de los hallazgos y orientan las recomendaciones.

REFERENCIAS BIBLIOGRÁFICAS

Aiken, L. R. (1985). Three Coefficients for Analyzing the Reliability and Validity of Ratings. *Educational and Psychological Measurement*, *45*(1), 131–142. https://doi.org/10.1177/0013164485451012

Al-Debagy, O., & Martinek, P. (2018). A Comparative Review of Microservices and Monolithic Architectures. *18th IEEE International Symposium on Computational Intelligence and Informatics, CINTI 2018 - Proceedings*, 149–154. https://doi.org/10.1109/CINTI.2018.8928192

Arias, F. G. (2012). *El proyecto de investigación: Introducción a la metodología científica* (6th ed.). Episteme.

Arteaga, J., Bonilla, R., & Flores, C. (2019). *Propuesta arquitectónica para la migración de un sistema monolítico a un sistema distribuido utilizando domain driven design y microservicios: caso de estudio*.

AWS. (2023). *Patrón Strangler Fig para cargas modernas | Blog de Amazon Web Services (AWS)*. https://aws.amazon.com/es/blogs/aws-spanish/patron-strangler-fig-para-cargas-modernas/

AWS. (2024a). *Monolítico frente a microservicios: diferencia entre arquitecturas de desarrollo de software - AWS*. https://aws.amazon.com/es/compare/the-difference-between-monolithic-and-microservices-architecture/

AWS. (2024b). *Patrones para la descomposición de monolitos - AWS Guía prescriptiva*. https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/modernization-decomposing-monoliths/decomposing-patterns.html

AWS. (2025). *Patrón de higos estranguladores - AWS Guía prescriptiva*. https://docs.aws.amazon.com/es_es/prescriptive-guidance/latest/cloud-design-patterns/strangler-fig.html

Bernal, C. A. (2010). *Metodología de la investigación Administración, economía, humanidades y ciencias sociales* (Tercera). Pearson Educación.

Bragagnolo, S., Anquetil, N., Ducasse, S., Seriai, A., & Derras, M. (2021). *Software Migration: A Theoretical Framework.* Inria Lille Nord Europe-Laboratoire CRIStAL-Université de Lille.

Brito, M., Cunha, J., & Saraiva, J. (2021). Identification of microservices from monolithic applications through topic modelling. *Proceedings of the ACM Symposium on Applied Computing*, 1409–1418. https://doi.org/10.1145/3412841.3442016

Cerda, H. (1991). *Los elementos de la investigación: Cómo reconocerlos, diseñarlos y construirlos*. Editorial El Búho.

Chandler, H. (2024a). *Arquitectura de microservicios | Atlassian*. https://www.atlassian.com/es/microservices/microservices-architecture

Chandler, H. (2024b). *Comparación entre la arquitectura monolítica y la arquitectura de microservicios*. https://www.atlassian.com/es/microservices/microservices-architecture/microservices-vs-monolith

Corral, Y. (2009). Validez y confiabilidad de los instrumentos de investigación para la recolección de datos. *Revista Ciencias de La Educación*, *19*(33), 228–247.

Creswell, J. W., & Plano Clark, V. L. (2011). *Designing and Conducting Mixed Methods Research* (2nd ed.). Sage Publications.

Cruz, J., Baez, M., & Flores, A. P. (2024). Migración de serverbox hacia un sistema de microservicios aplicando arquitectura vertical. *Revista de Investigación En Tecnologías de La Información*, *12*(27 Especial), 53–66. https://doi.org/10.36825/RITI.12.27.006

Escurra Mayaute, L. M. (1988). Cuantificación de la validez de contenido por criterio de jueces. *Revista de Psicología*, *6*(1–2), 103–111.

Evans, E. (2003). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.

Fowler, M. (2004). *Strangler Fig*. https://martinfowler.com/bliki/StranglerFigApplication.html

Freitas, F., Ferreira, A., & Cunha, J. (2023). A methodology for refactoring ORM-based monolithic web applications into microservices. *Journal of Computer Languages*, *75*. https://doi.org/10.1016/j.cola.2023.101205

Geovany, J., & Averos, A. (2025). *Diseño de arquitectura distribuida para migrar software de administración de informes y recomendaciones de una empresa de telefonía celular a una arquitectura moderna de microservicios*. https://dspace.ups.edu.ec/handle/123456789/29505

Hasan, M. H., Osman, M. H., Admodisastro, N. I., & Muhammad, M. S. (2023). Legacy systems to cloud migration: A review from the architectural perspective. *Journal of Systems and Software*, *202*, 111702. https://doi.org/10.1016/j.jss.2023.111702

Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M. (2014). *Metodología de la investigación* (I. E. S.A, Ed.; 6ta ed.). Mc Graw Hill Education.

IBM. (2024). *¿Qué es la migración de aplicaciones? | IBM*. https://www.ibm.com/es-es/topics/application-migration

Kumar, S. (2024, June 10). *Microservice 101: The Strangler Fig pattern*. https://www.linkedin.com/pulse/microservice-101-strangler-figpattern-shanoj-kumar-v-p7ssc/

Li, C. Y., Ma, S. P., & Lu, T. W. (2020). Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System. *Proceedings - 2020 International Computer Symposium, ICS 2020*, 519–524. https://doi.org/10.1109/ICS51289.2020.00107

Ma, S. P., Li, C. Y., Lee, W. T., & Lee, S. J. (2022). Microservice Migration Using Strangler Fig Pattern and Domain-Driven Design. *Journal of Information Science and Engineering*, *38*(6), 1285–1303. https://doi.org/10.6688/JISE.202211_38(6).0010

Malton, A. J. (2001). The software migration barbell. *ASERC Workshop on Software Architecture*.

Mendoza Vallejos, O. A., & Alva Caffo, A. A. (2024). Migración de los sistemas relacionados a la cobranza de la TUUA (Tarifa Única de Uso de Aeropuerto) para el mejoramiento de la infraestructura de software de un aeropuerto internacional. *Universidad Peruana de Ciencias Aplicadas (UPC)*. https://renati.sunedu.gob.pe/handle/renati/418842

Microsoft. (2024). *¿Qué es la migración de aplicaciones? | Microsoft Azure*. https://azure.microsoft.com/es-mx/resources/cloud-computing-dictionary/what-is-application-migration

Microsoft Learn. (2024). *Patrón Strangler Fig - Azure Architecture Center | Microsoft Learn*. https://learn.microsoft.com/es-es/azure/architecture/patterns/strangler-fig

Mordor Intelligence Research & Advisory. (2023). *Tamaño del mercado de microservicios en la nube y análisis de acciones - Informe de investigación de la industria - Tendencias de crecimiento*. https://www.mordorintelligence.com/es/industry-reports/cloud-microservices-market

Newman, S. (2021a). *Building microservices : designing fine-grained systems*. O'Reilly Media.

Newman, S. (2021b). *Monolith to Microservices*. https://books.google.com/books/about/Monolith_to_Microservices.html?hl=es&id=iul3wQEACAAJ

Noriega Yarlequé, H. J. (2022). Migración del sistema web ePlanning de una arquitectura monolítica a una arquitectura de microservicios en la nube para una corporación de belleza. *Universidad Nacional Mayor de San Marcos*. https://renati.sunedu.gob.pe/handle/renati/492577

Oracle. (2024). *Microservicios*. https://docs.oracle.com/es-ww/iaas/Content/cloud-adoption-framework/microservices.htm

Oracle. (2019). *Oracle Database 19c Technical Architecture*.

Pimienta Prieto, J. H., & la Orden Hoz, A. (2017). *Metodología de la investigación* (3rd ed.). Pearson.

RedHat. (2023, March 15). *¿Qué es una arquitectura de aplicaciones? Tipos de arquitecturas de aplicaciones*. https://www.redhat.com/es/topics/cloud-native-apps/what-is-an-application-architecture

RedHat. (2024). *Migración de las aplicaciones*. https://www.redhat.com/es/topics/cloud-native-apps/application-migration

Ren, Z., Wang, W., Wu, G., Gao, C., Chen, W., Wei, J., & Huang, T. (2018). Migrating web applications from monolithic structure to microservices architecture. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3275219.3275230

Richardson, C. (2019). *Microservices Patterns*. Manning Publications Co.

Schwaber, K., & Sutherland, J. (2020). *La Guía Scrum*.

Stake, R. E. (1995). *The Art of Case Study Research*. SAGE Publications.

Stuurman, S., Passier, H., & Barendsen, E. (2016). Analyzing students' software redesign strategies. *ACM International Conference Proceeding Series*, 110–119. https://doi.org/10.1145/2999541.2999559

Thamma, T., Nitayaprapha, T., Meesumrarn, T., Phanthunane, P., Preeprem, S., Suwatmakin, A., Pannarunothai, S., & Wongthai, W. (2025). TRANSFORMING SOFTWARE ARCHITECTURE: A STRANGLER FIG PATTERN APPROACH. *ICIC Express Letters, Part B: Applications*, *16*(8), 865–872. https://doi.org/10.24507/icicelb.16.08.865

Yépez, W. L. P., Alegría, A. F. S., Bandi, A., & Alegría, J. A. H. (2024). Visión de las competencias en arquitectura de software integrando las perspectivas de la industria y la academia. *REVISTA COLOMBIANA DE TECNOLOGIAS DE AVANZADA (RCTA)*, *1*(43), 9–23. https://doi.org/10.24054/RCTA.V1I43.2798

Yin, R. K. (2014). *Case Study Research: Design and Methods* (5th ed.). SAGE Publications.

ANEXOS

ANEXO A. INSTRUMENTO DE RECOLECCIÓN DE DATOS

Ficha de Análisis Documental — Patrón Strangler Fig – Se aplica por Sprint y Fase.

DATOS DE IDENTIFICACIÓN	
Sprint N°:	
Microservicio:	
Módulos migrados:	
Período:	

SECCIÓN B — TRANSFORMACIÓN			
Historia de usuario			
Endpoints implementados			
Método	**Endpoint**	**Descripción**	**Respuesta**
Criterios de aceptación			
N°	**Criterio**	**Resultado**	**Estado**
Observaciones técnicas del sprint (decisiones técnicas, dependencias o deuda técnica relevante):			

SECCIÓN C — COEXISTENCIA Y ELIMINACIÓN		
C.1 — COEXISTENCIA		
Enrutamiento en el API Gateway		
Ruta	**Destino**	**Estado**
Estrategia de BD:		
Incidencias:		

C.2 — ELIMINACIÓN		
Condiciones verificadas antes de eliminar		
Condición	**Evidencia**	**¿Cumple?**
Proceso de retiro:		
Monolito restante:		
Observaciones técnicas finales (cierre de la migración del módulo):		

Nota. El instrumento registra sistemáticamente el proceso de migración con el Patrón Strangler Fig en sus tres fases (Transformación, Coexistencia, Eliminación) por cada sprint ejecutado.

ANEXO B. MATRIZ DE CONSISTENCIA

TITULO: **APLICACIÓN DEL PATRÓN STRANGLER FIG EN LA MIGRACIÓN DE UN SISTEMA WEB MONOLÍTICO A MICROSERVICIOS, AYACUCHO, 2025**.

PROBLEMAS	OBJETIVOS	HIPÓTESIS	VARIABLES E INDICADORES	MÉTODO DE INVESTIGACIÓN
1. PROBLEMA PRINCIPAL ¿Cómo se aplica el patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios? **2. PROBLEMAS SECUNDARIOS** a. ¿Cómo se desarrolla la transformación de componentes de un sistema web monolítico a microservicios con el Patrón Strangler Fig? b. ¿Cómo se gestiona la coexistencia entre los componentes del sistema monolítico y los	**1. OBJETIVO GENERAL** Describir el proceso de aplicación del Patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios, en el caso de estudio del Sistema de Grados y Títulos de la Universidad Nacional de San Cristóbal de Huamanga, Ayacucho, 2025. **2. OBJETIVOS ESPECÍFICOS** a. Describir cómo se desarrolla la fase de Transformación del Patrón Strangler Fig en la migración de componentes de un sistema monolítico a microservicios. b. Describir como se gestiona la fase de Coexistencia del Patrón Strangler Fig durante el proceso de migración.	**1. HIPÓTESIS GENERAL** Este estudio no presenta hipótesis.	**1. VARIABLE DE INTERÉS** X: Aplicación del Patrón Strangler Fig (proceso de migración) **Variables Descriptivas** (Sub variables) **X1: Transformación** **X2: Coexistencia** **X3: Eliminación**	**1. ENFOQUE DE INVESTIGACIÓN:** Mixto con predominancia cualitativa. **1. TIPO DE INVESTIGACIÓN** Aplicada. **2. NIVEL DE INVESTIGACIÓN** Descriptivo **4. DISEÑO** Estudio de caso único con implementación, de corte longitudinal **5. UNIDAD DE ANÁLISIS** El Sistema de Grados y Títulos UNSCH (caso único)

microservicios durante el proceso de migración con el Patrón Strangler Fig? c. ¿Cómo se realiza la eliminación del monolito en el proceso de migración con el Patrón Strangler Fig?	c. Describir la ejecución de la fase de Eliminación del Patrón Strangler Fig en el proceso de migración.			**6. UNIDADES DE OBSERVACIÓN** Cada uno de los 8 sprints de migración ejecutados. Alcance de la migración: los 16 módulos del sistema monolítico (estudio censal, sin muestreo) **7. TÉCNICAS** ▪ Análisis Documental. **8. INSTRUMENTOS** ▪ Ficha de Análisis Documental.

ANEXO C. MATRIZ DE OPERACIONALIZACIÓN DE VARIABLES

Variable	Dimensión (Sub-variable)	Indicador	Ítem	Tipo de dato	Escala
Aplicación del Patrón Strangler Fig (proceso de migración)	X1: Transformación	Microservicio implementado	Sección B (se verifica cuando todos los criterios de aceptación de la Sección B están en estado "Cumple")	Cualitativo	Nominal
		Endpoints REST implementados	Sección B — Tabla de endpoints	Cuantitativo	Razón
		Criterios de aceptación cumplidos	Sección B — Tabla de criterios (Estado)	Cuantitativo	Razón
		Decisiones técnicas documentadas	Sección B — Observaciones	Cualitativo	Nominal
		Historia de usuario documentada	Sección B — Historia de usuario	Cualitativo	Nominal
	X2: Coexistencia	Rutas activas en el API Gateway por sprint	Sección C — Tabla de enrutamiento (Estado)	Cuantitativo	Razón
		Estrategia de base de datos	Sección C — Estrategia de BD	Cualitativo	Nominal
		Incidencias detectadas	Sección C — Incidencias	Cualitativo	Nominal
	X3: Eliminación	Condiciones verificadas antes de eliminar	Sección C — Tabla de condiciones	Cualitativo	Nominal
		Módulos PHP desactivados	Sección C — Proceso de retiro	Cualitativo	Nominal

	Módulos restantes en el monolito por sprint	Sección C — Monolito restante	Cuantitativo	Razón

Nota. La escala Nominal aplica a datos descriptivos sin jerarquía. La escala Nominal comprende asimismo las condiciones de verificación pre-eliminación, registradas de forma dicotómica (Sí / No). La escala Razón aplica a conteos numéricos con cero absoluto (endpoints implementados, criterios cumplidos, módulos restantes).

ANEXO D. PROCESO SCRUM ADOPTADO

Marco de trabajo ágil: Scrum

ROLES:

Rol de Trabajo	Responsable
Scrum Master	Julio M. Ayala Bellido
Product Owner	Julio M. Ayala Bellido
Scrum Team	Julio M. Ayala Bellido

Parámetros del marco Scrum adaptado al proceso de migración

Parámetro	Descripción
Rol	Único: el investigador actúa como Product Owner, Scrum Master y desarrollador.
Duración — Sprints 1 y 2	3 semanas, 40 horas efectivas por sprint (mayor complejidad técnica).
Duración — Sprints 3 al 8	2 semanas por sprint (infraestructura ya validada, menor complejidad).
Sprint Planning	Definición del Sprint Backlog por Sprint.
Sprint Review	Verificación de criterios de aceptación al finalizar cada sprint.
Definición de Hecho (DoD)	Criterios cumplidos en Postman + servicio en Eureka + Gateway actualizado + módulo PHP retirado.
Artefactos	Product Backlog (orden por scoring), Sprint Backlog, Incremento (microservicio desplegado).

Nota: El marco Scrum fue adaptado a una investigación individual. Los eventos Daily Scrum fueron omitidos por tratarse de un único investigador.

SPRINT BACKLOG GENERAL

Sprint	Microservicio	Historia de usuario principal	Tareas técnicas principales	
1	FileService	Gestionar expedientes académicos por nivel vía API REST independiente	Análisis módulo PHP, diseño entidades, implementación capas, validación unicidad, pruebas Postman	OK
2	DocumentService	Generar documentos oficiales (lista CU, resolución, diploma) individual y masivamente vía API REST	Implementación generación Word y pruebas equivalencia documental	OK
3	PersonService	Gestionar información del interesado con consulta automática al API UNSCH por DNI	Implementación cliente HTTP API UNSCH, registro manual como fallback, pruebas Postman	OK
4	SesionService	Gestionar sesiones universitarias con restricción de sesión única activa y sin fechas duplicadas	Implementación reglas de negocio en capa de servicio, pruebas de casos límite	OK
5	AcademicService	Gestionar facultades, escuelas y denominaciones con logging automático	Implementación tres controladores, configuración, pruebas de endpoints	OK
6	UserService	Autenticar usuarios con JWT y gestionar roles diferenciados	Implementación login BCrypt, generación JWT con claims, validación token en Gateway, pruebas de seguridad	OK
7	DiligenceService + ParametersService	Gestionar la diligencia de cada sesión y centralizar parámetros de configuración del sistema	Implementación restricción una diligencia por sesión, centralización tablas de parámetros, comunicación inter-servicio	OK
8	Separación de BD y Comunicación HTTP	Separar la base de datos para cada microservicio e implementar la comunicación por HTTP	Aplicar Database per Service y Comunicación por API REST entre microservicios.	OK

Nota. El detalle completo de criterios de aceptación, enrutamiento y resultados de cada sprint se muestran en las Fichas de Análisis Documental correspondientes.

ANEXO E. FICHA DE ANÁLISIS DOCUMENTAL EN FILE SERVICE

FICHA DE ANÁLISIS DOCUMENTAL *Patrón Strangler Fig — Sistema de Grados y Títulos UNSCH*

DATOS DE IDENTIFICACIÓN	
Sprint N°:	1
Microservicio:	FileService — Puerto 8082
Módulos migrados:	Bachiller (04), Título (05), Maestría (06), Doctorado (07), Segunda Especialidad (08)
Período:	Semanas 1–3 · 40 horas efectivas

SECCIÓN B — TRANSFORMACIÓN			
Historia de usuario			
Como usuario del sistema, quiero gestionar los expedientes académicos a través de una API REST independiente, para registrar, consultar, actualizar y eliminar expedientes sin depender del sistema monolítico.			
Endpoints implementados			
Método	**Endpoint**	**Descripción**	**Respuesta**
GET	/api/file	Lista todos los expedientes	200
GET	/api/file/id/{id}	Obtiene un expediente por ID	200 / 404
GET	/api/file/bySession	Filtra por sesión, nivel y generación	200 / 400
POST	/api/file/create	Registra un expediente	200 / 400
PATCH	/api/file/update	Actualiza un expediente	200 / 404
DELETE	/api/file/{id}	Elimina un expediente	200 / 404
GET	/api/file/health	Verifica el estado del microservicio	200 / 500
Criterios de aceptación			
N°	**Criterio**	**Resultado**	**Estado**
1	El servicio se registra en Eureka al iniciar	Visible como FILESERVICE en el dashboard de Eureka	Cumple
2	POST valida que no exista expediente duplicado por persona/nivel/generación/escuela	Retorna 409 Conflict con mensaje descriptivo	Cumple
3	GET /bySession retorna solo expedientes de la sesión, nivel y generación solicitados	Filtra correctamente los tres parámetros	Cumple
4	Los endpoints retornan JSON con la estructura esperada	Validado con colección Postman	Cumple
Observaciones técnicas del sprint	La principal dificultad fue la validación de unicidad con distintas combinaciones de parámetros. Se crearon casos de prueba para cada combinación posible. La		

(decisiones técnicas, dependencias o deuda técnica relevante):	infraestructura fue validada por primera vez en este sprint. Además de los endpoints listados, FileService implementó el endpoint de conteo de expedientes por sesión y los tres endpoints de reporte que absorbieron la funcionalidad del módulo Reportes, con lo que el servicio totalizó once endpoints.

