UNIVERSIDAD CATÓLICA DE CUENCA

UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD DE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE SOFTWARE

DESARROLLO DEL BACKEND PARA UN SISTEMA DE AUTOMATIZACIÓN DE LA GESTIÓN DE PROYECTOS DE VINCULACIÓN EN LA UNIVERSIDAD CATÓLICA DE CUENCA UTILIZANDO METODOLOGÍAS ÁGILES E INTELIGENCIA ARTIFICIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SOFTWARE

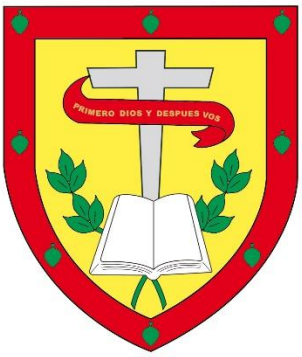
AUTOR: KEVIN OSWALDO GUAICHA VÁSQUEZ

DIRECTOR: ING. JUAN PABLO CUENCA TAPIA, MSC.

CUENCA - ECUADOR

2025

DIOS, PATRIA, CULTURA Y DESARROLLO



UNIVERSIDAD CATÓLICA DE CUENCA

Comunidad Educativa al Servicio del Pueblo

UNIDAD ACADÉMICADE INFORMÁTICA, CIENCIAS DE LA COMPUTACIÓN E INNOVACIÓN TECNOLÓGICA

CARRERA DE SOFTWARE

DESARROLLO DEL BACKEND PARA UN SISTEMA DE AUTOMATIZACIÓN DE LA GESTIÓN DE PROYECTOS DE VINCULACIÓN EN LA UNIVERSIDAD CATÓLICA DE CUENCA UTILIZANDO METODOLOGÍAS ÁGILES E INTELIGENCIA ARTIFICIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO DE SOFTWARE

AUTOR: KEVIN OSWALDO GUAICHA VÁSQUEZ

DIRECTOR: ING. JUAN PABLO CUENCA TAPIA, MSC.

CUENCA - ECUADOR

AÑO

DIOS, PATRIA, CULTURA Y DESARROLLO

Universidad Católica de Cuenca

DECLARATORIA DE AUTORÍA Y RESPONSABILIDAD

Declaratoria de Autoría y Responsabilidad

Kevin Oswaldo Guaicha Vásquez portador(a) de la cédula de ciudadanía Nº **0105886949**. Declaro ser el autor de la obra: "**Desarrollo del backend para un sistema de automatización de la gestión de proyectos de vinculación en la Universidad Católica de Cuenca utilizando metodologías ágiles e inteligencia artificial"**, sobre la cual me hago responsable sobre las opiniones, versiones e ideas expresadas. Declaro que la misma ha sido elaborada respetando los derechos de propiedad intelectual de terceros y eximo a la Universidad Católica de Cuenca sobre cualquier reclamación que pudiera existir al respecto. Declaro finalmente que mi obra ha sido realizada cumpliendo con todos los requisitos legales, éticos y bioéticos de investigación, que la misma no incumple con la normativa nacional e internacional en el área específica de investigación, sobre la que también me responsabilizo y eximo a la Universidad Católica de Cuenca de toda reclamación al respecto.

Cuenca, **15 de septiembre de 2025**

F: .....

Kevin Oswaldo Guaicha Vásquez

C.I. **0105886949**

CERTIFICADO

Certifico que el trabajo titulado **"Desarrollo del backend para un sistema de automatización de la gestión de proyectos de vinculación en la Universidad Católica de Cuenca utilizando metodologías ágiles e inteligencia artificial"** fue desarrollado por Kevin Oswaldo Guaicha Vásquez bajo mi supervisión.

F: .....

Ing. Juan Pablo Cuenca Tapia. Msg.

TUTOR DEL TRABAJO DE TITULACIÓN UNIVERSIDAD CATÓLICA DE CUENCA.

Dedicatoria

A mi madre, por su amor incondicional, por cada sacrificio silencioso y por enseñarme que la disciplina también es una forma de cariño. A mis abuelos: a mi abuelo, por su ejemplo de trabajo honesto y su paciencia infinita; y a mi abuela, que desde el cielo ilumina mis pasos y me recuerda que los sueños se alcanzan con fe y constancia. A mis tíos, por sus consejos oportunos, su apoyo en los momentos difíciles y por creer en mí cuando más lo necesité.

Agradecimiento

Expreso mi sincero agradecimiento al Ing. Juan Pablo Cuenca, tutor de esta tesis, por su acompañamiento comprometido, su paciencia y su alto estándar profesional. Sus comentarios precisos, las reuniones de seguimiento y las recomendaciones metodológicas fueron decisivas para estructurar el proyecto, superar obstáculos técnicos y mantener el rumbo cuando surgieron dudas. Gracias por inspirar disciplina, criterio y responsabilidad.

Resumen

Esta investigación presenta el desarrollo de un backend que automatiza la gestión de los proyectos de vinculación en la Universidad Católica de Cuenca. La plataforma captura la información solicitada por el formato institucional F-VS-41, verifica la coherencia presupuestaria y genera el documento oficial en Word utilizando plantillas dinámicas y bloques HTML mediante la técnica altChunk. Los anexos se almacenan de forma centralizada en AWS S3 y las consultas en lenguaje natural se resuelven con Oracle Select AI sobre vistas controladas en Oracle Autonomous Database, evitando que los usuarios deban revisar manualmente el documento completo.

La construcción del sistema siguió una metodología híbrida basada en Scrum y complementada con elementos del Proceso Unificado Racional (RUP). Se desarrolló en ocho sprints que abarcaron la autenticación con JWT, la persistencia con JPA, la generación dinámica de documentos, la gestión de anexos y la integración de inteligencia artificial. La ingeniería de requisitos y la evaluación de la solución se apoyaron en el modelo FURPS+. Las pruebas funcionales (Postman) y de carga (JMeter) con 100 y 200 usuarios concurrentes evidenciaron un rendimiento aceptable dentro de las limitaciones del entorno. En las mediciones individuales, las operaciones de autenticación (registro y login) responden en el orden de cientos de milisegundos; bajo carga sostenida la latencia puede aumentar a unos pocos segundos debido al uso de versiones gratuitas de la base de datos, S3 y la plataforma de despliegue. La generación del documento F-VS-41 varía en torno a 20–30 s cuando se simulan 100 usuarios y se acerca a 50 s con 200 usuarios. A pesar de ello, la cobertura de pruebas alcanza el 82 % y no se observaron divergencias entre los montos almacenados y los que se reflejan en el archivo final.

El sistema desarrollado elimina errores de transcripción, evita el retrabajo y acelera la elaboración del F-VS-41, además de sentar una base escalable para incorporar extensiones como firma electrónica, una interfaz web y analítica del impacto social. Esta automatización contribuye a fortalecer la trazabilidad y la transparencia en la gestión de proyectos de vinculación, permitiendo demostrar con datos verificables la pertinencia social de las iniciativas universitarias.

Palabras clave: *automatización documental, backend, vinculación universitaria, Scrum, RUP, FURPS+, Select AI.*

Abstract

This research presents the development of a backend that automates the management of outreach projects at the Catholic University of Cuenca. The platform takes the information requested by the institutional format F-VS-41, verifies budgetary coherence, and generates the official Word document using dynamic templates and HTML blocks via the *altChunk* technique. Annexes are stored centrally in AWS S3, and natural language queries are resolved with Oracle Select AI on controlled views in Oracle Autonomous Database, preventing users from having to manually review the entire document.

The system's structure followed a hybrid methodology based on Scrum and complemented by elements of the Rational Unified Process (RUP). It was developed in eight sprints covering authentication with JWT, persistence with JPA, dynamic document generation, attachment management, and artificial intelligence integration. Requirements engineering and solution evaluation were supported by the FURPS+ model. Functional (Postman) and load (JMeter) tests with 100 and 200 concurrent users demonstrated acceptable performance within the environment's limitations. In individual measurements, authentication operations (registration and login) respond in the order of hundreds of milliseconds; under sustained load, latency can increase to a few seconds due to the use of free versions of the database, S3, and the deployment platform. The generation of the F-VS-41 document varies around 20–30 s when simulating 100 users and approaches 50 s with 200 users. Despite this, test coverage reaches 82%, and no discrepancies were observed between the amounts stored and those reflected in the final file.

The developed system eliminates transcription errors, prevents rework, and accelerates the creation of the F-VS-41, while also establishing a scalable foundation for incorporating extensions such as electronic signatures, a web interface, and social impact analytics. This automation

contributes to strengthening traceability and transparency in the management of outreach projects, allowing for the demonstration, with verifiable data, of the social relevance of university initiatives.

Keywords: *Document automation, backend, university linkage, Scrum, RUP, FURPS+, Select AI.*

Índice

Lista de tablas

Lista de figuras

Capítulo 1

1. Marco Referencial

1.1 Introducción

La presente tesis describe el desarrollo de un backend que automatiza la gestión integral de los proyectos de vinculación con la sociedad en la Universidad Católica de Cuenca (UCACUE). Esta plataforma recoge todos los datos requeridos por el formato institucional F-VS-41, verifica de manera automática la coherencia presupuestaria y genera o regenera el documento oficial en Word a través de plantillas dinámicas y bloques HTML utilizando la funcionalidad altChunk. Los anexos se almacenan de forma centralizada en AWS S3 y las consultas en lenguaje natural se resuelven mediante Oracle Select AI sobre vistas controladas en Oracle Autonomous Database, evitando que los usuarios deban revisar manualmente el documento completo. De esta manera, se reducen drásticamente los plazos de elaboración y se responde a la directriz institucional de modernizar los procesos administrativos.

El interés por abordar esta solución surgió al constatar que la construcción manual del F-VS-41 ocasionaba errores numéricos, duplicidad de datos, desalineaciones de formato y la proliferación de versiones inconsistentes remitidas por correo. Estas deficiencias incrementaban el retrabajo y retrasaban la aprobación de iniciativas orientadas a generar impacto social. Automatizar el flujo desde la captura estructurada de información hasta la generación del archivo final y ofrecer una interfaz de consulta basada en lenguaje natural se vuelve imprescindible para garantizar exactitud, trazabilidad y oportunidad en la toma de decisiones. La solución propuesta también establece un repositorio unificado para los anexos, evitando su dispersión y facilitando el acceso a documentación clave.

En términos metodológicos, la investigación se concibió como un proyecto aplicado con un enfoque ágil e iterativo. Se adoptó Scrum a lo largo de ocho sprints de dos semanas, complementado con elementos del Proceso Unificado Racional (RUP) y el modelo de calidad FURPS+. Este enfoque híbrido permitió avanzar de forma incremental: se implementaron la autenticación basada en JWT, la persistencia con JPA, la construcción progresiva del documento utilizando Apache POI, la gestión de anexos en la nube y la integración de Select AI. Cada iteración fue sometida a inspección temprana y se incorporaron de manera continua las recomendaciones institucionales. Las pruebas funcionales y de carga con 100 y 200 usuarios concurrentes evidenciaron que, en condiciones controladas, las operaciones de autenticación responden en el orden de cientos de milisegundos, aunque bajo carga sostenida estos tiempos se elevan a unos segundos debido a las limitaciones de las versiones gratuitas de la base de datos y la infraestructura. La generación de documentos complejos se sitúa en torno a 20–30 s en la prueba de 100 usuarios y puede aproximarse a 50 s con 200 usuarios. Con una cobertura de pruebas del 82 %, estos resultados confirman que la solución no solo elimina el retrabajo manual, sino que ofrece un rendimiento estable y escalable dentro del entorno disponible.

La organización del documento refleja este enfoque integral. El Capítulo 1 contextualiza la problemática, expone los objetivos generales y específicos, y delimita el alcance funcional, señalando tanto las funcionalidades incluidas como las excluidas. El Capítulo 2 compila los fundamentos teóricos sobre arquitectura backend, automatización documental, inteligencia artificial aplicada a la gestión de documentos, los modelos de proceso (Scrum y RUP) y el marco de calidad FURPS+ que sustentan la propuesta. El Capítulo 3 describe detalladamente la metodología de investigación, el modelado de requisitos y datos, la arquitectura propuesta y la planificación de los sprints. En el Capítulo 4 se presentan los resultados de verificación y

validación, abarcando pruebas unitarias, funcionales, de rendimiento y el análisis de desempeño. Finalmente, se exponen las conclusiones, las principales contribuciones de la investigación y las líneas de trabajo futuro, entre las que destacan la incorporación de una interfaz web, la automatización completa de anexos y formularios, la firma electrónica y la integración con otros sistemas universitarios.

En síntesis, esta solución persigue reducir errores y tiempos de elaboración del F-VS-41, mejorar la trazabilidad de la información y sentar una base escalable para futuras extensiones como firma digital, circuitos de aprobación en línea y analítica de impacto. Al demostrar la viabilidad de un flujo digital que genera documentos coherentes en pocos minutos, se aporta una herramienta robusta para fortalecer la capacidad de la UCACUE en la gestión de proyectos de vinculación y para evidenciar con datos verificables la pertinencia social de sus iniciativas.

1.2 Planteamiento del problema

En la Universidad Católica de Cuenca la formulación de proyectos de vinculación se documenta en un archivo institucional extenso en Word que, además de los datos generales, integra bloques como la Matriz de Resultados del Proyecto, el Cronograma de Actividades, el Cuadro de Recursos Necesarios (donde se detallan los costos por carrera o componente) y un apartado de Tiempo y Presupuesto que consolida montos globales; sin embargo, en la práctica la mayoría de estos cuadros se calculan primero en hojas de Excel separadas, cada responsable aplica sus propias fórmulas, agrega columnas o filas según necesidad y luego copia manualmente los valores resultantes al documento Word.

Este traslado manual genera varios problemas encadenados: los totales de la sección Tiempo y Presupuesto con frecuencia no cuadran con las sumas detalladas en el Cuadro de Recursos; al ajustar filas en la MdR o en el Cronograma se desconfiguran estilos, bordes y

alineaciones; campos que deberían repetir el mismo dato en distintas secciones terminan con valores divergentes porque alguien los volvió a teclear; y las correcciones sucesivas producen múltiples versiones del mismo archivo circulando por correo. Así mismo, los anexos no se gestionan en un repositorio único, sino que llegan por canales informales y deben incorporarse a mano, lo cual retrasa la consolidación final del proyecto. Cuando una autoridad o un docente necesita verificar algo tan simple como "¿cuál es el presupuesto total?", "¿qué instituciones participan?" o "¿ya está cargado el documento firmado?", debe abrir el archivo completo y buscar entre muchas páginas, con el riesgo de consultar una versión desactualizada.

De esta manera, el proceso manual actual implica retrabajo, inconsistencias numéricas, pérdida de trazabilidad y demoras administrativas, especialmente a medida que aumenta el número de proyectos gestionados. Por su puesto, esto abre la necesidad de una solución que capture la información de manera estructurada en una base de datos, calcule los totales directamente a partir de los registros (sin depender de copiar desde Excel), genere el documento institucional en automático incluyendo la MdR, el Cronograma y los resúmenes presupuestarios, administre centralmente los anexos y permita recuperar información clave sin revisar el documento completo, incluso mediante consultas en lenguaje natural apoyadas en Select AI.

En síntesis, la dependencia de procesos manuales y dispersos para construir el documento de vinculación provoca errores, demoras y falta de confiabilidad en la información institucional, constituyendo el problema que aborda la presente tesis.

1.3 Justificación

La Universidad Católica de Cuenca exige que cada proyecto de vinculación se respalde con un documento institucional alineado con los programas de Vinculación con la Sociedad, el Plan Nacional de Desarrollo y los Objetivos de Desarrollo Sostenible. El documento también debe

demostrar que el proyecto aporta al territorio y cuenta con fuentes de financiamiento, por lo que la calidad, consistencia y trazabilidad de la información dejan de ser un aspecto operativo y se convierten en un requisito académico-administrativo.

La gestión actual se apoya en formularios y plantillas de Word llenados de manera manual, con el consiguiente riesgo de errores de transcripción y demoras administrativas. La manipulación manual de tablas como la Matriz de Resultados, el cronograma y el cuadro de recursos, así como el traslado de datos desde hojas de Excel, genera inconsistencias numéricas y proliferación de versiones. Esta situación, descrita en el planteamiento del problema, subraya la necesidad de automatizar.

El proyecto propone un backend que capture datos estructurados, genere automáticamente el documento institucional y gestione anexos; la solución se valida con pruebas unitarias, funcionales y de carga. El diseño se basa en un stack formado por *Oracle Autonomous Database* y *Select AI* para consultas en lenguaje natural, *AWS S3* para el almacenamiento de documentos y *Apache POI/altChunk* para la generación dinámica del formato Word

Incorporar *Select AI* representa un beneficio estratégico: usuarios sin conocimientos de *SQL* pueden recuperar montos, fechas o instituciones formulando preguntas en lenguaje natural; el motor enriquece la consulta con metadatos del esquema y devuelve resultados sin exponer los datos fuera del entorno seguro. Metodológicamente, el desarrollo se abordó como una investigación aplicada con iteraciones ágiles al estilo Scrum, permitiendo inspeccionar resultados parciales y ajustar en ciclos cortos.

En conclusión, el proyecto está justificado porque responde a requisitos institucionales formales, corrige ineficiencias del proceso manual, ofrece una solución técnica validada para

asegurar la consistencia de los datos y la trazabilidad de los anexos, incorpora acceso inteligente a la información mediante *IA* y aplica un proceso ágil que facilita la adaptación al cambio.

1.4 Objetivos

1.4.1 Objetivo general

Automatizar, mediante un backend ágil, la captura de datos, generación del documento institucional de proyectos de vinculación, gestión de anexos y consulta inteligente de información en la Universidad Católica de Cuenca.

1.4.2 Objetivos específicos

1. Levantar y estructurar los requerimientos funcionales y no funcionales del proceso institucional de vinculación, identificando campos obligatorios, secciones repetibles (MdR, Cronograma, Recursos) y reglas de consistencia numérica que deben preservarse en el sistema.
2. Diseñar la arquitectura del backend basada en el patrón *Controller–Service–Repository* y un modelo de datos relacional que permita capturar, actualizar y consultar toda la información necesaria para generar el documento institucional sin duplicación manual.
3. Implementar servicios seguros de autenticación y gestión de proyectos (registro, login, JWT, creación y actualización de datos de proyecto, participantes), garantizando que la información se almacene de forma íntegra antes de la generación del documento.
4. Automatizar la generación y regeneración del documento institucional (F-VS-41) reemplazando campos, construyendo tablas dinámicas (instituciones, matriz de resultados, etc.) e incrustando bloques complejos (cronograma, recursos, resumen

presupuestario) mediante técnicas programáticas con Apache POI y altChunk, almacenando las versiones resultantes en la nube.

5. Desarrollar la gestión centralizada de anexos y documento firmado vinculados a cada proyecto (carga simple y múltiple, descarga, eliminación controlada), asegurando trazabilidad y disponibilidad en un repositorio confiable (S3 u otro almacenamiento equivalente).
6. Integrar capacidades de consulta en lenguaje natural mediante Select AI, exponiendo vistas controladas de los datos del proyecto para que usuarios autorizados puedan recuperar información clave (montos, instituciones, fechas) sin revisar manualmente el documento completo.
7. Validar el backend mediante pruebas unitarias, funcionales y de carga, y documentar los resultados por sprint dentro de un proceso ágil incremental.

1.5. Alcance

Este trabajo se limita al desarrollo del backend que automatiza la gestión de proyectos de vinculación. Toda la captura de datos, la lógica para generar el documento institucional F-VS-41, el cálculo y validación de montos, la administración de anexos y el acceso a la información vía consultas (incluyendo la integración con Select AI) se implementan en la capa servidor y se verifican mediante pruebas en Postman y despliegue en la nube; no se desarrolla interfaz gráfica, por lo que la interacción se realiza a través de la API REST.

1.5.1 Alcance funcional incluido

- Gestión de usuarios básicos: registro, login, recuperación de contraseña y autenticación por JWT.

- Gestión de proyectos de vinculación: creación, actualización y consulta de datos generales; asociación de participantes.
- Persistencia estructurada masiva: recepción de cargas extensas (≈1000 campos distribuidos en entidades) con mapeo JPA.
- Generación automática del documento institucional F-VS-41: reemplazo de campos simples, inserción de tablas dinámicas con Apache POI, integración de bloques complejos mediante HTML + altChunk, regeneración del documento cuando cambian los datos y almacenamiento en la nube.
- Gestión de anexos y documento firmado: carga simple y múltiple, listado, descarga y eliminación controlada; metadatos asociados al proyecto.
- Servicios de consulta estructurada: endpoints que devuelven todos los datos del proyecto, anexos asociados y estado del documento.
- Consultas en lenguaje natural (Select AI): vistas controladas de la base para recuperar información clave sin revisar manualmente el documento.
- Contenerización y despliegue de demostración: empaquetado con Docker y despliegue del backend en Render para validación remota.
- Pruebas unitarias, funcionales y de rendimiento.

1.5.2 Fuera de alcance (no incluido)

- Frontend web o móvil para la captura final de usuarios; la interacción se simula con Postman u otros clientes de API.
- Flujo institucional de aprobación o rechazo de proyectos (firma digital, rutas de validación administrativa, notificaciones).

- Gestión avanzada de roles jerárquicos o permisos granulares más allá de propietario e invitados básicos.
- Gestión documental de versiones históricas (solo se conserva la versión vigente en S3).
- Edición colaborativa en línea del archivo Word; el documento se genera en el backend y se descarga, no se edita dentro del sistema.
- Integración con SIU, SGA u otros sistemas académicos externos (solo endpoints propios).
- Automatización de firma electrónica o verificación criptográfica de documentos firmados.
- Validación semántica profunda de contenidos textuales (el sistema no corrige redacción de justificación, objetivos o narrativas del proyecto).

1.5.3 Entornos y componentes tecnológicos considerados

- Base de datos: Oracle Autonomous Database como entorno principal de pruebas e integración con Select AI (portabilidad a PostgreSQL).
- Almacenamiento de archivos: AWS S3 o servicio compatible.
- Backend: Java 17 + Spring Boot 3 con arquitectura Controller-Service-Repository.
- Generación de documentos: Apache POI (XWPF) + altChunk para bloques HTML.
- Autenticación: JWT.
- Contenedorización: Docker; variables externas para credenciales y endpoints; imagen desplegable en Render.

- Pruebas: JUnit 5, Mockito, AssertJPostman y Apache JMeter.

1.6 Conceptos relacionados

Arquitectura backend

El backend constituye la parte de un sistema que opera del lado del servidor; incluye la lógica de negocio, la persistencia de datos y la exposición de servicios consumidos por clientes mediante APIs. Suele adoptar arquitecturas en capas (por ejemplo, Controller-Service-Repository) para separar responsabilidades, emplear frameworks como Spring Boot para la configuración rápida de servicios, y proteger sus endpoints con mecanismos de autenticación como JSON Web Token (JWT). Este diseño garantiza cohesión y trazabilidad entre la capa de presentación, la lógica y el acceso a datos.

Automatización documental

La automatización documental se refiere al uso de software para crear, completar y gestionar documentos de forma programada a partir de datos estructurados. Se basa en plantillas con marcadores (placeholders) que se sustituyen por valores de una base de datos y en la generación de tablas o secciones complejas mediante bloques HTML insertados en documentos de texto. Esta automatización asegura que los totales se calculen correctamente, reduce errores manuales y permite la regeneración de documentos cuando cambian los datos, manteniendo la consistencia y la trazabilidad.

Inteligencia artificial (NL2SQL)

Una aplicación destacada de la *inteligencia artificial* en bases de datos es la tecnología *NL2SQL*, que traduce preguntas en lenguaje natural a sentencias *SQL*. Según Microsoft, *NL2SQL* permite a los usuarios consultar bases de datos sin conocer el lenguaje de consulta, generando consultas correctas a partir de preguntas cotidianas. Este enfoque democratiza el acceso a la

información, acelera la toma de decisiones y mantiene las reglas de negocio al ejecutar las consultas generadas.

Visión computacional y OCR

La *visión computacional* aplicada a documentos se materializa en el *reconocimiento óptico de caracteres* (OCR), una técnica que extrae texto de imágenes o documentos escaneados. Adobe explica que el OCR digitaliza textos identificando letras y símbolos y convirtiéndolos en archivos editables; su funcionamiento se basa en el análisis y segmentación de la imagen, el reconocimiento de patrones y un postprocesamiento que corrige errores. Integrado con sistemas de gestión documental, el OCR facilita la automatización de formularios y archivos físicos.

Metodología RUP

El *Proceso Racional Unificado* (RUP) es una metodología de desarrollo de software iterativa e incremental propuesta por Rational Software (IBM). No se trata de un conjunto rígido de pasos, sino de un marco adaptable que estructura el ciclo de vida en cuatro fases (inicio, elaboración, construcción y transición) y se centra en la arquitectura y en los casos de uso. RUP promueve principios como la adaptación del proceso al proyecto, el equilibrio de prioridades entre partes interesadas, la entrega iterativa de valor y la integración temprana de actividades de garantía de calidad.

Modelo FURPS+

El modelo *FURPS* clasifica los atributos de calidad del software en cinco dimensiones: Funcionalidad (propósito y seguridad), Usabilidad (facilidad de uso, estética), Fiabilidad (disponibilidad y estabilidad), Rendimiento (eficiencia y capacidad) y Mantenibilidad (capacidad de prueba y modularidad). El “+” incorpora requisitos adicionales como restricciones de diseño, entorno o tecnología. Esta categorización ayuda a diferenciar y priorizar los requisitos no

funcionales frente a los funcionales, sirviendo como guía para la ingeniería de requisitos y la evaluación de la calidad del software.

1.7 Trabajos relacionados

La automatización de documentos emplea sistemas tecnológicos para generar, procesar y almacenar archivos de forma automática, minimizando la intervención manual y reduciendo errores humanos. Diversos estudios recientes destacan sus ventajas. Por ejemplo, Beltrán y Ortega (2024) desarrollaron un sistema inteligente de gestión documental para una organización benéfica, mediante aplicaciones web y móvil, logrando agilitar el acceso a la información y estandarizar la administración de los archivos.

De igual manera, Ruales (2024) combinó el modelado de procesos BPMN 2.0 con herramientas de Microsoft 365 (Power Apps, Power Automate, SharePoint, Azure SQL y Power BI) para digitalizar la gestión documental empresarial, lo que resultó en una mejora significativa de la eficiencia del proceso y en una mayor satisfacción de los usuarios.

En el ámbito universitario, Jara (2023) señala que las instituciones de educación superior están implementando estos sistemas para reducir los tiempos de respuesta a solicitudes y liberar al personal de tareas repetitivas, permitiéndoles enfocarse en actividades de mayor relevancia. Estos antecedentes confirman que la automatización documental disminuye errores y estandariza la información, además de acelerar los flujos de trabajo.

Scrum es un marco ágil de desarrollo que divide el trabajo en *sprints* cortos, fomenta la inspección continua y promueve la entrega incremental de valor. Varios trabajos evidencian sus beneficios en proyectos académicos. Macías y Mindiola (2018) aplicaron *Scrum* en el desarrollo de un sitio web de gestión de contenidos para Educación Continua en la ESPOL, obteniendo resultados funcionales en cada iteración gracias a la entrega rápida de incrementos del producto.

Por su parte, Loarte (2022) implementó *Scrum* en el desarrollo del *backend* de un sistema penitenciario en Ecuador, destacando que la clara definición de roles y artefactos de *Scrum* permitió llevar el desarrollo de forma ordenada, modular y enfocada en los objetivos.

Asimismo, Silva et al. (2021) mejoraron los trámites de titulación en la Universidad Técnica de Cotopaxi mediante un sistema web desarrollado en iteraciones con participación constante de estudiantes, tutores y autoridades, lo que evidenció mejoras en el flujo de trabajo respecto al proceso tradicional.

En otro caso, Sánchez (2023) reportó en su estudio un aumento del 28 % en la rentabilidad de un colegio privado tras adoptar *Scrum* para implementar un sistema de gestión académica. Igualmente, documentaron beneficios similares al introducir metodologías ágiles (Scrum) en la implantación de un sistema de gestión académica en escuelas técnicas de Brasil, observando mejoras en el desempeño de la gestión y en la ejecución del proyecto gracias a este enfoque.

En conjunto, los casos citados ilustran cómo Scrum facilita entregas funcionales frecuentes y adapta los proyectos a requisitos cambiantes de manera eficaz.

Las interfaces de *lenguaje natural* para *bases de datos* (ILNBD) permiten que usuarios sin conocimientos de *SQL* consulten información mediante preguntas en lenguaje cotidiano, haciendo más accesibles los datos estructurados. Bautista (2014) desarrolló uno de los primeros traductores de consultas en español a *SQL* que manejó adecuadamente consultas con funciones de agregación y agrupamiento, alcanzando niveles de precisión entre 83 % y 92 % en las respuestas.

Investigaciones posteriores en este campo añadieron componentes de diálogo iterativo para aclarar ambigüedades semánticas en las consultas, refinando así los resultados en caso de preguntas imprecisas o múltiples interpretaciones.

Con la llegada de los modelos de *lenguaje de gran tamaño* (LLM), la integración de *procesamiento de lenguaje natural* (PLN) con modelos de lenguaje avanzados ha dado un salto importante. Por ejemplo, Pérez et al. (2024) combinaron técnicas de procesamiento de lenguaje natural y un modelo de lenguaje de gran tamaño de código abierto (LLaMA) para generar documentos legales estandarizados, extrayendo información relevante de expedientes judiciales sin incluir datos sensibles, con el objetivo de mejorar la consistencia y la eficiencia en la elaboración de textos legales. Su herramienta utiliza plantillas personalizables y elementos opcionales que aseguran la uniformidad de los documentos resultantes, manteniendo la coherencia en la presentación de la información legal y eliminando detalles específicos de cada caso para evitar omisiones o variaciones indeseadas.

Estos trabajos demuestran que las tecnologías de *IA conversacional* pueden simplificar el acceso a datos estructurados y reducir drásticamente los tiempos de búsqueda de información específica, al permitir a los usuarios obtener respuestas rápidas sin necesidad de conocer lenguajes de consulta formales.

En conclusión, la literatura coincidente indica que la automatización documental reduce errores y tiempos en el manejo de información, *Scrum* provee una organización efectiva para el *desarrollo incremental* de sistemas (especialmente en el ámbito académico) y las interfaces impulsadas por *IA* facilitan la consulta de bases de datos sin necesidad de usar *SQL*. Estos hallazgos sirven de base para proponer nuevas soluciones orientadas a mejorar la gestión universitaria de forma integral.

Capítulo 2

2. Marco teórico

2.1 Fundamentos de arquitectura backend

2.1.1 Patrón Controller–Service–Repository

El patrón *Controller–Service–Repository* es un estilo arquitectónico ampliamente utilizado en backends de microservicios. En él, el *controlador* expone interfaces *HTTP* o *RPC* que reciben solicitudes y delegan la lógica de negocio a servicios, mientras que estos *servicios* encapsulan reglas de negocio y orquestan transacciones complejas. Por último, los *repositorios* se encargan de persistir y recuperar datos de almacenes como bases relacionales o documentales. La literatura sobre *microservicios* destaca que esta separación favorece el *principio de responsabilidad única* y permite que cada capa evolucione de manera independiente (Abdelfattah et al., 2025). En estudios recientes se clasifican las clases de una aplicación en *controladores, servicios y repositorios* según sus anotaciones (@Controller, @Service y @Repository) y se muestran ejemplos de código donde un controlador delega la creación y consulta de registros a un servicio y éste a un repositorio. Este patrón, al obligar a que las operaciones *CRUD* no se mezclen con la lógica de negocio ni con la persistencia, mejora la mantenibilidad y facilita la reutilización de código (Sun y Kim, 2022).

El *controlador* funciona como punto de entrada y valida parámetros antes de delegar al servicio. El *servicio* abstrae operaciones complejas, por ejemplo, calcular indicadores financieros o generar informes y controla las transacciones. El *repositorio* implementa la comunicación con el motor de base de datos o con servicios de almacenamiento externos (como Amazon S3), devolviendo entidades o colecciones de ellas. Esta estructura no sólo aplica a *microservicios*: los sistemas *monolíticos* también pueden beneficiarse de ella, aunque con menos *overhead*. Investigaciones recientes analizan la relación entre estas capas y la identificación automática de

dependencias, mostrando que mantener una arquitectura en capas reduce la complejidad ciclomática y mejora la comprensión del código. Por tanto, adoptar el *patrón Controller– Service–Repository* en el *backend* aporta orden, separa responsabilidades y hace más sencilla la evolución y prueba de cada componente (Abdelfattah et al., 2025; Sun y Kim, 2022).

2.1.2 Principios SOLID

Los principios *SOLID* son cinco directrices de diseño orientado a objetos que buscan garantizar que las clases sean cohesivas, modulables y fáciles de mantener. El principio de *responsabilidad única* establece que una clase debe tener un solo motivo para cambiar y es clave en la separación de capas del patrón *Controller–Service–Repository* (Abdelfattah et al., 2025).

El principio de *abierto/cerrado* sugiere que las clases deben estar abiertas a extensión, pero cerradas a modificación mediante herencia o composición. En estudios sobre comprensión de código se ha demostrado que la adopción de estos principios mejora significativamente la comprensión y mantenibilidad del código en proyectos de *aprendizaje automático* y en *aplicaciones de software* en general (Cabral et al., 2024). Por ejemplo, al aplicar el principio de sustitución de *Liskov*, las dependencias se diseñan para aceptar tipos más generales, lo que favorece la extensibilidad. El principio de segregación de interfaces propone dividir interfaces amplias en otras más específicas para evitar que los clientes dependan de métodos que no utilizan. Finalmente, la inversión de dependencias promueve que los módulos de alto nivel no se acoplen a módulos de bajo nivel, sino a abstracciones. Cabral et al. (2024) evidencian que la aplicación sistemática de *SOLID* en proyectos complejos reduce los errores de integración y aumenta la testabilidad; otros autores remarcan que estas reglas fomentan un diseño limpio y escalable, necesario en microservicios donde el código se despliega de forma independiente (Sun y Kim, 2022). Así, en

nuestra tesis los principios *SOLID* sirven como guía para estructurar las clases del backend y mejorar la calidad del software.