SECCIÓN C — COEXISTENCIA Y ELIMINACIÓN		
C.1 — COEXISTENCIA		
Enrutamiento en el API Gateway		
Ruta	**Destino**	**Estado**
/api/file/**	FileService :8082	Activo
/api/document/**	Monolito PHP	Pendiente
/api/person/**	Monolito PHP	Pendiente
/api/session/**	Monolito PHP	Pendiente
/api/academic/**	Monolito PHP	Pendiente
/api/users/**	Monolito PHP	Pendiente
/api/diligence/**	Monolito PHP	Pendiente
/api/report/**	Monolito PHP	Pendiente
Estrategia de BD:	Base de datos compartida MySQL 8.2. FileService accede a la tabla expediente. Se establece como propietario exclusivo de dicha tabla desde este sprint. Se separa la base de datos en Fase 2.	
Incidencias:	Se detectó respuesta lenta en el endpoint registrar al realizar la validación de unicidad. Corregido optimizando la consulta JPA en la capa de repositorio.	
C.2 — ELIMINACIÓN		
Condiciones verificadas antes de eliminar		
Condición	**Evidencia**	**¿Cumple?**
Criterios de aceptación completados	Tabla B cumplida al 100%	Sí
Gateway redirige correctamente a FileService	Verificado con Postman vía Gateway	Sí
Ningún módulo activo del monolito escribe en expediente	Revisión del código PHP confirma propiedad	Sí
Proceso de retiro:	Se desactivaron los controladores PHP de Bachiller, Título, Maestría, Doctorado y Segunda Especialidad. Se actualizaron las reglas del Gateway eliminando rutas residuales al monolito para /api/file/**.	
Monolito restante:	10 módulos restantes: Sesión Universitaria, Diligencia, Persona, Lista CU, Resoluciones, Diplomas, Escuela, Facultad, Denominación y Usuarios.	
Observaciones técnicas finales (cierre de la migración del módulo):	Sprint de mayor duración (3 semanas) por agrupar cinco módulos. La decisión de tratarlos como un único bounded context (FileService) demostró ser correcta al centralizar la lógica de validación sin duplicar código.	

ANEXO F. GUIA DE SCORING

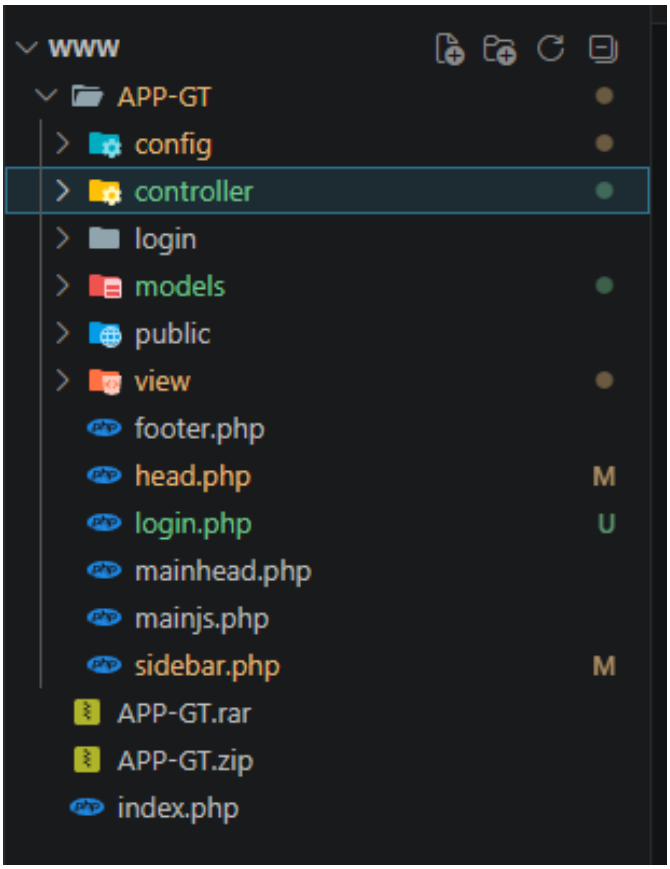
Table 1 Service value scoring guidelines

Criteria	Score	Rationales/Benefits	References
FM (Function to be Modified): The functionality involved in the service is about to be modified.	{0, 40}	Extracting this kind of services can avoid making changes directly to the monolithic system.	[2] [15] [16]
PP (Performance Problem): The service has performance problems.	[0, 20]	If a candidate service consumes many resources and may affect the performance of other modules, it should be extracted early.	[2] [15] [17]
CF (Core Function): The service contain the core functions of the system.	[0, 20]	If the extracted service includes core functions, the extraction usually brings more benefits, e. g., scalability or maintainability.	[16] [18]
RS (Related Service): The service is related to the service that has been migrated.	[0, 10]	It would be better if an extracted service collaborates with other migrated services.	[2]
SS (Separate Service): The service is obviously separated from the monolithic architecture.	[0, 10]	Extracting this kind of services is easier than other candidates.	[16]

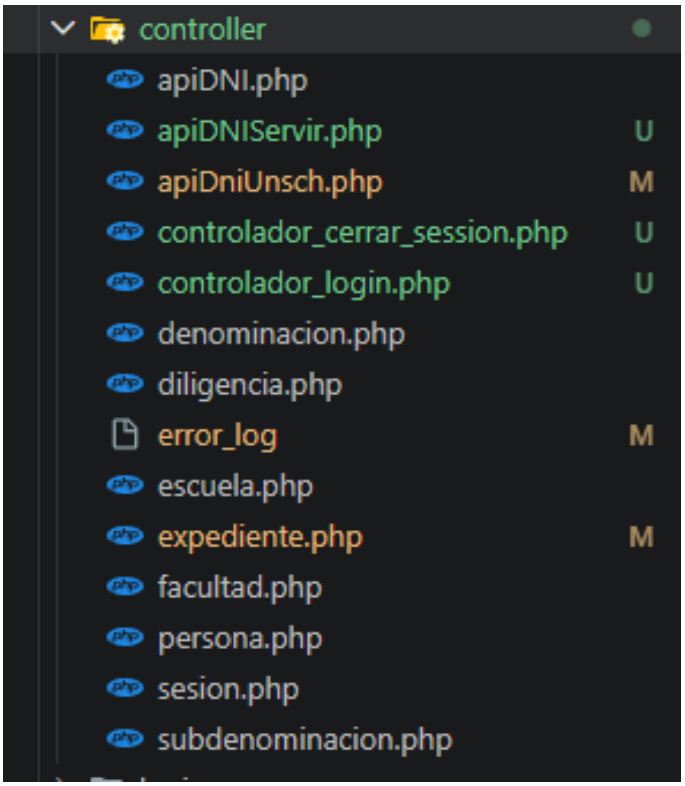
Nota: Obtenido de (Ma et al., 2022).

ANEXO G. PROYECTO MONOLITO Y MICROSERVICIOS

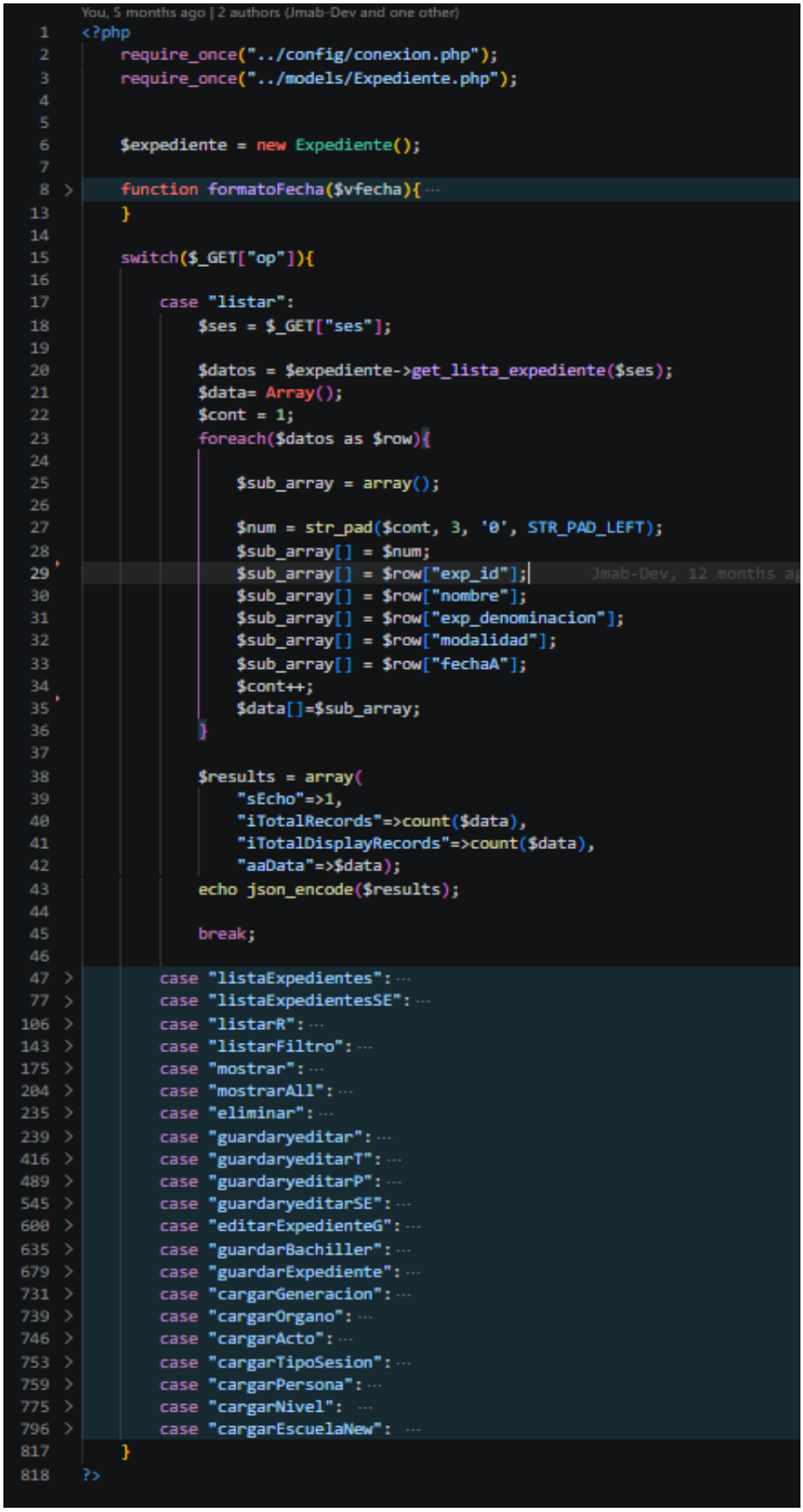
- Estructura de Proyecto Monolito



- Estructura del Controlador



- Controlador PHP Expediente



```
You, 5 months ago | 2 authors (Jmab-Dev and one other)
<?php
    require_once("../config/conexion.php");
    require_once("../models/Expediente.php");


    $expediente = new Expediente();

    function formatoFecha($vfecha){ ···
    }

    switch($_GET["op"]){

        case "listar":
            $ses = $_GET["ses"];

            $datos = $expediente->get_lista_expediente($ses);
            $data= Array();
            $cont = 1;
            foreach($datos as $row){

                $sub_array = array();

                $num = str_pad($cont, 3, '0', STR_PAD_LEFT);
                $sub_array[] = $num;
                $sub_array[] = $row["exp_id"];      Jmab-Dev, 12 months a
                $sub_array[] = $row["nombre"];
                $sub_array[] = $row["exp_denominacion"];
                $sub_array[] = $row["modalidad"];
                $sub_array[] = $row["fechaA"];
                $cont++;
                $data[]=$sub_array;
            }

            $results = array(
                "sEcho"=>1,
                "iTotalRecords"=>count($data),
                "iTotalDisplayRecords"=>count($data),
                "aaData"=>$data);
            echo json_encode($results);

            break;

        case "listaExpedientes": ···
        case "listaExpedientesSE": ···
        case "listarR": ···
        case "listarFiltro": ···
        case "mostrar": ···
        case "mostrarAll": ···
        case "eliminar": ···
        case "guardaryeditar": ···
        case "guardaryeditarT": ···
        case "guardaryeditarP": ···
        case "guardaryeditarSE": ···
        case "editarExpedienteG": ···
        case "guardarBachiller": ···
        case "guardarExpediente": ···
        case "cargarGeneracion": ···
        case "cargarOrgano": ···
        case "cargarActo": ···
        case "cargarTipoSesion": ···
        case "cargarPersona": ···
        case "cargarNivel": ···
        case "cargarEscuelaNew": ···
    }
?>
```

- Controlador PHP Sesión

APP-GT > controller > sesion.php

Jmab-Dev, 12 months ago | 2 authors (Jmab-Dev and one other)

```php
<?php
    require_once("../config/conexion.php");
    require_once("../models/Sesion.php");

    $sesion = new Sesion();

    switch($_GET["op"]){

        case "listar":
            $sf = $_GET["appat"];
            $datos;
            if ($sf == " ") {
                $datos = $sesion->get_sesion();
                # code...
            }else{
                $datos = $sesion->get_sesion_x_fecha($sf);
            }

            $data= Array();

            foreach($datos as $row){
                $sub_array = array();

                $sub_array[] = $row["ses_id"];
                $sub_array[] = $row["ses_fecha"];
                $sub_array[] = $row["sesTipo_nombre"];
                $sub_array[] = $row["org_nombre"];
                $sub_array[] = $row["ses_estado"];

                $sub_array[] = '<button type="button" onClick="editar('.$row["ses_id"].');"  id="'.$row["ses_id"].'
                " class="btn btn-outline-primary btn-icon"><div><i class="fa fa-edit"></i></div></button>';
                $sub_array[] = '<button type="button" onClick="eliminar('.$row["ses_id"].');"  id="'.$row["ses_id"].'
                " class="btn btn-outline-danger btn-icon"><div><i class="fa fa-trash"></i></div></button>';
                $data[]=$sub_array;
            }

            $results = array(···
            echo json_encode($results);

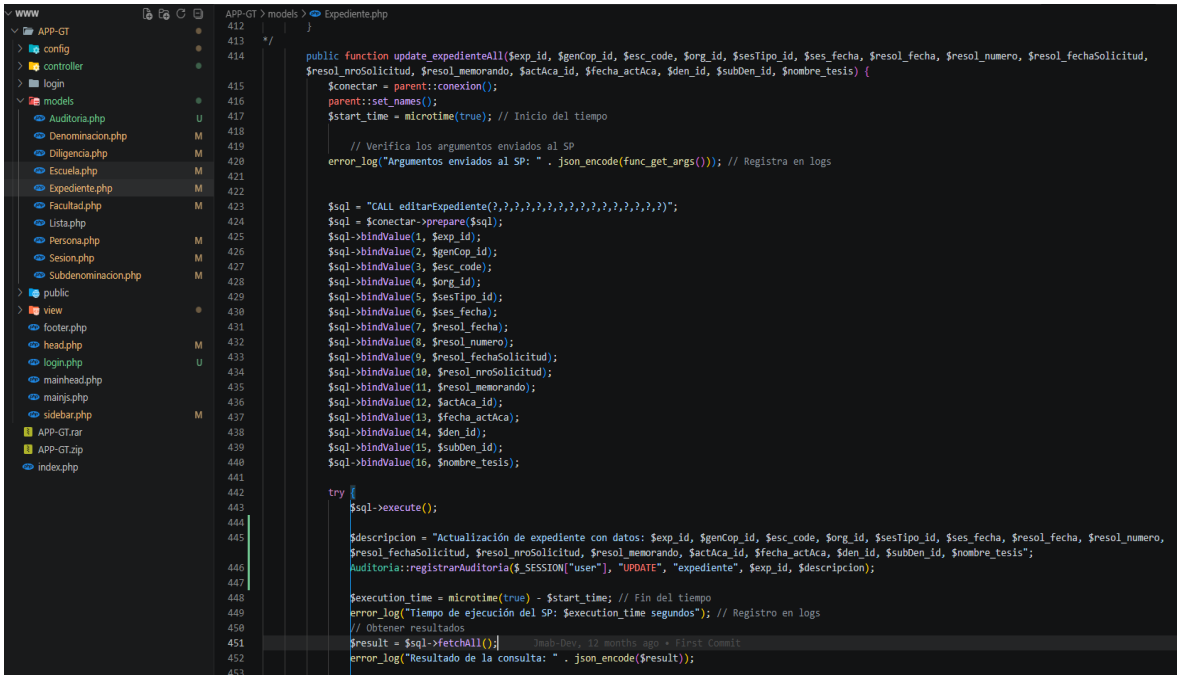
            break;

        case "guardaryeditar":···
        case "mostrar":···
        case "eliminar":
            $sesion->delete_sesion($_POST["ses_id"]);
            break;

        case "sesionActual":
            $fila = $sesion->get_sesion_actual();
            echo json_encode($fila);

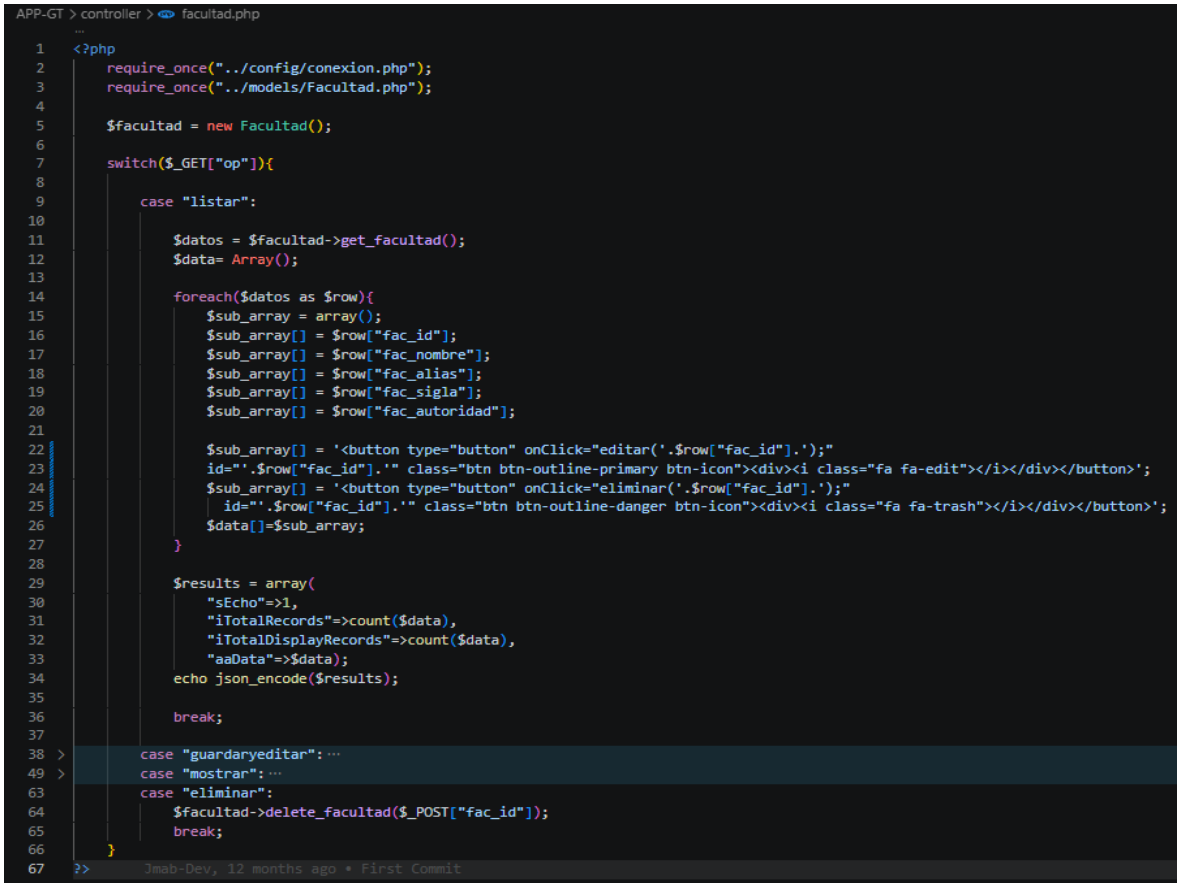
        Jmab-Dev, 12 months ago • First Commit
            break;
    }
```

- Estructura del Modelo - Expediente



```
APP-GT > models > Expediente.php
412         }
413     */
414     public function update_expedienteAll($exp_id, $genCop_id, $esc_code, $org_id, $sesTipo_id, $ses_fecha, $resol_fecha, $resol_numero, $resol_fechaSolicitud,
        $resol_nroSolicitud, $resol_memorando, $actAca_id, $fecha_actAca, $den_id, $subDen_id, $nombre_tesis) {
415         $conectar = parent::conexion();
416         parent::set_names();
417         $start_time = microtime(true); // Inicio del tiempo
418
419             // Verifica los argumentos enviados al SP
420         error_log("Argumentos enviados al SP: " . json_encode(func_get_args())); // Registra en logs
421
422
423         $sql = "CALL editarExpediente(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
424         $sql = $conectar->prepare($sql);
425         $sql->bindValue(1, $exp_id);
426         $sql->bindValue(2, $genCop_id);
427         $sql->bindValue(3, $esc_code);
428         $sql->bindValue(4, $org_id);
429         $sql->bindValue(5, $sesTipo_id);
430         $sql->bindValue(6, $ses_fecha);
431         $sql->bindValue(7, $resol_fecha);
432         $sql->bindValue(8, $resol_numero);
433         $sql->bindValue(9, $resol_fechaSolicitud);
434         $sql->bindValue(10, $resol_nroSolicitud);
435         $sql->bindValue(11, $resol_memorando);
436         $sql->bindValue(12, $actAca_id);
437         $sql->bindValue(13, $fecha_actAca);
438         $sql->bindValue(14, $den_id);
439         $sql->bindValue(15, $subDen_id);
440         $sql->bindValue(16, $nombre_tesis);
441
442         try {
443             $sql->execute();
444
445             $descripcion = "Actualización de expediente con datos: $exp_id, $genCop_id, $esc_code, $org_id, $sesTipo_id, $ses_fecha, $resol_fecha, $resol_numero,
                $resol_fechaSolicitud, $resol_nroSolicitud, $resol_memorando, $actAca_id, $fecha_actAca, $den_id, $subDen_id, $nombre_tesis";
446             Auditoria::registrarAuditoria($_SESSION["user"], "UPDATE", "expediente", $exp_id, $descripcion);
447
448             $execution_time = microtime(true) - $start_time; // Fin del tiempo
449             error_log("Tiempo de ejecución del SP: $execution_time segundos"); // Registro en logs
450             // Obtener resultados
451             $result = $sql->fetchAll();     Jmab-Dev, 12 months ago • First Commit
452             error_log("Resultado de la consulta: " . json_encode($result));
```

- Controlador PHP Facultad, Escuela y Denominacion



```
APP-GT > controller > facultad.php
1   <?php
2       require_once("../config/conexion.php");
3       require_once("../models/Facultad.php");
4
5       $facultad = new Facultad();
6
7       switch($_GET["op"]){
8
9           case "listar":
10
11              $datos = $facultad->get_facultad();
12              $data= Array();
13
14              foreach($datos as $row){
15                  $sub_array = array();
16                  $sub_array[] = $row["fac_id"];
17                  $sub_array[] = $row["fac_nombre"];
18                  $sub_array[] = $row["fac_alias"];
19                  $sub_array[] = $row["fac_sigla"];
20                  $sub_array[] = $row["fac_autoridad"];
21
22                  $sub_array[] = '<button type="button" onClick="editar('.$row["fac_id"].');"
23                  id="'.$row["fac_id"].'" class="btn btn-outline-primary btn-icon"><div><i class="fa fa-edit"></i></div></button>';
24                  $sub_array[] = '<button type="button" onClick="eliminar('.$row["fac_id"].');"
25                    id="'.$row["fac_id"].'" class="btn btn-outline-danger btn-icon"><div><i class="fa fa-trash"></i></div></button>';
26                  $data[]=$sub_array;
27              }
28
29              $results = array(
30                  "sEcho"=>1,
31                  "iTotalRecords"=>count($data),
32                  "iTotalDisplayRecords"=>count($data),
33                  "aaData"=>$data);
34              echo json_encode($results);
35
36              break;
37
38          case "guardaryeditar": ···
49          case "mostrar": ···
63          case "eliminar":
64              $facultad->delete_facultad($_POST["fac_id"]);
65              break;
66      }
67  ?>      Jmab-Dev, 12 months ago • First Commit
```

APP-GT > controller > escuela.php

Jmab-Dev, 12 months ago | 1 author (Jmab-Dev)

```php
<?php
    require_once("../config/conexion.php");
    require_once("../models/Escuela.php");

    $escuela = new Escuela();

    switch($_GET["op"]){

        case "listar":···
        case "guardaryeditar":···
        case "mostrar":
            $datos=$escuela->get_escuela_x_id($_POST["esc_id"]);
            if(is_array($datos)==true and count($datos)>0){
                foreach($datos as $row){
                    $output["esc_id"] = $row["esc_id"];
                    $output["esc_code"] = $row["esc_code"];
                    $output["esc_nombre"] = $row["esc_nombre"];
                    $output["esc_sigla"] = $row["esc_sigla"];
                    $output["esc_alias"] = $row["esc_alias"];
                    $output["fac_id"] = $row["fac_id"];
                }
                echo json_encode($output);
            }
            break;

        case "eliminar":
            $escuela->delete_escuela($_POST["esc_id"]);
            break;

        case "cargarFacultad":
            $datos = $escuela->cargarFacultad();

            $data= Array();

            foreach ($datos as $row) {
                # code...
                $fac_id = $row['fac_id'];
                $fac_sigla = $row['fac_sigla'];
                $data[] = array('fac_id'=>$fac_id, 'fac_sigla'=>$fac_sigla);
            }

            $results = array("data"=>$data);
            //$results = array($data);


            echo json_encode($data);
            //echo json_encode($results);
            break;
    }
?>
```

Jmab-Dev, 12 months ago • First Commit

```php
<?php
    require_once("../config/conexion.php");
    require_once("../models/Denominacion.php");


    $denominacion = new Denominacion();

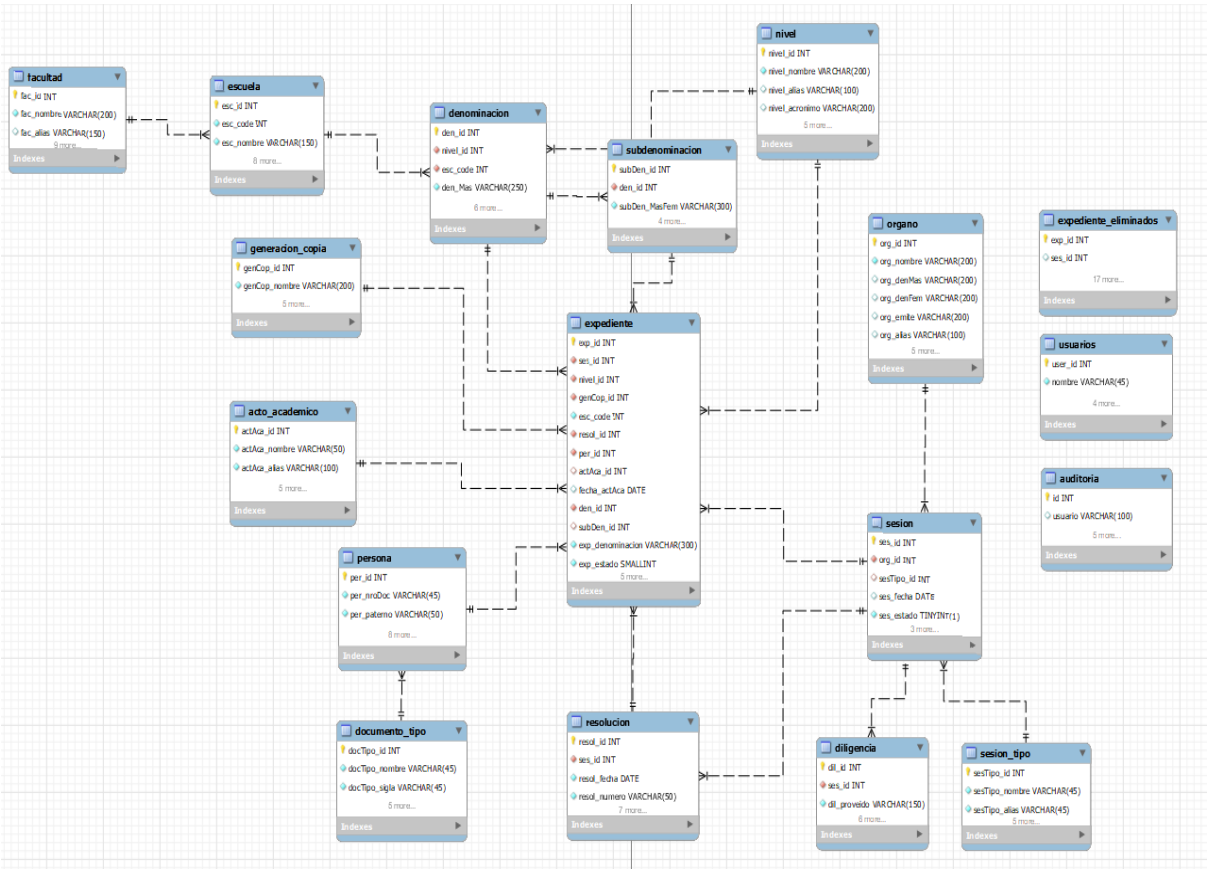
    switch($_GET["op"]){

        case "listar": ···
        /**/
        case "guardaryeditar": ···
        case "mostrar":
            $datos=$denominacion->get_denominacion_x_id($_POST["den_id"]);
            if(is_array($datos)==true and count($datos)>0){
                foreach($datos as $row){
                    $output["den_id"] = $row["den_id"];
                    $output["nivel_id"] = $row["nivel_id"];
                    $output["esc_code"] = $row["esc_code"];
                    $output["den_Mas"] = $row["den_Mas"];
                    $output["den_Fem"] = $row["den_Fem"];
                    $output["den_MasFem"] = $row["den_MasFem"];
                }
                echo json_encode($output);
            }
            break;

        case "eliminar":
            $denominacion->delete_denominacion($_POST["den_id"]);
            break;

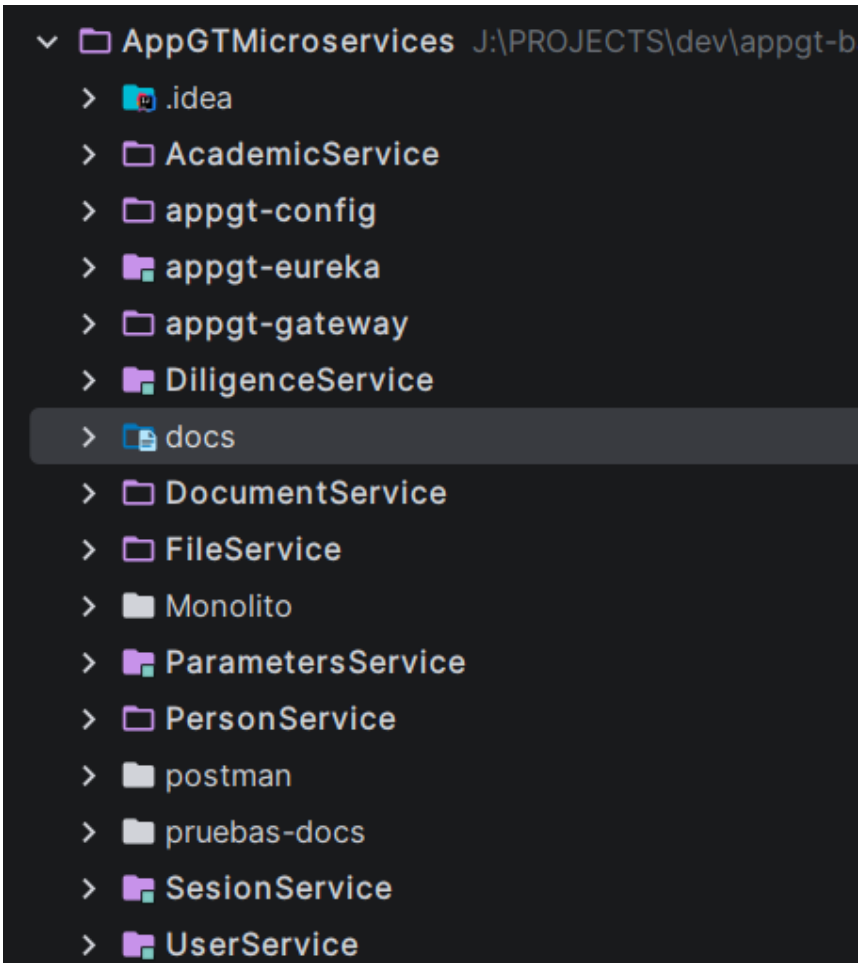
        case "cargarEscuela": ···
        case "cargarEscuelaBT": ···
        case "cargarEscuelaE": ···
        case "cargarEscuelaNew": ···
    }
?>
```

- Base de datos Compartida – MySQL



ANEXO H. PROYECTO MICROSERVICIOS

- Estructura de Proyecto Multi Módulo – Microservicios en JAVA Spring



- Configuración de Eureka Server:

appgt-gateway\...\application.yml | appgt-config\...\application.yml | appgt-eureka\...\application.yml

```yaml
server:
  port: 8081

spring:
  application:
    name: ms-eureka
  config:
    import: optional:configserver:http://localhost:8888

eureka:
  server:
    enable-self-preservation: false
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
    server-url:
      defaultZone: http://localhost:${server.port}/eureka/
```

- Configuración de Config Server:

appgt-gateway\...\application.yml | appgt-config\...\application.yml

```yaml
server:
  port: 8888

spring:
  application:
    name: appgt-config
  cloud:
    config:
      server:
        git:
          uri: https://github.com/jmab-dev/appgt-config-repo
          default-label: main
          search-paths: '{application}'

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8081/eureka/
```

- Configuración de API Gateway Final

appgt-gateway\...\application.yml | appgt-config\...\application.yml

```yaml
server:
  port: 8087

spring:
  config:
    import: optional:configserver:http://localhost:8888
  application:
    name: appgt-gateway
  cloud:
    gateway:
      globalcors:
        corsConfigurations: {1 key}
      discovery: {1 key}
      routes:
        # ========== MICROSERVICIOS ==========

        # FileService - Microservicio
        - {4 keys}

        # DocumentService - Microservicio
        - {4 keys}

        # PersonService - Microservicio
        - {4 keys}
        # SesionService - Microservicio
        - {4 keys}

        # AcademicService - Microservicio
        - {4 keys}

        # UserService - Microservicio
        - {4 keys}
```

appgt-gateway\...\application.yml × | appgt-config\...\application.yml | appgt-eureka\...\application.yml

```yaml
spring:
  cloud:
    gateway:
      routes:

        # Monolito
        - id: monolito-php-deleted
          uri: no://op
          predicates:
            - Path=/api/APP-GT/controller/**
          filters:
            - SetStatus=410
            - AddResponseHeader=X-Deprecated-Endpoint, migrated-to-microservice

        # ========== MONOLITO PHP ==========
        # Todas las rutas del monolito empiezan con /controller/
        - id: monolito-php
          uri: http://localhost:80
          predicates:
            - Path=/api/APP-GT/controller/**
          filters:
            - StripPrefix=1

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8081/eureka/
    fetchRegistry: true
    registerWithEureka: true

logging:
  level:
    org.springframework.cloud.gateway: DEBUG
    reactor.netty: DEBUG
```