2.1.3 Inyección de dependencias en Spring

La *inversión de control* (Inversion of Control, IoC) es un principio arquitectónico en el que el flujo de control de un programa no está determinado por el código del usuario sino por un *contenedor* o *framework* que inyecta las dependencias necesarias. La *inyección de dependencias* (Dependency Injection, DI) es un patrón que implementa IoC permitiendo suministrar objetos requeridos por una clase desde fuera, en vez de crearlos en su interior. Fowler (2004) explica que DI separa la configuración del uso y favorece que las clases dependan de abstracciones, facilitando así pruebas unitarias y permitiendo cambiar implementaciones sin modificar el código consumidor. Sun y Kim (2022) demuestran con métricas estáticas que DI aumenta la mantenibilidad y reduce el acoplamiento, introduciendo el índice DCBO que evalúa la mejora en mantenimiento cuando se adopta DI.

En *Spring*, la inyección de dependencias se implementa mediante anotaciones como @Autowired, @Component, @Service o @Repository. El contenedor de *Spring* crea las instancias y las inyecta en las clases consumidoras según el ciclo de vida configurado (singleton, prototipo, etc.). Este mecanismo facilita la implementación de pruebas, ya que las dependencias se pueden sustituir por *mocks*. Parejo et al. (2023) señala que al depender de interfaces y no de clases concretas, las aplicaciones pueden intercambiar componentes, como repositorios que usan distintas bases de datos, sin tocar la lógica de negocio. Además, el estudio sobre mantenibilidad de Sun y Kim (2022) concluye que DI disminuye la complejidad del código al externalizar la creación de objetos y mejora la cohesión.

2.2 Tecnologías habilitadoras

2.2.1 Spring Boot

Spring Boot es un *framework* para crear aplicaciones *Java* de manera rápida siguiendo el principio de convención sobre configuración. Proporciona un conjunto de bibliotecas llamadas 'starters' que agrupan dependencias comunes y un motor de *auto-configuración* que detecta qué componentes se encuentran en el *classpath* para habilitar automáticamente configuraciones sensatas. El marco simplifica la creación de microservicios al incorporar un *servidor embebido* (Tomcat, Jetty) que permite empaquetar y ejecutar la aplicación como un único *archivo JAR*. Investigaciones sobre microservicios con *Spring Boot* resaltan que sus capacidades de autoconfiguración reducen el tiempo de desarrollo y mejoran la flexibilidad al permitir anular configuraciones predeterminadas cuando se requieren comportamientos específicos (Ashok Lama, 2025).

Además de autoconfiguración, *Spring Boot* incorpora características de producción como el módulo Actuator, que expone métricas, indicadores de salud y rutas de monitorización para facilitar el despliegue y la operación de servicios. Su integración con herramientas de observabilidad y contenedores hace posible desplegar microservicios escalables. La literatura señala que el uso de *Spring Boot* en combinación con arquitecturas de contenedores simplifica la construcción de pipelines de *integración continua y despliegue continuo* (CI/CD) y favorece la observabilidad mediante el registro estructurado y la exposición de 'endpoints' de estado (Abdelfattah et al., 2025; Ashok Lama, 2025).

2.2.2 JSON Web Token (JWT)

Un *JSON Web Token* (JWT) es un estándar abierto (RFC 7519) que define un formato compacto y seguro para la transmisión de afirmaciones (claims) entre dos partes. Un *token* contiene

un *encabezado*, un *payload y* una *firma digital o clave de autenticación*; puede estar firmado con un algoritmo simétrico (por ejemplo, HS256) o asimétrico (por ejemplo, RS256). Según Kang et al. (2023), en el marco de la teoría de *seguridad de confianza cero* (Zero Trust), cada solicitud debe estar respaldada por credenciales verificables y los tokens se validan en cada microservicio sin confiar en la red interna.

La arquitectura *Zero Trust* plantea principios como 'nunca confiar, siempre verificar' y exige un control estricto del acceso a recursos, inspeccionando y registrando todo el tráfico. Dentro de este contexto, los *JWT* ofrecen varias ventajas: son stateless (no requieren mantener sesiones en el servidor), portables al poder enviarse a través de encabezados *HTTP* o *cookies*, y escalables para arquitecturas distribuidas. Sin embargo, presentan desventajas como la dificultad de revocación, ya que el servidor no puede invalidar un *token* sin mecanismos adicionales de listas negras, y la necesidad de proteger la clave privada para evitar la falsificación. Investigaciones como las de Kang et al. (2023) y Jones et al. (2015), destacan que, al combinar *Zero Trust* con *JWT* firmados mediante algoritmos asimétricos, se reduce el riesgo de compromiso y se fortalece la autenticación entre microservicios.

2.2.3 Oracle ADB + Select AI

Oracle Autonomous Database (ADB) es un servicio gestionado que automatiza tareas de administración de bases de datos como aprovisionamiento, copia de seguridad y optimización. En 2023 Oracle incorporó la funcionalidad *Select AI*, que permite a los usuarios realizar consultas en *lenguaje natural*. Rittman Mead describe que *Select AI* se apoya *en modelos de lenguaje grandes* (LLM) y en el metadata del esquema para generar consultas *SQL* adaptadas al contexto de cada base de datos (Oluwafemi, 2025). El servicio puede devolver la respuesta de manera narrativa, mostrar la sentencia SQL generada o integrarse en un chatbot que conversará con el usuario.

Desde el punto de vista de teoría de interacción lenguaje–datos, los sistemas text-to-SQL buscan traducir preguntas en *lenguaje natural* a consultas *SQL* válidas. En la revisión de Mohammadjafari et al. (2025), se expone que estas técnicas democratizan el acceso a bases de datos para usuarios sin conocimientos de *SQL*, reduciendo la barrera de entrada y acelerando la toma de decisiones. *Select AI* aplica estas ideas a *ADB*, aportando también la capacidad de restringir el alcance de las consultas mediante vistas seguras y metadatos, lo que mejora la seguridad y el cumplimiento normativo. No obstante, su exactitud depende de la calidad del modelo y de los datos de entrenamiento, y por ello se recomienda revisar manualmente las consultas generadas (Mohammadjafari et al., 2025; Oluwafemi, 2025).

2.2.4 Amazon S3

Amazon Simple Storage Service (S3) es un servicio de almacenamiento de objetos diseñado para almacenar y recuperar datos a cualquier escala. A diferencia de las bases de datos relacionales, los objetos en *S3* se organizan en *buckets* y son identificados de forma única por claves. Una de las características más citadas es su durabilidad de 11 nueves (99.999999999%), lograda mediante replicación geográfica y técnicas de corrección de errores. Asimismo, S3 ofrece diferentes clases de almacenamiento con disponibilidades del 99,99 % para la clase estándar y 99,9 % para clases de menor frecuencia de acceso (Khande et al., 2023).

La arquitectura orientada a objetos y la alta durabilidad convierten a *S3* en una opción ideal para almacenar archivos como reportes generados, documentos PDF o imágenes asociadas a registros. Sin embargo, el uso de *S3* supone considerar aspectos como la eventual consistencia en operaciones de lectura posterior a escritura, los costos asociados al tráfico saliente y la seguridad de acceso mediante políticas de *Identity and Access Management* (IAM). Según Izurieta et al. (2024), investigaciones sobre aplicaciones en la nube recomiendan utilizar mecanismos de cifrado

y control de versiones, así como implementar reglas de ciclo de vida que automaticen la transición de objetos a clases más económicas.

2.2.5 Apache POI y altChunk

Apache POI es un proyecto de código abierto que proporciona bibliotecas en Java para leer y escribir formatos de *Microsoft Office* tales como Word, Excel y PowerPoint. La misión del proyecto es ofrecer *APIs* tanto para los formatos *OLE2* (HSSF para Excel, HWPF para Word, etc.) como para *Office Open XML* (XSSF, XWPF, XSLF). Estas bibliotecas permiten generar documentos dinámicos desde código, lo que resulta esencial para automatizar la creación de informes, contratos o certificados. Uno de los elementos de la especificación *OOXML* es, un marcador que indica dónde debe insertarse contenido externo en un documento Word. Según la especificación, este elemento define la ubicación de un archivo externo y ordena al procesador de Word que integre su contenido en el documento principal (İRen et al., 2021).

La combinación de *POI* con permite incrustar fragmentos generados previamente o plantillas completas en un documento final, facilitando la generación de reportes complejos sin reconstruir todo el archivo desde cero. No obstante, existen limitaciones: el contenido externo debe cumplir las restricciones de relación de *OOXML* y el tamaño de los fragmentos puede afectar al rendimiento de la generación. Según (Achachlouei et al., 2021) y (İRen et al., 2021), algunos estudios sobre automatización documental recomiendan emplear para insertar secciones estáticas y usar *APIs de alto nivel* (XWPF) para completar los datos variables.

2.2.6 Buenas prácticas de logging y manejo de excepciones

El *logging* y el manejo de *excepciones* son preocupaciones transversales en microservicios. Estudios sobre observabilidad en *Java* destacan que un sistema de monitorización y registro debe proporcionar información en tiempo real sobre actividad de la aplicación, indicadores de

rendimiento y trazado de errores. Lama (2025), revisa prácticas de *monitoreo y logging* en microservicios *Java* y concluye que la adopción de herramientas como Prometheus, Grafana y la pila ELK (Elasticsearch, Logstash, Kibana) permite recopilar métricas y registros de forma centralizada y estructurada, facilitando la detección proactiva de fallos y la mejora de la calidad del *software*. El estudio subraya la importancia de emplear *logs* estructurados y correlacionar eventos mediante identificadores de solicitud para reconstruir el flujo de llamadas.

Para el manejo de excepciones en *Spring*, autores como (De Padua y Shang, 2017) señalan que no deben capturarse en los métodos del controlador, sino delegarse a componentes especializados (por ejemplo, @ControllerAdvice) que traduzcan las excepciones a respuestas *HTTP* coherentes. Esta separación evita duplicar código y permite una respuesta uniforme ante fallos. Lama (2025) también enfatiza que el uso de *tracing* distribuido junto con logs estructurados mejora la observabilidad en ecosistemas de microservicios. En resumen, la combinación de *logging* centralizado, trazabilidad y manejo global de excepciones contribuye a un backend robusto y mantenible.

2.3 Automatización documental

2.3.1 Concepto de automatización documental

La *automatización documental* se refiere al conjunto de tecnologías y procesos que permiten generar documentos de manera automática a partir de datos e información proveniente de diferentes fuentes. Un estudio sobre arquitecturas y tecnologías de *automatización documental* define que la automatización reduce el esfuerzo manual al ensamblar documentos mediante plantillas predefinidas y la integración de entradas de datos heterogéneas. El objetivo es garantizar que los documentos resultantes cumplan con formatos y normas establecidas, disminuyendo errores humanos y tiempos de elaboración (Achachlouei et al., 2021). Desde una perspectiva

empresarial, la *automatización documental* digitaliza y simplifica la creación, población y ruta de documentos, acelerando los ciclos de aprobación y mejorando el servicio al cliente (Gani et al., 2024).

Los sistemas de automatización suelen apoyarse en motores de plantillas que combinan variables y estructuras condicionales, permitiendo generar versiones personalizadas de un mismo formulario o contrato. Para lograrlo se requiere vincular bases de datos o servicios de *backend* que suministren la información. La investigación de Achachlouei et al. (2021) destaca que un sistema de *automatización documental* debe integrarse con servicios de *autenticación* (por ejemplo, JWT) y almacenamiento (por ejemplo, S3) para gestionar tanto la seguridad como el almacenamiento masivo de archivos.

2.3.2 Sistemas de Gestión Documental (EDMS)

Los *sistemas de gestión electrónica de documentos* (EDMS, por sus siglas en inglés) son plataformas que permiten almacenar, organizar y recuperar documentos de forma segura. Se originaron en la década de 1980 para gestionar *información electrónica no estructurada*, combinando aplicaciones para el almacenamiento, organización, recuperación y eliminación de documentos. En el contexto actual, los *EDMS* no sólo facilitan la clasificación y búsqueda de archivos, sino que también registran el historial de modificaciones, gestionan permisos de acceso y automatizan flujos de trabajo (Gani et al., 2024).

Las ventajas de los *EDMS* incluyen la transparencia, el soporte a sistemas de gestión de calidad, la reducción del espacio de almacenamiento físico, el acceso rápido a la información, la flexibilidad para los usuarios y la optimización de procesos de negocio. Sin embargo, existen desventajas como la necesidad de que toda la información esté en formato electrónico, la resistencia al cambio de los empleados y los costos iniciales de implantación. Según estudios de

Suárez y Vázquez (2021) y Roque (2021), indican que un *EDMS* eficaz debe ofrecer indexación automática, *control de versiones*, integración con *sistemas de autenticación* y cumplimiento de normativas como GDPR o la legislación local.

2.3.3 Beneficios y riesgos de la automatización documental

Implementar *automatización documental* aporta beneficios significativos. Las investigaciones muestran que reduce tareas manuales repetitivas y elimina errores de entrada de datos, lo que disminuye costos operativos y acelera los *ciclos de aprobación*. Además, fortalece el *cumplimiento normativo* al garantizar que los documentos generados sigan formatos y fórmulas predefinidas y permite rastrear modificaciones para auditorías, como señalan Suárez y Vázquez (2021). En un contexto de procesos educativos y gubernamentales, la automatización mejora la transparencia y la experiencia del usuario al ofrecer documentos consistentes y oportunos.

No obstante, existen riesgos. Algunos autores señalan que depender de plantillas rígidas puede conducir a una falta de flexibilidad y a un posible 'lock-in' con determinadas tecnologías o proveedores. La calidad de los documentos generados depende de la precisión de los datos de entrada y de las reglas de negocio programadas; cualquier error en estas fuentes se replicará de forma masiva. Además, la automatización conlleva consideraciones de seguridad, ya que los sistemas deben proteger datos sensibles y cumplir con regulaciones de protección de datos. Por ello Roque (2021) recomienda que la planificación de la *automatización documental* debe incluir estrategias de validación de datos, control de versiones y mecanismos de retroalimentación para mantener la calidad.

2.4 Inteligencia artificial y visión computacional aplicada a documentos

2.4.1 NL2SQL con modelos de lenguaje

Los sistemas de *traducción de lenguaje natural* a *SQL* (NL2SQL) utilizan *modelos de lenguaje* (LLM) para interpretar consultas en *lenguaje natural* y generar sentencias *SQL* que puedan ser ejecutadas en bases de datos. Mohammadjafari et al. (2025) explican que estos sistemas facilitan que usuarios sin conocimientos técnicos consulten datos complejos, democratizando el acceso y reduciendo la dependencia de especialistas. En particular, los modelos de *transformers* y las *técnicas de aprendizaje por refuerzo* han mejorado la precisión de las traducciones de *NL2SQL*, aunque requieren grandes volúmenes de datos etiquetados para entrenar.

Oracle Select AI, mencionado anteriormente, aplica NL2SQL dentro de una base de datos autónoma. Al combinar los metadatos del esquema con un *LLM* alojado en *Oracle Cloud*, el sistema genera consultas semánticamente correctas y restringidas según los privilegios del usuario. Oluwafemi (2025) señala que los sistemas *NL2SQL* deben lidiar con ambigüedades del lenguaje natural, referencias implícitas y sinónimos. Por ello, incorporan técnicas de desambiguación y retroalimentación donde el modelo sugiere varias interpretaciones y el usuario puede seleccionar la correcta.

2.4.2 OCR y extracción de texto

El *reconocimiento óptico de caracteres* (OCR) es una tecnología que convierte imágenes de texto en texto editable. Un estudio comparativo de bibliotecas *OCR* de código abierto destaca que *Tesseract* es una de las herramientas más precisas y adaptables, con soporte para múltiples idiomas y una alta exactitud en distintos sistemas de escritura. Otras bibliotecas como EasyOCR, MMOCR y PaddleOCR ofrecen interfaces modernas y entrenamiento personalizado, pero Tesseract sobresale por su madurez y comunidad (Nguyen et al., 2022).

En contextos de *automatización documental*, la *OCR* se utiliza para extraer datos de formularios escaneados o capturas de pantalla. La calidad de la imagen y el preprocesamiento (eliminación de ruido, corrección de inclinación) influyen significativamente en la precisión. Según Jayatilleke y Silva (2025) la integración de *OCR* con sistemas de análisis semántico y revisión humana proporciona un equilibrio entre automatización y exactitud.

2.5 Modelos de proceso de desarrollo

2.5.1 Scrum

Scrum es un *marco de trabajo ágil* para gestionar proyectos complejos que se centra en la entrega *incremental de valor*. Schwaber y Sutherland definieron *Scrum* como un *marco ligero* en el que un equipo multidisciplinar trabaja en iteraciones llamadas *sprints*, usualmente de dos a cuatro semanas, para entregar incrementos de producto potencialmente utilizables. Aunque la guía oficial no está indexada en una revista académica, diversos estudios sistemáticos sobre *Scrum* reconocen sus pilares de *transparencia, inspección y adaptación* y destacan su aplicación en entornos de *software*, como lo señala Zalimben (2022). Sassa et al. (2023) muestran que *Scrum* fomenta la colaboración continua con los clientes, prioriza la adaptabilidad ante cambios y reduce la documentación excesiva.

En *Scrum* existen roles definidos: el *Product Owner*, responsable de maximizar el valor del producto; el *Scrum Master*, que guía al equipo en el marco, y el *Equipo de Desarrollo*, que es autoorganizado. Las ceremonias comprenden la planificación del *sprint*, las reuniones diarias (daily stand-up), la revisión del *sprint* y la retrospectiva. Varios trabajos de investigación, como los Bautista (2022) y , comparan *Scrum* con *metodologías tradicionales* y concluyen que su adopción mejora la satisfacción del cliente y acelera la entrega, aunque su éxito depende del compromiso del equipo y de la madurez organizacional.

2.5.2 Proceso Unificado Racional (RUP)

El *Proceso Unificado Racional* (RUP) es un proceso iterativo de desarrollo de *software* propuesto originalmente como Rational Objectory Process y luego adoptado por *IBM*. El análisis realizado por Anwar (2014) lo describe como un marco orientado a objetos y habilitado para la web, diseñado para proporcionar directrices, plantillas y ejemplos para todas las fases del desarrollo. *RUP* define cuatro fases macro: inicio, elaboración, construcción y transición, y dentro de cada fase se abordan nueve disciplinas (requisitos, análisis y diseño, implementación, pruebas, etc.). Este enfoque estructurado se apoya en la definición de roles y artefactos y usa la notación UML para modelar los componentes.

Los defensores de *RUP* argumentan que su carácter *iterativo* y su énfasis en la arquitectura permiten gestionar riesgos de manera temprana y asegurar la calidad desde las primeras iteraciones. Anwar (2014) destaca que *RUP* es un proceso rico en herramientas y plantillas, lo que puede resultar pesado para proyectos pequeños. Comparado con *Scrum*, *RUP* requiere mayor documentación y una planificación más detallada, pero ofrece mayor control y rastreabilidad. Diversos estudios sugieren que una combinación híbrida, adoptando la flexibilidad de *Scrum* y la disciplina de *RUP*, puede ser apropiada para proyectos medianos y grandes (Anwar, 2014; Jatnika et al., 2023).

2.5.3 Comparativa Scrum vs RUP

Aunque *Scrum* y *RUP* comparten principios iterativos, difieren en su formalidad y en la cantidad de documentación requerida. Sassa et al. (2023) observan que *Scrum* es ligero y se centra en entregar valor rápidamente, mientras que *RUP* es un proceso prescriptivo que prescribe roles, artefactos y fases. El uso de *RUP* puede proporcionar una mayor gobernanza y visibilidad, útil en organizaciones con exigencias regulatorias. Por otra parte, *Scrum* permite adaptarse a los cambios

de requisitos y fomenta la autoorganización. Las investigaciones de Arregocés et al. (2022) recomiendan seleccionar o combinar estos enfoques según el tamaño y complejidad del proyecto, la cultura organizacional y las exigencias de conformidad.

2.6 Modelo de calidad FURPS+

2.6.1 Dimensiones FURPS+

El modelo *FURPS* fue creado por Robert Grady en Hewlett-Packard para categorizar los requisitos de *software* en cinco dimensiones: Funcionalidad (F), Usabilidad (U), Fiabilidad (R, del inglés Reliability), Rendimiento (P, Performance) y Soportabilidad (S, Supportability). Eeles amplió el modelo a *FURPS+* para incluir aspectos adicionales como restricciones de diseño, requisitos físicos y cuestiones de usabilidad. Este modelo se incorporó posteriormente al *Proceso Unificado* de *IBM* y sirve como guía para clasificar y priorizar requisitos no funcionales (Suhartono y Indriyanti, 2025).

En *FURPS+*, la dimensión de funcionalidad se refiere a las características necesarias para cumplir con el propósito del sistema; usabilidad abarca la facilidad de uso, accesibilidad y experiencia del usuario; fiabilidad incluye disponibilidad, seguridad y manejo de errores; rendimiento contempla tiempos de respuesta y uso de recursos; y soportabilidad aborda la capacidad de mantenimiento, portabilidad y compatibilidad. Los requisitos adicionales (‘+’) incluyen restricciones de entorno, normativas y limitaciones tecnológicas. Diversos estudios, como los de Suhartono e Indriyanti (2025) e Yungan Gualli et al. (2019), demuestran que la aplicación sistemática de *FURPS+* facilita la trazabilidad entre requisitos, diseño y pruebas.

2.6.2 Uso de FURPS+ en la ingeniería de requisitos

La *ingeniería de requisitos utiliza FURPS+* para identificar y documentar expectativas de los usuarios y restricciones técnicas. El modelo ayuda a separar requisitos funcionales de no

funcionales y permite priorizar en función de la criticidad. Izurieta et al. (2024) señalan que incluir FURPS+ en las historias de usuario facilita la verificación y validación, ya que cada historia puede asociarse con criterios específicos de fiabilidad o rendimiento.

2.6.3 Conexión FURPS+ – historias de usuario – pruebas

Para asegurar la coherencia entre *requisitos* y verificaciones, es recomendable mapear cada historia de usuario con sus atributos *FURPS+*. Así, por ejemplo, una historia sobre la generación de reportes se vincula con la usabilidad (facilidad de uso), fiabilidad (manejo de errores) y rendimiento (tiempo de respuesta). Cada atributo se convierte en criterio de aceptación y se traduce en pruebas unitarias, de *integración* y de *aceptación*. Yungan Gualli et al. (2019) y Suhartono e Indriyanti (2025) subrayan que esta trazabilidad no sólo ayuda a validar los entregables, sino que también facilita la comunicación con los interesados y prioriza la mejora continua.

2.7 Contexto de proyectos de vinculación

2.7.1Marco normativo ecuatoriano

En el contexto ecuatoriano, los proyectos de *vinculación universitaria* están regulados por la Ley Orgánica de Educación Superior (LOES) y las resoluciones del Consejo de Educación Superior (CES). Estas normas establecen que las *instituciones de educación superior* deben desarrollar proyectos que contribuyan al bienestar de la sociedad y fortalezcan la formación integral de los estudiantes. Los proyectos deben articular la *docencia*, *investigación* y *vinculación*, generando soluciones a problemas locales. Aunque no existen artículos científicos que discutan en detalle la implementación de LOES, las universidades publican informes institucionales que describen sus experiencias de vinculación (Universidad de Cuenca, 2024; Universidad de Azuay, 2023).

2.7.2 Ciclo de vida e indicadores de impacto

Los proyectos de vinculación siguen un *ciclo de vida* que incluye las fases de diagnóstico, planificación, ejecución, monitoreo y cierre. En cada fase se establecen objetivos específicos y se recogen indicadores de impacto social, como el número de beneficiarios, la mejora en procesos institucionales o la generación de oportunidades laborales. En la tesis se emplean indicadores propuestos por la Universidad de Cuenca y alineados con los Objetivos de Desarrollo Sostenible. Estos indicadores permiten evaluar la pertinencia y sostenibilidad de la solución propuesta y sirven para reportar resultados a los organismos reguladores.

Capítulo 3

3. Marco Metodológico

3.1 Desarrollo

El desarrollo del sistema se realizó de forma *iterativa* e *incremental*, priorizando entregas funcionales verificables al final de cada iteración. Se adoptó *Scrum* como marco principal para gestionar el *ciclo de vida*, complementándolo con elementos de *RUP* que aportaron rigor en la definición de artefactos (casos de uso, modelos UML, lineamientos de calidad y criterios de aceptación). Esta combinación permitió mantener un ritmo sostenido de construcción, asegurar trazabilidad entre requisitos y pruebas, y sostener un nivel de documentación alineado con las exigencias académicas y de auditoría interna.

3.1.1 Enfoque general (Scrum + elementos RUP)

Cadencia y ceremonias. El proyecto se ejecutó en ocho sprints de dos semanas cada uno. En cada sprint se llevaron a cabo: *planning* para priorizar *historias de usuario* y estimarlas; *dailies* para coordinar avances y desbloquear impedimentos; *reviews* con demostraciones de *incrementos* potencialmente desplegables; y retrospectives para incorporar mejoras de proceso. El *product backlog se* refinó de forma continua para mantener una priorización basada en valor y riesgo técnico.

Criterios de aceptación (Definition of Done). Cada historia de usuario se consideró completada únicamente cuando:

El *endpoint* correspondiente respondió conforme a especificación (códigos HTTP esperados, validaciones de entrada y manejo de errores consistente).

- Los datos se persistieron de forma íntegra y verificable (consultas posteriores devolvieron el estado esperado).

- Se generó o regeneró el documento institucional sin bloquear el hilo principal, dejando evidencia de su disponibilidad para descarga.
- Se registraron pruebas asociadas (unitarias/funcionales) y la historia quedó trazada en la matriz FURPS+ ↔ HU ↔ Pruebas.

Aportes de RUP al proceso. Paralelo a las iteraciones de Scrum, se mantuvieron artefactos formales: modelo de casos de uso con actores desagregados y relaciones <<include>>/<<extend>>, diagramas de secuencia con flujos alternos y excepciones, y lineamientos de calidad (validación estricta, manejo de errores, reintentos en servicios externos y procesamiento asíncrono). Esto permitió que cada incremento tuviera respaldo de diseño, documentación y criterios claros de verificación.

Justificación del híbrido Scrum/RUP. *Scrum* aportó ligereza y entrega temprana de valor mediante iteraciones cortas, feedback continuo y priorización adaptativa; *RUP* añadió gobernanza documental y rigurosidad en la especificación y el modelado, indispensables para proyectos académicos y contextos con auditoría. En conjunto, el híbrido garantizó velocidad de construcción sin sacrificar trazabilidad, calidad técnica ni claridad de artefactos.

3.1.2 Comparativa metodológica

Tabla 1

Comparación entre modelo en cascada y Scrum aplicados al contexto del proyecto

Criterio	Modelo en cascada	Scrum	Relevancia para este trabajo
Planificación	Línea temporal fija por fases secuenciales.	Iteraciones cortas con ajuste continuo del alcance.	Necesitábamos reaccionar a cambios de requisitos y hallazgos técnicos sin

			replanificar todo el proyecto.
Gestión del riesgo	Riesgos se abordan temprano en documentación, pero impacto se ve tarde.	Riesgos se reducen con entregas frecuentes y feedback por sprint.	La validación temprana de endpoints y documentos redujo retrabajo y aceleró correcciones.
Entrega de valor	Entrega al final del ciclo.	Entrega incremental al finalizar cada sprint.	Se mostraron incrementos verificables (API + documento) de forma periódica.
Documentación	Extensa y previa a la construcción.	Suficiente y viva; se actualiza por iteración.	Mantuvimos documentación esencial (UML, criterios, trazabilidad) sin frenar el avance.
Cambios de alcance	Costosos y tardíos.	Esperados y gestionados por prioridad.	Ajustamos historias y criterios según pruebas y revisiones docentes.
Calidad técnica	Dependiente de controles al final.	Integrada (DoD, pruebas y revisiones por sprint).	La calidad (validaciones, errores, rendimiento) se aseguró iteración a iteración.

Visibilidad del progreso	Basada en documentos y hitos de fase.	Basada en incrementos demostrables y métricas de sprint.	Las *reviews* con demostraciones facilitaron decisiones informadas.
Adecuación al contexto	Útil en requisitos muy estables y procesos lineales.	Ideal en entornos con aprendizaje y evolución del producto.	El proyecto requería prueba rápida de generación documental y ajustes de dominio.
Riesgo de retrabajo	Alto si surgen cambios tardíos.	Bajo-moderado por iteraciones y feedback continuo.	Se corrigieron supuestos de negocio a tiempo, evitando refactorizaciones masivas.
Gobernanza	Fuerte control por fase, riesgo de burocracia.	Ligero, riesgo de documentación insuficiente si no se cuida.	Se compensó la ligereza de Scrum con artefactos RUP (UML, trazabilidad, criterios).

Nota. Dado que el dominio y los requisitos evolucionaron durante el desarrollo, Scrum fue más adecuado para asegurar entregas tempranas y ajuste continuo, mientras que los elementos de RUP garantizaron la formalidad necesaria en modelado, documentación y control de calidad. Este balance permitió sostener el ritmo sin perder trazabilidad ni rigor metodológico. Elaboración propia.

3.2 Requisitos y trazabilidad (FURPS+ - Historias de Usuario - Pruebas)

Tabla 2

Autenticación & Usuarios — HU ↔ RF/NF ↔ FURPS+

HU	RF/NF	U	R	P	S	+
HU-01 Registro	RF-01 · NF-01/02/03	✓	✓			✓
HU-02 Login	RF-01 · NF-02/03	✓				✓
HU-03 Recuperar/Reset	RF-01 · NF-03	✓	✓			
HU-04 Listar usuarios	RF-02 · NF-02/03	✓				✓
HU-05 Mis proyectos	RF-03 · NF-02/03	✓				✓

Nota. Los criterios están redactados en GWT y anclados a códigos HTTP para verificación objetiva. Elaboración propia.

Tabla 3

Autenticación & Usuarios — Criterios

HU	Given/When (acción)	Then (HTTP esperado)	Validación / Errores
HU-01	POST /api/auth/registro con datos válidos	**201** (o 200) + usuario	**400** validación; **409** email duplicado.
HU-02	POST /api/auth/login	**200** + JWT	**401** credenciales inválidas.
HU-03	POST /forgot-password / reset-password	**200**	**400** token inválido/expirado.
HU-04	GET /api/usuarios con JWT	**200**	**401** sin token

HU-05	GET /api/proyectos/mis-proyectos	**200** (lista del owner)	**401** sin token

Nota. Los criterios están redactados en GWT y anclados a códigos HTTP para verificación objetiva. Elaboración propia.

Tabla 4

Autenticación & Usuarios — Pruebas

HU	Unit (qué cubre)	Func (endpoints y códigos)	Carga (si aplica)
HU-01	Validadores de DTO	Registro: 200/201, 400, 409	N/A
HU-02	Servicio de auth	Login: 200, 401	N/A
HU-03	Flujo de token	Forgot/Reset: 200, 400	N/A
HU-04	Usuario service	Listar: 200, 401	N/A
HU-05	Proyecto service	Mis proyectos: 200, 401	Lecturas

Nota. Cada HU declara el mínimo de pruebas necesarias para evidenciar cumplimiento; "Carga" solo cuando aporta valor. Elaboración propia.

Tabla 5

Datos de Proyecto — HU ↔ RF/NF ↔ FURPS+

HU	RF/NF	U	R	P	S	+
HU-06 Crear datos	RF-04 · NF-01/02/03	✓				✓
HU-07 Actualizar datos	RF-05 · NF-01/02/03	✓				✓
HU-08 Detalle	RF-06 · NF-02/03	✓				✓

Nota. Se exige validación estricta (NF-01) y permisos (NF-02) antes de tocar recursos sensibles. Elaboración propia.

Tabla 6

Datos de Proyecto — Criterios

HU	Given/When (acción)	Then (HTTP esperado)	Validación / Errores
HU-06	POST /api/datosProyecto/guardar (multipart válido)	**201**	**400** validación; **401** sin token; **403** sin permisos.
HU-07	PUT /api/datosProyecto/{id}	**200**	**400** validación; **401** sin token; **403** sin permisos; **404** id inválido.
HU-08	GET /api/datosProyecto/detalle/{id}	**200**	**401** sin token; **403** no miembro/owner; **404** no existe.

Nota. Los criterios dejan claro quién puede ejecutar cada acción (autorización previa explícita). Elaboración propia.

Tabla 7

Datos de Proyecto — Pruebas

HU	Unit	Func	Carga
HU-06	DTO/servicio	201, 400, 403	N/A
HU-07	Servicio	200, 400, 403, 404	N/A
HU-08	Repo/servicio	200, 403, 404	Lecturas

Nota. Las pruebas funcionales cubren códigos de error además del caso feliz. Elaboración propia.

Tabla 8

Documentos DOCX — HU ↔ RF/NF ↔ FURPS+

HU	RF/NF	U	R	P	S	+

HU-09 Generar DOCX (sync)	RF-07 · NF-01/03/04	✓		✓		✓
HU-10 Descargar DOCX (sync)	RF-08 · NF-02/03/04/05	✓	✓	✓		✓
HU-11 DOCX asíncrono (Job/SSE)	RF-09 · NF-02/03/04/06	✓	✓	✓	✓	✓

Nota. HU-11 introduce asincronía y SSE; HU-10 aplica reintentos S3 (NF-05). Elaboración propia.

Tabla 9

Documentos DOCX — Criterios

HU	Given/When (acción)	Then (HTTP esperado)	Validación / Errores
HU-09	POST /api/documento/generar/{id} (owner)	**200** (doc generado)	**400** validación; **403** permisos; **500** plantillas.
HU-10	GET /api/documento/descargar/{id}	**200** binario	**401** sin token; **404** no existe en S3; **503** tras reintentos (S3).
HU-11	POST /api/documento/jobs ⇒ crea; GET /jobs/{jobId}; GET /jobs/{jobId}/events; GET /jobs/{jobId}/download	**202** (crear); **200** (estado/SSE/descarga si DONE)	**401** sin token; **409** no listo; **403** otro usuario; **404** job inexistente.

Nota. Se modela el ciclo de vida del job: PENDING→RUNNING→DONE|ERROR y descarga solo en DONE. Elaboración propia.

Tabla 10

Documentos DOCX — Pruebas

HU	Unit	Func	Carga
HU-09	Generador DOCX	200, 400, 403, 500	p95 objetivo
HU-10	S3 (retry/@Recover)	200, 404, 503	Descargas
HU-11	Worker async / Job store	202, 200, 409, 403, 404	Concurrencia (crear/estado)

Nota. En carga, reportar p50/p95/p99, RPS y error%. Elaboración propia.

Tabla 11

Documentación, Anexos, Firmado e Invitaciones — HU ↔ RF/NF ↔ FURPS+

HU	RF/NF	U	R	P	S	+
HU-12A Subir anexo	RF-10 · NF-02/03/05	✓	✓			✓
HU-12B Actualizar anexo (reemplazo/metadata)	RF-10 · NF-01/02/03/05	✓	✓			✓
HU-12C Eliminar anexo	RF-10 · NF-02/03/05	✓	✓			✓
HU-13A Listar metadata de anexos	RF-11 · NF-02/03/05	✓	✓			✓
HU-13B Descargar anexo	RF-11 · NF-02/03/05	✓	✓			✓
HU-14A Cargar documento firmado (crear)	RF-12 · NF-02/03	✓	✓			✓
HU-14B Actualizar documento firmado (metadata)	RF-12 · NF-01/02/03	✓	✓			✓
HU-15A Obtener metadata documento firmado	RF-13 · NF-02/03/05	✓	✓			✓
HU-15B Descargar documento firmado	RF-13 · NF-02/03/05	✓	✓			✓
HU-15C Reemitir documento firmado	RF-13 · NF-02/03/05	✓	✓			✓

HU-16A Listar invitaciones	RF-14/15 · NF-02/03	✓	✓
HU-16B Agregar invitación	RF-14/15 · NF-01/02/03	✓	✓
HU-16C Quitar invitación	RF-14/15 · NF-02/03	✓	✓

Nota. Se aplican reglas de permisoService (dueño, miembro, invitado). Operaciones con binarios usan S3 con retry controlado. Elaboración propia.

Tabla 12

Documentación/Anexos/Firmado/Invitaciones — Criterios

HU	Given/When (acción)	Then (HTTP esperado)	Validación / Errores
HU-12A	**POST** /api/proyectos/{proyectoId}/anexos (owner)	**201 Created**	**400** archivo inválido; **401** sin token; **403** sin permisos; **413** tamaño; **503** tras reintentos (S3)
HU-12B	**PUT** /api/proyectos/{proyectoId}/anexos/{anexoId}	**200**	**400** validación; **401** sin token; **403**; **404** anexo; **413**; **503** tras reintentos (S3)
HU-12C	**DELETE** /api/proyectos/{proyectoId}/anexos/{anexoId}	**204 No Content**	**401** sin token; **403**; **404** anexo; **503** tras reintentos (S3)
HU-13A	**GET** /api/proyectos/{proyectoId}/anexos/metadata	**200**	**401** sin token; **403**
HU-13B	**GET** /api/proyectos/{proyectoId}/anexos/{anexoId}/descargar	**200** (binario)	**401** sin token; **403**; **404** no existe; **503** tras reintentos (S3)
HU-14A	**POST** /documento-firmado (cargar firmado)	**201 Created**	**400** validación; **401** sin token; **403**

HU-14B	**PUT** /documento-firmado/{docFirmadoId}	**200**	**400** validación; **401** sin token; **403**; **404** inexistente
HU-15A	**GET** /documento-firmado/{docFirmadoId}/metadata	**200**	**401** sin token; **403**; **404** inexistente
HU-15B	**GET** /documento-firmado/{docFirmadoId}/descargar	**200** (binario)	**401** sin token; **403**; **404**; **503** tras reintentos (S3)
HU-15C	**PUT** /documento-firmado/{docFirmadoId}/reemitir	**200**	**401** sin token; **403**; **404**; **503** tras reintentos (S3)
HU-16A	**GET** /documento-firmado/{docFirmadoId}/invitaciones	**200** (lista)	**401** sin token; **403**; **404** doc
HU-16B	**POST** /documento-firmado/{docFirmadoId}/invitaciones	**201 Created**	**400** validación (email/rol); **401** sin token; **403**; **404** doc; **409** ya invitado
HU-16C	**DELETE** /documento-firmado/{docFirmadoId}/invitaciones/{invitacionId}	**204 No Content**	**401** sin token; **403**; **404** invitación/doc

Nota. En anexos/firmado, los errores S3 siguen la política de reintentos y 503 final controlado.

Elaboración propia.

Tabla 13

Documentación/Anexos/Firmado/Invitaciones — Pruebas

HU	Unit	Func	Carga
HU-12A	Validador archivo (tipo/tamaño)	200, 400, 403, 413, 503	N/A

HU-12B	Servicio actualización anexo	200, 400, 403, 404, 413, 503	N/A
HU-12C	Servicio eliminación anexo	204, 403, 404, 503	N/A
HU-13A	Permisos listado metadata	200, 403	Lecturas
HU-13B	S3 retry en descarga	200, 403, 404, 503	Descargas concurrentes
HU-14A	Servicio “cargar firmado”	2xx, 400, 403	N/A
HU-14B	Validación/servicio actualización firmado	200, 400, 403, 404	N/A
HU-15A	Reglas permisos metadata	200, 403, 404	Lecturas
HU-15B	S3 retry descarga firmado	200, 403, 404, 503	Descargas
HU-15C	Servicio reemisión/versionado	200, 403, 404, 503	N/A
HU-16A	Permisos listar invitaciones	200, 403, 404	N/A
HU-16B	Validación invitación (email/rol/duplicado)	201, 400, 403, 404, 409	N/A
HU-16C	Servicio eliminar invitación	204, 403, 404	N/A

Nota. En Func siempre se incluyen rutas de error (no solo “happy path”). Elaboración propia.

Tabla 14

Objetivos, Select AI y Croquis — HU ↔ RF/NF ↔ FURPS+

HU	RF/NF	U	R	P	S	+
HU-17 Objetivos (general/específicos)	RF-16 · NF-02/03	✓				✓
HU-18 Select AI (consulta NL)	RF-17 · NF-01/02/03/05	✓	✓			✓

HU-19 Descargar croquis	RF-18 · NF-02/03/05	✓	✓	✓

Nota. Para AI, la política de "sin resultados" es 200 OK con resultado vacío (no 204). Elaboración propia.

Tabla 15

Objetivos, Select AI y Croquis — Criterios

HU	Given/When (acción)	Then (HTTP esperado)	Validación / Errores
HU-17	GET /api/proyectos/{proyectoId}/objetivos/general **y** GET /api/proyectos/{proyectoId}/objetivos/especificos	**200 OK** (lista)	**401** sin token; **403** no miembro/owner; **404** proyecto/recurso inexistente
HU-18	GET /api/ai/ask?q=...&mode=... (miembro)	**200 OK** (**[]** si vacío)	**400** q/mode inválidos; **401** sin token; **403** no miembro; **503** caída de fuente externa
HU-19	GET /api/croquis/descargar/{croquisId}	**200 OK** (binario)	**401** sin token; **403** si no autorizado; **404** no existe en S3; **503** tras reintentos (S3)

Nota. En AI, "Then" explicita la UX: OK incluso sin filas; los errores son por validación, permisos o dependencia externa. Elaboración propia.

Tabla 16

Objetivos, Select AI y Croquis — Pruebas

HU	Unit	Func	Carga
HU-17	Servicio de objetivos	200, 403, 404	Lecturas
HU-18	Mapper/handler AI	200 (vacío), 400, 403, 503	N/A
HU-19	S3 / permisos	200, 403, 404, 503	Descargas

Nota. Prioriza reproducibilidad: fija datos semilla y casos bordes (IDs inválidos, sin permisos). Elaboración propia.

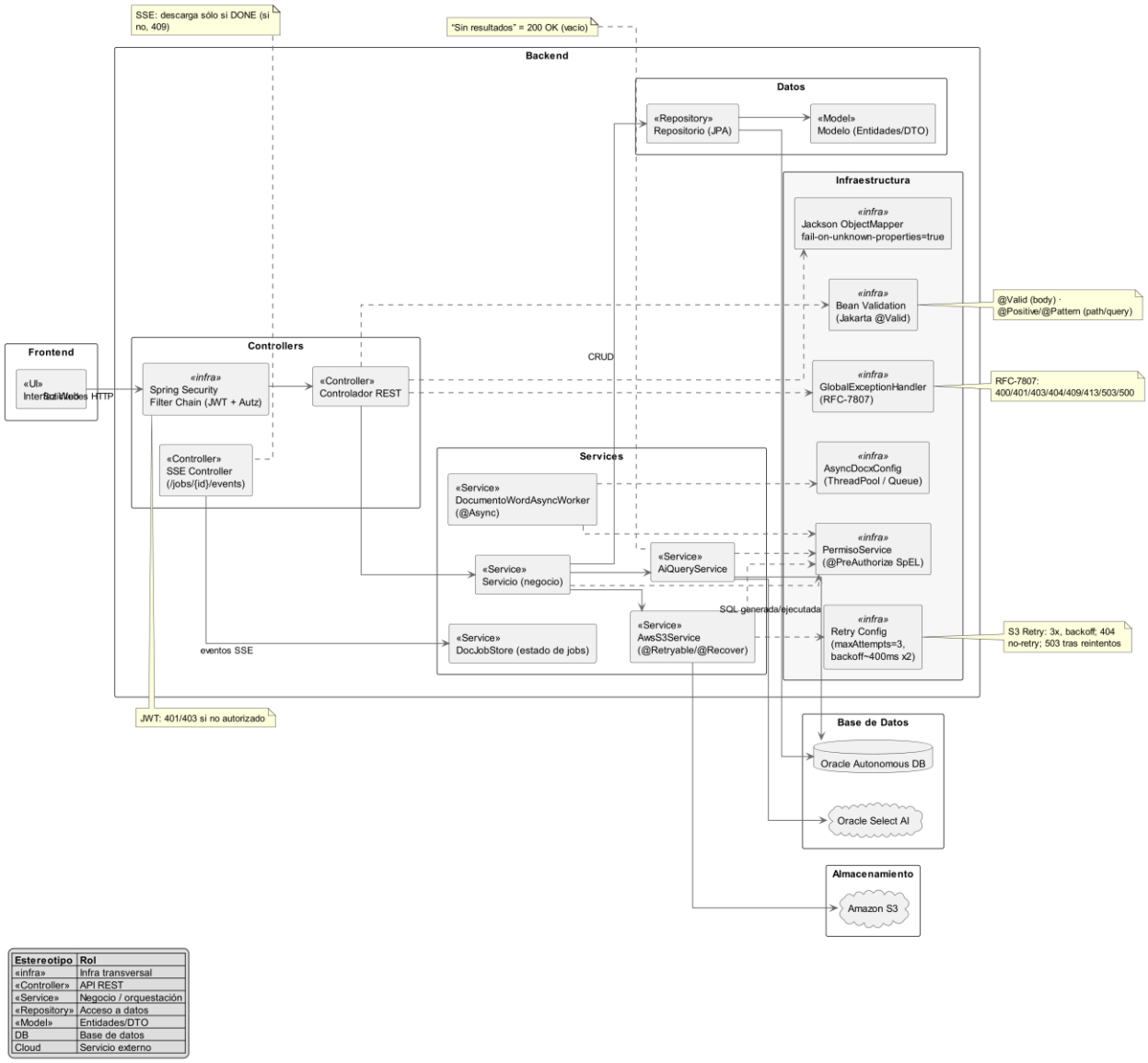
Esta matriz funciona como contrato verificable entre la intención (HU + RF/NF) y la evidencia (pruebas), alineando decisiones de diseño exigir @Validated/@Valid, filtrar permisos con @PreAuthorize, modelar asincronía con *jobs* y *SSE*, incorporar reintentos con *backoff* en dependencias con criterios de aceptación medibles (Given–When–Then con códigos *HTTP* y métricas p50/p95/p99). En implementación, se convierte en la *Definition of Done*: una *HU* solo "termina" si cumple esos criterios y queda cubierta por pruebas unitarias (reglas/validadores), funcionales/API (endpoints + errores) y, cuando aplica, de carga (latencias, throughput, error%). En verificación y auditoría, la trazabilidad HU↔RF/NF↔Pruebas permite detectar brechas, justificar decisiones y demostrar calidad de extremo a extremo con resultados reproducibles.

3.3 Modelado UML del sistema

3.3.1 Arquitectura lógica (resumen)

Figura 1

Arquitectura lógica del backend



Nota. La banda Infraestructura agrupa componentes transversales: JWT (autenticación/autorización), validación estricta (@Valid en body y @Positive/@Pattern en path/query, Jackson con fail-on-unknown-properties=true), GlobalExceptionHandler (RFC-7807),

PermisoService (@PreAuthorize), configuración Async (pool/cola) y Retry a S3 (3 intentos, backoff; 404 no reintenta; 503 tras reintentos). Flechas sólidas = flujo principal; punteadas = dependencias transversales. SSE publica progreso y la descarga solo procede en DONE (si no, 409). En Select AI, "sin resultados" se devuelve como 200 OK con cuerpo vacío. Elaboración propia.

El sistema adopta una *arquitectura por capas* que separa responsabilidades y hace explícitas las dependencias externas. Toda solicitud del *Frontend* atraviesa la cadena de seguridad (JWT) antes de llegar a los controladores *REST*, donde se aplican validaciones de datos y se normalizan errores mediante *RFC-7807*. La capa de servicios concentra la lógica de negocio y los permisos declarativos (*@PreAuthorize*), orquestando tanto operaciones sincrónicas (CRUD vía repositorios JPA sobre la BD relacional) como asíncronas: la generación de documentos se delega a un worker con pool/cola, su estado queda en *DocJobStore* y el *SSE* emite eventos de progreso hasta permitir la descarga; si se intenta descargar antes de *DONE* se responde 409. Para almacenamiento, *AwsS3Service* implementa reintentos con *backoff* (NF-05); cuando el objeto no existe (404) no reintenta, y si la dependencia permanece indisponible se retorna 503 tras agotar intentos. La integración con *Select AI* traduce consultas en lenguaje natural a *SQL* que se ejecuta en la base de datos y, por diseño de *UX*, "sin resultados" se comunica con 200 OK y cuerpo vacío (no 204). Esta estructuración satisface las observaciones de los pares: validación estricta, manejo uniforme de errores, permisos por recurso, *asincronía* observable (jobs + SSE) y resiliencia ante fallos externos, manteniendo bajo acoplamiento y alta trazabilidad hacia los *RF/NF*.

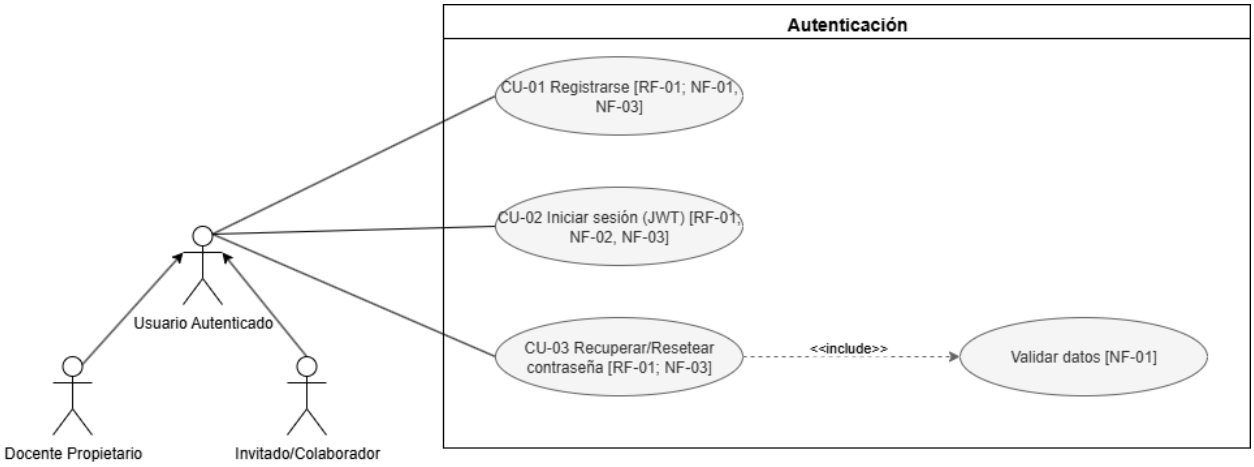
3.3.2 Diagrama de caso de uso

Convenciones (aplican a todas las figuras)

- **Actores:** Docente Propietario e Invitado/Colaborador heredan de Usuario Autenticado.
- **Autenticación:** <<include>> Iniciar sesión (JWT) en CU protegidos.
- **Validación:** <<include>> Validar datos en registrar/crear/actualizar/reemitir/agregar invitación.
- **S3 Retry:** Reintentar descarga (S3 Retry) extiende (<<extend>>) a los CU de descarga.
- **Trazabilidad:** cada CU anota [RF-xx; NF-yy] visible.
- **Política "sin resultados" (AI):** 200 OK con lista vacía.

Figura 2

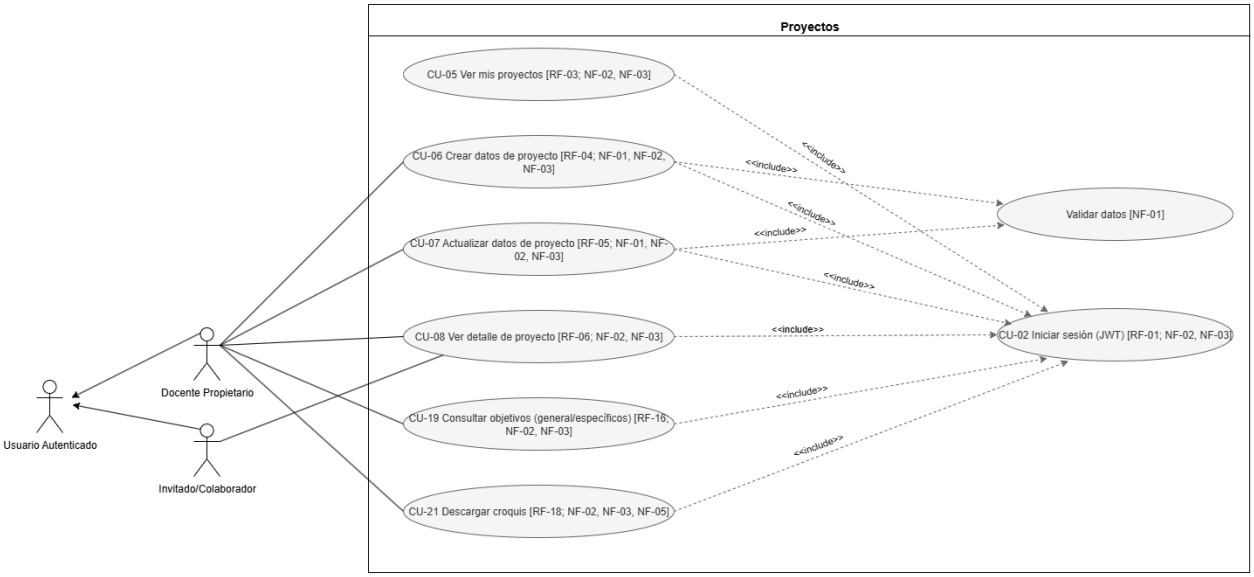
Usuarios - Casos de uso



Nota. Registro y Recuperar/Resetear incluyen Validar datos (NF-01); Login entrega JWT. Se explicita la generalización hacia Usuario Autenticado para mantener coherencia con la autorización del backend (Spring Security).

Figura 3

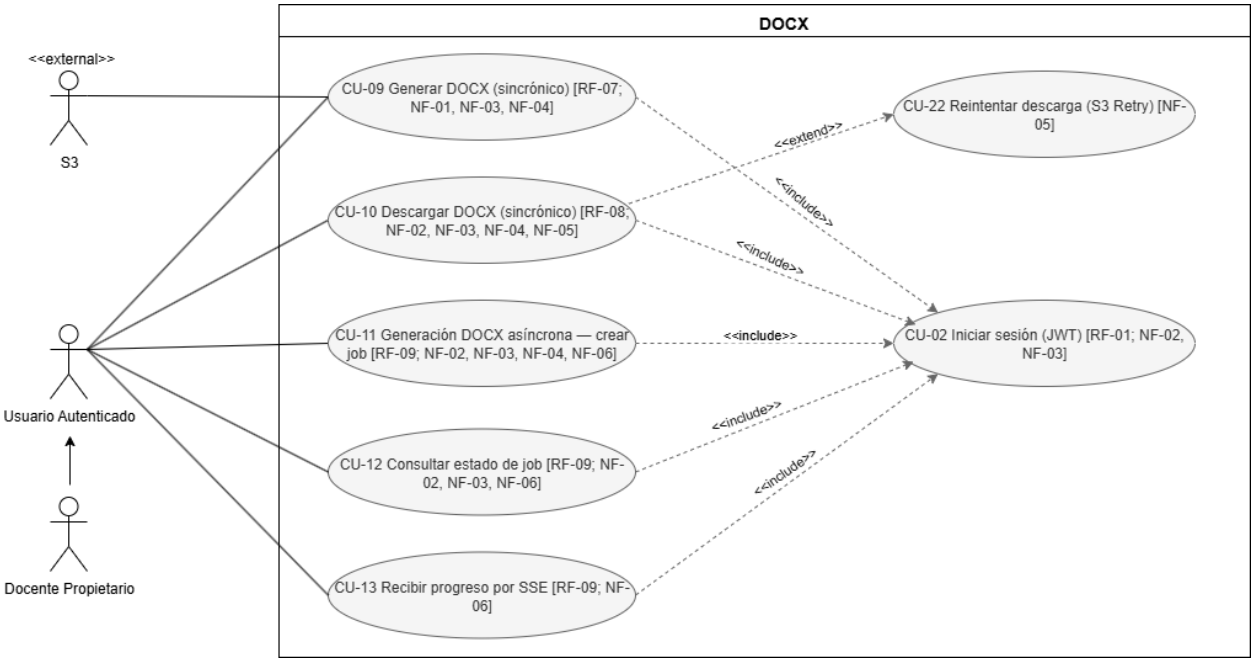
Proyectos - Casos de uso



Nota. Se modelan mis proyectos, crear/actualizar datos, ver detalle, consultar objetivos (general/específicos) y descargar croquis. Los CU de crear/actualizar incluyen Validar datos; todos los protegidos incluyen Iniciar sesión. Trazabilidad: RF-03/04/05/06/16/18 y NF-01/02/03/05.

Figura 4

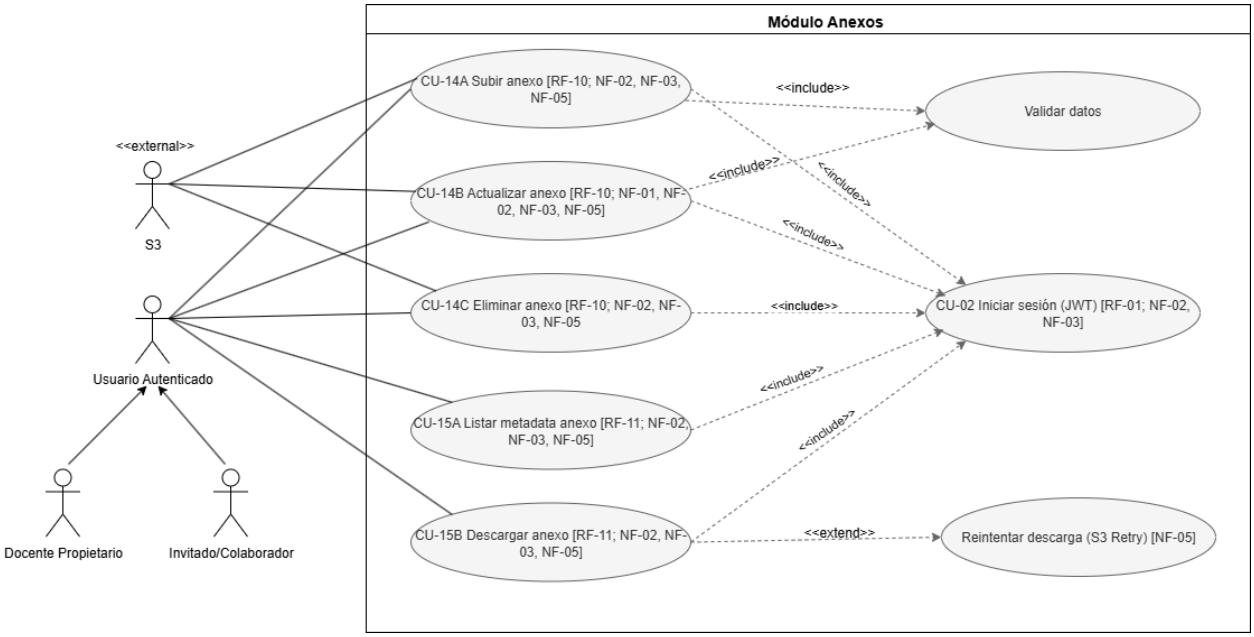
DOCX - Casos de uso



Nota. Generar DOCX (síncrono) y Generación asíncrona (crear job, consultar estado, SSE). Descargar DOCX es extendido por Reintentar descarga (S3 Retry) para fallos transitorios. Todos los CU protegidos incluyen Iniciar sesión.

Figura 5

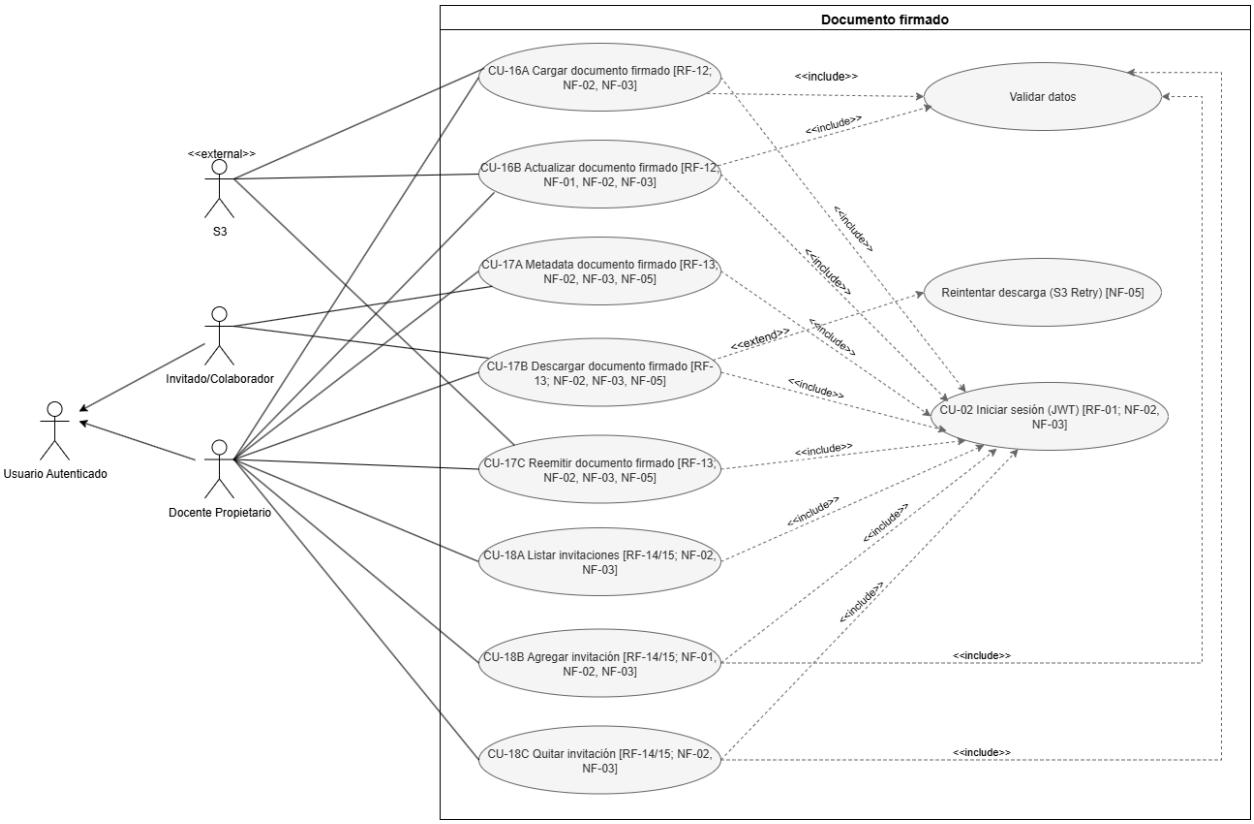
Anexos - Casos de uso



Nota. CRUD granular: Subir (14A), Actualizar (14B), Eliminar (14C); y Listar metadata (15A) / Descargar (15B). Crear/Actualizar incluyen Validar datos. Descargar es extendido por Reintentar descarga (S3 Retry). Accesos a S3 se muestran como actor externo.

Figura 6

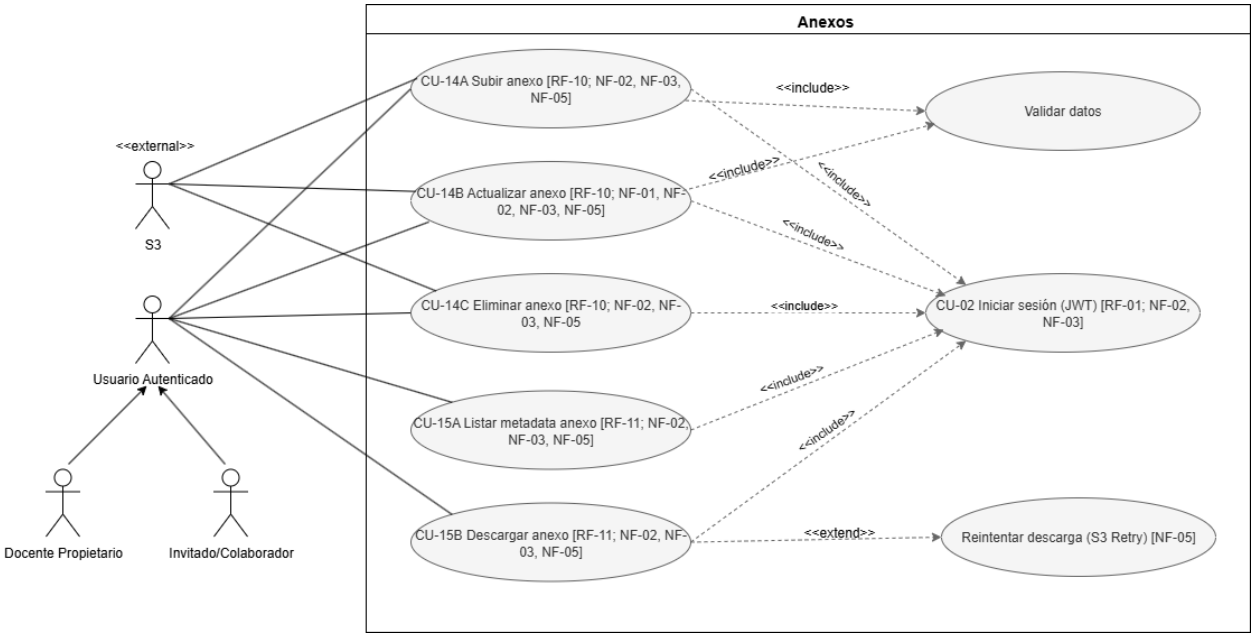
Documento firmado e invitaciones - Casos de uso



Nota. Firmados: Cargar (16A), Actualizar (16B), Metadata (17A), Descargar (17B) (extendido por S3 Retry), Reemitir (17C) (incluye Validar datos). Invitaciones: Listar (18A), Agregar (18B) (incluye Validar datos), Quitar (18C). Invitado/Colaborador puede ver/descargar si fue invitado; la gestión es del Propietario.

Figura 7

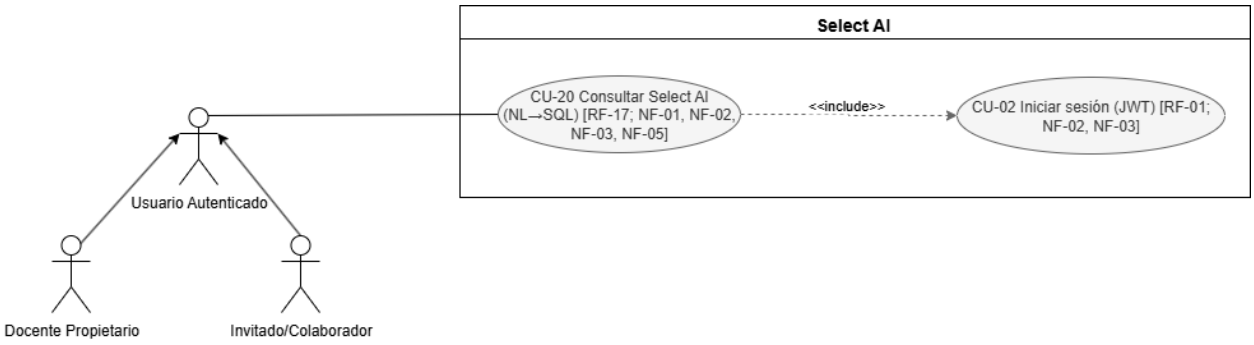
Anexos - Casos de Uso



Nota. CRUD granular: Subir (14A), Actualizar (14B), Eliminar (14C); y Listar metadata (15A) / Descargar (15B). Crear/Actualizar incluyen Validar datos. Descargar es extendido por Reintentar descarga (S3 Retry). Accesos a S3 se muestran como actor externo.

Figura 8

Select AI - Casos de Uso



Nota. Consultar Select AI (NL→SQL) incluye Iniciar sesión (JWT) y aplica la política "sin resultados" (200 OK con lista vacía si no hay filas). Se explicita Select AI como actor externo; el control de permisos se hace antes de ejecutar la consulta.