- Configuración de FileService

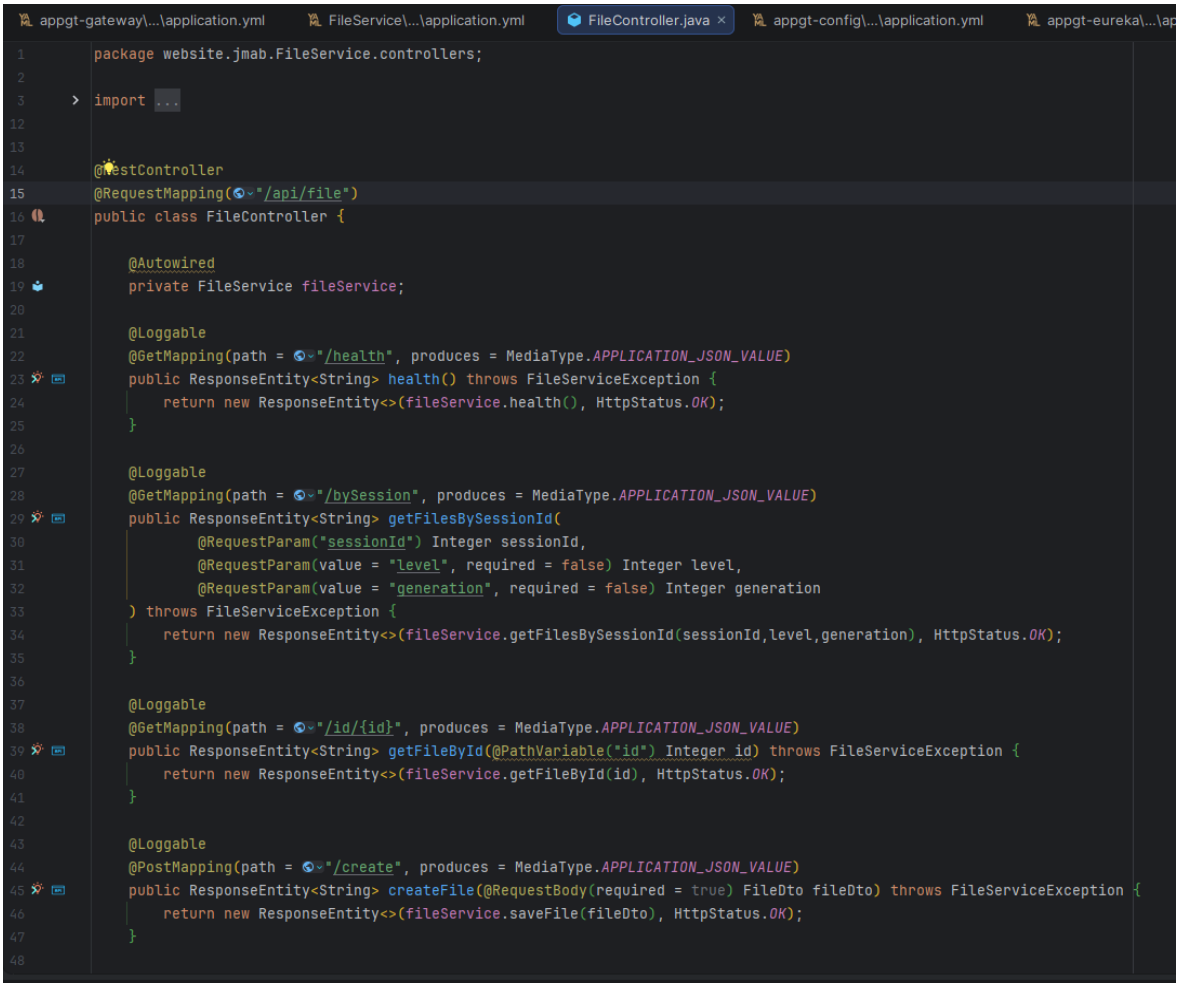
appgt-gateway\...\application.yml | FileService\...\application.yml × | appgt

```yaml
server:
  port: 8082

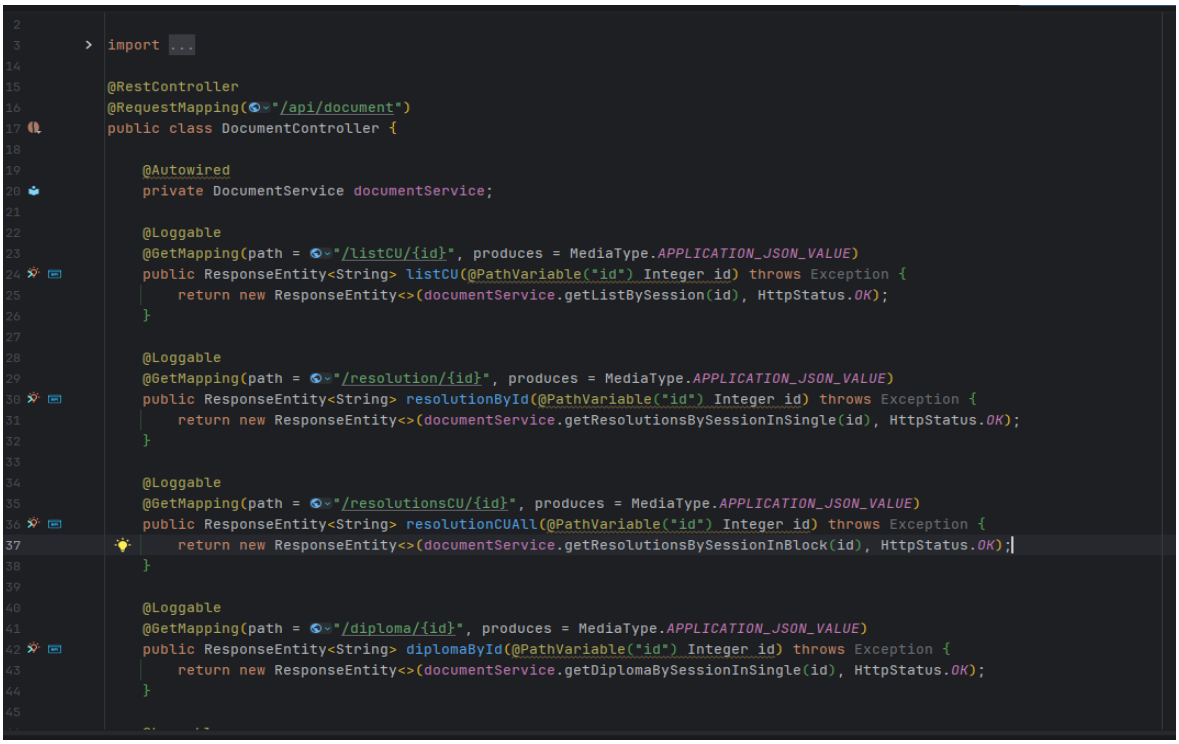
spring:
  cloud:
    gateway:
      enabled: false
  application:
    name: FileService
  config:
    import: optional:configserver:http://localhost:8888
  servlet:
    multipart:
      max-file-size: 10MB
      max-request-size: 10MB

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8081/eureka/
  instance:
    instance-id: ${spring.application.name}:${server.port}
```

- Controlador de FileService

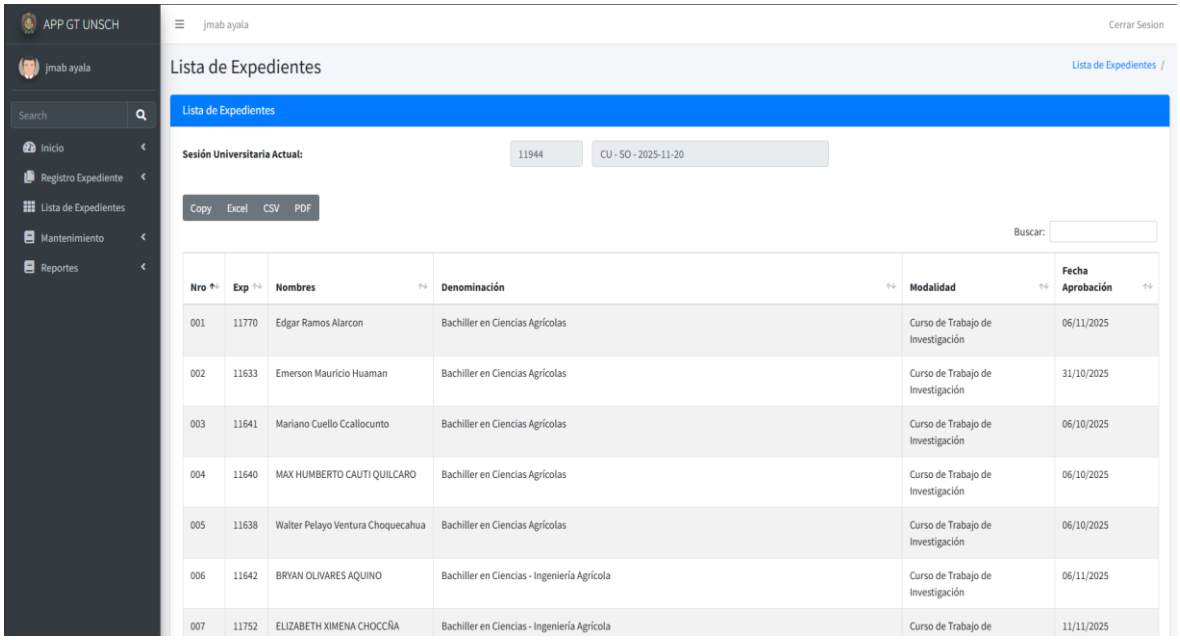


- Controlador de DocumentService

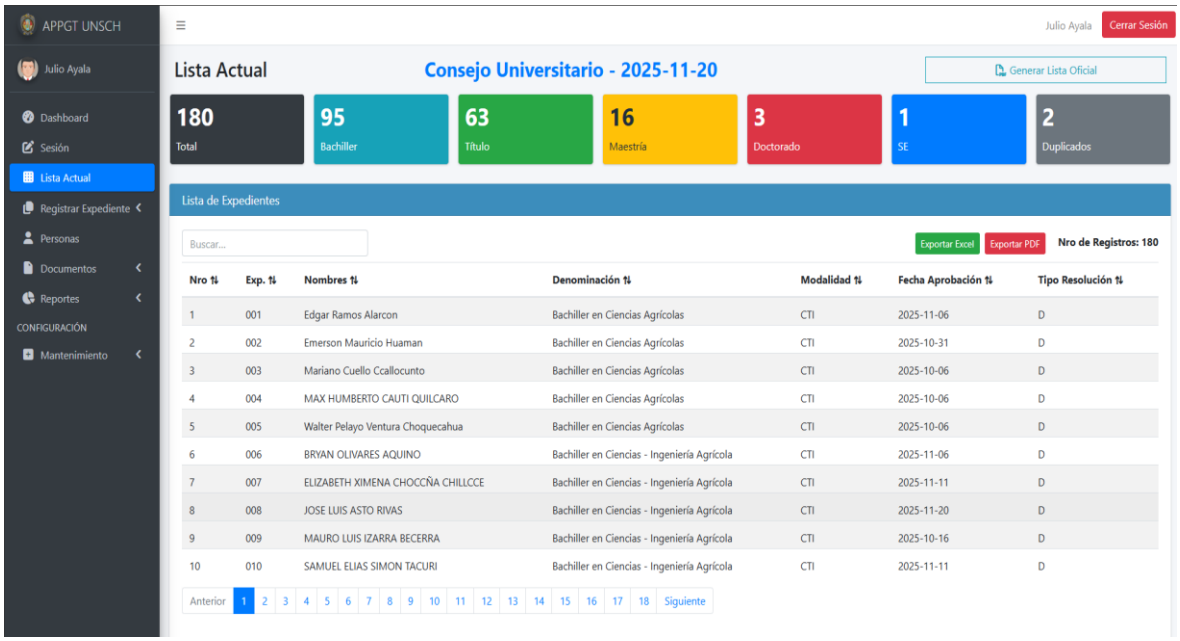


ANEXO I. VISTAS DEL SISTEMA

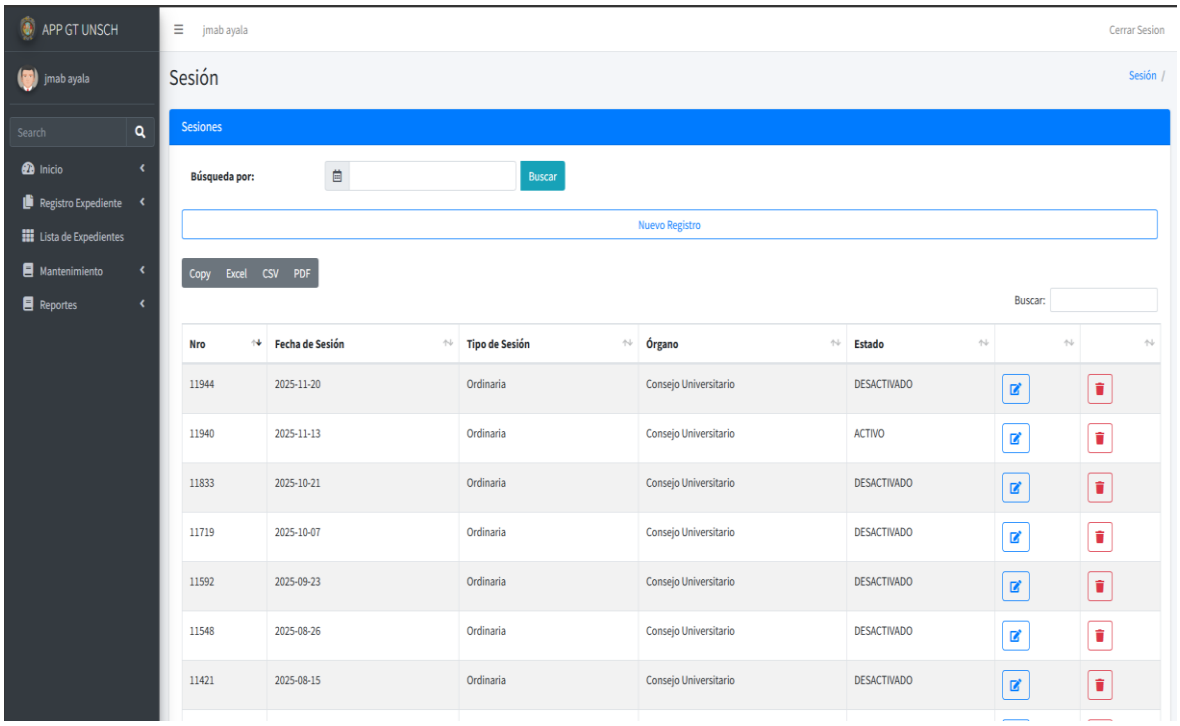
- APPGT – Vista de Listado de Expedientes Monolito



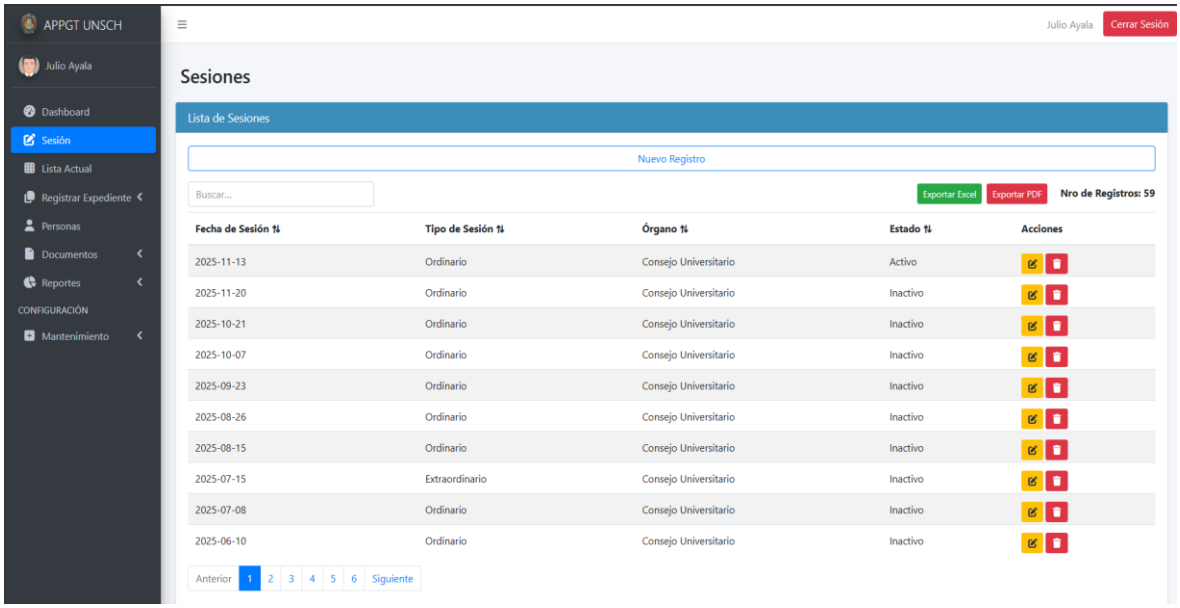
- APPGT – Vista de Listado de Expedientes Migrado