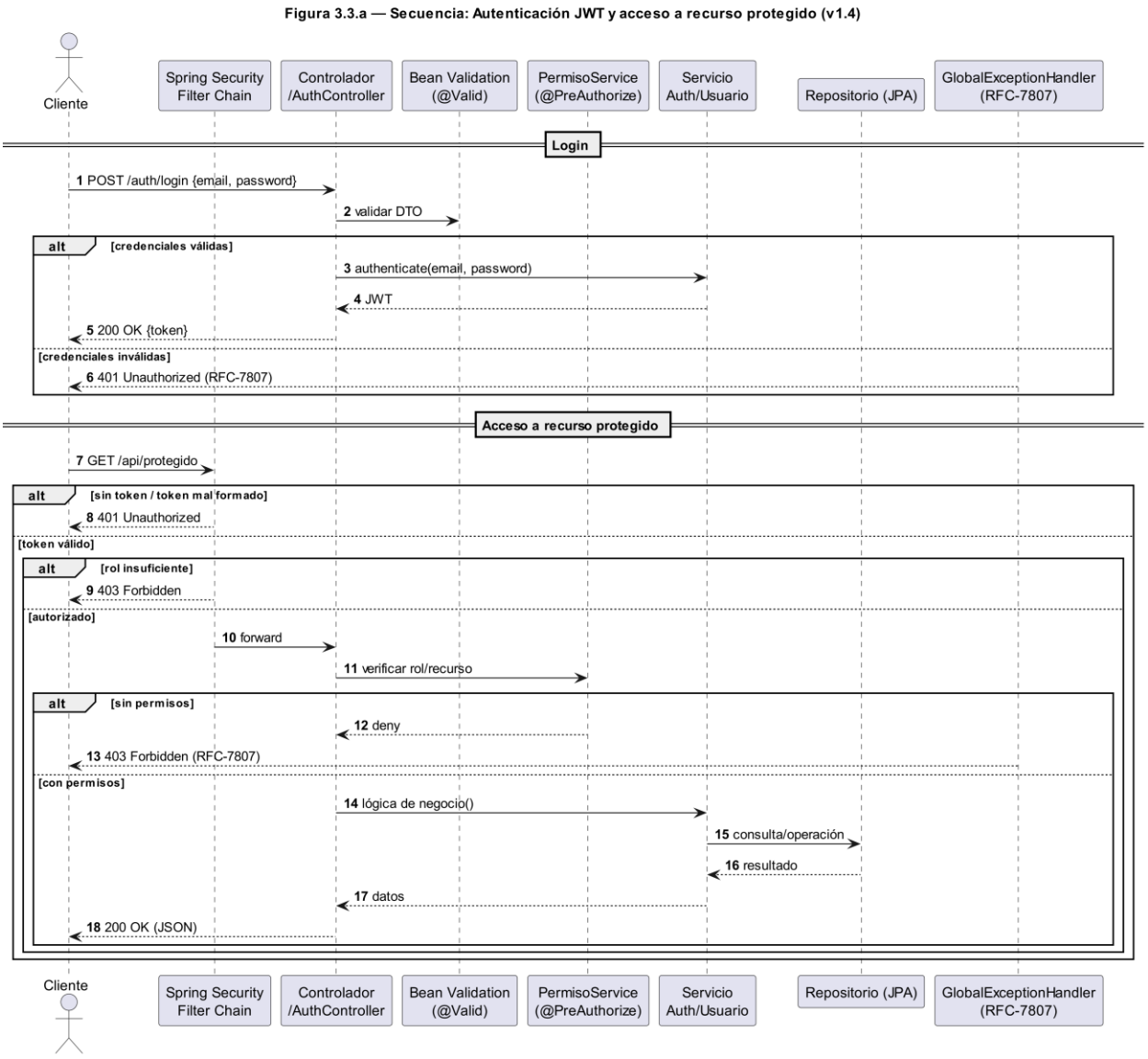
La desagregación de actores, la inclusión explícita de Iniciar sesión (JWT) y Validar datos, y la extensión Reintentar descarga (S3 Retry) alinean los casos de uso con los *filtros de seguridad*, *validaciones* y *mecanismos de resiliencia* implementados en el backend. La subdivisión CRUD evita CU monolíticos y facilita la *trazabilidad* RF/NF y la *verificación* (códigos HTTP esperados) en las pruebas funcionales y de carga.

3.3.3 Diagramas de secuencia

Criterios de modelado. Los diagramas representan flujos principales, alternativas y excepciones con sus códigos *HTTP* (400/401/403/404/409/500/503) en formato *RFC-7807*. Se incluyen pasos intermedios de validación y autorización antes de acceder a recursos sensibles. La generación de *DOCX* se modela mediante job *asíncrono* y *SSE* para progreso, y la descarga implementa reintentos *S3* (máx. 3; 404 sin retry; 503 tras reintentos). La consulta *Select AI* adopta la política "sin resultados" (200 OK con lista vacía). Todos los diagramas se versionan (v1.4) para reflejar la evolución del backend.

Figura 9

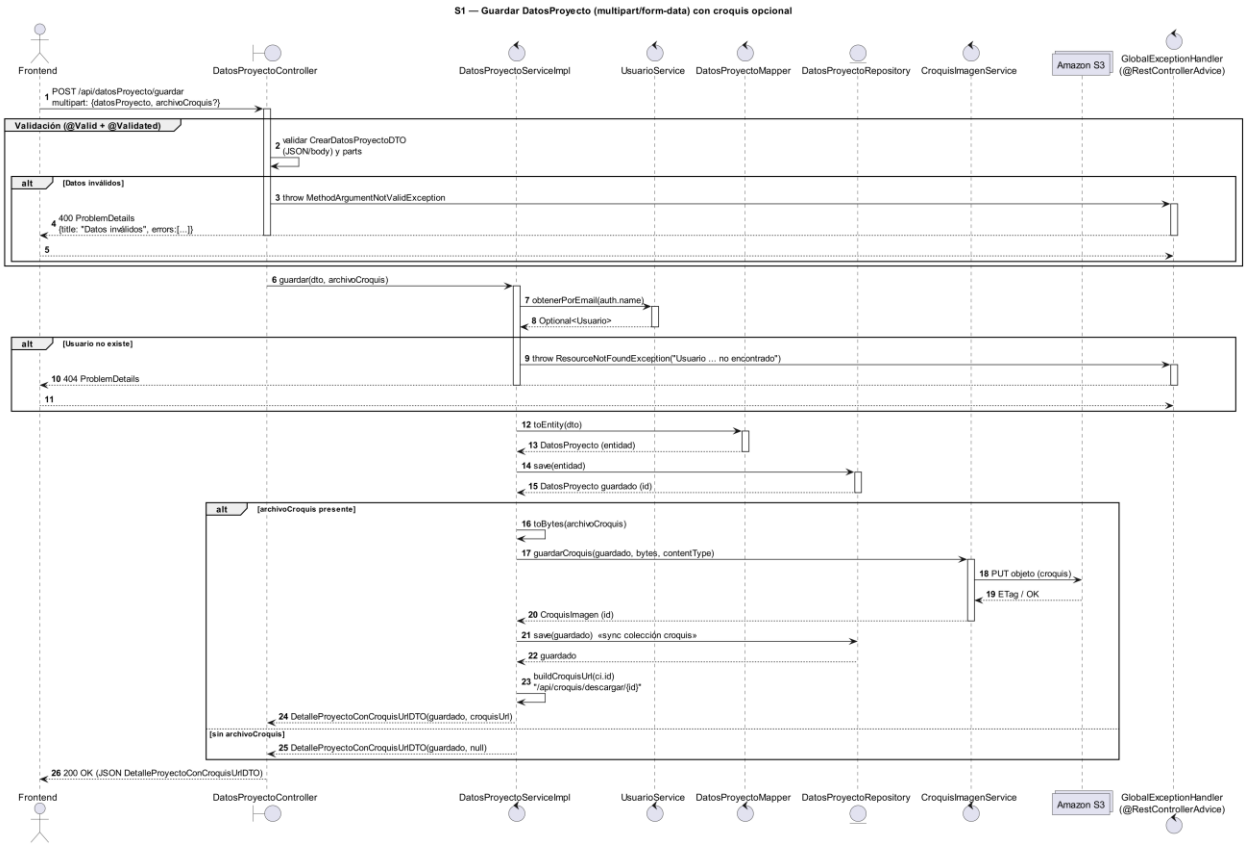
Secuencia: Autenticación JWT y acceso a recurso protegido (v1.4)



Nota. El flujo ilustra validación de entrada (@Valid) en login y la emisión de JWT (200). El acceso a recursos protegidos se filtra en la Security Filter Chain: 401 por token ausente/mal formado y 403 por rol insuficiente mediante @PreAuthorize. Solo tras autorizar se ejecuta la lógica de negocio y consultas a repositorio (200). Los errores se devuelven con detalle en RFC-7807. Elaboración propia.

Figura 10

Secuencia: Guardar datos del proyecto (v1.4)



Nota. El flujo inicia cuando el Frontend envía POST /api/datosProyecto/guardar en multipart/form-data con dos parts: datosProyecto (JSON validado con @Valid) y archivoCroquis (opcional). El Controller valida entrada y delega en DatosProyectoServiceImpl, que (1) obtiene el usuario autenticado desde SecurityContextHolder vía UsuarioService; (2) mapea el DTO a entidad con DatosProyectoMapper; (3) persiste la entidad en DatosProyectoRepository.

Si hay archivo, el servicio convierte a bytes y llama a CroquisImagenService, que almacena el objeto en S3 y devuelve la metadata (CroquisImagen). Se sincroniza la colección en la entidad y se re-guarda para reflejar el vínculo. Finalmente se construye la URL de descarga con

ServletUriComponentsBuilder (/api/croquis/descargar/{id}) y se retorna DetalleProyectoConCroquisUrlDTO (con o sin croquisUrl según exista archivo).

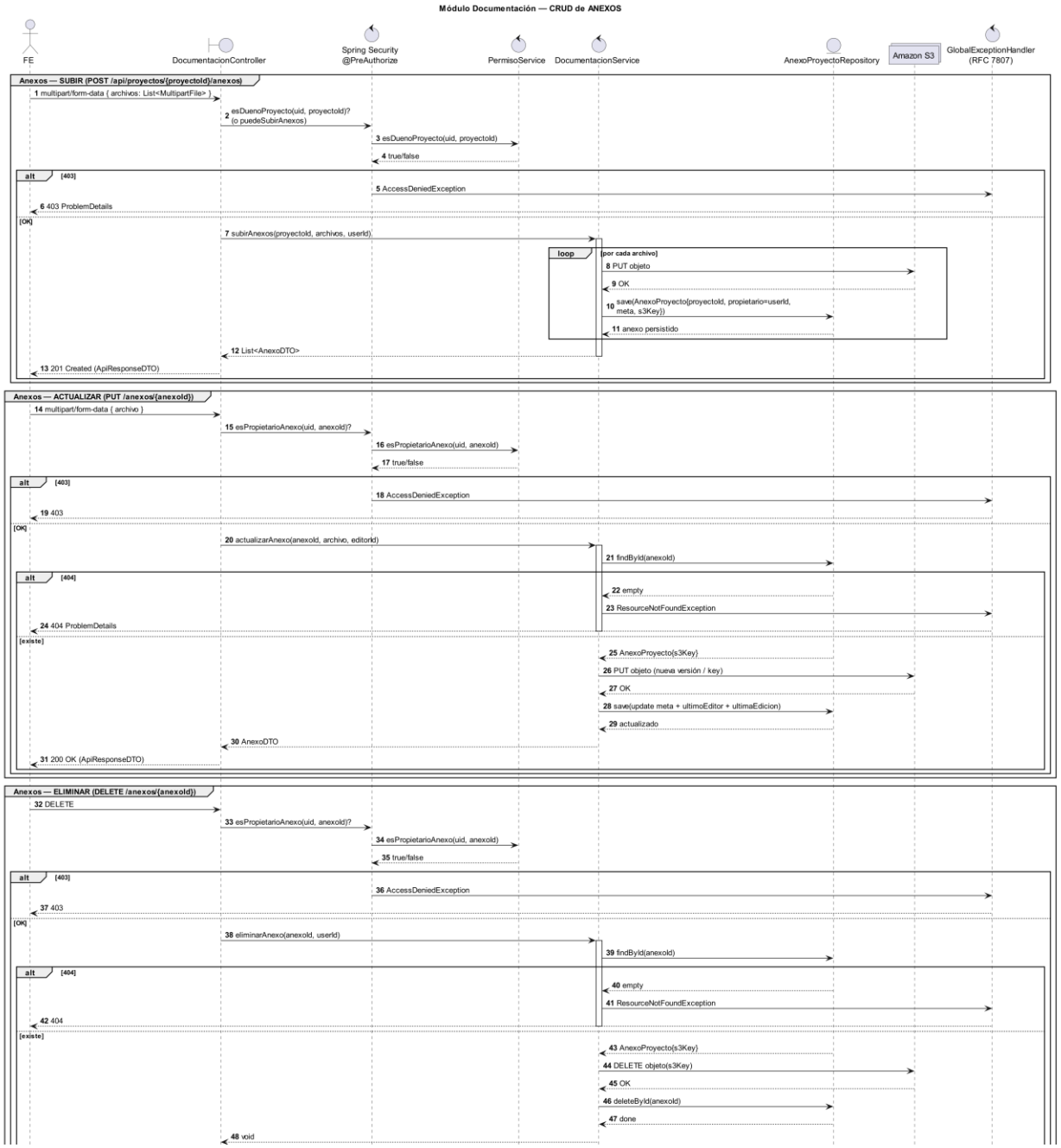
Los errores se gestionan con Problem Details (RFC 7807): 400 por validaciones de body/params o multipart inválido, 404 si no se encuentra el usuario/autorizado o recursos derivados, 413 si el archivo excede el tamaño permitido, 503 por indisponibilidad de S3/BD (timeouts, throttling o transacciones), y 500 ante fallos inesperados (p. ej., E/S). Respuesta exitosa: 200 OK con el DTO. Elaboración propia.

Figura 11

Secuencia: CRUD de documento firmado (v1.4)

Figura 12

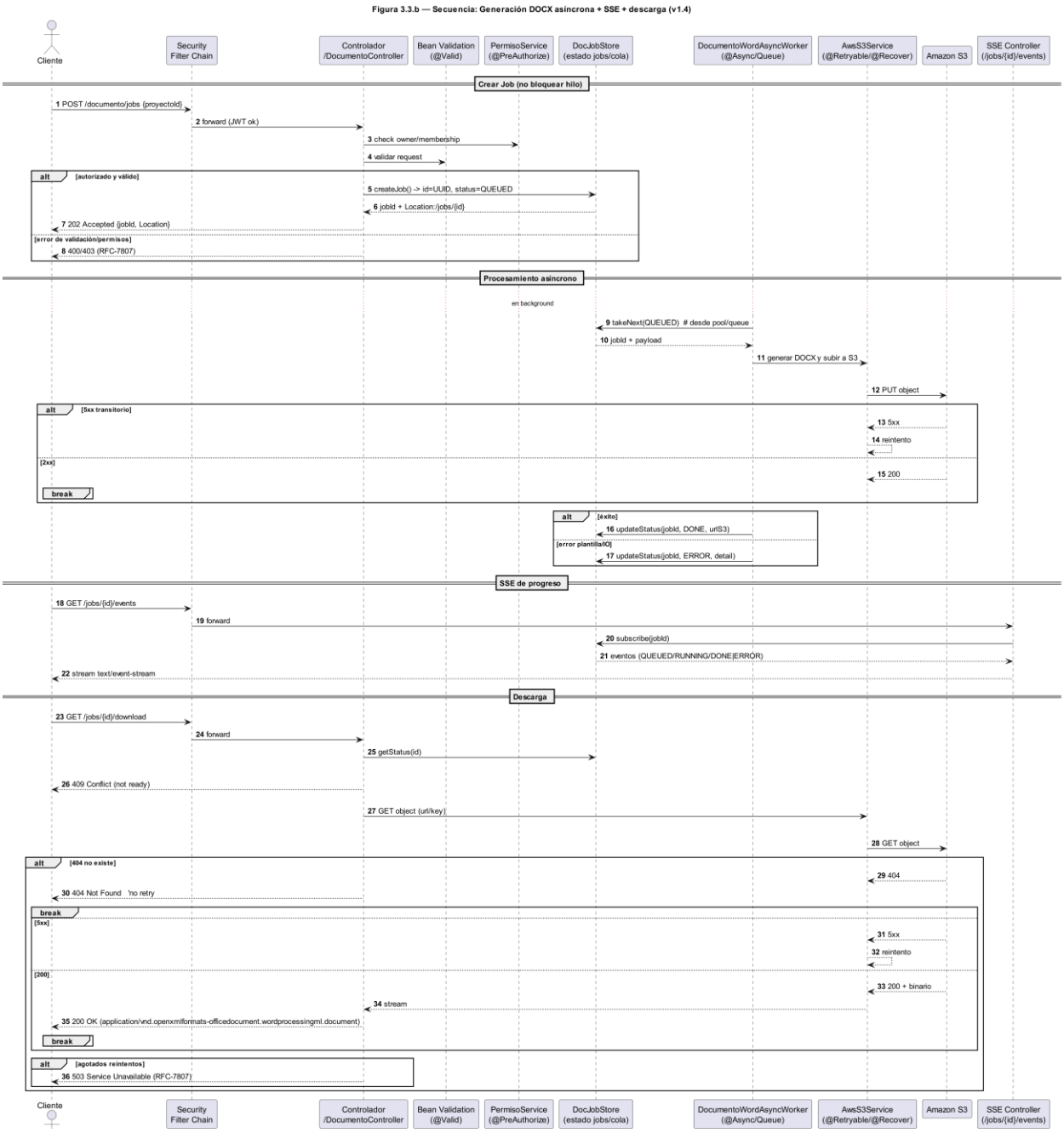
Secuencia: CRUD de Anexos (v1.4)



Nota. Se controlan permisos por rol/propiedad, se suben/actualizan/borran objetos en S3, se persiste metadata en APR, y se exponen descargas con headers correctos. Errores via Problem Details (RFC 7807). Elaboración propia.

Figura 13

Secuencia: Generación DOCX asíncrona con cola/Job, SSE y descarga (v1.4)

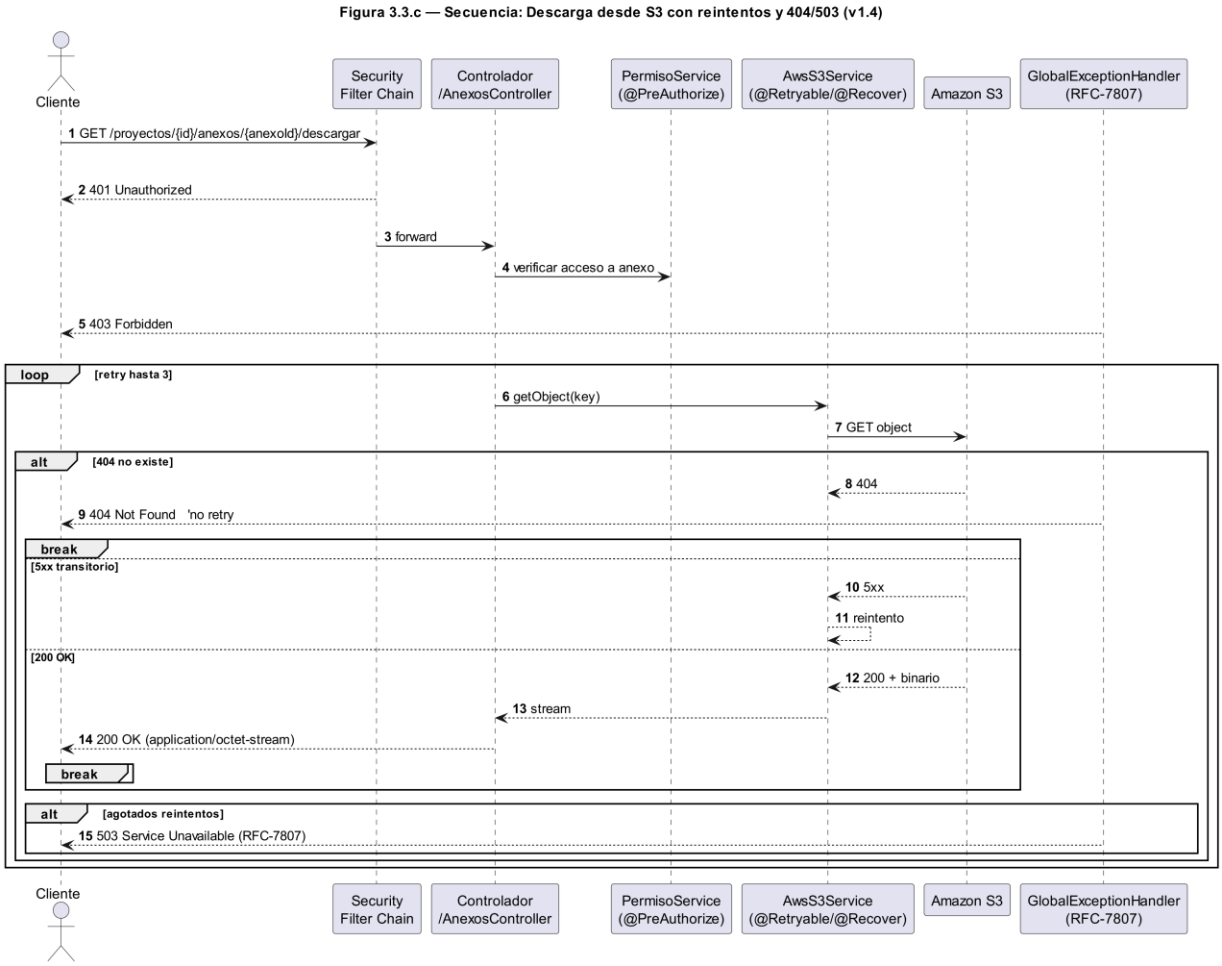


Nota. La creación del job retorna 202 Accepted con Location y no bloquea el hilo principal. Un worker @Async toma la tarea desde la cola, genera el DOCX y lo sube a S3 con política de

reintentos (hasta 3; backoff; 404 sin retry; 503 si se agotan). El progreso se expone por SSE (QUEUED/RUNNING/DONE/ERROR). La descarga solo procede en DONE (si no, 409); respuestas de error en RFC-7807. Elaboración propia.

Figura 14

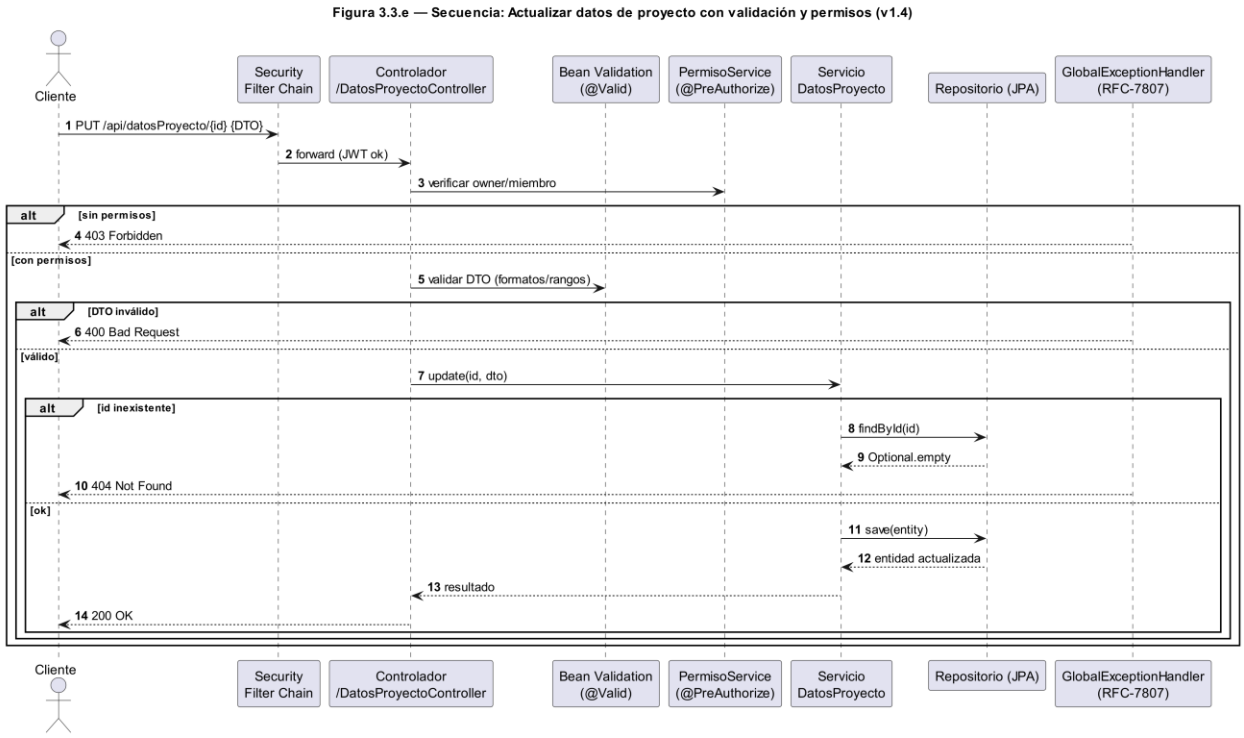
Secuencia: Descarga desde S3 con reintentos y 404/503 (v1.4)



Nota. Antes de acceder al objeto se verifica autorización con @PreAuthorize. La obtención desde S3 aplica reintentos ante fallos transitorios (máx. 3); el 404 Not Found corta el ciclo sin reintento; si persisten errores, se responde 503 Service Unavailable. En éxito, el binario se envía por stream (200). Formato de errores conforme a RFC-7807. Elaboración propia.

Figura 15

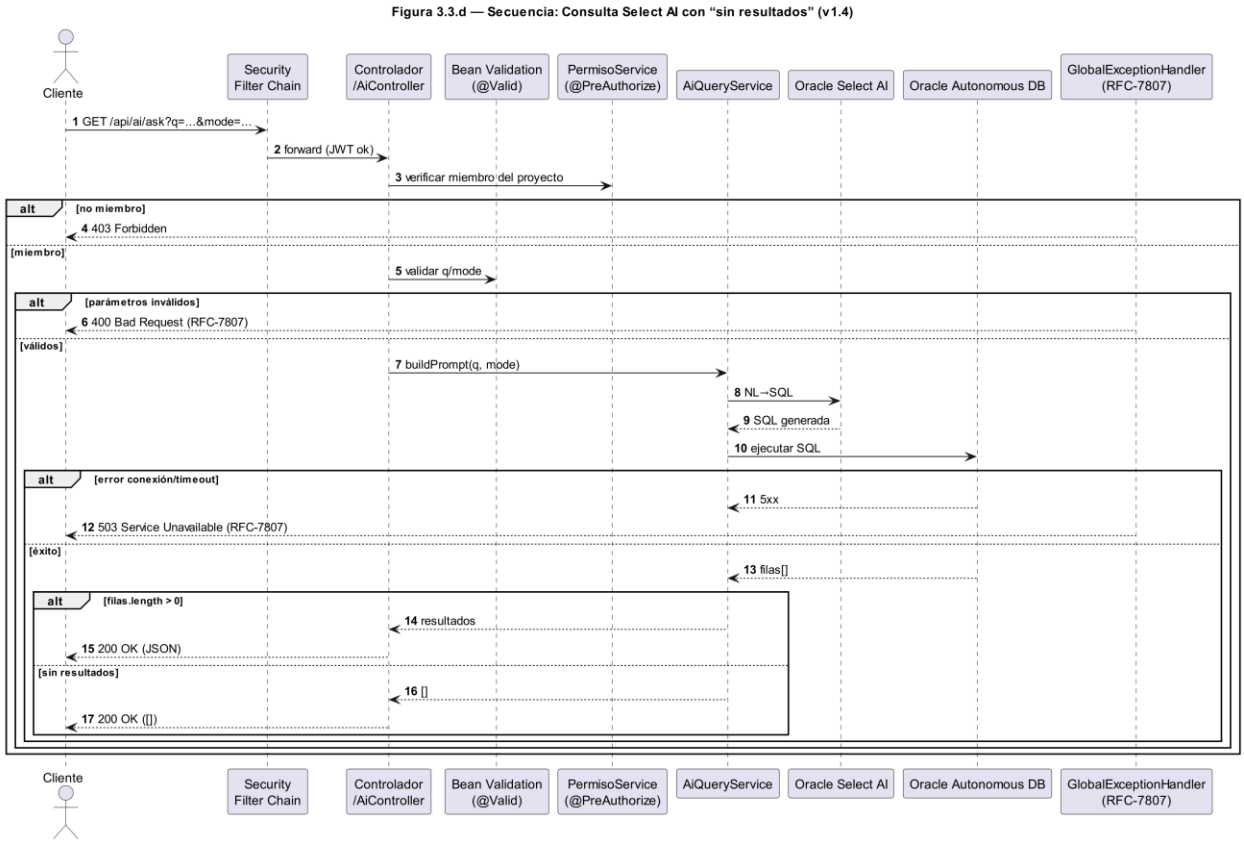
Secuencia: Actualizar datos de proyecto con validación y permisos (v1.4)



Nota. El controlador verifica permisos (403) y ejecuta validación estricta del DTO (400) antes de tocar el repositorio. Si el identificador no existe, se responde 404; en caso exitoso se persiste y retorna 200 OK. El modelado hace explícitos los pasos de filtrado de entrada y autorización previos al acceso a datos, con errores detallados en RFC-7807. Elaboración propia.

Figura 16

Secuencia: Consulta Select AI (NL→SQL) con "sin resultados" (v1.4)



Nota. Se controla pertenencia al proyecto (403 si no) y se valida q/mode (400). Select AI transforma la consulta a SQL, que se ejecuta en la base de datos. Ante fallos de conectividad se retorna 503. Política adoptada para "sin resultados": 200 OK con lista vacía []; en caso de filas, 200 OK con el conjunto correspondiente. Errores formateados como RFC-7807. Elaboración propia.

Los diagramas modelan los cinco flujos clave del backend con camino principal, alternativas y excepciones claramente identificadas: (a) Autenticación *JWT* con rechazo 401 (token ausente/mal formado) y 403 (rol insuficiente); (b) Generación de *DOCX* asíncrona mediante *job* en cola + worker @Async, estados QUEUED/RUNNING/DONE/ERROR, 202 Accepted al crear

y 409 si se intenta descargar antes de *DONE*, además de *SSE* para el progreso; (c) Descarga desde *S3* con reintentos (máx. 3, backoff), política de 404 sin reintento y 503 si se agotan; (d) *Select AI* (NL→SQL) con validación de q/mode, control de membresía del proyecto, manejo de 503 por caída del proveedor y política de "sin resultados" como 200 OK con lista vacía; y (e) Actualización de datos de proyecto con @Valid y verificación de permisos antes de persistir, devolviendo 400/403/404 según corresponda. En todos los flujos, las validaciones de entrada y la autorización (@PreAuthorize) se ubican antes de acceder a recursos sensibles o servicios externos, y los errores se responden en formato *RFC-7807* (400/401/403/404/409/500/503).

Asimismo, los títulos incorporan versionado visible (v1.4) para fijar el estado funcional documentado. Este conjunto de secuencias alinea el comportamiento del sistema con la arquitectura *Controller–Service–Repository*, hace verificables los criterios (códigos HTTP, estados de job, políticas de retry) y sirve como guía directa para las *pruebas unitarias/funcionales/de carga*, asegurando coherencia entre diseño, implementación y evidencia de calidad.

3.4. Implementación de la metodología Scrum

3.4.1 Enfoque híbrido y roles

Se aplicó *Scrum* como marco ágil de organización del trabajo, complementado con prácticas ligeras de *RUP* para garantizar gobernanza documental, trazabilidad y control de riesgos. Dado el carácter unipersonal del proyecto, los roles se mapearon de forma práctica: el Tutor/Coordinación actuó como *Product Owner* (priorización por valor y aceptación del incremento), y el autor asumió las funciones de *Scrum Master* y *Development Team* (planificación, construcción y verificación).

Justificación del híbrido. Scrum proporcionó *cadencia* y *entrega incremental*, mientras que *RUP*, en modalidad "lite", aportó artefactos y puertas de calidad (visión, modelo de casos de uso, arquitectura lógica, matriz de requisitos y riesgos). Esta combinación permitió documentar decisiones y evidenciar cumplimiento de atributos *FURPS*+ (usabilidad, confiabilidad, rendimiento, soporte y requisitos adicionales como seguridad/legal), además de explicitar validaciones, manejo de errores y políticas de reintentos que se reflejaron en criterios de aceptación y diagramas de secuencia.

3.4.2. Artefactos y trazabilidad (Scrum ⇄ RUP ⇄ FURPS+)

- **Product Backlog.** Se redactó en Historias de Usuario (HU) y tareas técnicas, etiquetadas con FURPS+, para asegurar que cada ítem contemplara atributos de calidad junto con la funcionalidad.
- **Artefactos RUP (versión ligera**). Se mantuvieron la Visión y Alcance, el Modelo de Casos de Uso (actores desagregados y relaciones <<include>>/<<extend>>), la Arquitectura Lógica (capas Controller–Service–Repository, seguridad, validación, asincronía y S3), y una Lista de Riesgos revisada por sprint.
- **Especificación de requisitos y trazabilidad.** Se consolidó la matriz HU ↔ RF/NF ↔ FURPS+ ↔ Pruebas (unitarias/funcionales/de carga), con criterios de aceptación medibles (códigos HTTP, validaciones por formato/rango, política S3 con reintentos y política de "sin resultados").
- **Control de cambios y versionado.** Las figuras UML (casos de uso y secuencias) se versionaron (**v1.4**) para fijar el alcance documentado del backend.

Nota metodológica. Cada HU registró *Given–When–Then* con umbrales verificables (p. ej., 409 si el *Job* no está listo, 404 sin reintento en S3, 503 tras agotar reintentos, 200 [] para consultas sin filas) y las reglas de autorización previas a cualquier acceso a recursos sensibles.

3.4.3 Cadencia y eventos

Se trabajó con sprints de dos semanas. En cada iteración se realizó un *Sprint Planning* breve con el tutor (selección de *HU* por valor y dependencia técnica), seguimiento asíncrono de avance, *Sprint Review* con demostraciones en *Postman* (códigos 2xx/4xx/5xx), descarga de binarios desde *S3*, y *Retrospective* enfocada en mejoras técnicas (por ejemplo, separar la generación *DOCX* en etapas, uso *de POI/altChunk*, ajuste de *thread pool* para jobs).

En paralelo, se utilizaron hitos *RUP-lite*: *Inception* (visión y alcance), *Elaboration* (casos de uso y arquitectura lógica), *Construction* (implementación y pruebas). Este enfoque aseguró que las decisiones arquitectónicas y la gestión de riesgos quedaran asentadas tempranamente y que la evidencia de calidad aumentara en cada incremento.

3.4.4 Definición de Hecho (DoD) y criterios de aceptación mínimos

Un ítem se consideró *terminado* cuando cumplió, al menos, los siguientes criterios:

1. *API* operativa con pruebas positivas y negativas (colección Postman/Newman) y códigos *HTTP* esperados.
2. Persistencia *JPA* sin errores y consistencia del modelo de datos.
3. Seguridad aplicada en recursos protegidos (JWT, filter chain, @PreAuthorize).
4. Documento *DOCX* íntegro y con campos correctos cuando la HU lo requería.

Esta DoD se complementó con políticas transversales:

- Errores en formato RFC-7807 (400/401/403/404/409/500/503).
- Reintentos S3 controlados (hasta 3; 404 sin retry; 503 tras agotar intentos).

- Asincronía para generación y descarga de documentos pesados (Job + worker @Async + SSE de progreso; 409 si se intenta descargar antes de DONE).
- "Sin resultados" en Select AI como 200 OK con lista vacía [].

Estas reglas quedaron reflejadas y verificables en los diagramas de secuencia (Sección 3.3.3) y en las matrices HU–Pruebas.

3.4.5 Flujo de trabajo por ítem

Se ejecutó un ciclo corto y repetible para cada HU/tarea:

1. **Refinamiento.** se precisó el alcance con FURPS+, RF/NF y riesgos; se definieron criterios GWT y los códigos HTTP esperados.
2. **Planificación.** se priorizó en el Sprint Planning considerando dependencias (seguridad → persistencia → generación DOCX → anexos/firmado → Select AI).
3. **Implementación.** Se desarrolló en la arquitectura Controller–Service–Repository, cuidando validación de entrada (@Valid, restricciones de formato/rango) y mapeo DTO/entidades.
4. **Autorización.** Se aplicaron permisos antes de tocar recursos sensibles (@PreAuthorize, reglas por rol/propiedad del recurso).
5. Pruebas:
    a. **Unitarias.** Servicios, validadores, handlers).
    b. **Funcionales (Postman).** Casos 2xx y escenarios 4xx/5xx, incluyendo 401/403, 404/409 y 503 tras reintentos S3.
    c. **De carga.** Cuando aportó valor (p50/p95/p99, RPS, error%).
6. **Procesos especiales.** (generación/validación DOCX, Jobs asíncronos y SSE; verificación de descarga con reintentos S3.

7. **Evidencia.** Se registraron **capturas**, **binarios** (DOCX en S3) y colecciones de prueba, asegurando trazabilidad contra la matriz HU ↔ RF/NF ↔ Pruebas.

Este *pipeline* permitió sostener el avance incluso en componentes complejos (por ejemplo, *Apache POI*, *altChunk*, reintentos *S3*) y mantuvo la coherencia entre diseño, implementación y verificación.

3.5 Definición de los Sprints

3.5.1 Cadencia y criterio de agrupación (Scrum ⇄ RUP)

Se trabajó con *sprints bisemanales*. La *planificación* agrupó historias por dependencia técnica y madurez arquitectónica: primero seguridad y persistencia, luego generación de documento, después anexos/firmado sobre *S3*, y finalmente *Select AI* y despliegue. El avance se mapeó a un *RUP-lite*: S1 *Inception*, S2–S3 Elaboration, S4–S7 *Construction*, S8 *Transition*. Cada *sprint* se aceptó con una *Definición de Hecho* (DoD) homogénea: *endpoints* con pruebas positivas/negativas*, persistencia JPA* estable, *JWT/autorización en protegidos*, errores *RFC-7807*, y, cuando aplicó, documento *DOCX* válido. Esta cadencia permitió integrar tempranamente validación estricta, reintentos *S3* (máx. 3; 404 sin retry; 503 tras agotar intentos), *asincronía* Job+SSE y la política *AI* "200 OK []".

Tabla 17

Mapa de sprints (Scrum) ↔ Fase RUP ↔ Objetivo ↔ Entregables

Sprint	Fase RUP (lite)	Objetivo principal	Entregables claves
S1	Inception	Visión, alcance, riesgos y arquitectura objetivo	Visión y alcance; bosquejo de casos de uso; arquitectura lógica (Controller–Service–

			Repository, seguridad, validación, S3, asincronía)
S2	Elaboration I	Autenticación básica + errores normalizados	HU-01/02/03 (registro/login/reset); Security Filter Chain + JWT; RFC-7807 en errores; JPA base; colección Postman (2xx/4xx)
S3	Elaboration II	Usuarios y "Mis proyectos" con permisos y validación	HU-04/05; @PreAuthorize por rol/propietario; validación estricta DTO; actualización matriz HU↔RF/NF↔FURPS+
S4	Construction I	Datos de proyecto (crear/actualizar/detalle) y objetivos	HU-06/07/08; HU-17 (objetivos general/específicos); pruebas 200/400/401/403/404
S5	Construction II	DOCX síncrono y descarga con política S3	HU-09/10; POI/altChunk; S3 con retry (máx=3; 404 sin retry; 503 tras reintentos); p95 objetivo de generación
S6	Construction III	DOCX asíncrono (Job + SSE)	HU-11: crear job (202/Location), estados QUEUED/RUNNING/DONE/ERROR, SSE; **409** si descarga antes de DONE
S7	Construction IV	Documentación: anexos, firmado, invitaciones, croquis	HU-12A/B/C, HU-13A/B, HU-14A/B, HU-15A/B/C, HU-16A/B/C, HU-19; permisos; descargas con S3 Retry

S8	Transition	Select AI + empaquetado/despliegue + cierre	HU-18 (AI NL→SQL con **200 []** si vacío); contenedor y despliegue; estabilización de errores; UML versionado (v1.4)

Nota. Mapa de sprints y fases RUP (lite) con objetivos y entregables observables. Elaboración propia.

3.5.2 Resumen global de sprints (síntesis)

La secuencia de *sprints* consolidó incrementos verificables (API + datos + documento cuando aplicó), con evidencia: *colecciones Postman*, binarios .docx en *S3*, *logs* de *POI/altChunk* y artefacto contenedor desplegado. Se mantuvo una tabla de sprints con objetivo, *RF/NF* cubiertos, *HU* y evidencia.

Tabla 18

Resumen global de sprints (síntesis)

Sprint	**Objetivo**	**HU (id/título)**	**RF/NF cubiertos**	**Incremento verificable**
S1	Alinear visión y arquitectura	— (pre-HU)	—	Documento de visión; diagrama lógico inicial
S2	Autenticación + errores normalizados	HU-01 Registro; HU-02 Login; HU-03 Reset	RF-01; NF-01/02/03	Login/registro operativos; errores RFC-7807; JPA base

S3	Listados y contexto del usuario	HU-04 Listar usuarios; HU-05 Mis proyectos	RF-02/03; NF-02/03	Endpoints protegidos con @PreAuthorize; validación DTO
S4	Datos de proyecto + objetivos	HU-06 Crear; HU-07 Actualizar; HU-08 Detalle; HU-17 Objetivos	RF-04/05/06/16; NF-01/02/03	CRUD parcial estable; objetivos de solo lectura
S5	DOCX sincrónico + descarga	HU-09 Generar DOCX; HU-10 Descargar DOCX	RF-07/08; NF-01/02/03/04/05	DOCX válido; descarga con retry S3 (404/503)
S6	DOCX asíncrono (Job/SSE)	HU-11 Job + SSE + descarga condicionada	RF-09; NF-02/03/04/06	202+Location; estados de job; **409** si no DONE
S7	Documentación integral	HU-12A/B/C; HU-13A/B; HU-14A/B; HU-15A/B/C; HU-16A/B/C; HU-19	RF-10/11/12/13/14/15/18; NF-01/02/03/05	Anexos y firmados con permisos; descargas con retry S3
S8	AI + despliegue	HU-18 Select AI	RF-17; NF-01/02/03/05	Política **"200 []"** si vacío; servicio contenedorizado y desplegado

Nota. La columna "Incremento verificable" apunta a evidencias concretas (colecciones Postman, binarios en S3, registros de ejecución). Elaboración propia.

3.5.3 Descripción breve por sprint

A continuación, se describe, el alcance real de S1–S8, con mapeo *Scrum* ⇄ *RUP* y la evidencia asociada. Cuando correspondió, se incluyeron políticas transversales (RFC-7807, reintentos *S3*, *asincronía* Job+SSE y 200 [] en AI).

Tabla 19

Evidencia por sprint

Sprint	Evidencia técnica	Cómo se verificó
S1	Visión/alcance; diagrama lógico v1	Revisión documental.
S2	Colección Postman auth; RFC-7807; entidades base JPA	200/401/400 en login/registro/reset
S3	Colección usuarios/proyectos; reglas @PreAuthorize	200/401/403; filtros por propietario/miembro
S4	Colección datosProyecto; DTO con @Valid	201/200/400/403/404; aserciones sobre validaciones
S5	DOCX generado; logs POI/altChunk; descarga S3	200 (doc), 404 (no existe), 503 (tras reintentos)
S6	JobStore (estados); SSE en tiempo real	202 (crear), 200 (estado/SSE), 409 (descarga anticipada)
S7	Anexos/firmado/invitaciones; croquis; S3 retry	201/200/204/400/401/403/404/413/503

S8	AI GET /api/ai/ask; contenedor y despliegue	200 [] si vacío; 400/403/503; servicio operativo

Nota. Las evidencias respaldaron la DoD: funcionalidad operativa, seguridad, manejo de errores, S3 con retry y, cuando aplicó, binarios válidos en S3. Elaboración propia.

La planificación bisemanal permitió integrar, por iteraciones, seguridad, persistencia, generación/descarga de documentos con resiliencia (S3 Retry) y asincronía (Job+SSE). Las tablas muestran el encaje *Scrum⇄RUP*, la trazabilidad *HU↔RF/NF*, la evidencia por *sprint* y las métricas objetivo, asegurando verificación objetiva de la *DoD* y alineación con los requisitos de calidad (FURPS+).

3.6 Planificación de los Sprints

3.6.1 Escala de estimación utilizada

Se empleó una escala relativa simple (1–3–5) adecuada al contexto unipersonal. La unidad expresa comparación de esfuerzo, no horas. Esta escala permitió balancear carga por sprint y dividir épicas cuando superaban la capacidad.

Tabla 20

Escala de estimación

Puntos	Significado	Criterios guía	Ejemplos en este proyecto
1	Bajo	Cambios locales, bajo riesgo, sin integración externa	Ajuste de DTO/validación, listado simple con filtro existente
3	Medio	Lógica de negocio + persistencia y/o seguridad	CRUD con validación estricta y @PreAuthorize
5	Alto	Integración externa / asincronía / binarios	DOCX (POI/altChunk), S3 con retry, Job+SSE, Select AI

Nota. La escala es referencial; no se mapeó a horas. Elaboración propia.

3.6.2 Capacidad por sprint

Se planificó una capacidad referencial de 12–16 puntos por *sprint* (≈20–30 h efectivas/2 semanas). El compromiso real se ajustó según riesgo técnico y dependencias (S3, AI, plantillas DOCX).

Tabla 21

Capacidad por sprint (planificada vs. comprometida)

Sprint	Capacidad planificada	Capacidad comprometida	Porcentaje cumplimiento (SP)	Observaciones de ajuste
S1	12–16	14	100%	Definición de alcance/arquitectura (pre-HU)
S2	12–16	14	100%	Autenticación + RFC-7807
S3	12–16	15	100%	Usuarios y Mis proyectos con permisos
S4	12–16	15	98%	Datos de proyecto + objetivos (validación estricta)
S5	12–16	15	100%	DOCX síncrono + S3 Retry
S6	12–16	15	100%	DOCX asíncrono (Job+SSE)
S7	12–16	14	100%	Anexos/firmado/invitaciones/croquis
S8	12–16	13	100%	Select AI + empaquetado/despliegue

Nota. La variación entre planificada y comprometida refleja el ajuste continuo por riesgo (por ejemplo, plantillas DOCX, S3, AI), manteniendo la cadencia de entrega. Elaboración propia.

3.6.3 Planificación de historias por sprint

Se listaron *HU/tareas* por sprint con sus *RF/NF* asociados y estimación (1–3–5). Esta planificación reconstruye el orden técnicamente óptimo observado en ejecución. Se añaden métricas a reportar por sprint (porcentaje de HU cumplidas, cobertura de errores 4xx/5xx y p95 cuando aplica) como evidencia objetiva de calidad.

Tabla 22

Plan de historias por sprint — HU ↔ RF/NF ↔ Estimación (1–3–5)

Sprint	HU	RF/NF vinculados	Estimación	Pruebas previstas
S2	HU-01 Registro; HU-02 Login; HU-03 Reset	RF-01; NF-01/02/03	3, 3, 1	Unit (servicio/validadores); Func (200/400/401)
S3	HU-04 Listar usuarios; HU-05 Mis proyectos	RF-02/03; NF-02/03	1, 3	Func (200/401/403); Unit (permisoService)
S4	HU-06 Crear; HU-07 Actualizar; HU-08 Detalle; HU-17 Objetivos	RF-04/05/06/16; NF-01/02/03	3, 3, 1, 1	Func (201/200/400/403/404); Unit (DTO/servicio)
S5	HU-09 Generar DOCX; HU-10 Descargar DOCX	RF-07/08; NF-01/02/03/04/05	5, 3	Func (200/404/503); Carga (p95 generar/descargar)

S6	HU-11 DOCX asíncrono (Job+SSE) HU-12A/B/C Anexos; HU-13A/B Metadata/Descarga; HU-14A/B Firmado; HU-15A/B/C Firmado (ver/desc/rees); HU-16A/B/C Invitaciones; HU-19 Croquis	RF-09; NF-02/03/04/06	5	Func (202/200/409/403/404); Concurrencia
S7		RF-10/11/12/13/14/15/18; NF-01/02/03/05	3/3/3, 1/3, 3/3, 1/3/3, 1/3/1, 3	Func (2xx/4xx/5xx); Descargas con S3 Retry
S8	HU-18 Select AI (NL→SQL)	RF-17; NF-01/02/03/05	5	Func (200 [] / 400 / 403 / 503)

Nota. Las estimaciones son referenciales de planificación; los resultados se reportan en la evidencia (colecciones, binarios, métricas). Elaboración propia.

3.6.4 Dependencias clave en la planificación

Se priorizó completar seguridad antes de manipular datos sensibles; persistencia antes de generación de documento; *POI* antes de *altChunk*; anexos/descargas *S3* cuando existían documentos disponibles; y *Select AI* cuando el modelo de datos estuvo maduro. Este orden minimizó retrabajo y facilitó pruebas de carga realistas.

Tabla 23

Dependencias y justificación

Bloque	Depende de	Justificación técnica	Riesgo mitigado
Seguridad (JWT/403)	—	Protección temprana del perímetro	Acceso indebido durante pruebas
Persistencia (JPA)	Seguridad básica	Modelo estable antes de exponer API	Cambios de esquema a destiempo
DOCX (POI/altChunk)	Persistencia	Plantillas requieren datos consistentes	Inconsistencias en placeholders
S3 (descargas/retry)	DOCX disponible	Descarga valida binarios reales	Falsos positivos en pruebas
Job+SSE	DOCX y S3 operativos	Asincronía sobre proceso estable	Bloqueos del hilo principal
Select AI	Modelo de datos estable	SQL generada con esquema final	Consultas inválidas o vacías

Nota. Las dependencias se planificaron para reducir riesgo y encadenar evidencias útiles (binarios, métricas p95, errores manejados). Elaboración propia.

El enfoque *Scrum+RUP* combinó agilidad (entrega iterativa, evidencia por sprint) con gobernanza (artefactos y puertas de calidad). Las tablas de planificación integran notas explicativas y métricas objetivo (DoD, %HU, p95, manejo de errores), respondiendo a los requisitos de trazabilidad, validación estricta y documentación de fallos. Con ello, las secciones 3.4–3.6 quedan alineadas con la arquitectura y la evidencia de pruebas del sistema.

3.7 Desarrollo de la App

Convenciones de presentación. En cada sprint se reporta: Objetivo, Historias, Trabajo realizado, Estándares aplicados (con fragmentos de código o pseudocódigo), Manejo de fallos, Pruebas/Evidencia y Observaciones. La solución sigue arquitectura *Controller–Service–Repository*, *JWT* para seguridad de endpoints, @Valid para entradas y respuestas de error normalizadas (formato problem+json tipo RFC-7807).

3.7.1 Sprint 1 – Inception (visión y arquitectura)

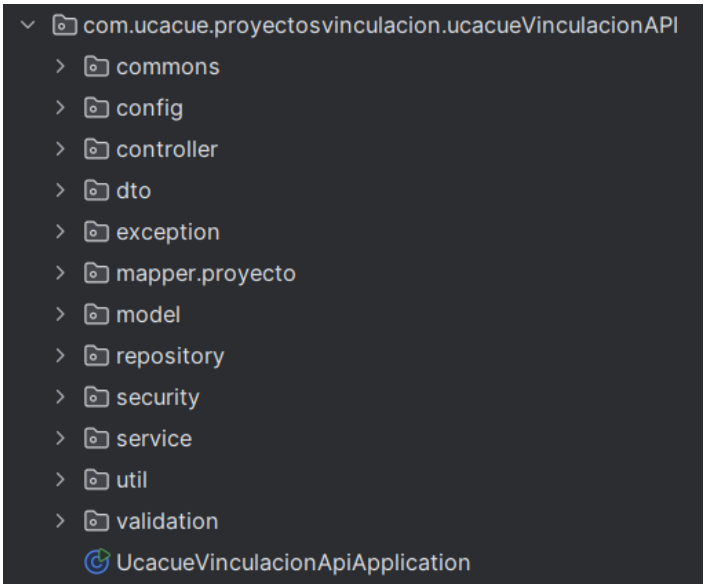
Objetivo. linear la visión y definir la arquitectura base. Se acordó una arquitectura en capas con paquetes lógicos que separan la seguridad, la web, los servicios, los repositorios y la infraestructura.

Historias ejecutadas. Pre-HU (sin endpoints).

Estándares (fragmentos):

Figura 17

Convención de paquetes



Nota. La figura ilustra la separación por capas (Controller, Service y Repository) empleada para garantizar mantenibilidad y escalabilidad del sistema. Elaboración propia.

Decisiones clave:

- **Errores unificados.** El manejador global *GlobalExceptionHandler* construye respuestas *ProblemDetail* con campos status, title y detail. Por ejemplo, una validación fallida retorna 400 con un título "Datos inválidos" y una lista de errores, mientras que un acceso no autorizado genera un 403 con el título "Acceso denegado". La especificación *RFC-7807* recomienda usar estos campos para que los clientes puedan interpretar los errores de forma uniforme.
- **JWT y seguridad.** El proyecto usa *tokens JWT* para autenticar a los usuarios. Un JWT es un mecanismo compacto y seguro para transportar reclamaciones como JSON que puede firmarse o cifrarse. El filtro *JwtTokenFilter* extrae el token del encabezado Authorization, verifica su validez mediante *JwtProvider* y carga las credenciales del usuario.
- **Asincronía y reintentos.** Se planificó soportar operaciones largas como la generación de documentos Word de forma no bloqueante. *Spring* permite habilitar métodos *asíncronos* con la anotación @EnableAsync; si se define un *Executor* personalizado se controla el número de *hilos* y la *cola* de tareas. Para operaciones con fallos temporales, como el acceso a *S3*, se decidió usar @Retryable con reintentos exponenciales; de forma predeterminada *Spring* realiza hasta tres intentos con un segundo de espera.

3.7.2 Sprint 2 – Autenticación y manejo de errores

Objetivo. mplementar registro, inicio de sesión y restablecimiento de contraseña con validación estricta y respuestas de error normalizadas.

Historias. HU-01, HU-02, HU-03.

Trabajo realizado. Se crearon DTO con anotaciones de *Bean Validation*, un controlador de autenticación y un manejador global para traducir excepciones a *RFC-7807*.

Estándares aplicados (fragmentos):

Figura 18

DTO de registro con validación estricta (RegistroDto.java)

```
public record RegistroDto(
  @NotBlank @Email String email,
  @NotBlank @Size(min=8, max=64) String password,
  @NotBlank String nombre
) {}
```

Nota. Los mensajes de error personalizados se definen en messages.properties para que al fallar la validación se devuelva un 400 con detalle de los campos inválidos. Elaboración propia.

Nota. Captura de Postman que muestra una solicitud POST a api/auth/registro con cuerpo JSON mal formado (campo "nombr" en lugar de "nombre"). El backend aplica validación y el Handler global normaliza el error devolviendo 400 Bad Request en formato application/problem+json (Problem Details): type:"about:blank", title:"Solicitud inválida", status:400, detail:"El contenido de la solicitud es inválido o no se pudo procesar", instance:"/api/auth/registro", timestamp:"2025-08-19T23:31:05.067819400-05:00". Latencia ≈ 12 ms, tamaño 661 B. Elaboración propia.

Figura 19

Controlador de autenticación con validación y JWT

```java
@RestController
@RequestMapping("/api/auth")
@Validated
public class AutenticacionController {

    @PostMapping("/registro")
    public ResponseEntity<?> registrar(@Valid @RequestBody RegistroDTO dto) {
        var usuario = authService.registrar(dto);
        return ResponseEntity.status(HttpStatus.CREATED).body(usuario);
    }

    @PostMapping("/login")
    public ResponseEntity<JwtDTO> login(@Valid @RequestBody LoginDTO dto) {
        return ResponseEntity.ok(authService.login(dto));
    }

    // … endpoints de forgot/reset password …
}
```

Nota. El servicio AuthServiceImpl genera el JWT usando JwtProvider y comprueba la contraseña con AuthenticationManager. El JWT contiene el correo como subject y se firma con una clave secreta. El filtro JwtTokenFilter intercepta cada solicitud, extrae el token y, si es válido, carga el UsuarioPrincipal en el contexto de seguridad. Elaboración propia.

Figura 20

Endpoints de registro y login con @Valid (AuthController.java)

```java
@RestController
@RequestMapping("/api/auth")
public class AuthController {

  @PostMapping("/registro")
  public ResponseEntity<?> registrar(@Valid @RequestBody RegistroDto dto) {
    var usuario = authService.registrar(dto);
    return ResponseEntity.status(HttpStatus.CREATED).body(usuario); // 201
  }

  @PostMapping("/login")
  public ResponseEntity<TokenDto> login(@Valid @RequestBody LoginDto dto) {
    return ResponseEntity.ok(authService.login(dto)); // 200 con JWT
  }
}
```

Nota. Aplica @Valid y códigos HTTP claros (201 creado, 200 login). Demuestra criterios de aceptación medibles y DoD en endpoints base. Elaboración propia.

Figura 21

Manejador global de errores con RFC-7807

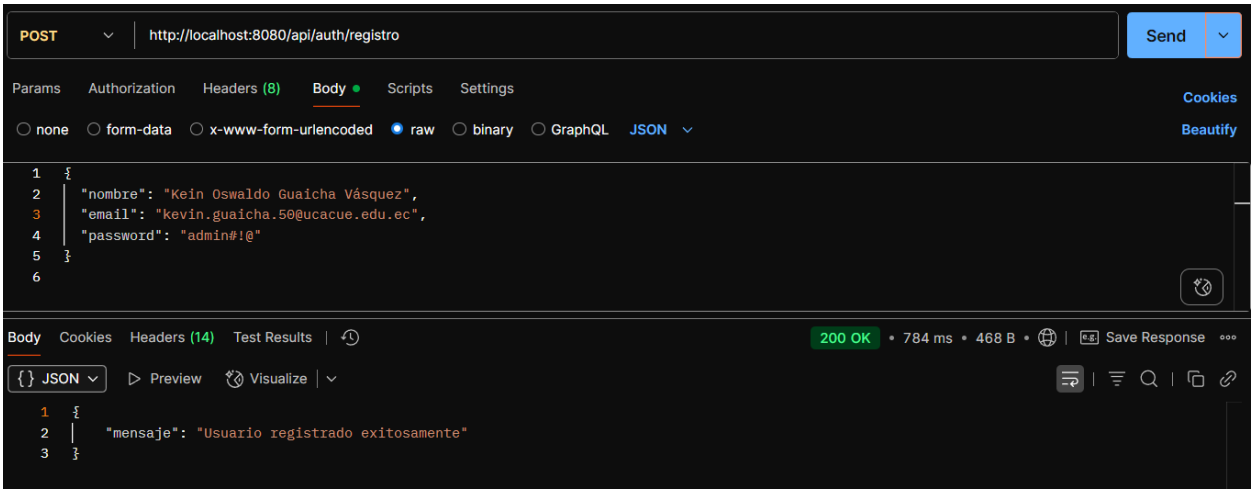
```java
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ProblemDetail> handleBodyValidation(MethodArgumentNotValidException ex,
                                                              HttpServletRequest req) {
        List<String> errors = ex.getBindingResult().getFieldErrors().stream()
                .map(f -> f.getField() + ": " + Optional.ofNullable(f.getDefaultMessage()).orElse("inválido"))
                .toList();
        ProblemDetail pd = ProblemDetail.forStatus(HttpStatus.BAD_REQUEST);
        pd.setTitle("Datos inválidos");
        pd.setDetail("Uno o más campos no cumplen las reglas de validación");
        pd.setProperty("errors", errors);
        // añade timestamp y path
        enrich(pd, req);
        return ResponseEntity.badRequest().body(pd);
    }

    @ExceptionHandler(BadCredentialsException.class)
    public ResponseEntity<ProblemDetail> handleBadCred(BadCredentialsException ex, HttpServletRequest req) {
        ProblemDetail pd = ProblemDetail.forStatus(HttpStatus.UNAUTHORIZED);
        pd.setTitle("Autenticación fallida");
        pd.setDetail("Credenciales inválidas o token no válido");
        enrich(pd, req);
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(pd);
    }
    // … otros manejadores 403/404/409/413/503 …
}
```

Nota. El método enrich inserta un timestamp y la ruta en el cuerpo, proporcionando información contextual al consumidor del API. Elaboración propia.

Pruebas y evidencia:

Figura 22

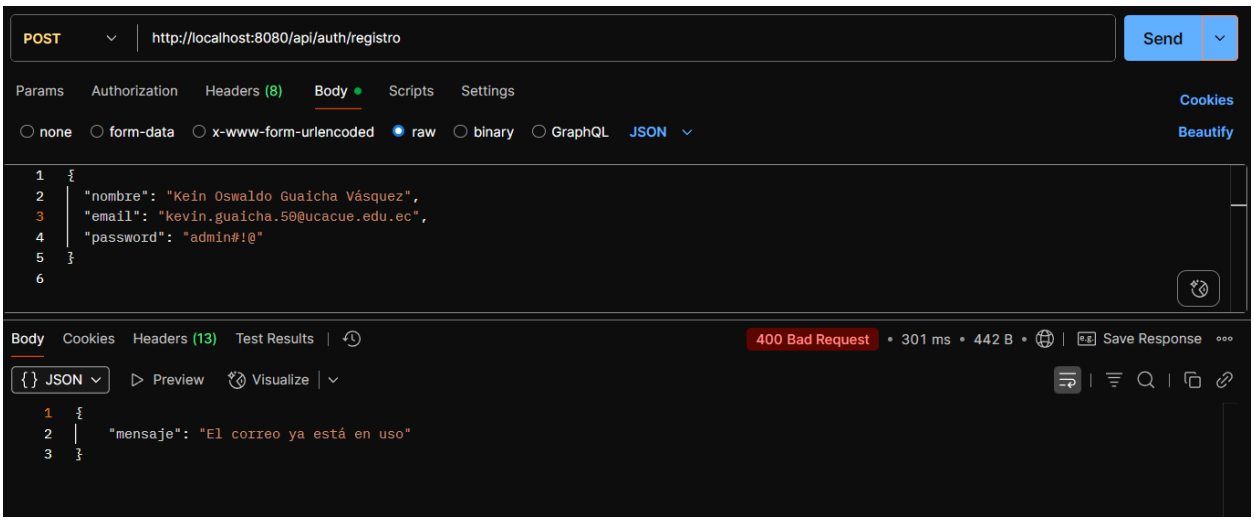
Registro de usuario exitoso en el endpoint /api/auth/registro (Postman)



Nota. Captura de Postman que muestra una solicitud POST /api/auth/registro con cuerpo JSON (campos: nombre, email del dominio @ucacue.edu.ec, password). El servicio responde 200 OK con {"mensaje":"Usuario registrado exitosamente"}; tiempo de respuesta ≈ 784 ms y tamaño 468 B. Entorno local (puerto 8080). Elaboración propia.

Figura 23

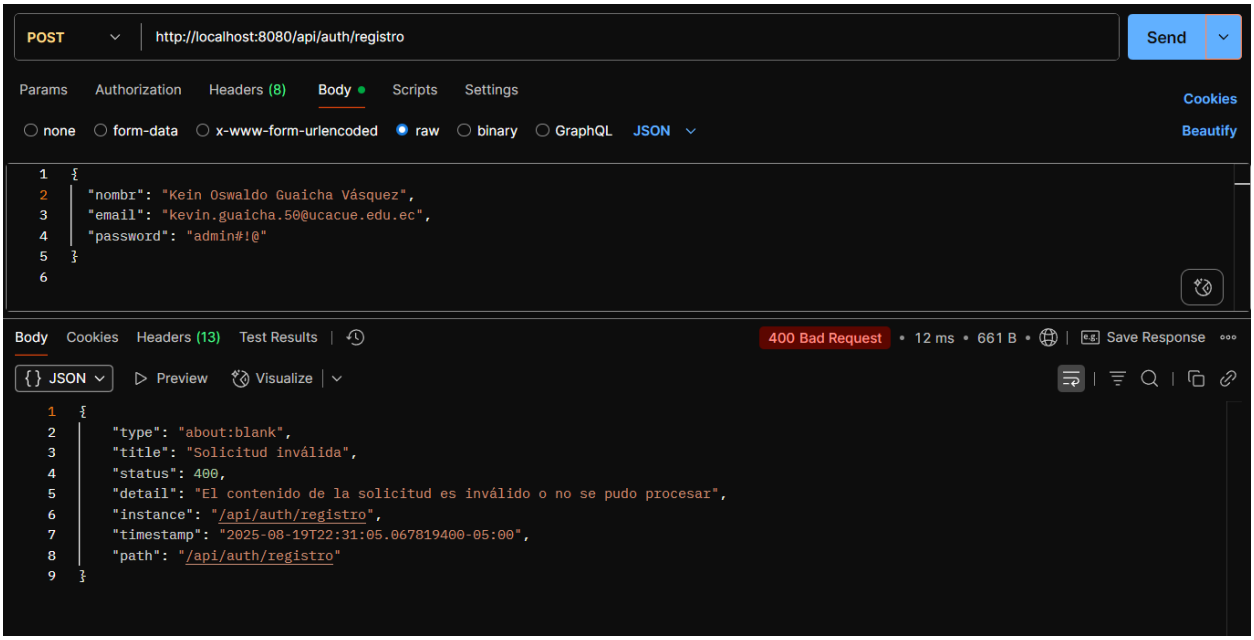
Rechazo de registro por correo duplicado en el endpoint /api/auth/registro (Postman)



Nota. Captura de Postman que evidencia la validación de unicidad de email: ante una solicitud POST a /api/auth/registro con cuerpo JSON (nombre, email @ucacue.edu.ec, password), el servicio responde 400 Bad Request con {"mensaje":"El correo ya está en uso"}; tiempo ≈ 301 ms y tamaño 442 B. Elaboración propia.

Figura 24

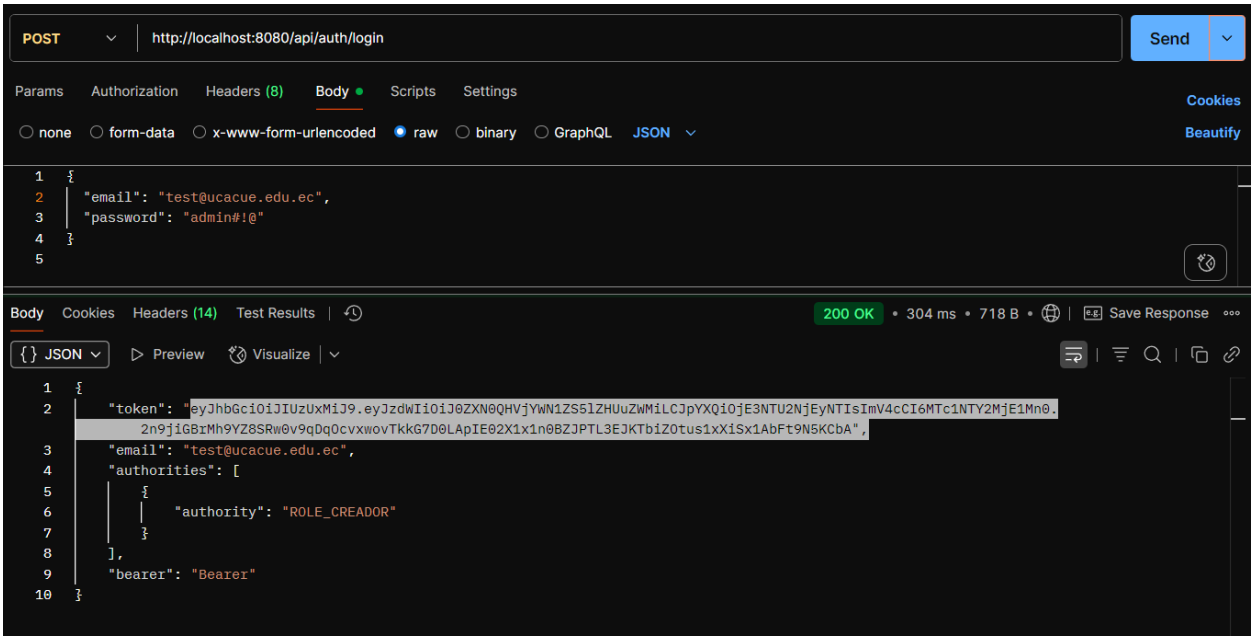
Rechazo por payload inválido y respuesta RFC 7807 del endpoint /api/auth/registro (Postman)



Nota. Captura de Postman que muestra una solicitud POST a /api/auth/registro con cuerpo JSON mal formado (campo "nombr" en lugar de "nombre"). El backend aplica validación y el Handler global normaliza el error devolviendo 400 Bad Request en formato application/problem+json (Problem Details): type:"about:blank", title:"Solicitud inválida", status:400, detail:"El contenido de la solicitud es inválido o no se pudo procesar", instance:"/api/auth/registro", timestamp:"2025-08-19T23:31:05.067819400-05:00". Latencia ≈ 12 ms, tamaño 661 B. Elaboración propia.