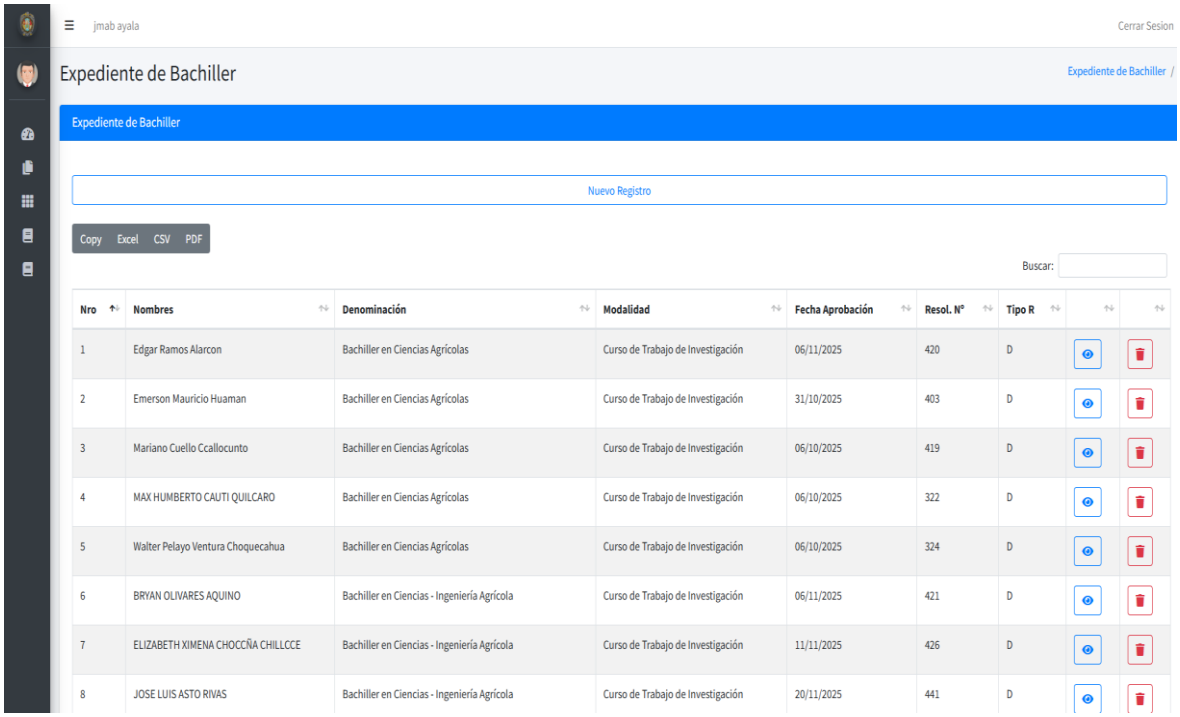
- APPGT – Vista de Sesiones Monolito

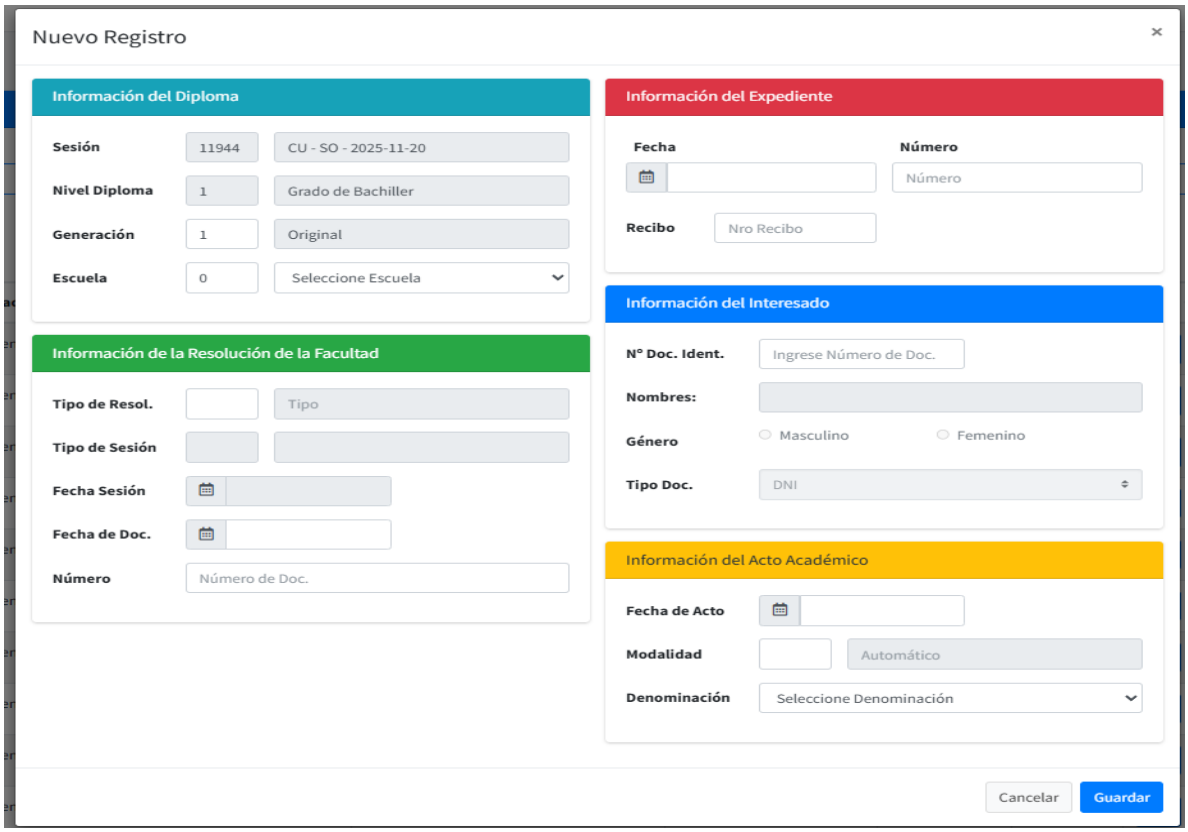


- APPGT – Vista de Sesiones Migrado

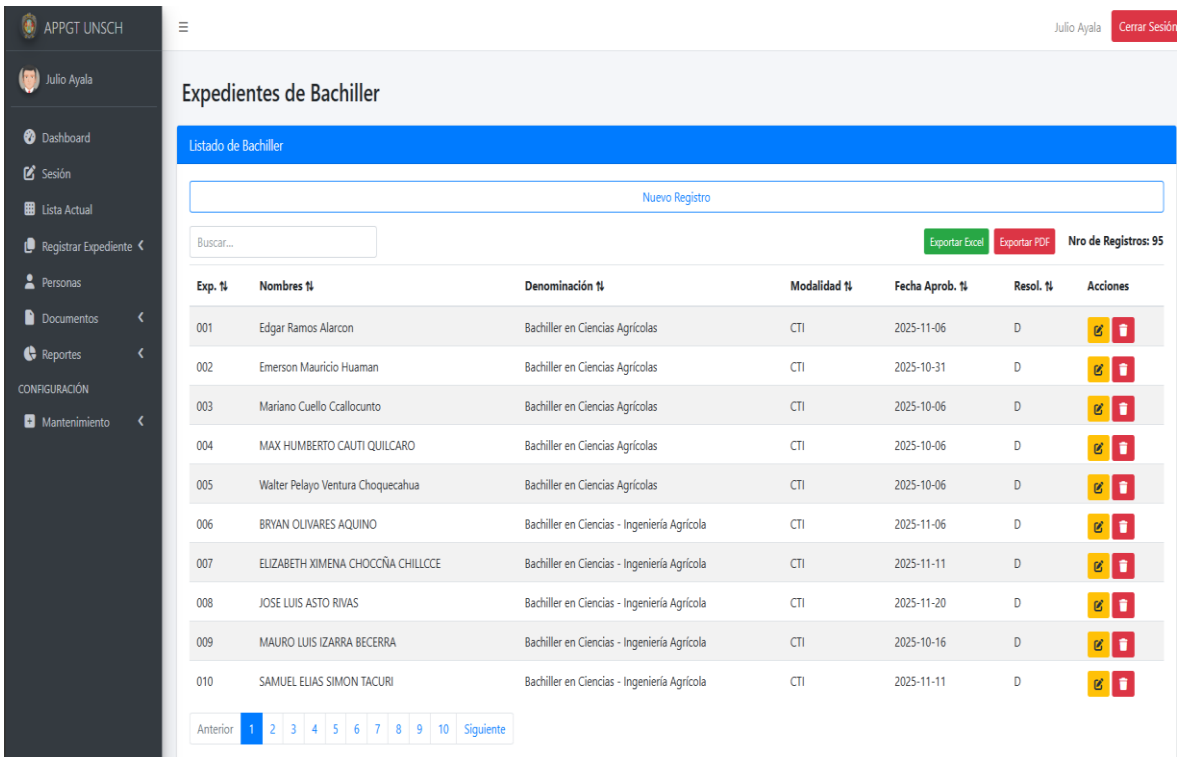


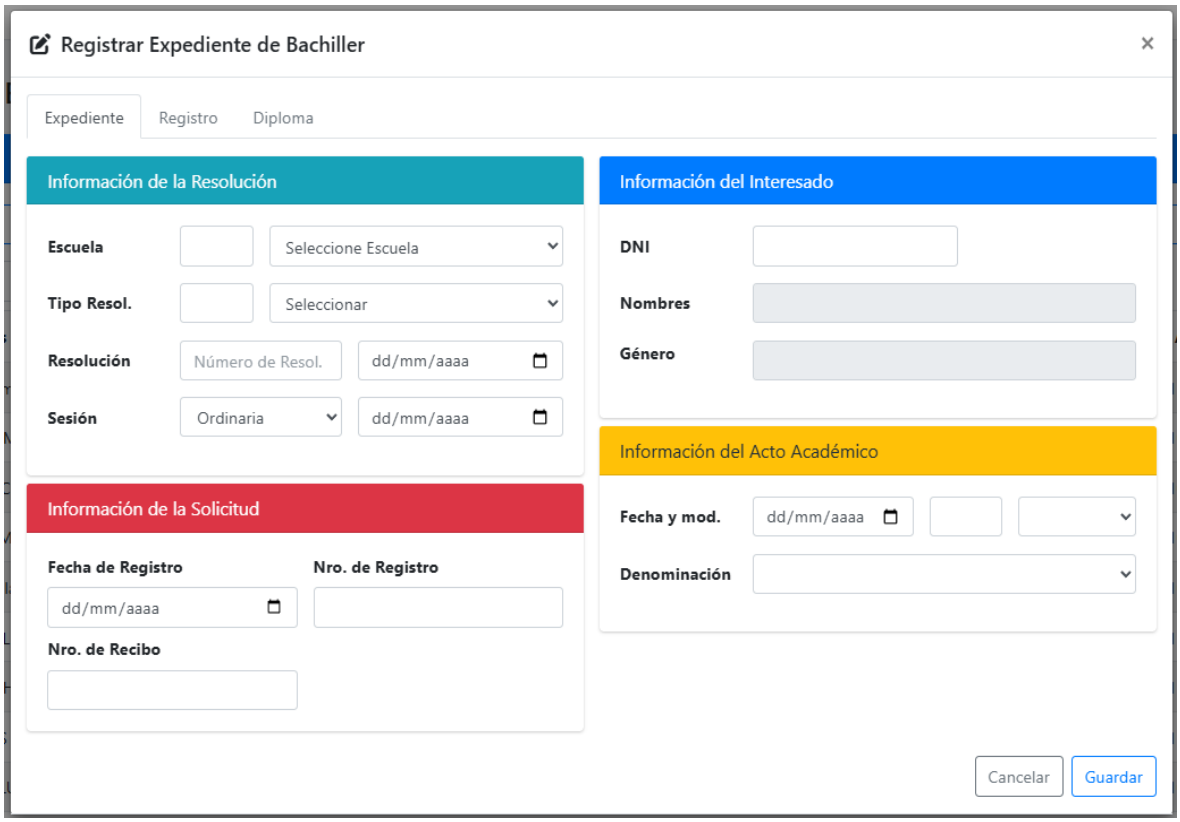
-
- APPGT – Vista de Módulo Bachiller Monolito





- APPGT – Vista de Módulo Bachiller Migrado





ANEXO J. VALIDACIÓN DEL INSTRUMENTO POR JUICIO DE EXPERTOS

Validación Nº 1 - Experto 1

VALIDACIÓN DE INSTRUMENTO DE INVESTIGACIÓN POR JUICIO DE EXPERTOS

Ficha de Análisis Documental — Patrón Strangler Fig

I. DATOS INFORMATIVOS DE LA INVESTIGACIÓN

Título de la investigación	Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios
Autor	Julio Martí Ayala Bellido
Objetivo general	Describir el proceso de aplicación e implementación del Patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios.
Variable de estudio	Aplicación del Patrón Strangler Fig (proceso de migración)

II. DATOS DEL EXPERTO (a ser completado por el experto)

Apellidos y nombres	PERALTA SOTOMAYOR KAREL
Carrera profesional	ING. DE SISTEMAS - MG. ING. INFORMATICA
Especialidad / área de experiencia	ING. SOFTWARE – GESTION DE PROYECTOS TI.
Institución donde labora	UNSCH
Cargo actual	DOCENTE.
Años de experiencia profesional	15 AÑOS

III. PROPÓSITO DE LA VALIDACIÓN

Se solicita al experto evaluar el instrumento "Ficha de Análisis Documental" (Sección VI) empleado en la investigación descrita arriba, calificando la pertinencia, claridad y suficiencia de cada campo respecto a la fase que registra (Sección VII), y emitiendo un juicio de aplicabilidad general (Sección VIII).

IV. INSTRUCCIONES PARA EL EXPERTO

Marque Sí/No en cada criterio de la Sección VII — Pertinencia: el campo es relevante para la fase que pretende registrar. Claridad: el campo está formulado con lenguaje comprensible y sin ambigüedad. Suficiencia: los campos de la sección son suficientes, en cantidad y calidad, para recoger evidencia completa.

V. VARIABLE, DIMENSIONES E INDICADORES DEL ESTUDIO

Dimensión	Indicador	Tipo de dato
X1: Transformación	Microservicio implementado	Cualitativo
	Endpoints REST implementados	Cuantitativo
	Criterios de aceptación cumplidos	Cuantitativo
	Decisiones técnicas documentadas	Cualitativo
	Historia de usuario documentada	Cualitativo
X2: Coexistencia	Rutas activas en el API Gateway por sprint	Cuantitativo
	Estrategia de base de datos	Cualitativo
	Incidencias detectadas	Cualitativo
X3: Eliminación	Condiciones verificadas antes de eliminar	Cualitativo
	Módulos PHP desactivados	Cualitativo
	Módulos restantes en el monolito por sprint	Cuantitativo

Variable: Aplicación del Patrón Strangler Fig (proceso de migración).

VII. MATRIZ DE VALIDACIÓN DE CONTENIDO

CAMPO DEL INSTRUMENTO	PERTINENCIA[1]	CLARIDAD[2]	SUFICIENCIA[3]	OBSERVACIONES
SECCIÓN B — TRANSFORMACIÓN				
Historia de usuario	Si	Si	Si	
Endpoints implementados (tabla)	Si	Si	Si	
Criterios de aceptación (tabla)	Si	Si	Si	
Observaciones	Si	Si	Si	
SECCIÓN C — COEXISTENCIA Y ELIMINACIÓN				
C.1 — Enrutamiento en el API Gateway (tabla)	Si	Si	Si	
Estrategia de BD	Si	Si	Si	
Incidencias	Si	Si	Si	
C.2 — Condiciones verificadas antes de eliminar (tabla)	Si	Si	Si	
Proceso de retiro	Si	Si	Si	
Monolito restante	Si	Si	Si	
Observaciones finales	Si	Si	Si	

[1] El campo es relevante y corresponde a la fase o dimensión que pretende registrar.

[2] El campo está formulado con lenguaje comprensible, preciso y sin ambigüedad.

[3] El conjunto de campos de la sección es suficiente, en cantidad y calidad, para recoger evidencia completa de la fase correspondiente.

VIII. JUICIO DE APLICABILIDAD GENERAL DEL INSTRUMENTO

(X) Aplicable ( ) Aplicable después de incorporar observaciones ( ) No aplicable

Observaciones generales:
NINGUNA

IX. CONSTANCIA DE VALIDACIÓN

Quien suscribe hace constar que ha revisado el instrumento descrito en este documento y emite el juicio de validez consignado en la Sección VIII.

Lugar y fecha: AYACUCHO 6 DE JULIO 2026.



Firma del experto

Validación Nº 1 - Experto 2

VALIDACIÓN DE INSTRUMENTO DE INVESTIGACIÓN POR JUICIO DE EXPERTOS

Ficha de Análisis Documental — Patrón Strangler Fig

I. DATOS INFORMATIVOS DE LA INVESTIGACIÓN

Título de la investigación	Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios
Autor	Julio Martí Ayala Bellido
Objetivo general	Describir el proceso de aplicación e implementación del Patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios.
Variable de estudio	Aplicación del Patrón Strangler Fig (proceso de migración)

II. DATOS DEL EXPERTO (a ser completado por el experto)

Apellidos y nombres	Terraza Huaman Edem Jersson.
Carrera profesional	Ing. de Sistemas - M6. Seguridad informatica
Especialidad / área de experiencia	Seguridad Informatica - Software - TIC.
Institución donde labora	UNSCH
Cargo actual	Docente.
Años de experiencia profesional	4 años de experiencia

III. PROPÓSITO DE LA VALIDACIÓN

Se solicita al experto evaluar el instrumento "Ficha de Análisis Documental" (Sección VI) empleado en la investigación descrita arriba, calificando la pertinencia, claridad y suficiencia de cada campo respecto a la fase que registra (Sección VII), y emitiendo un juicio de aplicabilidad general (Sección VIII).

IV. INSTRUCCIONES PARA EL EXPERTO

Marque Sí/No en cada criterio de la Sección VII — Pertinencia: el campo es relevante para la fase que pretende registrar. Claridad: el campo está formulado con lenguaje comprensible y sin ambigüedad. Suficiencia: los campos de la sección son suficientes, en cantidad y calidad, para recoger evidencia completa.

V. VARIABLE, DIMENSIONES E INDICADORES DEL ESTUDIO

Dimensión	Indicador	Tipo de dato
X1: Transformación	Microservicio implementado	Cualitativo
	Endpoints REST implementados	Cuantitativo
	Criterios de aceptación cumplidos	Cuantitativo
	Decisiones técnicas documentadas	Cualitativo
	Historia de usuario documentada	Cualitativo
X2: Coexistencia	Rutas activas en el API Gateway por sprint	Cuantitativo
	Estrategia de base de datos	Cualitativo
	Incidencias detectadas	Cualitativo
X3: Eliminación	Condiciones verificadas antes de eliminar	Cualitativo
	Módulos PHP desactivados	Cualitativo
	Módulos restantes en el monolito por sprint	Cuantitativo

Variable: Aplicación del Patrón Strangler Fig (proceso de migración).

VII. MATRIZ DE VALIDACIÓN DE CONTENIDO

CAMPO DEL INSTRUMENTO	PERTINENCIA[1]	CLARIDAD[2]	SUFICIENCIA[3]	OBSERVACIONES
SECCIÓN B — TRANSFORMACIÓN				
Historia de usuario	Si	Si	Si	
Endpoints implementados (tabla)	Si	Si	Si	
Criterios de aceptación (tabla)	Si	Si	Si	
Observaciones	Si	Si	Si	
SECCIÓN C — COEXISTENCIA Y ELIMINACIÓN				
C.1 — Enrutamiento en el API Gateway (tabla)	Si	Si	Si	
Estrategia de BD	Si	Si	Si	
Incidencias	Si	Si	Si	
C.2 — Condiciones verificadas antes de eliminar (tabla)	Si	Si	Si	
Proceso de retiro	Si	Si	Si	
Monolito restante	Si	Si	Si	
Observaciones finales	Si	Si	Si	

[1] El campo es relevante y corresponde a la fase o dimensión que pretende registrar.

[2] El campo está formulado con lenguaje comprensible, preciso y sin ambigüedad.

[3] El conjunto de campos de la sección es suficiente, en cantidad y calidad, para recoger evidencia completa de la fase correspondiente.

VIII. JUICIO DE APLICABILIDAD GENERAL DEL INSTRUMENTO

(X) Aplicable ( ) Aplicable después de incorporar observaciones ( ) No aplicable

Observaciones generales:
Ninguna.

IX. CONSTANCIA DE VALIDACIÓN

Quien suscribe hace constar que ha revisado el instrumento descrito en este documento y emite el juicio de validez consignado en la Sección VIII.

Lugar y fecha:

Ayacucho 3 de Julio 2026.

Firma del experto

Validación Nº 3 - Experto 3

VALIDACIÓN DE INSTRUMENTO DE INVESTIGACIÓN POR JUICIO DE EXPERTOS

Ficha de Análisis Documental — Patrón Strangler Fig

I. DATOS INFORMATIVOS DE LA INVESTIGACIÓN

Título de la Investigación	Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios
Autor	Julio Martí Ayala Bellido
Objetivo general	Describir el proceso de aplicación e implementación del Patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios.
Variable de estudio	Aplicación del Patrón Strangler Fig (proceso de migración)

II. DATOS DEL EXPERTO (a ser completado por el experto)

Apellidos y nombres	Vargas Perales Yrvin Orlando
Carrera profesional	Ingeniería de Sistemas e Informática
Especialidad / área de experiencia	Ingeniería de Software, Microservicios, Liderazgo de proyectos tecnológicos, arquitectura de soluciones, desarrollo de software Full Stack, integración de plataformas e-commerce y transformación digital.
Institución donde labora	Cencosud Perú SA
Cargo actual	Delivery Lead
Años de experiencia profesional	14 años

III. PROPÓSITO DE LA VALIDACIÓN

Se solicita al experto evaluar el instrumento "Ficha de Análisis Documental" (Sección VI) empleado en la investigación descrita arriba, calificando la pertinencia, claridad y suficiencia de cada campo respecto a la fase que registra (Sección VII), y emitiendo un juicio de aplicabilidad general (Sección VIII).

IV. INSTRUCCIONES PARA EL EXPERTO

Marque Sí/No en cada criterio de la Sección VII — Pertinencia: el campo es relevante para la fase que pretende registrar. Claridad: el campo está formulado con lenguaje comprensible y sin ambigüedad. Suficiencia: los campos de la sección son suficientes, en cantidad y calidad, para recoger evidencia completa.

V. VARIABLE, DIMENSIONES E INDICADORES DEL ESTUDIO

Dimensión	Indicador	Tipo de dato
X1: Transformación	Microservicio implementado	Cualitativo
	Endpoints REST implementados	Cuantitativo
	Criterios de aceptación cumplidos	Cuantitativo
	Decisiones técnicas documentadas	Cualitativo
	Historia de usuario documentada	Cualitativo
X2: Coexistencia	Rutas activas en el API Gateway por sprint	Cuantitativo
	Estrategia de base de datos	Cualitativo
	Incidencias detectadas	Cualitativo
X3: Eliminación	Condiciones verificadas antes de eliminar	Cualitativo
	Módulos desactivados	Cualitativo
	Módulos restantes en el monolito por sprint	Cuantitativo

Variable: Aplicación del Patrón Strangler Fig (proceso de migración).

VII. MATRIZ DE VALIDACIÓN DE CONTENIDO

CAMPO DEL INSTRUMENTO	PERTINENCIA[1]	CLARIDAD[2]	SUFICIENCIA[3]	OBSERVACIONES
SECCIÓN B — TRANSFORMACIÓN				
Historia de usuario	Si	Si	Si	
Endpoints implementados (tabla)	Si	Si	Si	
Criterios de aceptación (tabla)	Si	Si	Si	
Observaciones	Si	Si	Si	
SECCIÓN C — COEXISTENCIA Y ELIMINACIÓN				
C.1 — Enrutamiento en el API Gateway (tabla)	Si	Si	Si	
Estrategia de BD	Si	Si	Si	
Incidencias	Si	Si	Si	
C.2 — Condiciones verificadas antes de eliminar (tabla)	Si	Si	Si	
Proceso de retiro	Si	Si	Si	
Monolito restante	Si	Si	Si	Podría cuantificarse en número de módulos restantes, no solo de forma descriptiva.
Observaciones finales	Si	Si	Si	

[1] El campo es relevante y corresponde a la fase o dimensión que pretende registrar.
[2] El campo está formulado con lenguaje comprensible, preciso y sin ambigüedad.
[3] El conjunto de campos de la sección es suficiente, en cantidad y calidad, para recoger evidencia completa de la fase correspondiente.

VIII. JUICIO DE APLICABILIDAD GENERAL DEL INSTRUMENTO

(X) Aplicable ( ) Aplicable después de incorporar observaciones ( ) No aplicable

Observaciones generales:
Ninguna de fondo. El instrumento recoge de manera completa la evidencia de las tres fases del patrón Strangler Fig; las sugerencias anotadas son opcionales y no comprometen su validez. Asimismo, el instrumento presenta una estructura similar a los formatos de documentación empleados en procesos de migración.

IX. CONSTANCIA DE VALIDACIÓN

Quien suscribe hace constar que ha revisado el instrumento descrito en este documento y emite el juicio de validez consignado en la Sección VIII.