Figura 25

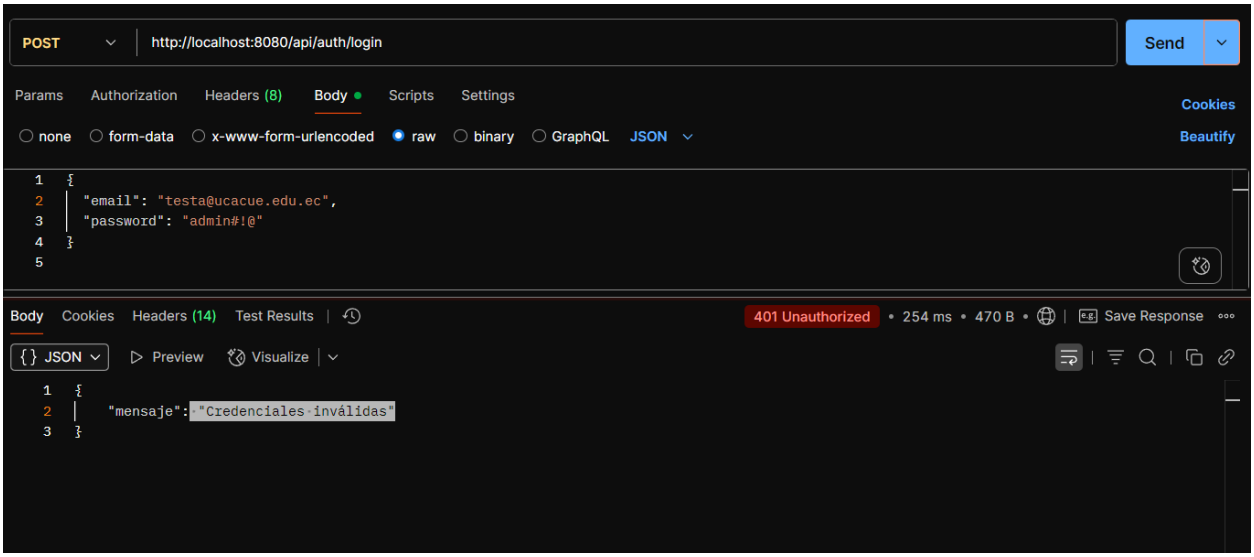
Inicio de sesión exitoso y emisión de JWT en el endpoint /api/auth/login (Postman)



Nota. Captura de Postman que muestra una solicitud POST a /api/auth/login con cuerpo JSON (email, password). El backend responde 200 OK con un token JWT (mostrado parcialmente), el email autenticado, la lista de authorities ["ROLE_CREADOR"] y el esquema bearer "Bearer". Tiempo ≈ 304 ms y tamaño 718 B. Este token se utiliza en llamadas posteriores vía Authorization: Bearer <token>. Elaboración propia.

Figura 26

Rechazo de autenticación por credenciales inválidas en el endpoint /api/auth/login (Postman)



Nota. Captura de Postman que muestra una solicitud POST a http://localhost:8080/api/auth/login con cuerpo JSON (email, password). Al enviar un correo incorrecto (testa@ucacue.edu.ec) el backend responde 401 Unauthorized con {"mensaje":"Credenciales inválidas"}; tiempo ≈ 254 ms y tamaño 470 B. El sistema no expone detalles sensibles, confirmando la política de seguridad ante fallos de autenticación. Elaboración propia.

3.7.3 Sprint 3 – Usuarios y "Mis proyectos" con permisos

Objetivo. Exponer listados en contexto de seguridad y validar acceso por rol/propiedad.

Historias. HU-04, HU-05.

Trabajo realizado. Se añadió un servicio de permisos (PermisoService) que consulta la base de datos para determinar si el usuario autenticado es el dueño de un proyecto, está invitado al documento o es miembro. Los controladores usan expresiones @PreAuthorize de Spring para restringir la ejecución de los métodos. La referencia de Spring Security indica que @PreAuthorize evalúa una expresión *SpEL* y sólo permite invocar el método si ésta se cumple.

Estándares aplicados (fragmentos):

Figura 27

Servicio de permisos con reglas de acceso

```java
@Service("permisoService")
public class PermisoService {
    public boolean esDuenoProyecto(Long usuarioId, Long proyectoId) {
        return datosRepo.existsByIdAndUsuarioId(proyectoId, usuarioId);
    }
    public boolean invitadoAlPdf(Long usuarioId, Long proyectoId) {
        return docRepo.findByDatosProyectoId(proyectoId)
                .map(doc -> invRepo.existsByDocumentoIdAndUsuarioId(doc.getId(), usuarioId))
                .orElse(false);
    }
    public boolean puedeVerDocumentoFirmado(Long usuarioId, Long proyectoId) {
        return esDuenoProyecto(usuarioId, proyectoId) || invitadoAlPdf(usuarioId, proyectoId);
    }
    public boolean esMiembroProyecto(Long usuarioId, Long proyectoId) {
        return esDuenoProyecto(usuarioId, proyectoId) || invitadoAlPdf(usuarioId, proyectoId);
    }
    public boolean esPropietarioAnexo(Long usuarioId, Long anexoId) {
        return anexoRepo.existsByIdAndPropietarioId(anexoId, usuarioId);
    }
    // … otros métodos …
}
```

Nota. Estos métodos se invocan en las expresiones *SpEL* de los controladores, por ejemplo: @PreAuthorize("@permisoService.esDuenoProyecto(principal.id,#proyectoId)"). De esta forma se asegura que sólo el propietario o un miembro puedan ver o modificar los recursos. Elaboración propia.

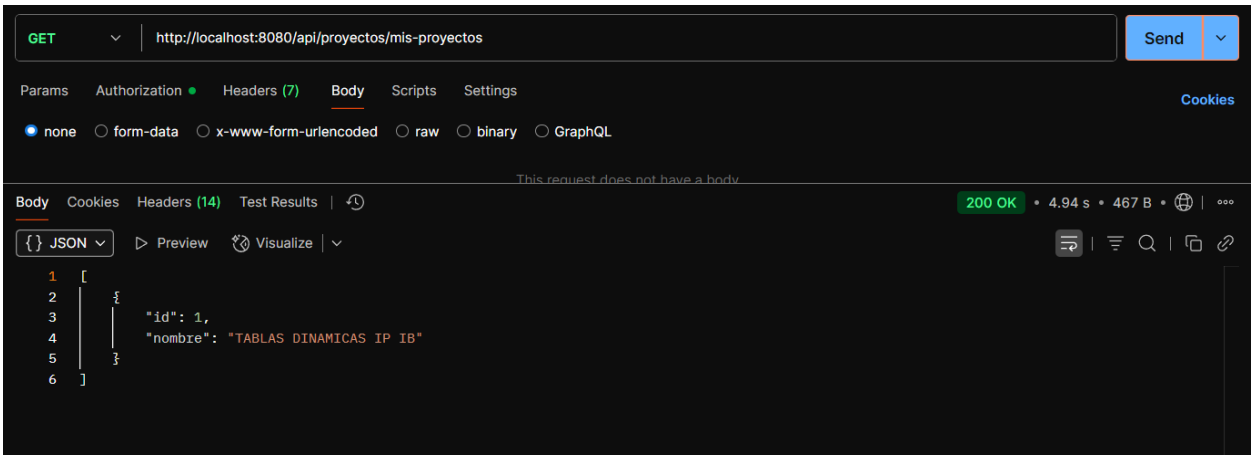
Figura 28

Controlador con validación de parámetros y autorización previa

```java
@Validated
@RestController
@RequestMapping("/api/proyectos")
public class DatosProyectoController {

    @GetMapping("/mis-proyectos")
    @PreAuthorize("@permisoService.esMiembroProyecto(principal.id, #ownerId)")
    public ResponseEntity<List<ProyectoDTO>> listar(
        @RequestParam @Positive Long ownerId,
        @AuthenticationPrincipal UsuarioPrincipal user) {
        return ResponseEntity.ok(service.misProyectos(ownerId));
    }
    // … otros endpoints de creación/actualización …
}
```

Nota. La anotación @Positive valida que el identificador sea un entero positivo (de lo contrario, se devuelve un 400), y el método esMiembroProyecto asegura que el usuario autenticado tenga derecho a acceder al listado. El uso de @PreAuthorize evita siquiera ejecutar la lógica del servicio si la expresión no es verdera. Elaboración propia.

Pruebas y evidencia:



Nota. Prueba de Postman “Mis proyectos” demuestran que un usuario autorizado recibe **200** con sus proyectos. Elaboración propia.

3.7.4 Sprint 4 – Datos de proyecto (crear/actualizar/detalle) + Objetivos

Objetivo del sprint. Permitir crear y modificar la información del proyecto incluyendo la subida de *croquis* y exponer un detalle con la *URL* del *croquis*.

Historias ejecutadas. HU-06, HU-07, HU-08, HU-17.

Trabajo realizado. Se implementaron métodos en *DatosProyectoServiceImpl* para guardar y actualizar los datos de un proyecto. El servicio obtiene el usuario autenticado a partir del contexto de seguridad, mapea el *DTO* a la entidad y, si se adjunta un archivo de croquis, lo guarda a través de *CroquisImagenService*. Se devuelven *DTO* enriquecidos con la *URL* del croquis.

Estándares aplicados (fragmentos):

Figura 29

Guardar datos de proyecto con croquis

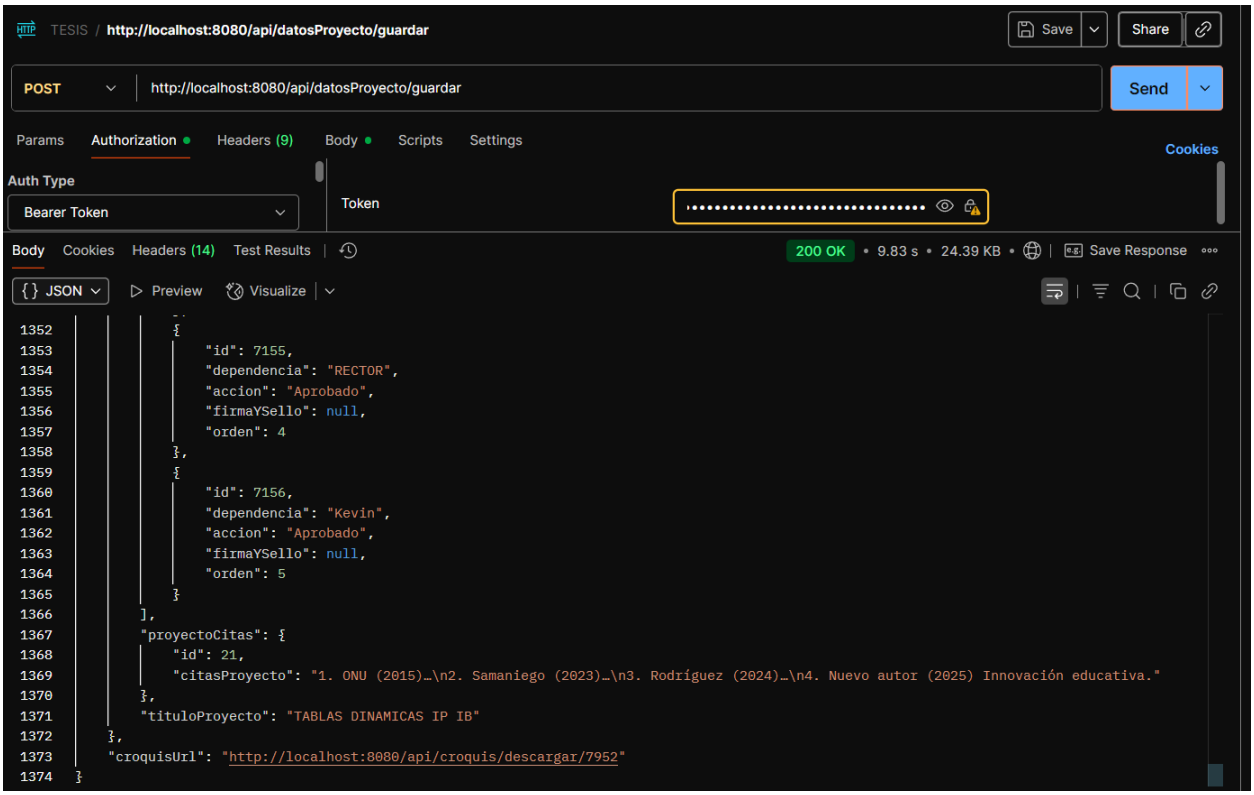
```java
@Service
@Transactional
public class DatosProyectoServiceImpl implements DatosProyectoService {

    @Override
    public DetalleProyectoConCroquisUrlDTO guardar(CrearDatosProyectoDTO dto, MultipartFile archivoCroquis) {
        Usuario usuario = getUsuarioAutenticado()
                .orElseThrow(() -> new ResourceNotFoundException("Usuario autenticado no encontrado"));
        DatosProyecto entidad = datosProyectoMapper.toEntity(dto);
        entidad.setUsuario(usuario);
        DatosProyecto guardado = datosProyectoRepository.save(entidad);
        String croquisUrl = null;
        if (archivoCroquis != null && !archivoCroquis.isEmpty()) {
            byte[] contenido = toBytes(archivoCroquis);
            CroquisImagen ci = croquisImagenService.guardarCroquis(guardado, contenido, archivoCroquis.getContentType());
            // actualizar colección para mantener consistencia en memoria
            guardado.getCroquisImagenes().clear();
            guardado.getCroquisImagenes().add(ci);
            datosProyectoRepository.save(guardado);
            // construir URL de descarga
            croquisUrl = ServletUriComponentsBuilder.fromCurrentContextPath()
                    .path("/api/croquis/descargar/")
                    .path(ci.getId().toString())
                    .build()
                    .toUriString();
        }
        return new DetalleProyectoConCroquisUrlDTO(guardado, croquisUrl);
    }
    // … métodos de actualizar() y detalle() …
}
```

Nota. Cualquier falta de proyecto o usuario devuelve una ResourceNotFoundException que el manejador global traduce a 404; la validación de DTO y de parámetros se realiza mediante anotaciones en el controlador. Para mantener trazabilidad, se actualiza la colección de croquis en memoria antes de persistir. Elaboración propia.

Pruebas y evidencia:

Figura 30

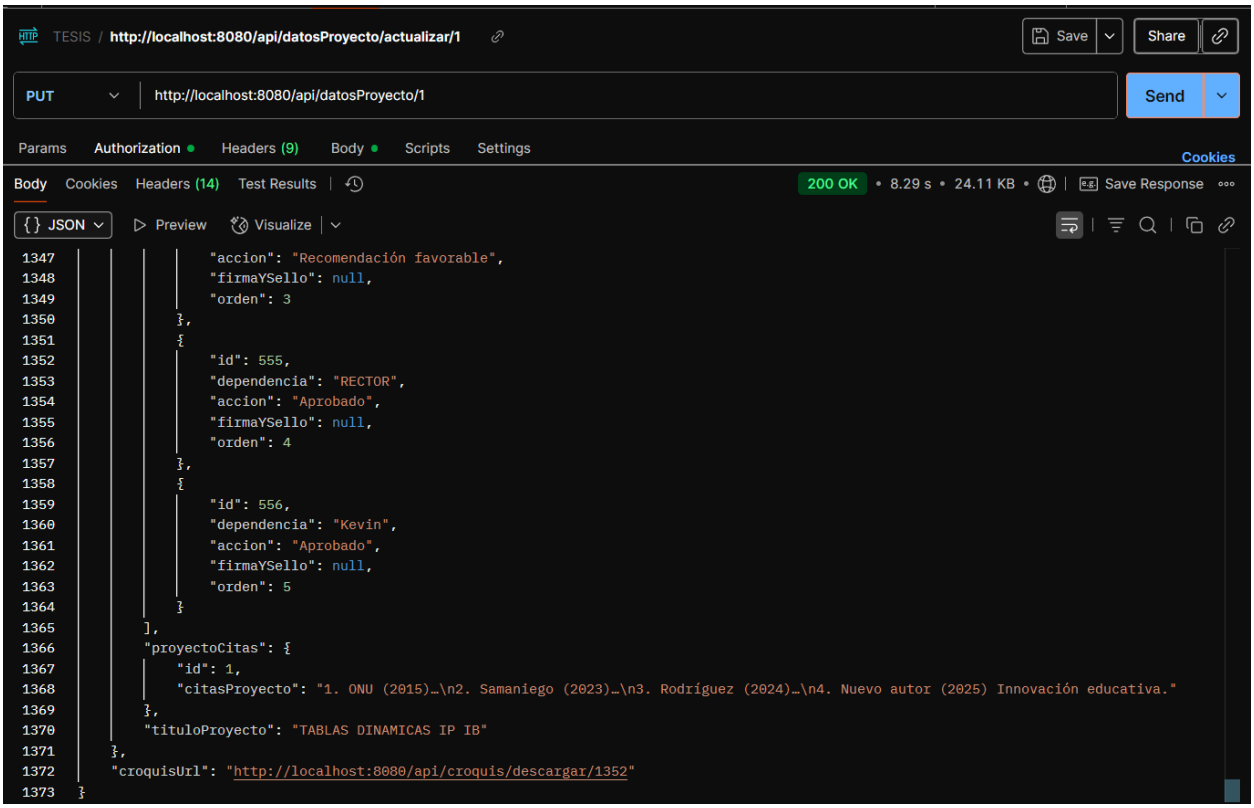
Guardar proyecto (crear/actualizar) - respuesta exitosa y URL de croquis (POST /api/datosProyecto/guardar, Bearer)



Nota. Captura de Postman que evidencia la consulta GET al endpoint /api/proyectos/mis-proyectos, dentro de un contexto autenticado, devolviendo 200 OK con un arreglo JSON de proyectos del usuario. Tiempo ≈ 4.94 s y tamaño 467 B. Elaboración propia.

Figura 31

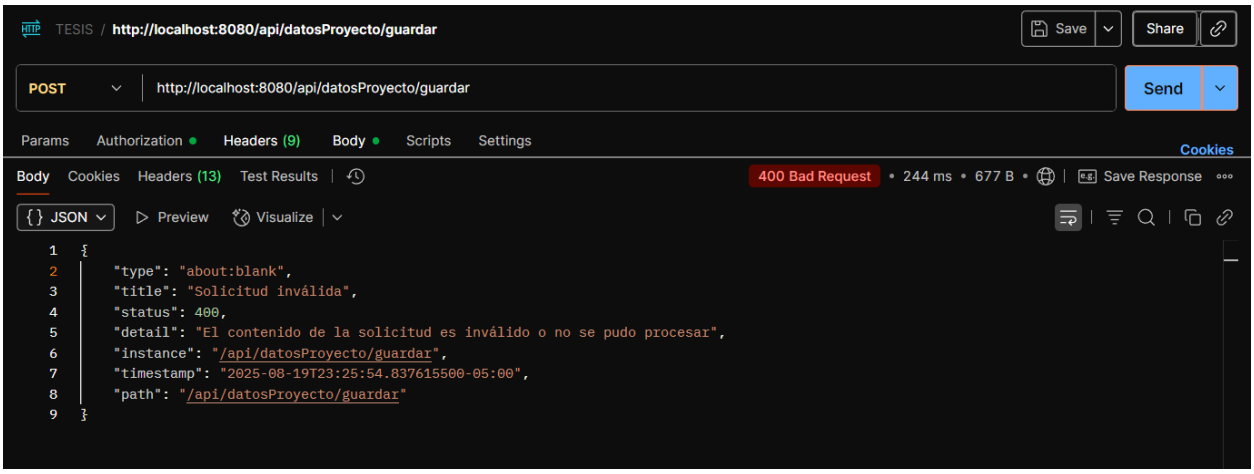
Actualización de proyecto — 200 OK y retorno de croquisUrl (PUT /api/datosProyecto/actualizar/1, Bearer)



Nota. Captura de Postman con autenticación Bearer que ejecuta PUT a /api/datosProyecto/actualizar/1. El backend responde 200 OK (≈ 8.29 s, 24.11 KB) con un JSON que incluye el tituloProyecto "TABLAS DINAMICAS IP IB" y la croquisUrl /api/croquis/descargar/1352. Elaboración propia.

Figura 32

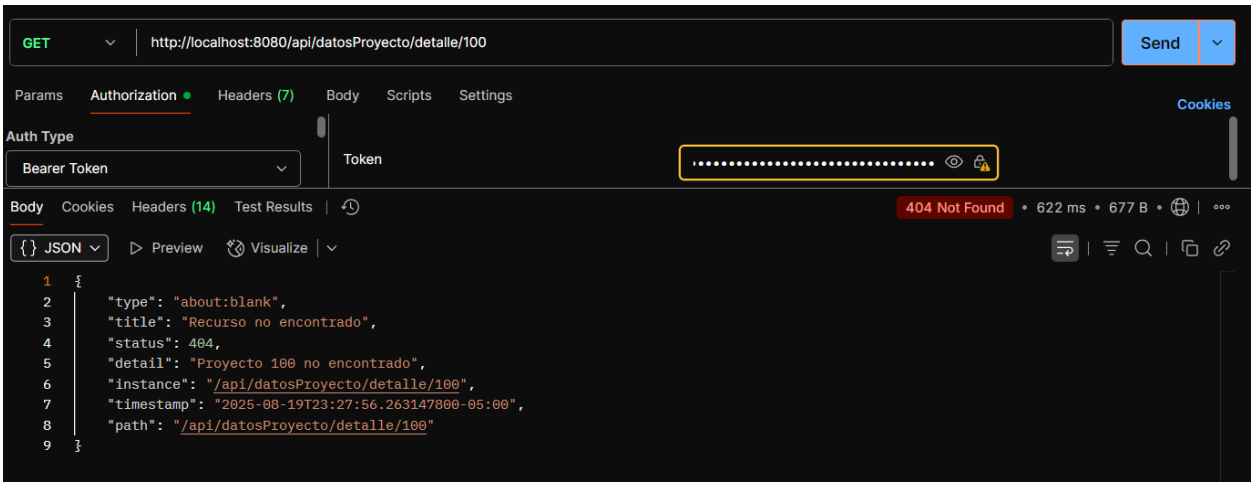
Error de validación en creación/actualización de proyecto - 400 Bad Request con Problem Details (RFC 7807) (POST /api/datosProyecto/guardar)



Nota. Captura de Postman que evidencia la respuesta estandarizada application/problem+json ante un payload inválido o campos requeridos faltantes en el endpoint /api/datosProyecto/guardar. El backend retorna: type:"about:blank", title:"Solicitud inválida", status:400, detail:"El contenido de la solicitud es inválido o no se pudo procesar", instance:"/api/datosProyecto/guardar", path:"/api/datosProyecto/guardar". Latencia ≈ 244 ms; tamaño ≈ 677 B. Elaboración propia.

Figura 33

Detalle de proyecto inexistente 404 Not Found con Problem Details (RFC 7807) (GET /api/datosProyecto/detalle/100, Bearer)



Nota. Captura de Postman que muestra una solicitud GET autenticada con Bearer Token a /api/datosProyecto/detalle/100. El backend responde 404 Not Found en application/problem+json: type:"about:blank", title:"Recurso no encontrado", status:404, detail:"Proyecto 100 no encontrado", instance:"/api/datosProyecto/detalle/100", timestamp:"2025-08-19T23:27:56.263147800-05:00", path:"/api/datosProyecto/detalle/100". Latencia ≈ 622 ms; tamaño ≈ 677 B. Elaboración propia.

3.7.5 Sprint 5 – Generación y descarga de DOCX sincrónico con reintentos S3

Objetivo del sprint. Generar un archivo Word del proyecto de forma sincrónica, almacenarlo en S3 y permitir su descarga con resiliencia ante fallos de red. Automatizar la inserción de secciones repetibles directamente en Word sin romper el formato institucional.

Historias ejecutadas. HU-09, HU-10.

Trabajo realizado. El controlador *DocumentoWordController* expone un *endpoint POST* para generar el documento y otro *GET* para descargarlo. La clase *DocumentoWordService* orquesta la generación mediante *GeneradorDocumentoProyectoService*, persiste la *metadata* en la base de

datos, sube el *binario* a S3 *usando AwsS3Service* y valida que el usuario sea propietario o administrador.

Estándares aplicados (fragmentos):

Figura 34

Servicio de generación y descarga con permiso y S3

```java
public class DocumentoWordService {
    @Transactional
    public void generarYGuardarDocumentoWord(Long proyectoId) {
        assertPermisoSobreProyecto(proyectoId);
        DatosProyecto proyecto = proyectoRepo.findById(proyectoId)
                .orElseThrow(() -> new DocumentoNoEncontradoException(proyectoId));
        generarYGuardarDocumentoWord(proyecto);
    }
    public DocumentoWordProyecto guardarDocumentoWord(Long proyectoId, byte[] docGenerado, String nombreArchivo) {
        DatosProyecto proyecto = proyectoRepo.findById(proyectoId)
                .orElseThrow(() -> new RuntimeException("No existe proyecto con ID: " + proyectoId));
        DocumentoWordProyecto docWord = wordRepo.findByDatosProyectoId(proyectoId).orElseGet(DocumentoWordProyecto::new);
        docWord.setDatosProyecto(proyecto);
        docWord.setNombreArchivo(nombreArchivo);
        docWord.setTipoContenido("application/vnd.openxmlformats-officedocument.wordprocessingml.document");
        String key = wordFolder + "/Proyecto_" + proyectoId + ".docx";
        awsS3Service.uploadFile(key, docGenerado, docWord.getTipoContenido());
        docWord.setS3Key(key);
        return wordRepo.save(docWord);
    }
    public ResponseEntity<byte[]> descargarResponseEntity(Long proyectoId) {
        assertPermisoSobreProyecto(proyectoId);
        DocumentoWordProyecto meta = obtenerDocumentoWord(proyectoId);
        byte[] contenido = descargarDocumentoWordBinario(proyectoId);
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.parseMediaType(meta.getTipoContenido()));
        headers.setContentDisposition(ContentDisposition.attachment().filename(meta.getNombreArchivo()).build());
        return ResponseEntity.ok().headers(headers).body(contenido);
    }
    private void assertPermisoSobreProyecto(Long proyectoId) {
        var usuario = usuarioService.autenticado();
        if (usuario == null) throw new AccessDeniedException("No autenticado.");
        var proyecto = proyectoRepo.findById(proyectoId)
                .orElseThrow(() -> new DocumentoNoEncontradoException(proyectoId));
        boolean esPropietario = proyecto.getUsuario() != null && proyecto.getUsuario().getId().equals(usuario.getId());
        boolean esAdmin = usuario.getRol() != null && usuario.getRol().name().equals("ADMIN");
        if (!esPropietario && !esAdmin) throw new AccessDeniedException("No tienes permisos para descargar este documento.");
    }
}
```

Nota. El método assertPermisoSobreProyecto evita que un usuario no autorizado genere o descargue documentos de otros proyectos. La subida y descarga se delegan a AwsS3Service, que usa @Retryable para reintentar en caso de fallos temporales de S3. Si al descargar S3 devuelve 404, el servicio lanza S3ObjetoNoEncontradoException que se mapea a 404 sin reintento; tras agotar reintentos se lanza ExternalServiceException que el manejador global traduce a 503. Elaboración propia.

Figura 35

Servicio S3 con reintentos y recuperación

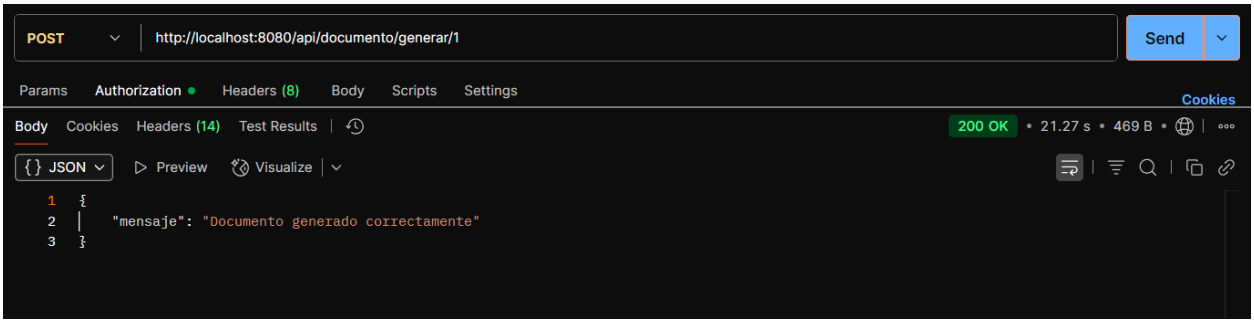
```java
@Service
public class AwsS3Service {
    @Retryable(
        include = {S3Exception.class, SdkClientException.class},
        exclude = {S3ObjetoNoEncontradoException.class},
        maxAttemptsExpression = "#{@s3Props.springRetry.maxAttempts}",
        backoff = @Backoff(
            delayExpression = "#{@s3Props.springRetry.initialDelayMs}",
            multiplierExpression = "#{@s3Props.springRetry.multiplier}",
            maxDelayExpression = "#{@s3Props.springRetry.maxDelayMs}",
            random = true
        )
    )
    public byte[] downloadFile(String key) {
        try {
            GetObjectRequest req = GetObjectRequest.builder().bucket(props.getBucket()).key(key).build();
            ResponseBytes<GetObjectResponse> bytes = s3.getObjectAsBytes(req);
            return bytes.asByteArray();
        } catch (S3Exception e) {
            if (e.statusCode() == 404) {
                throw new S3ObjetoNoEncontradoException("Archivo no encontrado en S3: " + key);
            }
            throw e;
        }
    }
    @Recover
    public byte[] recoverDownload(Exception ex, String key) {
        throw new ExternalServiceException("No se pudo descargar desde S3 tras reintentos", ex);
    }
    // métodos uploadFile() y deleteFile() con reintentos similares …
}
```

Nota. Spring Retry repetirá la operación hasta alcanzar maxAttempts; la política de backoff se configura en application.yml a través de s3Props.springRetry. Según la documentación de Spring, por defecto se reintentan tres veces con un retraso inicial de un segundo, pero aquí se establece un retraso inicial de 400 ms, un multiplicador de 2 y un retraso máximo. Elaboración propia.

Pruebas y evidencia:

Figura 36

Generación síncrona de documento - 200 OK (POST /api/documento/generar/1, Bearer)



Nota. Solicitud POST a /api/documento/generar/1 que responde 200 OK con {"mensaje":"Documento generado correctamente"}. Latencia ≈ 21.27 s; tamaño ≈ 469 B. Elaboración propia.

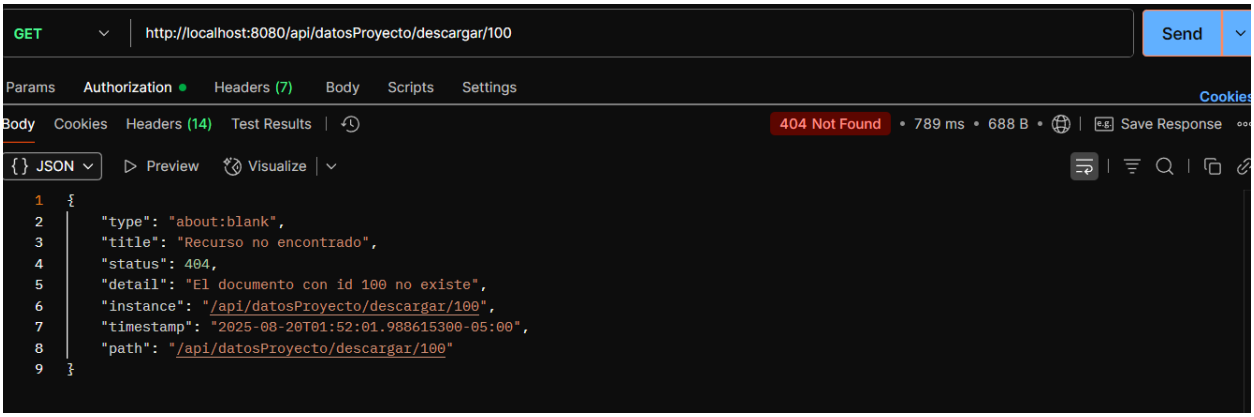
Figura 37

Descarga directa del .docx generado - 200 OK (GET /api/datosProyecto/descargar/1, Bearer)

Nota. Solicitud GET autenticada que devuelve el archivo Office Open XML (firma ZIP “PK” visible como binario). Tiempo ≈ 3.49 s; tamaño ≈ 707.48 KB. Se sugiere Save Response / Send and Download para guardar el documento. Elaboración propia.

Figura 38

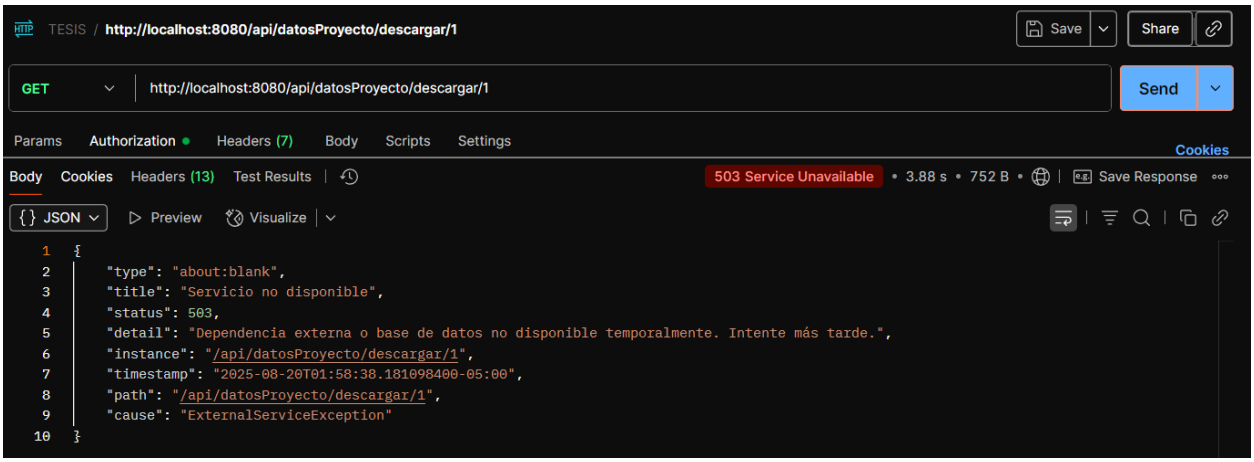
Documento no encontrado - 404 Not Found con Problem Details (RFC 7807) (GET /api/datosProyecto/descargar/100, Bearer)



Nota. El backend estandariza el error en application/problem+json: type:"about:blank", title:"Recurso no encontrado", status:404, detail:"El documento con id 100 no existe", instance:"/api/datosProyecto/descargar/100". Latencia ≈ 789 ms; tamaño ≈ 688 B. Elaboración propia.

Figura 39

Interrupción temporal del servicio - 503 Service Unavailable con Problem Details (RFC 7807) (GET /api/datosProyecto/descargar/1, Bearer)



Nota. Captura de Postman que evidencia el manejo de fallos transitorios al descargar un documento. La solicitud GET a /api/datosProyecto/descargar/1 devuelve 503 Service Unavailable en application/problem+json: type:"about:blank", title:"Servicio no disponible", status:503, detail:"Dependencia externa o base de datos no disponible temporalmente. Intente más tarde.", instance:"/api/datosProyecto/descargar/1", timestamp:"2025-08-20T01:58:38.181098400-05:00", path:"/api/datosProyecto/descargar/1", cause:"ExternalServiceException". Latencia ≈ 3.88 s; tamaño ≈ 752 B. La respuesta no expone detalles sensibles y está preparada para estrategias de reintento/backoff del cliente. Elaboración propia.

3.7.6 Sprint 6 – Generación de DOCX asíncrono con Job y SSE

Objetivo del sprint. Evitar que el hilo *HTTP* se bloquee durante la creación del Word. Se implementó un modelo de *trabajos* (Jobs) que permite al usuario lanzar la generación de un documento, consultar su estado mediante *polling* o eventos *SSE* (Server-Sent Events) y descargarlo una vez terminado.

Historias ejecutadas. HU-11.

Trabajo realizado. Se definió *DocJobStore* para almacenar trabajos en memoria, *DocumentoWordAsyncService* para crear trabajos y *DocumentoWordAsyncWorker* para procesarlos en un hilo separado. El controlador *DocumentoWordAsyncController* expone *endpoints REST* para crear trabajos, consultar el estado y descargar el resultado. El controlador *DocumentoWordSseController* ofrece un *endpoint /events* que envía actualizaciones de progreso vía *SSE*.

Estándares aplicados (fragmentos):

Figura 40

Creación del job y consulta del estado

```
@RestController
@RequestMapping("/api/documento/jobs")
public class DocumentoWordAsyncController {
    @PreAuthorize("@permisoService.esDuenoProyecto(principal.id,#proyectoId)")
    @PostMapping
    public ResponseEntity<ApiResponseDTO<DocJobDTO>> crearJob(
            @RequestParam @Positive Long proyectoId,
            @AuthenticationPrincipal UsuarioPrincipal user) {
        DocJob job = asyncService.start(proyectoId, user.getId());
        DocJobDTO dto = toDto(job);
        return ResponseEntity.accepted()
                .location(URI.create("/api/documento/jobs/" + job.getId()))
                .body(ApiResponseDTO.ok("Job aceptado", dto, HttpStatus.ACCEPTED));
    }
    @PreAuthorize("isAuthenticated()")
    @GetMapping("/{jobId}")
    public ResponseEntity<ApiResponseDTO<DocJobDTO>> estado(
            @PathVariable String jobId,
            @AuthenticationPrincipal UsuarioPrincipal user) {
        DocJob job = jobs.get(jobId);
        if (job == null) throw new ResourceNotFoundException("Job no encontrado");
        if (!job.getUserId().equals(user.getId())) throw new AccessDeniedException("No autorizado para este job");
        return ResponseEntity.ok(ApiResponseDTO.ok("OK", toDto(job), HttpStatus.OK));
    }
    @PreAuthorize("isAuthenticated()")
    @GetMapping("/{jobId}/download")
    public ResponseEntity<byte[]> descargar(@PathVariable String jobId,
                                            @AuthenticationPrincipal UsuarioPrincipal user) {
        DocJob job = jobs.get(jobId);
        if (job == null) throw new ResourceNotFoundException("Job no encontrado");
        if (!job.getUserId().equals(user.getId())) throw new AccessDeniedException("No autorizado para este job");
        if (job.getStatus() != JobStatus.DONE) throw new ConflictException("El documento aún no está listo");
        return documentoWord.descargarResponseEntity(job.getProyectoId());
    }
    // helper toDto() …
}
```

Nota. Al crear un job se devuelve 202 Accepted con la ubicación del recurso; consultar el estado devuelve 200 con información del job (status, percent, message) y la URL de descarga una vez

finalizado. Si se intenta descargar antes de que el job esté listo, se responde con 409 Conflict. Elaboración propia.

Figura 41

Worker asíncrono y envío de progreso por SSE

```java
@Component
public class DocumentoWordAsyncWorker {
    @Async("docxExecutor")
    @Transactional
    public void generateAsync(String jobId, Long proyectoId, Long userId) {
        jobs.update(jobId, j -> { j.setStatus(JobStatus.RUNNING); j.setMessage("Iniciado"); });
        sse.send(jobId, userId, new DocxProgress("RUNNING", 5, "Preparando…", OffsetDateTime.now()));
        try {
            sse.send(jobId, userId, new DocxProgress("RUNNING", 25, "Recopilando datos…", OffsetDateTime.now()));
            docService.generarYGuardarDocumentoWord(proyectoId);
            sse.send(jobId, userId, new DocxProgress("RUNNING", 90, "Subiendo a S3…", OffsetDateTime.now()));
            jobs.update(jobId, j -> {
                j.setStatus(JobStatus.DONE);
                j.setMessage("Documento generado correctamente");
                j.setFileName("proyecto-" + proyectoId + ".docx");
            });
            sse.send(jobId, userId, new DocxProgress("DONE", 100, "Listo", OffsetDateTime.now()));
            sse.complete(jobId, userId);
        } catch (Exception e) {
            jobs.update(jobId, j -> { j.setStatus(JobStatus.ERROR); j.setMessage(e.getMessage()); });
            sse.send(jobId, userId, new DocxProgress("ERROR", 100, e.getMessage(), OffsetDateTime.now()));
            sse.complete(jobId, userId);
        }
    }
}
```

Nota. El método está anotado con @Async("docxExecutor") para ejecutarse en un pool de hilos configurado en AsyncDocxConfig. Spring crea un ThreadPoolTaskExecutor con dos hilos base, cuatro máximos y una cola de 50 tareas. La documentación de Spring señala que si se define un bean Executor y se habilita @EnableAsync, los métodos anotados se ejecutan en hilos separados. El progreso se envía a los clientes suscritos mediante DocxSseService y SSE, permitiendo construir una barra de progreso en tiempo real. Elaboración propia.

Figura 42

Configuración del executor asíncrono

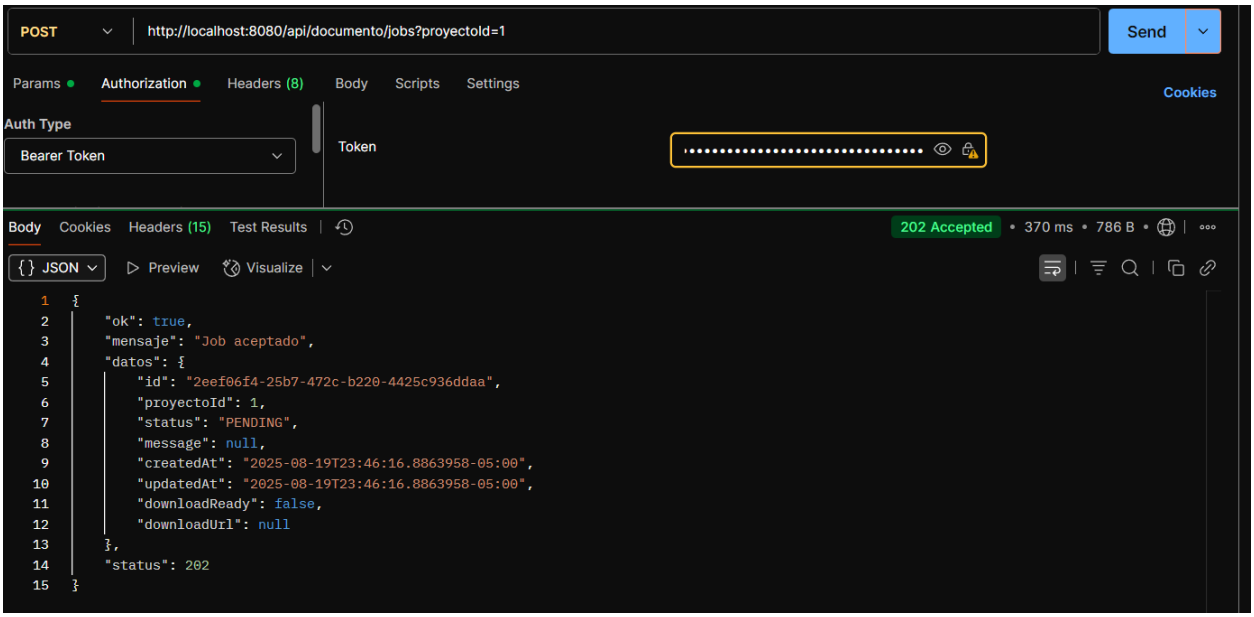
```java
@Configuration
@EnableAsync
public class AsyncDocxConfig {
    @Bean(name = "docxExecutor")
    public AsyncTaskExecutor docxExecutor(TaskDecorator securityContextTaskDecorator,
                                          TaskDecorator mdcTaskDecorator) {
        ThreadPoolTaskExecutor ex = new ThreadPoolTaskExecutor();
        ex.setThreadNamePrefix("docx-");
        ex.setCorePoolSize(2);
        ex.setMaxPoolSize(4);
        ex.setQueueCapacity(50);
        ex.setWaitForTasksToCompleteOnShutdown(true);
        ex.setTaskDecorator(compose(securityContextTaskDecorator, mdcTaskDecorator));
        ex.initialize();
        return ex;
    }
    // … decoradores para propagar SecurityContext y MDC …
}
```

Nota. Se combinan dos decoradores: uno copia el SecurityContext del hilo HTTP al hilo del executor y la otra copia el MDC (contexto de logging) para mantener la trazabilidad. Esto garantiza que las llamadas asíncronas respeten la seguridad y que los logs conserven el traceId del request. Elaboración propia.

Pruebas y evidencia:

Figura 43

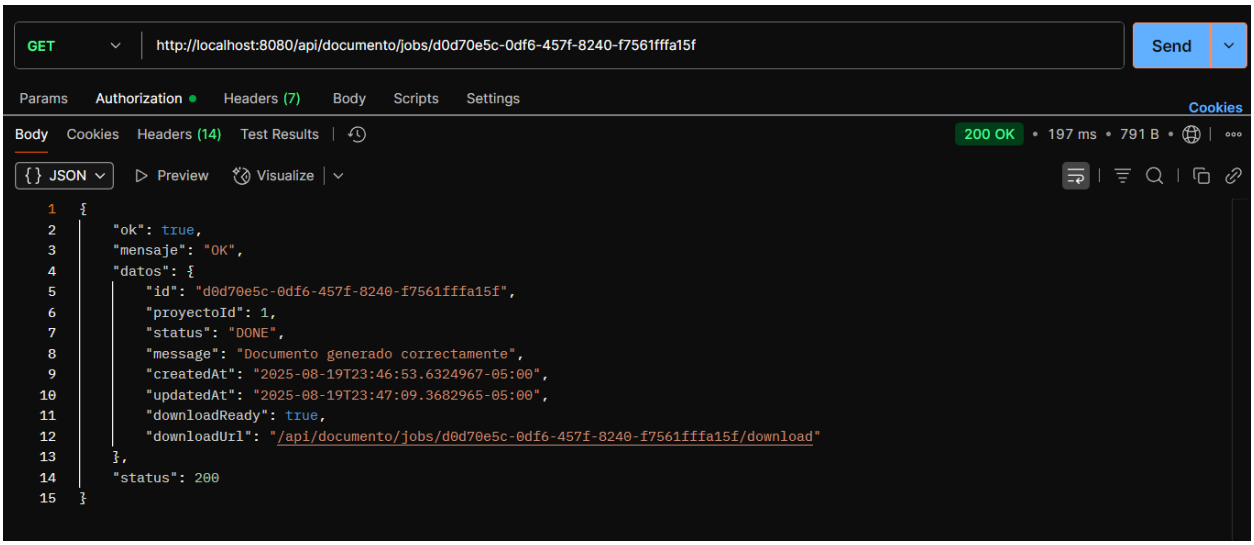
Encolado asíncrono de generación de documento - 202 Accepted (POST /api/documento/jobs?proyectoId=1, Bearer)



Nota. Captura de Postman que evidencia el patrón async: al invocar POST a http://localhost:8080/api/documento/jobs?proyectoId=1, el backend acepta el trabajo y responde 202 Accepted con ok:true, mensaje:"Job aceptado" y un objeto datos con id (UUID del job), proyectoId:1, status:"PENDING", downloadReady:false y downloadUrl:null. Latencia ≈ 370 ms, tamaño ≈ 786 B. Elaboración propia.

Figura 44

Consulta de estado hasta DONE y entrega de downloadUrl — 200 OK (GET /api/documento/jobs/{jobId})



Nota. Polling de estado mediante GET a http://localhost:8080/api/documento/jobs/{jobId}. La respuesta muestra ok:true, mensaje:"OK", status:"DONE", message:"Documento generado correctamente.", downloadReady:true y la downloadUrl /api/documento/jobs/d0d70e5c-…/download. Latencia ≈ 197 ms, tamaño ≈ 791 B. Este flujo confirma la no-bloqueante generación de documentos y la recuperación segura vía Bearer. Elaboración propia.

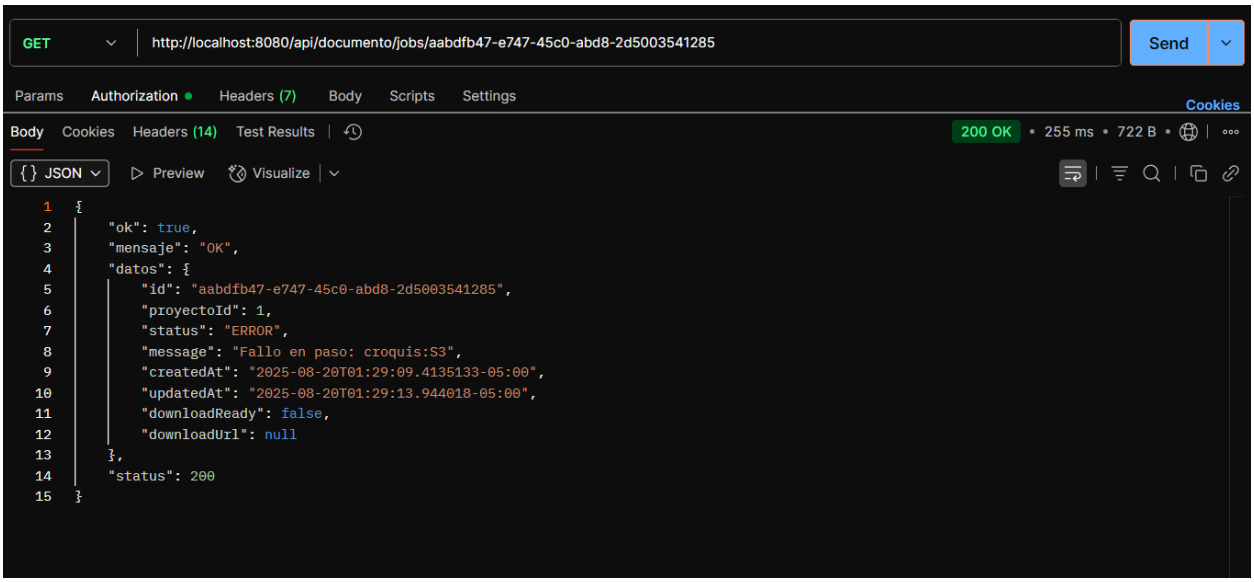
Figura 45

Descarga del documento generado - 200 OK (GET /api/documento/jobs/{jobId}/download, Bearer)

Nota. Captura de Postman que muestra la descarga del archivo .docx generado por el job: solicitud GET a http://localhost:8080/api/documento/jobs/{jobId}/download con autenticación Bearer. El backend responde 200 OK y cuerpo binario (firma ZIP “PK”, propio de Office Open XML), visible como caracteres no legibles en el visor. Tiempo ≈ 3.47 s; tamaño ≈ 707.48 KB. Para guardar el archivo se usa Save Response o Send and Download. Elaboración propia.

Figura 46

Polling de job con fallo de generación - estado ERROR (GET /api/documento/jobs/{jobId}, Bearer)



Nota. Consulta de estado vía GET a http://localhost:8080/api/documento/jobs/{jobId}. El servicio responde 200 OK (recurso consultado) con ok:true, mensaje:"OK", y en datos se reporta status:"ERROR", message:"Fallo en paso: croquis: S3", downloadReady:false y downloadUrl:null (sin enlace de descarga). Tiempos de auditoría createdAt/updatedAt visibles. Latencia ≈ 255 ms; tamaño ≈ 722 B. Elaboración propia.

3.7.7 Sprint 7 – Documentación: anexos, firmado e invitaciones

Objetivo del sprint. Gestionar documentos anexos (subir, actualizar, listar y eliminar), *PDF* firmado e invitaciones a firmar, con control de permisos y descargas resilientes.

Historias ejecutadas: HU-12A/B/C, HU-13A/B, HU-14A/B, HU-15A/B/C, HU-16A/B/C, HU-19.

Trabajo realizado. Se creó *DocumentacionController* que agrupa operaciones sobre anexos y documentos firmados bajo la ruta /api/proyectos/{proyectoId}. Se reutilizó

PermisoService para validar quién puede realizar cada acción (propietario, invitado o miembro). La subida de archivos usa *MultipartFile* y verifica el tamaño máximo; la descarga se respalda con reintentos de *S3*. Los invitados a firmar se gestionan mediante *endpoints* que agregan o eliminan usuarios. Se implementó la subida del documento firmado (único archivo especial marcado como tal) y la carga múltiple de *anexos* (varios archivos por proyecto, categorizables). Se creó el *CRUD* de *anexos* con validación por usuario participante. *Endpoints* para listar anexos, descargar archivos, eliminar autorizados y recuperar todos los datos del proyecto en una sola llamada (útil para verificación o reconstrucción del documento). Los archivos se almacenan en *S3* con metadatos (tipo, usuario, timestamp).

Estándares aplicados (fragmentos):

Figura 47

Subida de anexos y límites de tamaño

```
@PreAuthorize("@permisoService.esDuenoProyecto(principal.id,#proyectoId)")
@PostMapping(value = "/anexos", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public ResponseEntity<ApiResponseDTO<List<AnexoDTO>>> subirAnexos(
        @PathVariable @Positive Long proyectoId,
        @RequestPart("archivos") @NotNull List<MultipartFile> archivos,
        @AuthenticationPrincipal UsuarioPrincipal user) throws Exception {
    List<AnexoDTO> lista = service.subirAnexos(proyectoId, archivos, user.getId());
    return ResponseEntity.status(HttpStatus.CREATED)
            .body(ApiResponseDTO.ok("Anexos subidos", lista, HttpStatus.CREATED));
}

@PutMapping(value = "/anexos/{anexoId}", consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
@PreAuthorize("@permisoService.esPropietarioAnexo(principal.id,#anexoId)")
public ResponseEntity<ApiResponseDTO<AnexoDTO>> actualizarAnexo(
        @PathVariable @Positive Long anexoId,
        @RequestPart("archivo") @NotNull MultipartFile archivo,
        @AuthenticationPrincipal UsuarioPrincipal user) throws Exception {
    return ResponseEntity.ok(ApiResponseDTO.ok("Anexo actualizado",
            service.actualizarAnexo(anexoId, archivo, user.getId()), HttpStatus.OK));
}
```

Nota. La validación @NotNull garantiza que se envíe al menos un archivo; @Positive valida los identificadores y @PreAuthorize impide que un usuario no dueño cargue o actualice anexos. Si el

archivo supera el límite establecido en spring.servlet.multipart.max-file-size (10 MB), Spring lanza MaxUploadSizeExceededException, mapeada a 413 Payload Too Large en el manejador global. Los archivos se guardan en S3 mediante AwsS3Service con reintentos, por lo que las descargas son robustas. Elaboración propia.

Figura 48

Invitaciones a firmar documento

```
@PreAuthorize("@permisoService.esDuenoProyecto(principal.id,#proyectoId)")
@PostMapping("/documento-firmado/invitaciones/{invitadoId}")
public ResponseEntity<ApiResponseDTO<Void>> invitar(
        @PathVariable @Positive Long proyectoId,
        @PathVariable @Positive Long invitadoId,
        @AuthenticationPrincipal UsuarioPrincipal user) {
    service.invitarUsuario(proyectoId, invitadoId, user.getId());
    return ResponseEntity.ok(ApiResponseDTO.ok("Invitado agregado", null, HttpStatus.OK));
}

@DeleteMapping("/documento-firmado/invitaciones/{invitadoId}")
@PreAuthorize("@permisoService.esDuenoProyecto(principal.id,#proyectoId)")
public ResponseEntity<ApiResponseDTO<Void>> quitarInvitado(
        @PathVariable @Positive Long proyectoId,
        @PathVariable @Positive Long invitadoId,
        @AuthenticationPrincipal UsuarioPrincipal user) {
    service.eliminarInvitacion(proyectoId, invitadoId, user.getId());
    return ResponseEntity.ok(ApiResponseDTO.ok("Invitado eliminado", null, HttpStatus.OK));
}
```

Nota. El propietario del proyecto puede invitar o quitar invitados para firmar el PDF. PermisoService comprueba la propiedad del proyecto, y el manejador global devuelve 404 si el proyecto o el usuario invitado no existen. El listado de invitados y las descargas del PDF firmado usan reintentos S3 de manera similar al caso DOCX. Elaboración propia.

Pruebas y evidencia:

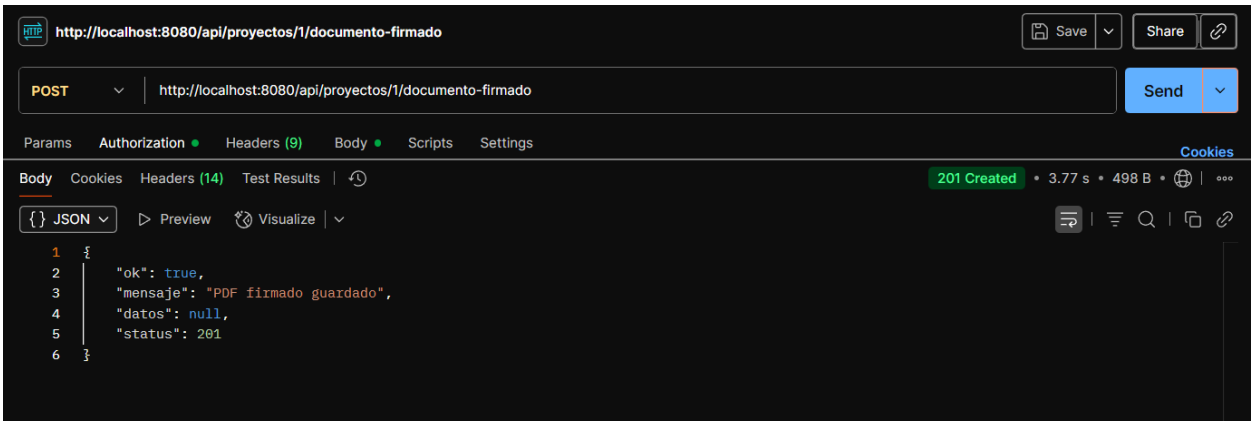
Figura 49

Carga de PDF firmado – interrupción temporal (503 Service Unavailable) en /api/proyectos/1/documento-firmado (POST, Bearer).

Nota. El backend devuelve application/problem+json ante una dependencia externa no disponible: title:"Servicio no disponible", status:503, detail:"Dependencia externa o base de datos no disponible temporalmente. Intente más tarde.", cause:"ExternalServiceException". Latencia ≈ 4.14 s, tamaño ≈ 760 B. Elaboración propia.

Figura 50

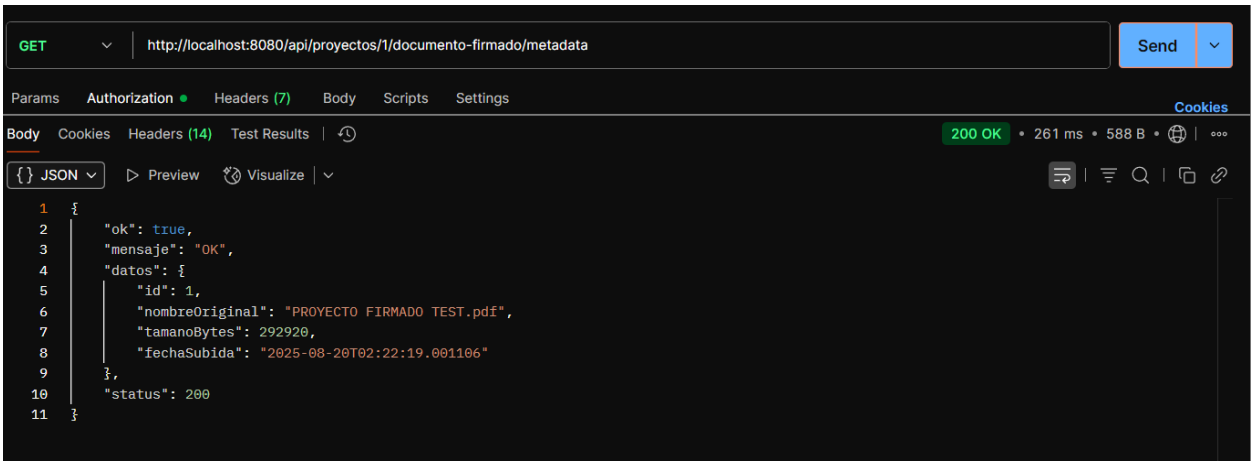
Carga de PDF firmado exitosa - 201 Created (POST /api/proyectos/1/documento-firmado, Bearer)



Nota. El servicio confirma la creación con {"ok":true,"mensaje":"PDF firmado guardado","status":201}. Latencia ≈ 3.77 s, tamaño ≈ 498 B. Elaboración propia.

Figura 51

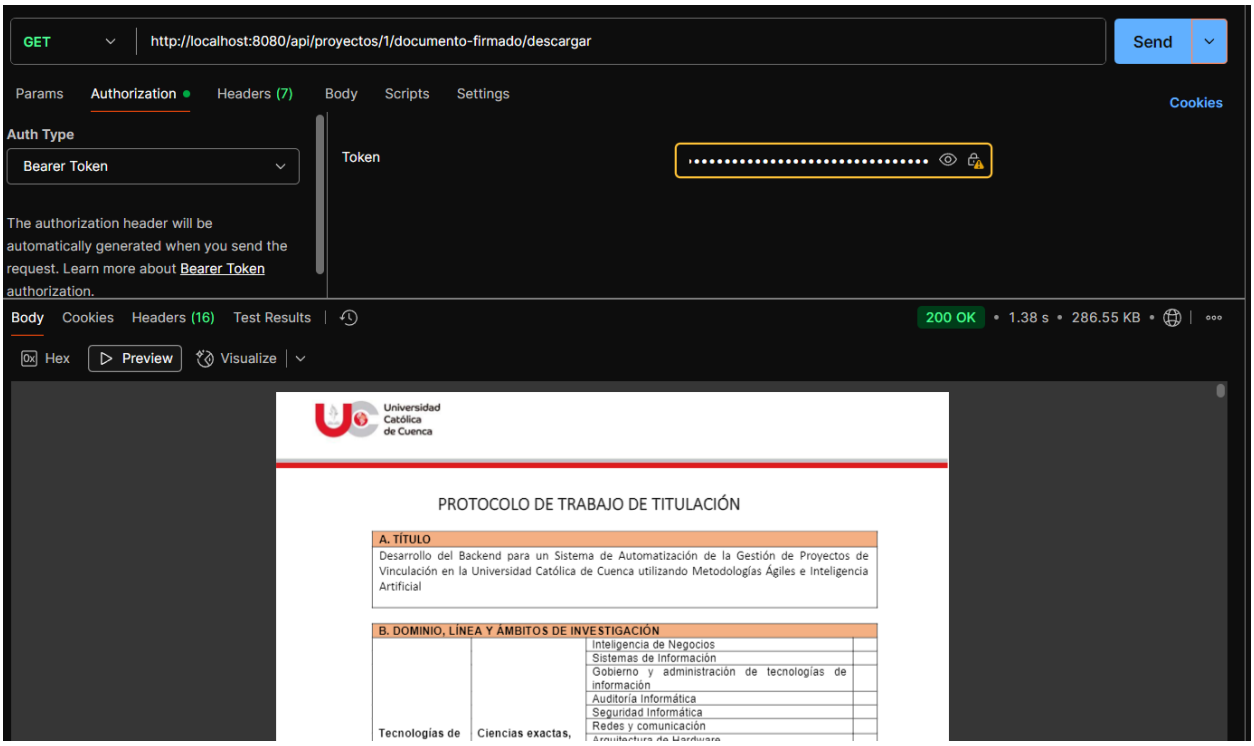
Consulta de metadatos del PDF firmado - 200 OK (GET /api/proyectos/1/documento-firmado/metadata, Bearer)



Nota. Respuesta con ok:true y datos (id:1, nombreOriginal:"PROYECTO FIRMADO TEST.pdf", tamanoBytes:292920, fechaSubida:"2025-08-20T02:22:19.001106"). Latencia ≈ 261 ms, tamaño ≈ 588 B. Elaboración propia.

Figura 52

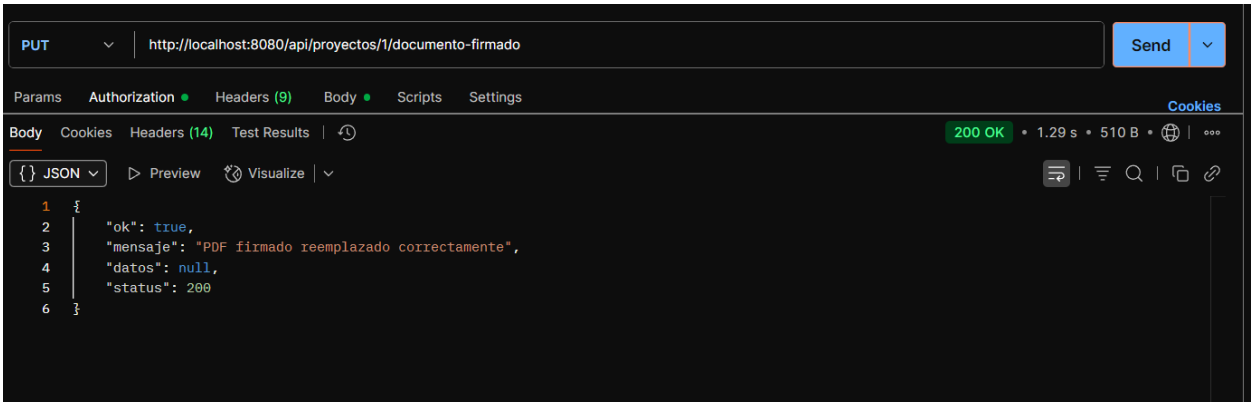
Descarga y previsualización del PDF firmado - 200 OK (GET /api/proyectos/1/documento-firmado/descargar, Bearer)



Nota. La respuesta entrega el binario PDF y Postman lo muestra en Preview. Latencia ≈ 1.38 s, tamaño ≈ 286.55 KB. Elaboración propia.