Lugar y fecha: Lima, 03 de julio de 2026



Yrvin Vargas Perales
Delivery Lead - ExCl
Gerencia de IT Perú
www.cencosud.com
Síguenos en Linkedin como Cencosud S.A

Validación Nº 4 – Experto 4

VALIDACIÓN DE INSTRUMENTO DE INVESTIGACIÓN POR JUICIO DE EXPERTOS

Ficha de Análisis Documental — Patrón Strangler Fig

I. DATOS INFORMATIVOS DE LA INVESTIGACIÓN

Título de la investigación	Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios
Autor	Julio Martí Ayala Bellido
Objetivo general	Describir el proceso de aplicación e implementación del Patrón Strangler Fig en la migración de un sistema web monolítico hacia una arquitectura de microservicios.
Variable de estudio	Aplicación del Patrón Strangler Fig (proceso de migración)

II. DATOS DEL EXPERTO (a ser completado por el experto)

Apellidos y nombres	Tijero Scamarone Daniella
Carrera profesional	Ingeniería de Sistemas
Especialidad / área de experiencia	Transformación Digital, Gestión de Tecnologías de Información, Modernización de Sistemas, Ciclo de Vida del Software
Institución donde labora	Cencosud Perú SA
Cargo actual	Sub Gerente IT
Años de experiencia profesional	25 años

III. PROPÓSITO DE LA VALIDACIÓN

Se solicita al experto evaluar el instrumento "Ficha de Análisis Documental" (Sección VI) empleado en la investigación descrita arriba, calificando la pertinencia, claridad y suficiencia de cada campo respecto a la fase que registra (Sección VII), y emitiendo un juicio de aplicabilidad general (Sección VIII).

IV. INSTRUCCIONES PARA EL EXPERTO

Marque Sí/No en cada criterio de la Sección VII — Pertinencia: el campo es relevante para la fase que pretende registrar. Claridad: el campo está formulado con lenguaje comprensible y sin ambigüedad. Suficiencia: los campos de la sección son suficientes, en cantidad y calidad, para recoger evidencia completa.

V. VARIABLE, DIMENSIONES E INDICADORES DEL ESTUDIO

Dimensión	Indicador	Tipo de dato
X1: Transformación	Microservicio implementado	Cualitativo
	Endpoints REST implementados	Cuantitativo
	Criterios de aceptación cumplidos	Cuantitativo
	Decisiones técnicas documentadas	Cualitativo
	Historia de usuario documentada	Cualitativo
X2: Coexistencia	Rutas activas en el API Gateway por sprint	Cuantitativo
	Estrategia de base de datos	Cualitativo
	Incidencias detectadas	Cualitativo
X3: Eliminación	Condiciones verificadas antes de eliminar	Cualitativo
	Módulos PHP desactivados	Cualitativo
	Módulos restantes en el monolito por sprint	Cuantitativo

Variable: Aplicación del Patrón Strangler Fig (proceso de migración).

VII. MATRIZ DE VALIDACIÓN DE CONTENIDO

CAMPO DEL INSTRUMENTO	PERTINENCIA[1]	CLARIDAD[2]	SUFICIENCIA[3]	OBSERVACIONES
SECCIÓN B — TRANSFORMACIÓN				
Historia de usuario	Si	Si	Si	Verificar que el formato siga siempre la estructura "Como… quiero …para …" en todos los sprints.
Endpoints implementados (tabla)	Si	Si	Si	
Criterios de aceptación (tabla)	Si	Si	Si	
Observaciones	Si	No	Si	Se sugiere precisar si el campo debe registrar información técnica o comentarios generales del sprint.
SECCIÓN C — COEXISTENCIA Y ELIMINACIÓN				
C.1 — Enrutamiento en el API Gateway (tabla)	Si	Si	Si	
Estrategia de BD	Si	Si	Si	
Incidencias	Si	Si	Si	
C.2 — Condiciones verificadas antes de eliminar (tabla)	Si	Si	Si	
Proceso de retiro	Si	Si	Si	
Monolito restante	Si	Si	Si	
Observaciones finales	Si	No	Si	

[1] El campo es relevante y corresponde a la fase o dimensión que pretende registrar.
[2] El campo está formulado con lenguaje comprensible, preciso y sin ambigüedad.
[3] El conjunto de campos de la sección es suficiente, en cantidad y calidad, para recoger evidencia completa de la fase correspondiente.

VIII. JUICIO DE APLICABILIDAD GENERAL DEL INSTRUMENTO

( ) Aplicable (X) Aplicable después de incorporar observaciones ( ) No aplicable

Observaciones generales:
El instrumento guarda correspondencia con la matriz de variable, dimensiones e indicadores. La observación registrada en claridad corresponde a un aspecto de forma del campo 'Observaciones', que no altera la validez de contenido del instrumento

IX. CONSTANCIA DE VALIDACIÓN

Quien suscribe hace constar que ha revisado el instrumento descrito en este documento y emite el juicio de validez consignado en la Sección VIII.

Lugar y fecha: 07 de julio de 2026



Firma de la experta

ANEXO K. TABLA DE CÁLCULO V DE AIKEN

CÁLCULO DEL COEFICIENTE V DE AIKEN — Ficha de Análisis Documental (Patrón Strangler Fig)

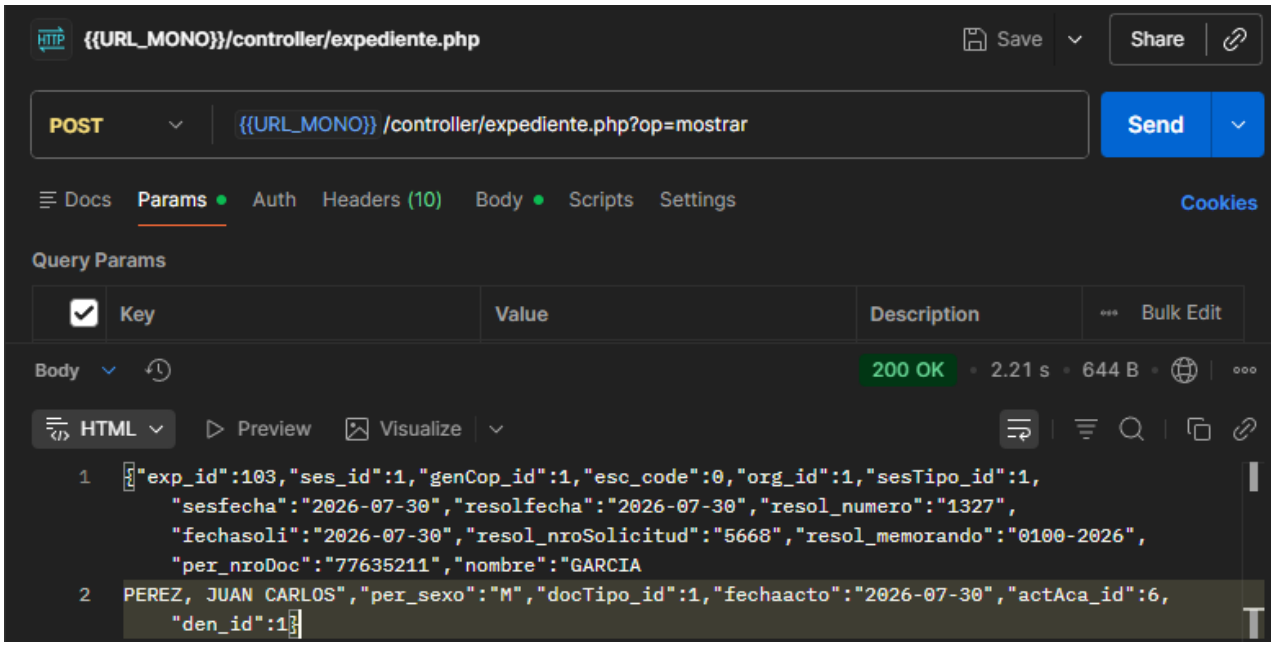
Escala dicotómica: Sí = 1, No = 0. n = 4 jueces, c = 2 categorías. Fórmula: V = S / [n · (c − 1)], con S = suma de acuerdos.
Jueces:
J1 = Karel P. S. (UNSCH)
J2 = Edem T. H. (UNSCH)
J3 = Yrvin V. P. (Cencosud)
J4 = Daniella T. S. (Cencosud)

N°	Campo del instrumento	Fase / Sección	PERTINENCIA					CLARIDAD					SUFICIENCIA					V promedio ítem
			J1	J2	J3	J4	V	J1	J2	J3	J4	V	J1	J2	J3	J4	V	
1	Historia de usuario	B — Transformación	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
2	Endpoints implementados (tabla)	B — Transformación	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
3	Criterios de aceptación (tabla)	B — Transformación	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
4	Observaciones (Sección B)	B — Transformación	1	1	1	1	1,00	1	1	1	0	0,75	1	1	1	1	1,00	0,92
5	Enrutamiento en el API Gateway (tabla)	C.1 — Coexistencia	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
6	Estrategia de BD	C.1 — Coexistencia	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
7	Incidencias	C.1 — Coexistencia	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
8	Condiciones verificadas antes de eliminar (tabla)	C.2 — Eliminación	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
9	Proceso de retiro	C.2 — Eliminación	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
10	Monolito restante	C.2 — Eliminación	1	1	1	1	1,00	1	1	1	1	1,00	1	1	1	1	1,00	1,00
11	Observaciones finales	C.2 — Eliminación	1	1	1	1	1,00	1	1	1	0	0,75	1	1	1	1	1,00	0,92
	V de Aiken general del instrumento						**1,00**					**0,95**					**1,00**	**0,98**

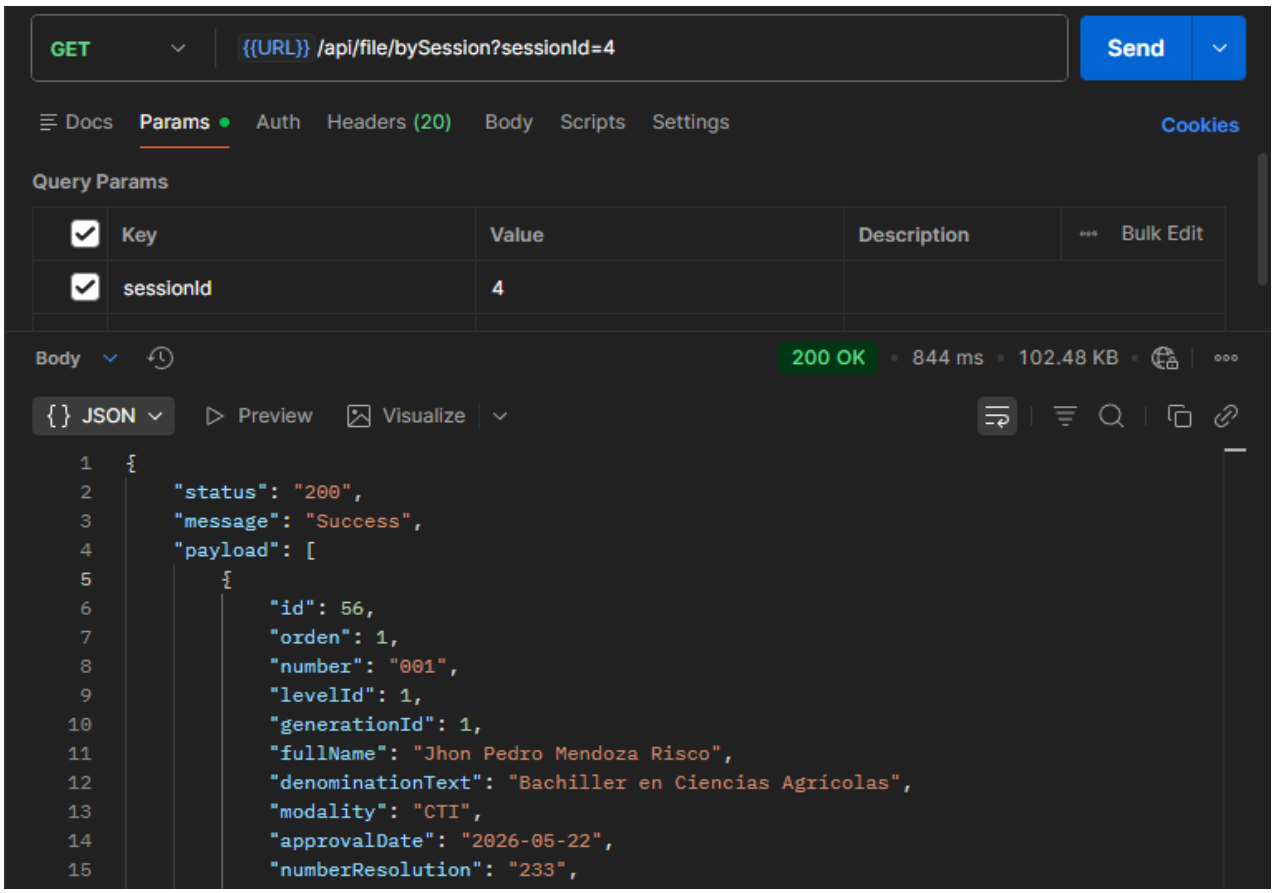
Interpretación: V ≥ 0.70 se considera evidencia aceptable de validez de contenido (Aiken, 1985). Resultado: Pertinencia = 1.00, Claridad = 0.95, Suficiencia = 1.00, V general = 0.98.

ANEXO L. CAPTURAS DE TIEMPOS DE RESPUESTA

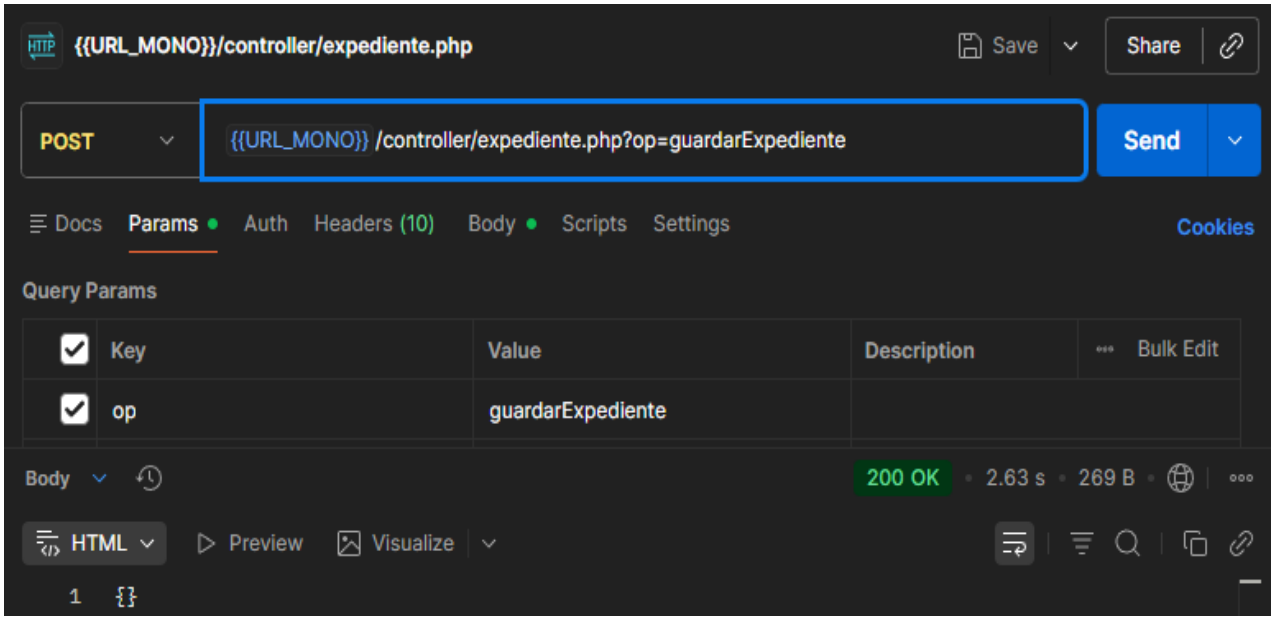
Listar Expedientes – Monolito



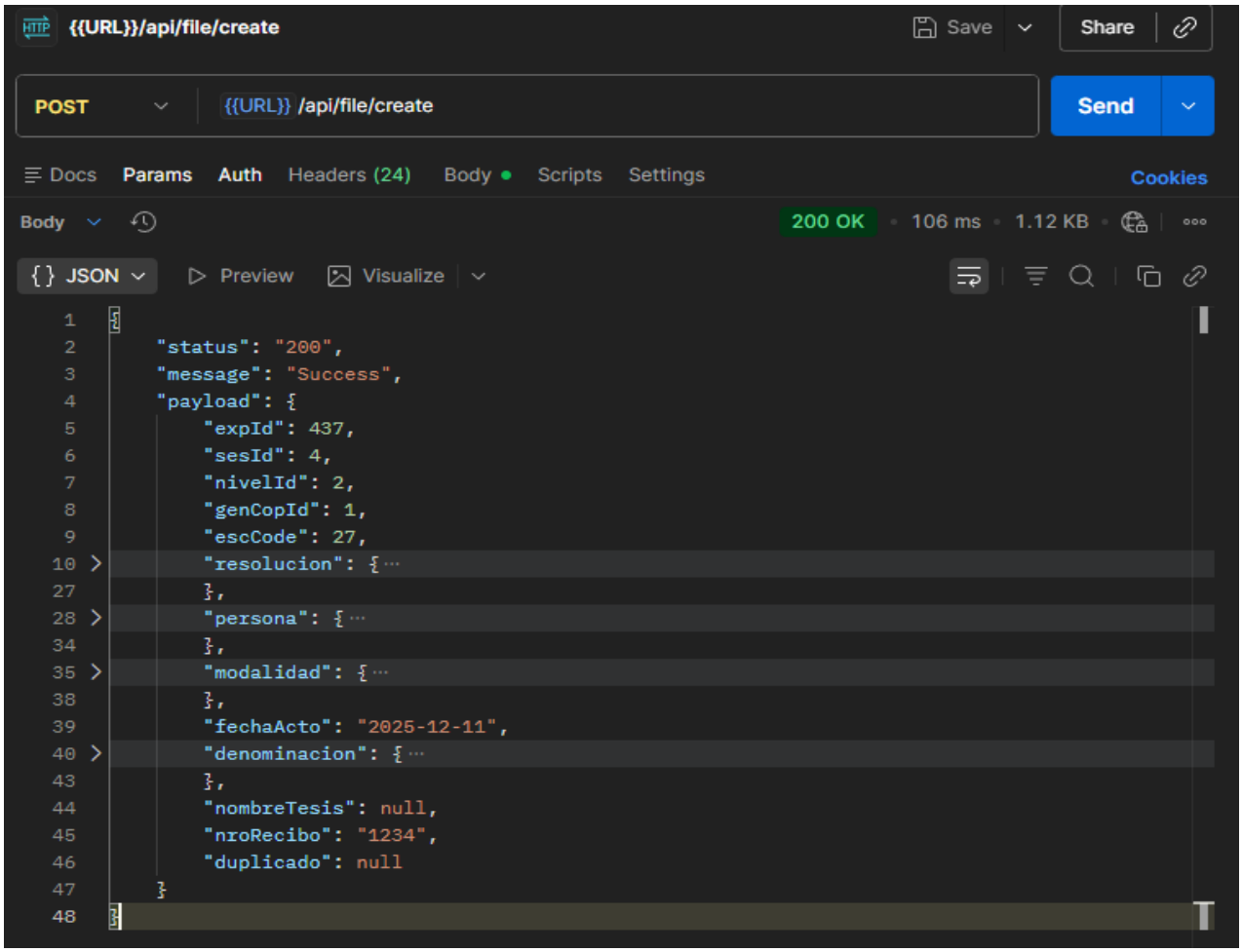
Listar Expedientes – Microservicio



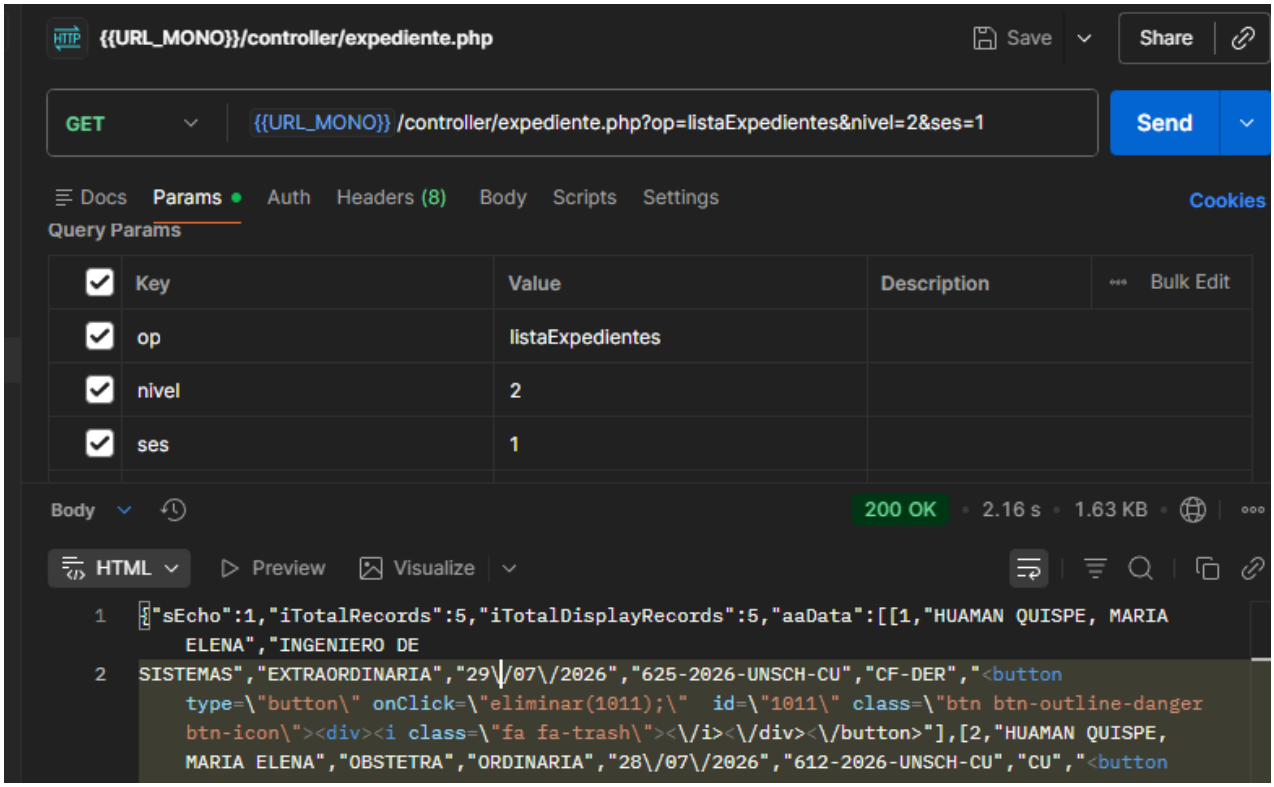
Registrar Expediente – Monolito:



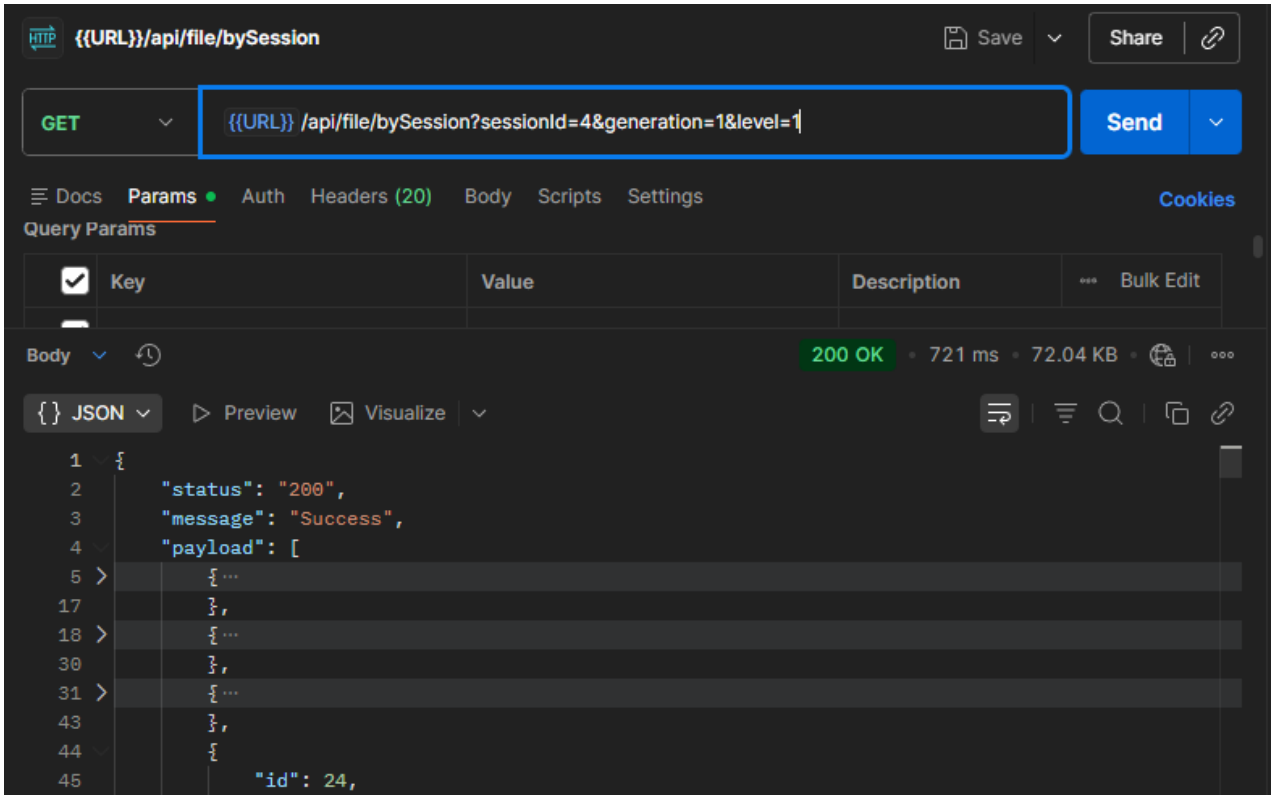
Registrar Expediente – Microservicio



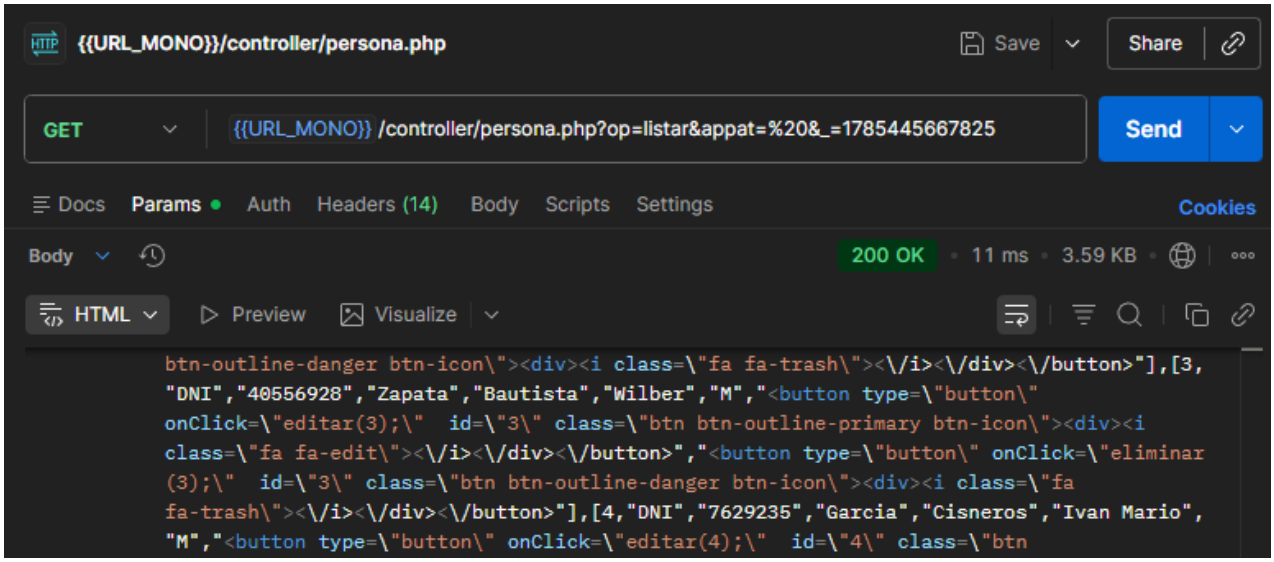
Listar Expedientes por Nivel – Monolito



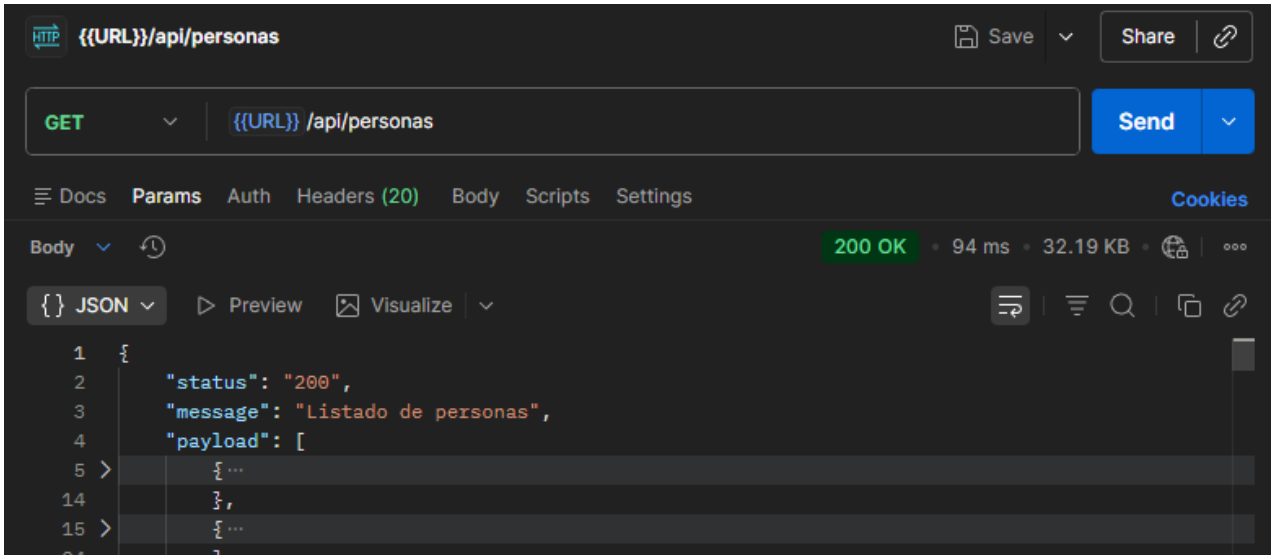
Listar Expedientes por Nivel – Microservicio



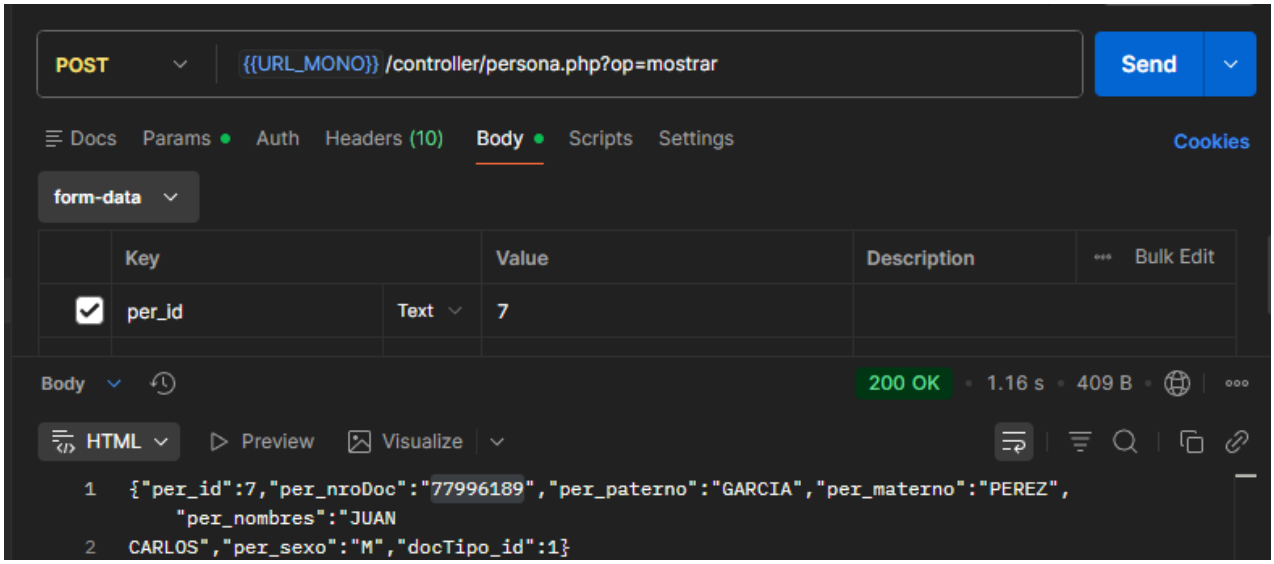
Listar Personas – Monolito



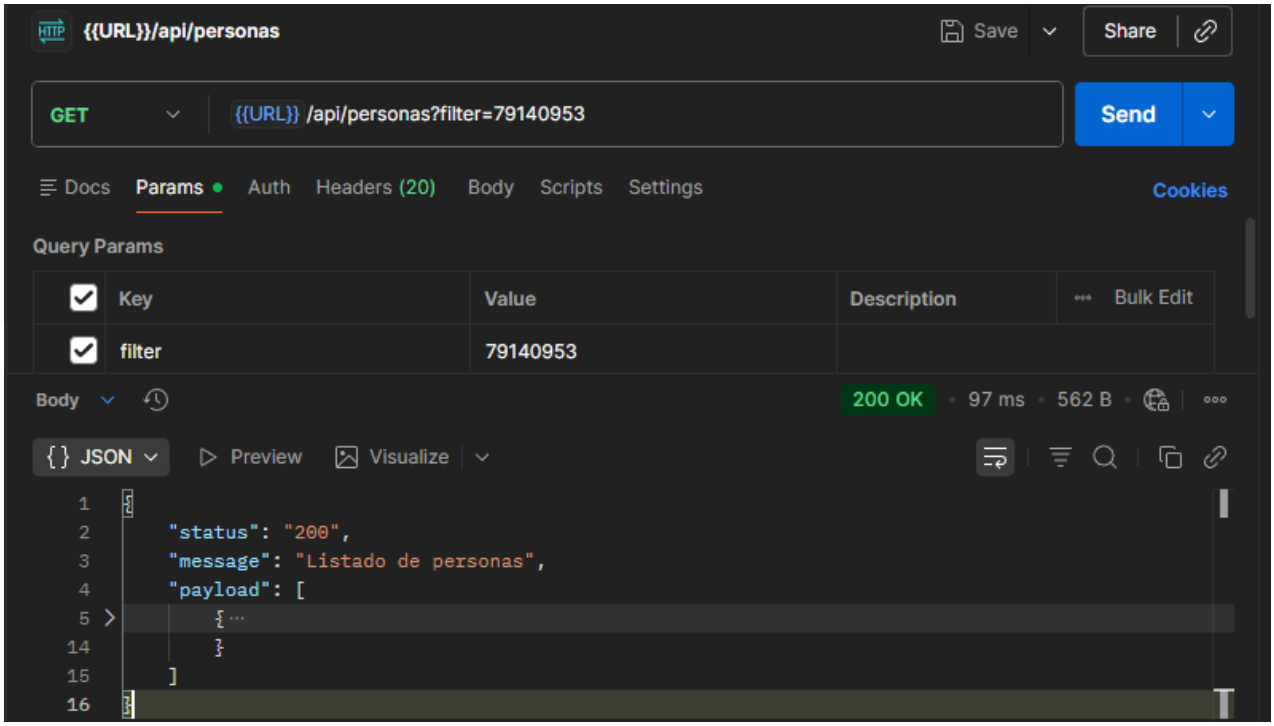
Listar Personas – Microservicio



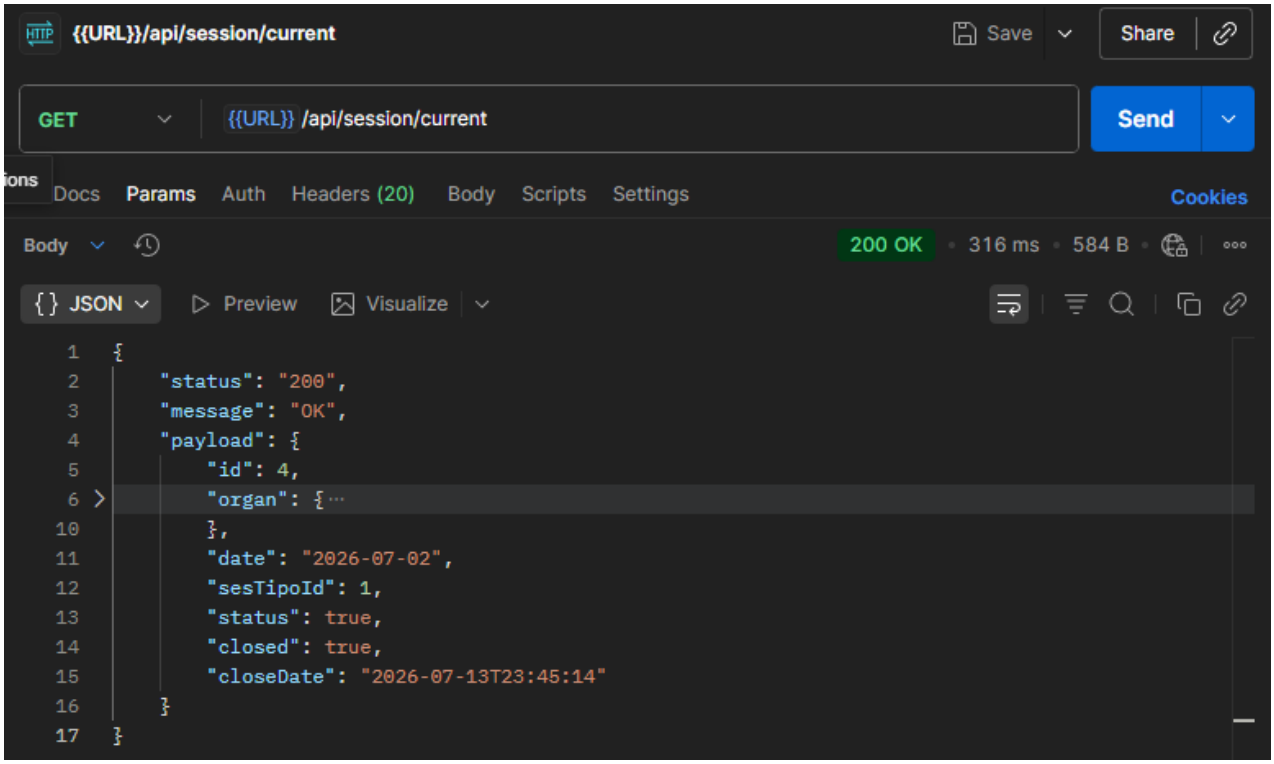
Buscar Persona por Id – Monolito



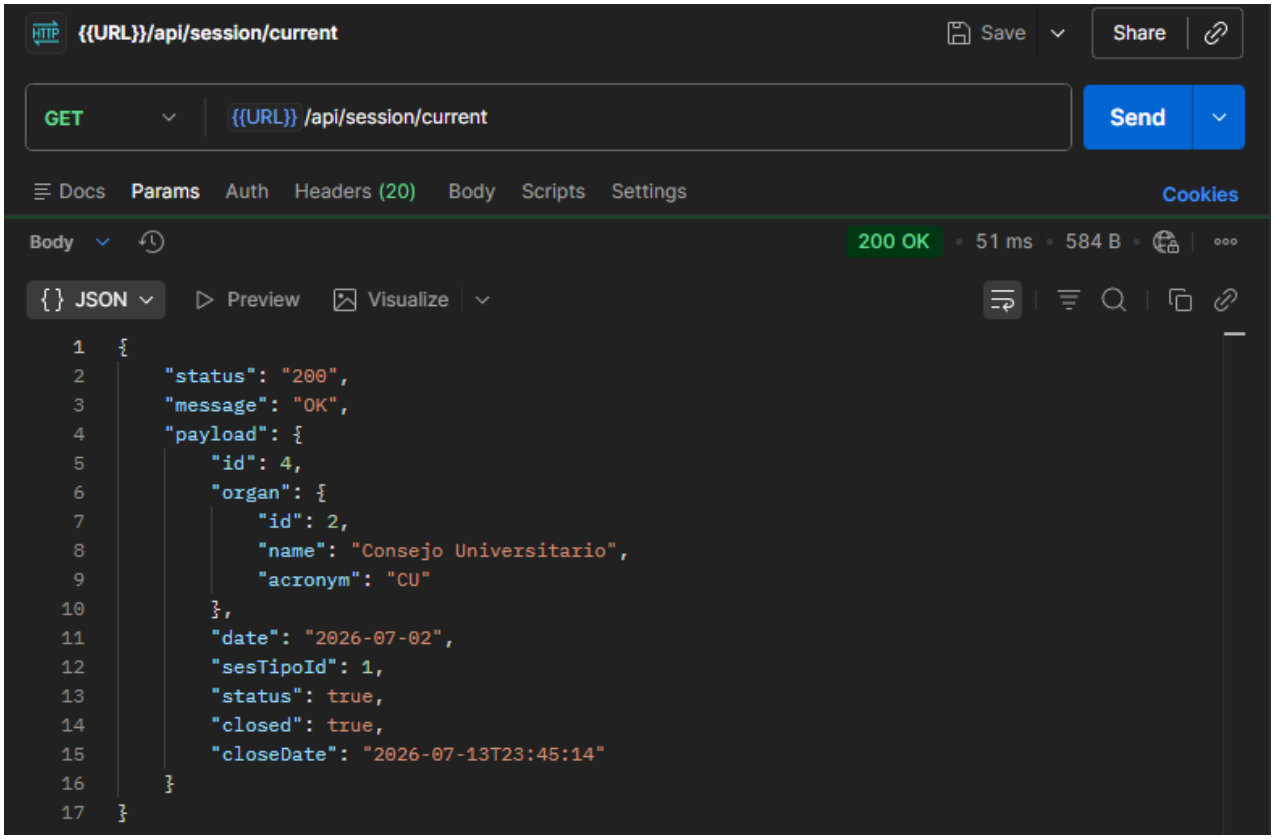
Buscar Persona por Id – Microservicio.



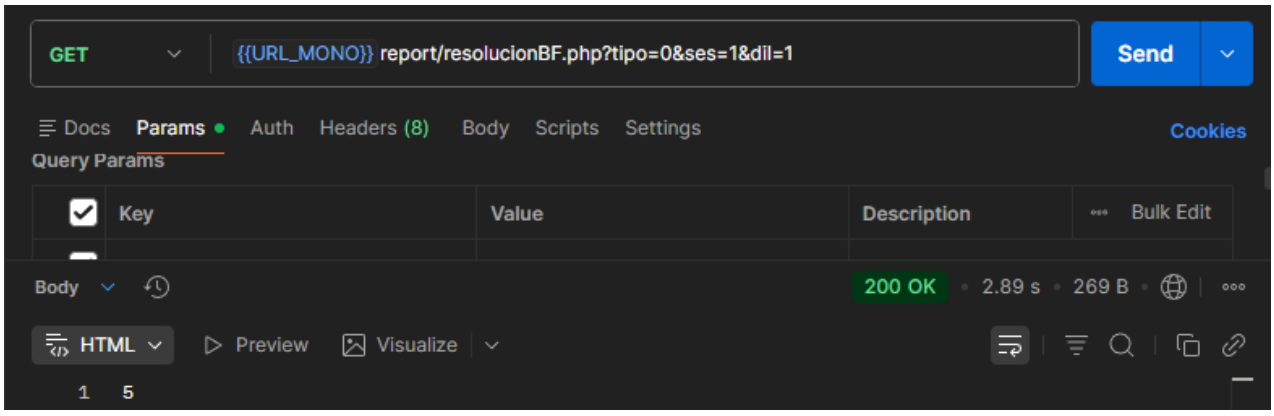
Sesión Actual – Monolito



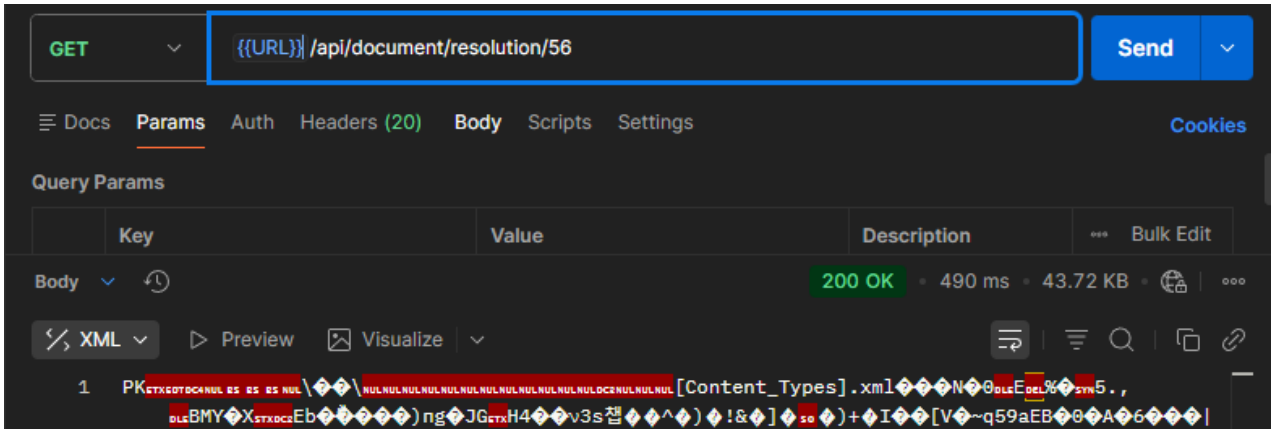
Sesión Actual – Microservicio



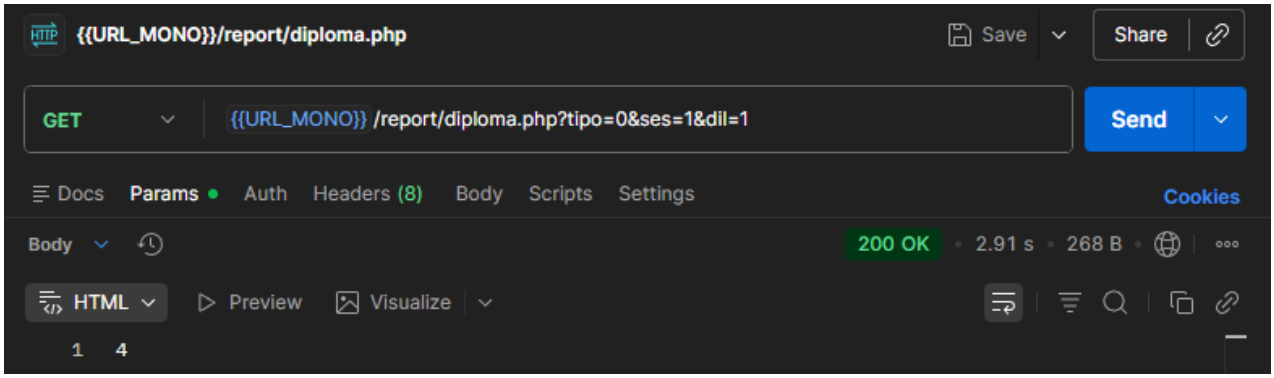
Generar Documento Resolución – Monolito



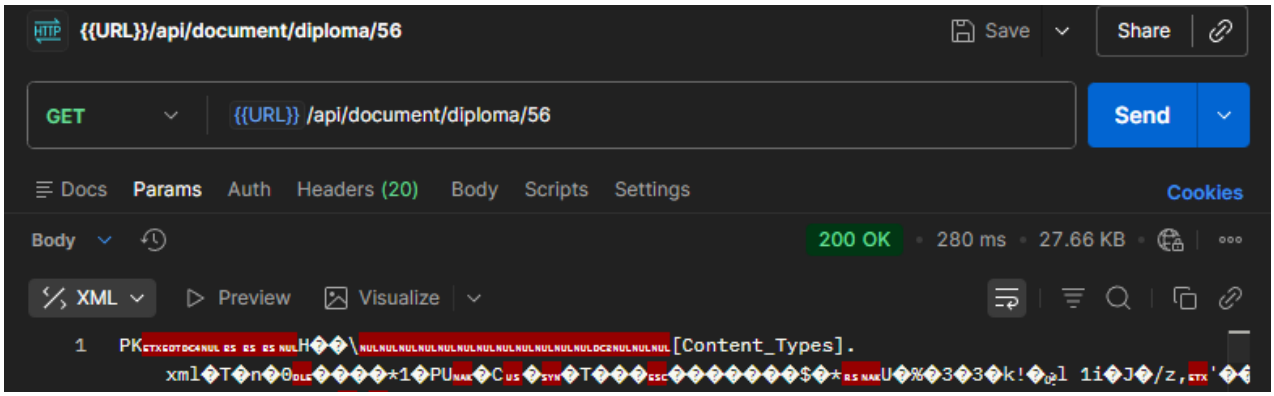
Generar Documento Resolución – Microservicio



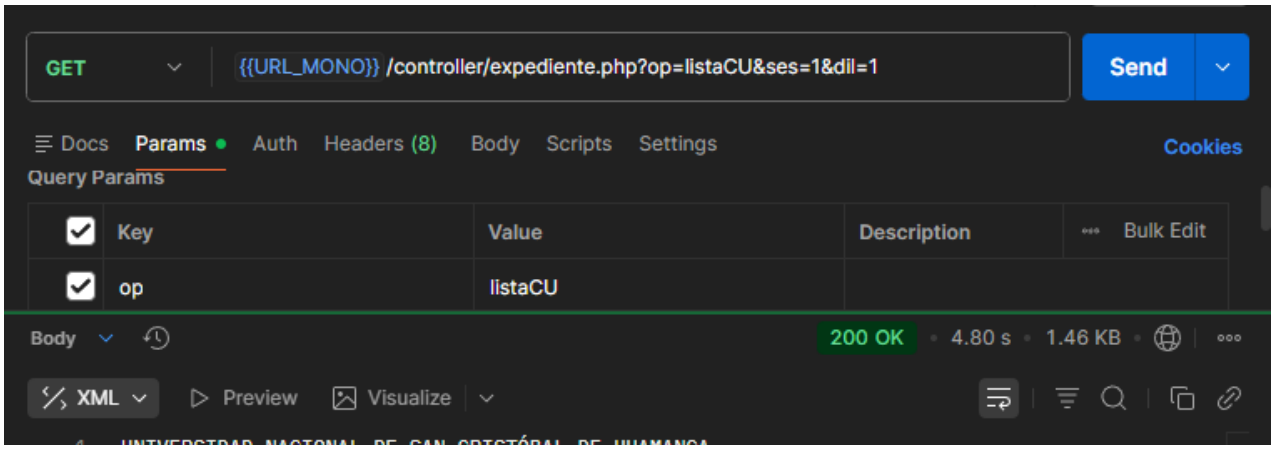
Generar Documento Diploma – Monolito



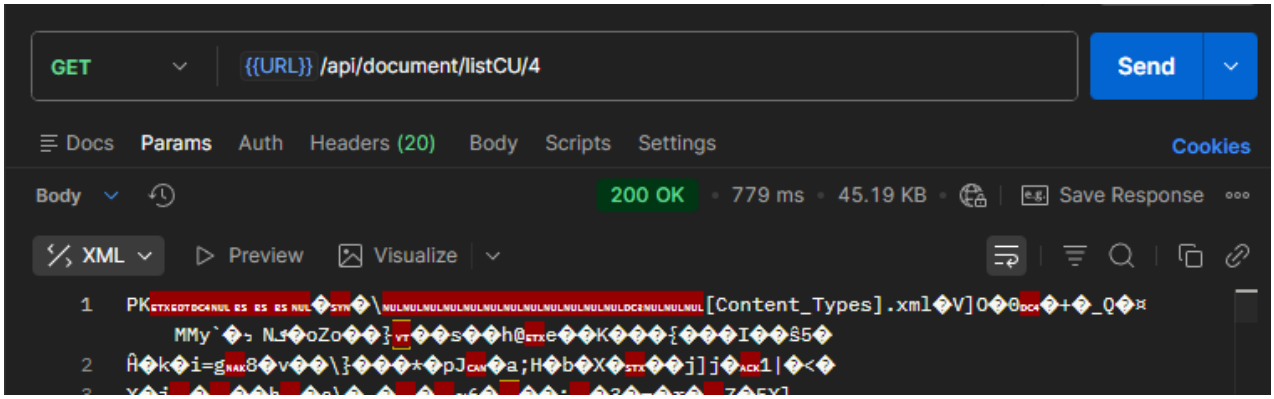
Generar Documento Diploma – Microservicio



Generar Documento – Lista CU – Monolito



Generar Documento – Lista CU – Microservicio



UNSCH FACULTAD DE INGENIERÍA DE MINAS, GEOLOGIA Y CIVIL

ACTA DE SUSTENTACIÓN DE TESIS N° 63-2026-FIMGC

PARA OPTAR EL TÍTULO PROFESIONAL DE INGENIERO DE SISTEMAS

En la Universidad Nacional de San Cristóbal de Huamanga, en la ciudad de Ayacucho, en cumplimiento a la **Resolución Decanal N° 317-2026-FIMGC-D**, a los **veinte días del mes de agosto de 2026**, siendo las **10:18 a.m.**, reunidos en el **Auditorio de la Escuela Profesional de Ingeniería de Minas**, bajo la presidencia del **MSc. José Ernesto ESTRADA CARDENAS**, y los miembros: **Mtra. Ingª. Elinar CARRILLO RIVEROS**, **Mg. Ingª. Edith Felicitas GUEVARA MOROTE** y **Dr. Manuel Avelino LAGOS BARZOLA**, actuando como secretario docente el **Ing. Saul Walter RETAMOZO FERNANDEZ**, para proceder a la sustentación de tesis para optar el **Título Profesional de Ingeniero de Sistemas**, del Bachiller en Ingeniería de Sistemas:

JULIO MARTI AYALA BELLIDO

Quien presentó la tesis denominada:

"Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios, Ayacucho, 2025"

Los señores miembros del jurado luego de expuesta la tesis y absueltas las preguntas, deliberaron y declararon:

Aprobado con 17 (diecisiete)

Siendo las **11:49 a.m.** del día **20 de agosto del 2026**, culmina el acto de sustentación de tesis, y en conformidad de lo actuado los miembros del jurado firmamos al pie del presente.

MSc. José Ernesto ESTRADA CARDENAS
Presidente

Mg. Ingª. Edith Felicitas GUEVARA MOROTE
Miembro

Dr. Manuel Avelino LAGOS BARZOLA
Miembro

Mtra. Ingª. Elinar CARRILLO RIVEROS
Miembro - Asesor

Ing. Saul Walter RETAMOZO FERNANDEZ
Secretario docente de la FIMGC

FACULTAD DE INGENIERÍA
DE MINAS Y CIVIL
Av. Independencia S/N
Ciudad Universitaria
Central Tel. 066 312510
Anexo 151

UNSCH FACULTAD DE INGENIERÍA DE MINAS, GEOLOGÍA Y CIVIL

turnitin

CONSTANCIA DE ORIGINALIDAD DE TRABAJO DE INVESTIGACIÓN

CONSTANCIA Nº 046-2026-KPS-FIMGC/UNSCH

El que suscribe; responsable verificador de originalidad de trabajos de tesis de pregrado con el software Turnitin, en segunda instancia para las **Escuelas Profesionales** de la **Facultad de Ingeniería de Minas, Geología y Civil**; en cumplimiento a la **Resolución de Consejo Universitario Nº 039-2021-UNSCH-CU**, Reglamento de Originalidad de Trabajos de Investigación de la Universidad Nacional San Cristóbal de Huamanga y **Resolución Decanal Nº 697-2024-FIMGC-D**, deja constancia de originalidad de trabajo de investigación, que el/la Sr./Srta.

Nombres y Apellidos	:	Julio Marti AYALA BELLIDO
Escuela Profesional	:	INGENIERÍA DE SISTEMAS
Título de la Tesis	:	"Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios, Ayacucho, 2025"
Evaluación de la Originalidad	:	**3%** Índice de Similitud
Identificador de la entrega	:	3059249193

Por tanto, según los Artículos 12, 13 y 17 del Reglamento de Originalidad de Trabajos de Investigación, es **PROCEDENTE** otorgar la **Constancia de Originalidad** para los fines que crea conveniente.

En señal de conformidad y verificación se firma la presente constancia

Ayacucho, 14 de octubre de 2026

FIRMA DIGITAL

Firmado digitalmente por:
PERALTA SOTOMAYOR Karel
FAU 20143660754 soft
Motivo: Soy el autor del documento
Fecha: 14/10/2026 07:40:02-0500

Con depósito para Sustentación y Tramites
C.c. Archivo

FACULTAD DE INGENIERIA DE MINAS, GEOLOGIA Y CIVIL
Av. Independencia S/N Ciudad Universitaria
Central Tel. 066 312510
Anexo 151

“Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios, Ayacucho, 2025”

por Julio Marti Ayala Bellido

Fecha de entrega: 13-oct-2026 09:03p. m. (UTC-0500)
Identificador de la entrega: 3059249193
Nombre del archivo: MEMORANDO_Nº_1062-2026-FIMGC-UNSCH-CERTIFICADO_DE_ORIGINALIDAD-_JULIO_MARTI_AYALA_BELLIDO.pdf (10.49M)
Total de palabras: 30482
Total de caracteres: 191042

“Aplicación del patrón Strangler Fig en la migración de un sistema web monolítico a microservicios, Ayacucho, 2025”

INFORME DE ORIGINALIDAD

3%	3%	1%	1%
INDICE DE SIMILITUD	FUENTES DE INTERNET	PUBLICACIONES	TRABAJOS DEL ESTUDIANTE

FUENTES PRIMARIAS

1	repositorio.unsch.edu.pe Fuente de Internet	1%
2	dspace.ups.edu.ec Fuente de Internet	<1%
3	sedici.unlp.edu.ar Fuente de Internet	<1%
4	www.campustudionline.com Fuente de Internet	<1%
5	repository.unipiloto.edu.co:8080 Fuente de Internet	<1%
6	huru.unsj.edu.ar Fuente de Internet	<1%
7	Submitted to Universidad de Guayaquil Trabajo del estudiante	<1%
8	www.mordorintelligence.com Fuente de Internet	<1%
9	produccioncientificaluz.org Fuente de Internet	<1%
10	Submitted to Universidad Nacional de San Cristóbal de Huamanga Trabajo del estudiante	<1%
11	cbdcona.blogspot.com Fuente de Internet	<1%
12	Submitted to Escuela de Posgrado Newman Trabajo del estudiante	<1%
13	repositorioinstitucional.buap.mx Fuente de Internet	<1%
14	repositorio.udh.edu.pe Fuente de Internet	<1%
	Submitted to Universidad Autonoma de Chile	

15	Trabajo del estudiante	<1%
16	apirepositorio.unh.edu.pe Fuente de Internet	<1%
17	repositorio.unap.edu.pe Fuente de Internet	<1%

Excluir citas	Activo	Excluir coincidencias	< 30 words
Excluir bibliografía	Activo		