Figura 53

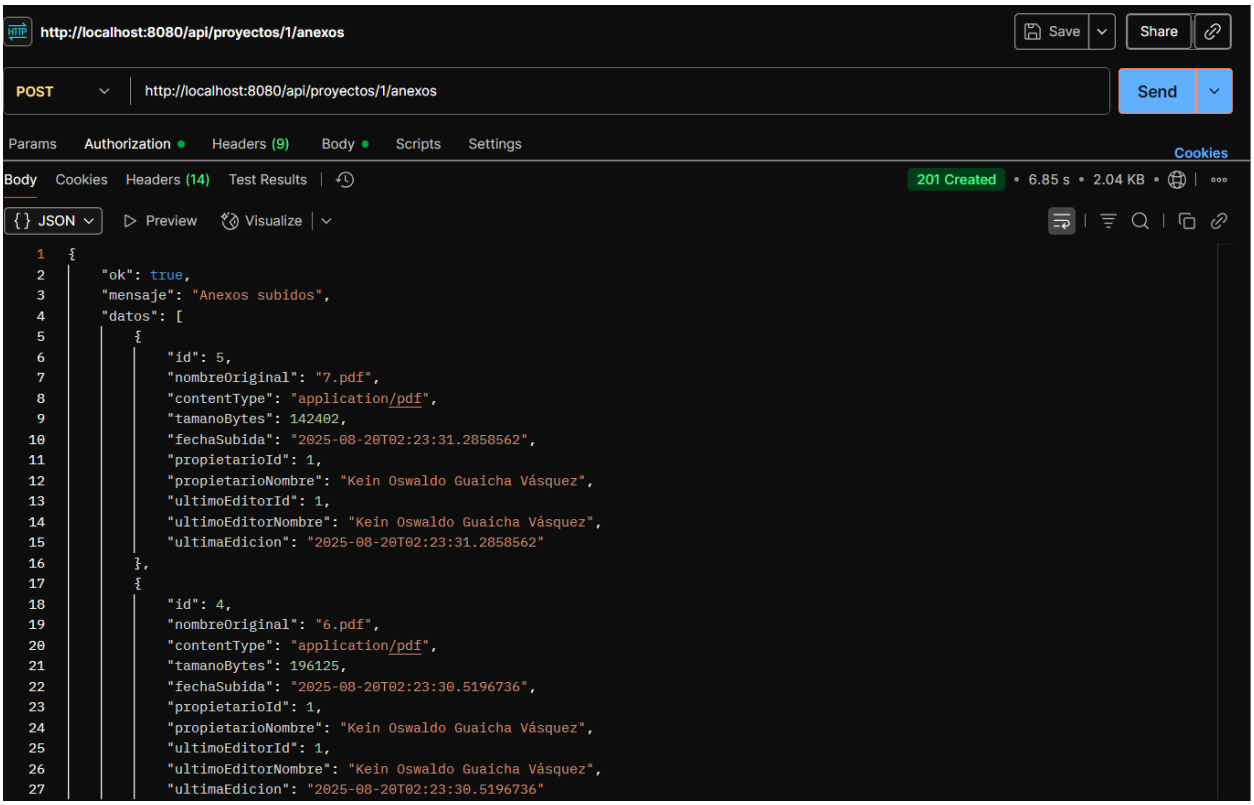
Reemplazo del PDF firmado - 200 OK (PUT /api/proyectos/1/documento-firmado, Bearer)



Nota. Respuesta de éxito: {"ok":true,"mensaje":"PDF firmado reemplazado correctamente","status":200}. Latencia ≈ 1.29 s, tamaño ≈ 510 B. Elaboración propia.

Figura 54

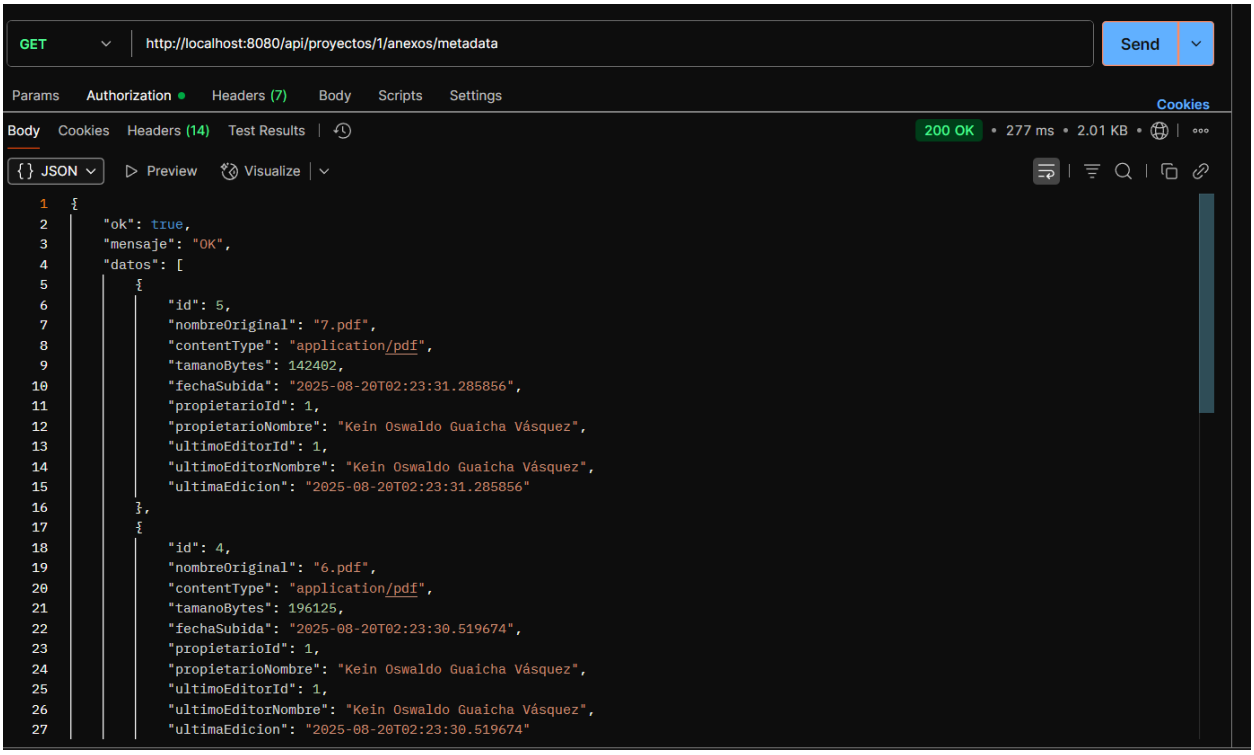
Carga múltiple de anexos — 201 Created (POST /api/proyectos/1/anexos, Bearer)



Nota. Se registran anexos con metadatos por archivo (id, nombreOriginal, contentType:"application/pdf", tamanoBytes, fechaSubida, propietario y último editor). Latencia ≈ 6.85 s, tamaño ≈ 2.04 KB. Elaboración propia.

Figura 55

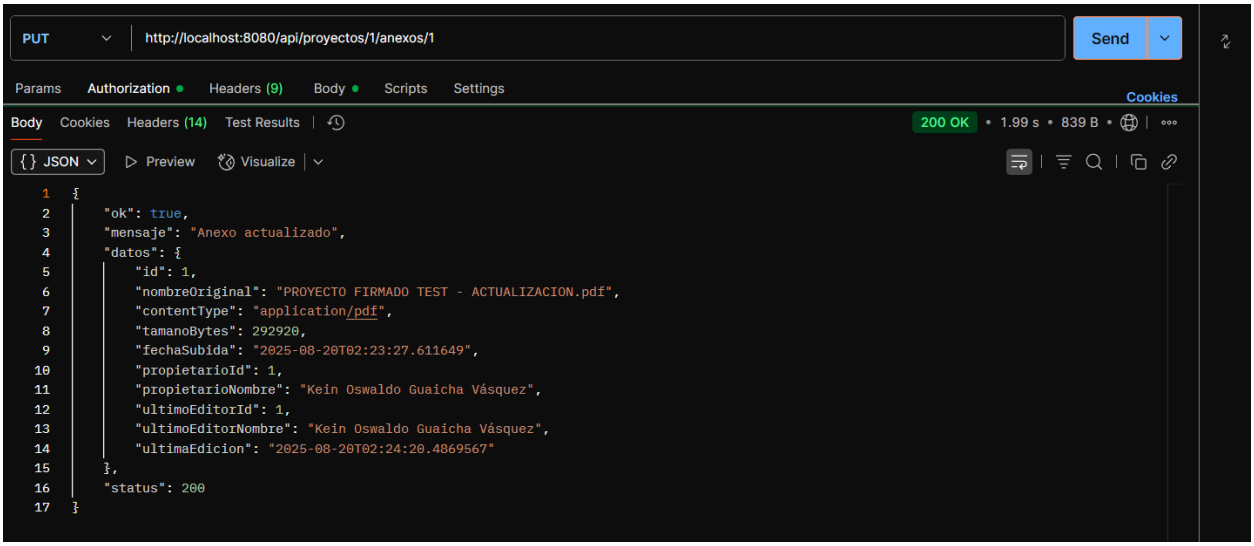
Listado de metadatos de anexos - 200 OK (GET /api/proyectos/1/anexos/metadata, Bearer)



Nota. Devuelve un arreglo con metadatos de cada anexo (ids 5 y 4, nombres originales, tamaños y marcas de tiempo), útil para auditoría y trazabilidad. Latencia ≈ 277 ms, tamaño ≈ 2.01 KB. Elaboración propia.

Figura 56

Actualización de anexo - 200 OK (PUT /api/proyectos/1/anexos/1, Bearer)



Nota. Respuesta con ok:true, mensaje:"Anexo actualizado" y metadatos actualizados (nombreOriginal:"PROYECTO FIRMADO TEST - ACTUALIZACION.pdf", tamanoBytes:292920, ultimaEdicion:...). Latencia ≈ 1.99 s, tamaño ≈ 839 B. Elaboración propia.

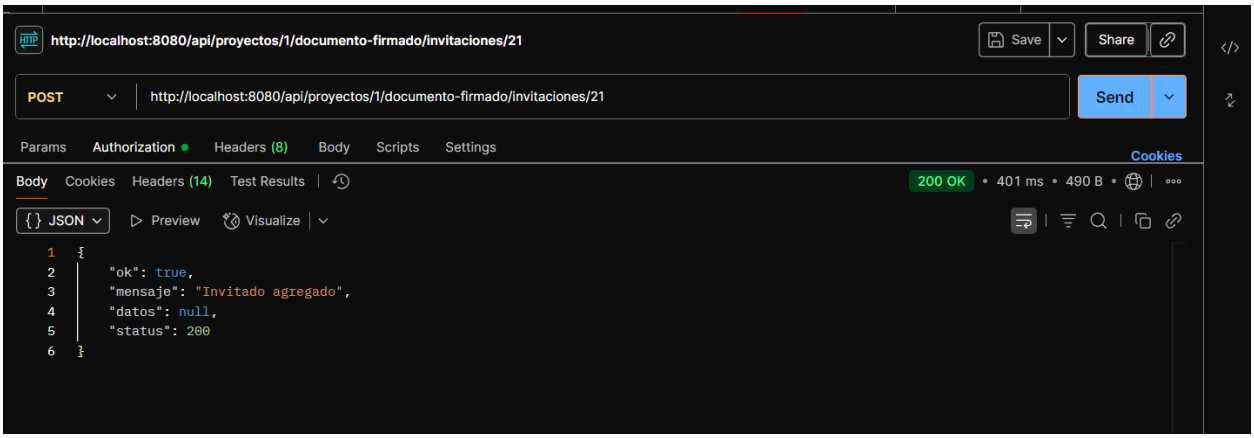
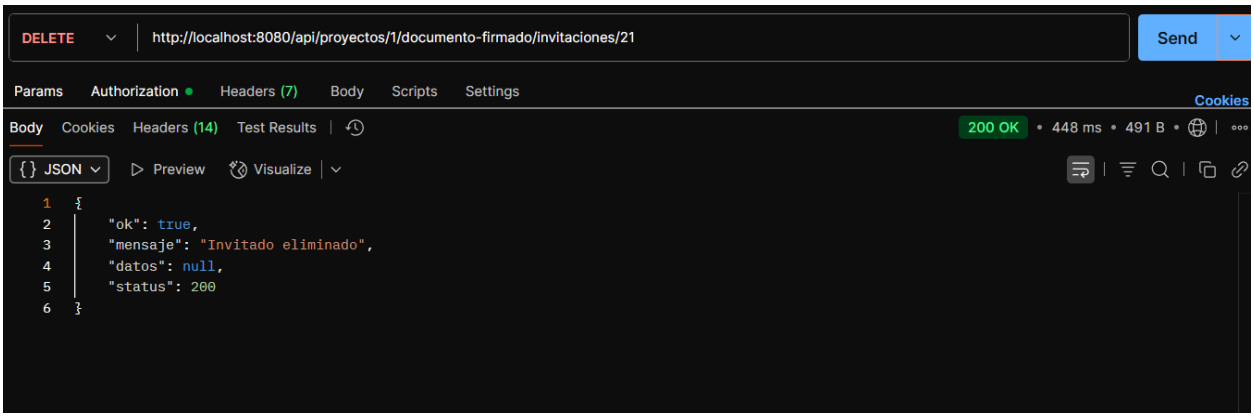


Figura 57

Gestión de invitados para el documento firmado - alta de invitado (200 OK) (POST /api/proyectos/1/documento-firmado/invitaciones/21, Bearer)



Nota. El sistema confirma {"ok":true,"mensaje":"Invitado agregado","status":200}. Latencia ≈ 401 ms, tamaño ≈ 490 B. Elaboración propia.

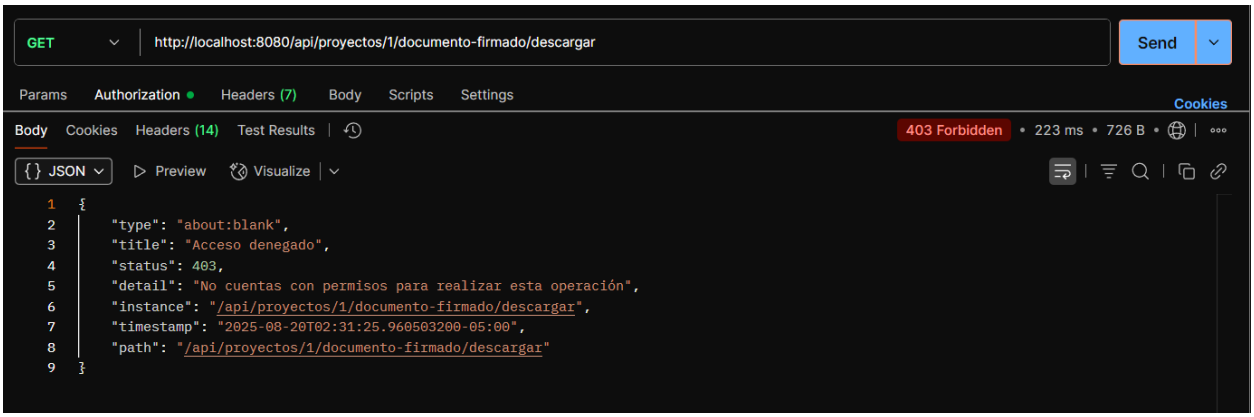
Figura 58

Gestión de invitados - baja de invitado (200 OK) (DELETE /api/proyectos/1/documento-firmado/invitaciones/21, Bearer)

Nota. Respuesta: {"ok":true,"mensaje":"Invitado eliminado","status":200}. Latencia ≈ 448 ms, tamaño ≈ 491 B. Elaboración propia.

Figura 59

Descarga de PDF firmado sin permisos - 403 Forbidden con Problem Details (RFC 7807) (GET /api/proyectos/1/documento-firmado/descargar, Bearer)



Nota. Captura de Postman que evidencia la autorización por roles/permisos: ante la solicitud GET a http://localhost:8080/api/proyectos/1/documento-firmado/descargar, el backend rechaza el acceso con 403 Forbidden y respuesta application/problem+json: type:"about:blank", title:"Acceso denegado", status:403, detail:"No cuentas con permisos para realizar esta operación", instance:"/api/proyectos/1/documento-firmado/descargar", "path:"/api/proyectos/1/documento-firmado/descargar". Latencia ≈ 223 ms; tamaño ≈ 726 B. La respuesta no expone datos sensibles y estandariza errores según RFC 7807. Elaboración propia.

3.7.8 Sprint 8 – Consulta Select AI (NL→SQL) y despliegue

Objetivo del sprint: Permitir a los usuarios consultar información de su proyecto mediante lenguaje natural usando *Oracle Select AI*, con validación de parámetros, permisos y política de "sin resultados".

Historias ejecutadas: HU-18.

Trabajo realizado: Se implementó *AiQueryController* con el endpoint /api/ai/ask. Este endpoint requiere que el usuario sea miembro del proyecto (dueño o invitado), valida que el

parámetro q no sea blanco y tenga máximo 500 caracteres y valida el parámetro opcional mode (runsql|showsql|narrate) usando una expresión regular. Dependiendo del modo, se ejecuta o se muestra la consulta *SQL* generada por el modelo de *AI* y se crearon vistas controladas en la base para limitar los campos que *Select AI* puede consultar. Se desarrolló una *API* que recibe una pregunta en lenguaje natural, la redirige a *Select AI* y retorna la respuesta estructurada (monto, fechas, etc.). Se generó *Dockerfile* y se desplegó en *Render*.

Estándares aplicados (fragmentos):

```java
@Validated
@RestController
@RequestMapping("/api/ai")
public class AiQueryController {
    @PreAuthorize("@permisoService.esMiembroProyecto(principal.id, #projectId)")
    @GetMapping("/ask")
    public ResponseEntity<ApiResponseDTO<AiResult>> ask(
            @RequestParam @Positive long projectId,
            @RequestParam("q") @NotBlank @Size(max = 500) String question,
            @RequestParam(name = "mode", required = false)
            @Pattern(regexp = "(?i)runsql|showsql|narrate", message = "mode debe ser runsql|showsql|narrate")
            String mode) {
        AiAction action = AiAction.from(mode);
        AiResult result = ai.ask(projectId, question, action);
        return ResponseEntity.ok(ApiResponseDTO.ok("OK", result, HttpStatus.OK));
    }
}
```

Nota. @Positive garantiza que el projectId sea válido; @NotBlank y @Size limitan la longitud de la pregunta; @Pattern valida el modo. Si no hay resultados, AiQueryService devuelve un arreglo vacío que se responde con 200 OK; si el proveedor AI está caído, se lanza ExternalServiceException y el manejador global devuelve 503. La política de permisos se centraliza en PermisoService.esMiembroProyecto. Elaboración propia.

Figura 60

Creación de perfil de Select AI con DBMS_CLOUD_AI.CREATE_PROFILE (Oracle)

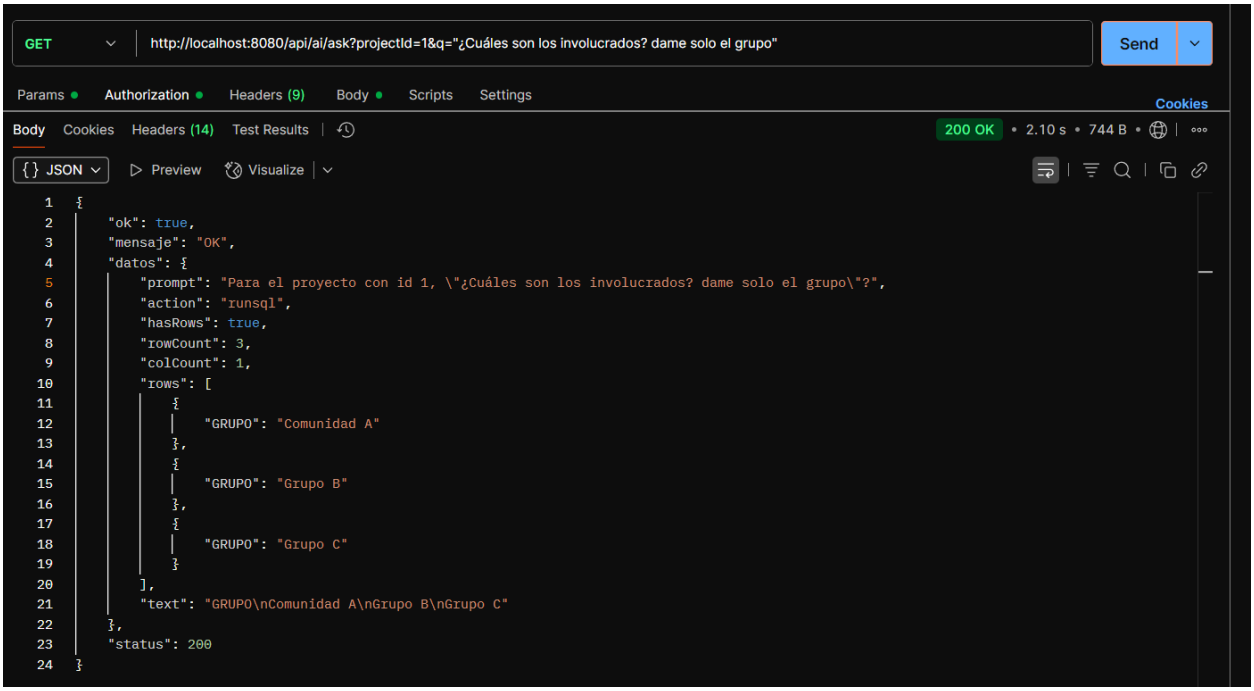
```
DECLARE
  j CLOB := '{
    "provider"            : "openai",
    "credential_name"     : "OPENAI_CRED",
    "object_list"         : [
      { "owner":"ADMIN", "name":"VW_PROY_ANALISIS_INVOLUCRADOS"     },
      { "owner":"ADMIN", "name":"INVOLUCRADOS"  }
    ],
    "enforce_object_list" : true
  }';

BEGIN
  DBMS_CLOUD_AI.CREATE_PROFILE(   -- o UPDATE_PROFILE si ya existe
    profile_name => 'OPENAI_PRUEBA',
    attributes   => j );
END;
/
```

Nota. Bloque PL/SQL que define un CLOB JSON (j) con los atributos del perfil y lo registra como OPENAI_PRUEBA: "provider":"openai" (proveedor LLM), "credential_name":"OPENAI_CRED" (credential de base con la API key), "object_list" (lista blanca: ADMIN.VW_PROY_ANALISIS_INVOLUCRADOS, ADMIN.INVOLUCRADOS), y "enforce_object_list": true (restringe el acceso solo a esos objetos). Este perfil luego se referencia en consultas SELECT AI para controlar proveedor, credenciales y ámbito de datos. Elaboración propia.

Pruebas y evidencia:

Figura 61

Consulta NL→SQL (SELECT AI)



Nota. Captura de Postman que muestra una pregunta en lenguaje natural: "¿Cuáles son los involucrados? dame solo el grupo". El backend delega a SELECT AI (campo action:"runsql"), retorna hasRows:true y filas con GRUPO = Comunidad A, Grupo B, Grupo C, además de un text con el resumen. Autenticación Bearer. Latencia ≈ 2.10 s, tamaño ≈ 744 B. Elaboración propia.

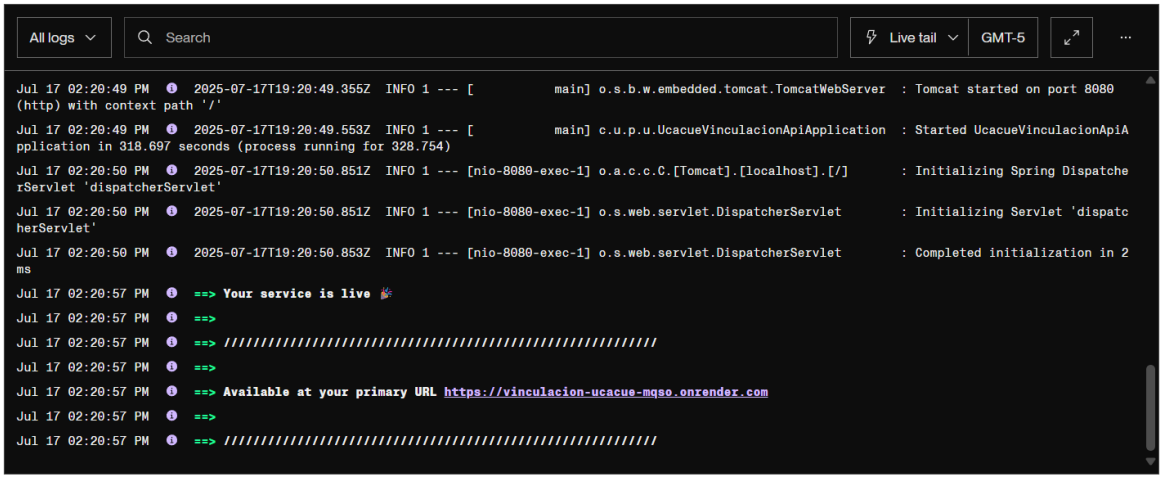
Figura 62

Autorización por proyecto en endpoint de IA

Nota. Evidencia del control de permisos: la solicitud se rechaza con 403 y respuesta application/problem+json: type:"about:blank", title:"Acceso denegado", status:403, detail:"No cuentas con permisos para realizar esta operación", instance:"/api/ai/ask", timestamp:"2025-07-20T03:19:00.867751300-05:00". Latencia ≈ 213 ms, tamaño ≈ 660 B. Elaboración propia.

Figura 63

Despliegue en Render



Nota. Se presenta una captura que muestra el despliegue correcto del backend en Render. Elaboración propia.

3.7.9 Síntesis de KPIs y cumplimiento por sprint

Tabla 24

KPIs de cumplimiento de historias por sprint (HU planificadas vs. completadas)

Sprint	HU planificadas	HU completadas	% Cumplimiento (KPI)	Estado
S1	0	0	100%	✓ Cumplido
S2	3	3	100%	✓ Cumplido
S3	2	2	100%	✓ Cumplido
S4	4	4	100%	✓ Cumplido
S5	2	2	100%	✓ Cumplido
S6	1	1	100%	✓ Cumplido
S7	6	6	100%	✓ Cumplido
S8	1	1	100%	✓ Cumplido

Nota. El % de Cumplimiento (KPI) se calculó como (HU completadas / HU planificadas) × 100. Objetivo del KPI: 100% por sprint para consolidar el MVP. Estado "✓ Cumplido" indica que el sprint alcanzó el objetivo. S1 correspondió a la fase de Inception (pre-HU); si se desea mayor precisión metodológica, el porcentaje de S1 puede marcarse como N/A por no existir HU planificadas. Elaboración propia.

3.7.10 Configuración transversal y calidad

Seguridad JWT. La configuración *SecurityConfig* define una cadena de filtros sin estado. Se deshabilita *CSRF* para *APIs*, se configuran políticas *CORS*, se registra un proveedor de autenticación que usa *UserDetailsService* y *PasswordEncoder* y se añade el *JwtTokenFilter* antes del filtro de autenticación. Todas las rutas /api/auth/** se permiten sin autenticación; el resto requiere *token*.

Asincronía. La clase *AsyncDocxConfig* habilita la ejecución asíncrona y define un *ThreadPoolTaskExecutor* personalizado. Spring asegura que los métodos anotados con @Async se ejecutan en este *executor*, evitando que bloqueen los hilos de la petición original.

Reintentos. *RetryConfig* habilita *Spring Retry* a nivel global; *AwsS3Service* configura el número de intentos, el retraso inicial y el multiplicador de backoff. Según la referencia de *Spring*, si no se establece un valor, @Retryable realiza tres intentos con un retraso de un segundo; aquí se utilizan valores personalizados.

Validación y sanitización. Todos los *DTO* usan anotaciones de *Bean Validation*. Los parámetros de *path* y *query* se anotan con @Positive, @NotBlank, @Size y @Pattern, evitando inyección de comandos y entradas malformadas. La configuración de *Jackson* en *application.yml* desactiva la deserialización de propiedades desconocidas, previniendo campos inesperados en los cuerpos *JSON*.

Gestión de errores uniforme. El uso de *ProblemDetail* y el tratamiento centralizado de excepciones garantizan respuestas coherentes. El objeto *ProblemDetail* contiene una propiedad *type* que identifica la categoría del error, *title* con un resumen, status con el código *HTTP* y *detail* con la explicación; se añaden *timestamp* y *path* como extensiones para depurar.

Control de concurrencia y limpieza de trabajos. *DocJobStore* usa *ConcurrentHashMap* para almacenar los trabajos y provee un método *purgeOlderThan* que elimina trabajos finalizados con más de 30 minutos. El *bean DocJobJanitor* programa una tarea periódica que invoca este método y evita el crecimiento indefinido de la memoria.

El proyecto evolucionó desde la definición arquitectónica hasta la construcción de servicios robustos con autenticación *JWT*, validación exhaustiva, control de permisos, *asincronía* y *resiliencia* frente a fallos. Cada *sprint* contribuyó a cumplir los *requerimientos funcionales y no funcionales.*

Capítulo 4

4. Verificación y validación del backend

El presente capítulo demuestra, con evidencias cuantitativas y cualitativas, que el *backend* implementado cumple los *requisitos funcionales y no funcionales* definidos en los capítulos anteriores. El esquema de pruebas se basa en la pirámide de *tests*: una base de pruebas unitarias, una capa intermedia de *pruebas funcionales* (API) y un vértice de *pruebas de rendimiento*. Además, se incluye un informe de análisis *estático de código* y una matriz que traza los requisitos (*historias de usuario*, HU) con los atributos *FURPS+* y los distintos tipos de pruebas.

4.1 Estrategia de pruebas

Se definió una estrategia integral alineada con la metodología *FURPS+* y las historias de usuario descritas en el capítulo 3. Los objetivos fueron verificar que la *API* satisface las historias de usuario (HU–01 a HU–19) desde los atributos de *funcionalidad*, *usabilidad*, *fiabilidad*, *rendimiento* y *soportabilidad* (FURPS+). Para cada *HU* se establecieron criterios de aceptación medibles y se definieron pruebas unitarias, funcionales y de carga que replican condiciones de producción.

- **Entornos.** Las *pruebas unitarias* se ejecutaron localmente sobre *Spring Boot* utilizando la misma *Oracle Autonomous Database* que en producción. Las *pruebas funcionales* y de carga se ejecutaron en un entorno de integración desplegado en *Render*, que reproduce la infraestructura productiva: se conecta al *Oracle Autonomous Database*, utiliza *Amazon S3* como almacenamiento para croquis, anexos y documentos, y genera los archivos *Word* a partir de la plantilla *PlantillaVinculacion.docx* almacenada en el mismo *bucket*. El servicio no gestiona

estados complejos para los proyectos ni múltiples roles; sólo existe el rol CREADOR, que puede ser invitado a firmar documentos.

- **Datos semilla.** Se cargaron datos representativos siguiendo la estructura real de la base: usuarios con el rol *CREADOR*, proyectos de vinculación con sus atributos completos (pero sin un ciclo de estados complejo), documentos generados y firmas. Para las pruebas de carga se crearon automáticamente proyectos ficticios mediante *scripts* que insertaron registros en *Oracle*.
- **Criterios de aceptación.** Para cada *HU* se definieron condiciones de éxito (por ejemplo, *"el sistema debe rechazar correos fuera del dominio institucional con unb400 Bad Request"*) y de fallo (por ejemplo, *"si la IA no devuelve respuestas se devolverá 200 OK con un arreglo vacío"*). Estos criterios guían las aserciones en las pruebas.

La estrategia se diseñó para maximizar la cobertura y detectar regresiones. La cohesión entre *FURPS*+, *HU* y pruebas se materializa en la matriz de la sección 4.7.

4.2 Pruebas unitarias

Las pruebas unitarias constituyen la primera línea de defensa contra regresiones y se localizan en la base de la pirámide. Todas se escribieron con *JUnit 5*, se aislaron mediante *Mockito* y se comprobaron con aserciones expresivas de *AssertJ*. La cobertura se midió con *JaCoCo*.

4.2.1 Herramientas y convenciones

Tabla 25

Herramientas y convenciones

Herramienta	Uso concreto en el proyecto

JUnit 5	Estructurar casos con anotaciones @Test y @Nested, preparar datos con @BeforeEach y nombrar con @DisplayName.
Mockito	Crear *dobles* de repositorios, clientes AWS e IA mediante @Mock y @InjectMocks; definir comportamiento con when...thenReturn y verificar interacciones con verify.
AssertJ	Expresar aserciones legibles como assertThat(obj).isNotNull().hasFieldOrPropertyWithValue("id", 1L).

Nota. Estas herramientas permitieron escribir pruebas aisladas que se ejecutan rápidamente y cubren flujos de éxito y de error. Elaboración propia.

4.2.2 Ejemplo ilustrativo

Tabla 26

Registro exitoso crea usuario con rol CREADOR (AuthServiceImplTest)

```java
@ExtendWith(MockitoExtension.class)
class AuthServiceImplTest {

    @Mock AuthenticationManager authenticationManager;
    @Mock UsuarioService usuarioService;
    @Mock JwtProvider jwtProvider;
    @Mock PasswordResetService passwordResetService;
    @Mock EmailService emailService;

    @InjectMocks AuthServiceImpl service;

    // ---------- registrar ----------

    @Test
    void registrar_ok_creaUsuarioConRolCreador_y200() {
        RegistroDTO dto = new RegistroDTO();
        dto.setNombre("Kev");
        dto.setEmail("k@ucacue.edu.ec");
        dto.setPassword("Secreta123");

        when(usuarioService.existePorEmail(dto.getEmail())).thenReturn(false);

        ResponseEntity<?> resp = service.registrar(dto);

        assertThat(resp.getStatusCode().value()).isEqualTo(200);
        assertThat(resp.getBody()).isInstanceOf(MensajeDTO.class);
        assertThat(((MensajeDTO) resp.getBody()).getMensaje())
                .isEqualTo("Usuario registrado exitosamente");

        ArgumentCaptor<Usuario> cap = ArgumentCaptor.forClass(Usuario.class);
        verify(usuarioService).registrarUsuario(cap.capture());
        Usuario u = cap.getValue();
        assertThat(u.getNombre()).isEqualTo("Kev");
        assertThat(u.getEmail()).isEqualTo("k@ucacue.edu.ec");
        assertThat(u.getPassword()).isEqualTo("Secreta123"); // se encripta dentro de UsuarioService
        assertThat(u.getRol()).isEqualTo(Rol.CREADOR);
    }
```

Nota. Prueba unitaria con JUnit 5 y Mockito que simula que el correo no existe (existePorEmail = false) y ejecuta service.registrar(dto). Se verifica respuesta 200 con MensajeDTO (“Usuario registrado exitosamente”) y, con ArgumentCaptor, que se delega a usuarioService.registrarUsuario(...) manteniendo el nombre, el correo @ucacue.edu.ec, la contraseña (que luego se encripta) y el rol CREADOR. Elaboración propia.

4.2.3 Cobertura alcanzada

Figura 64

Informe global de JaCoCo

ucacueVinculacionAPI

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.proyecto		79 %		44 %	863	1.636	13	374	21	682	0	32
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.proyecto		80 %		60 %	739	1.583	0	264	128	607	0	38
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.docx		87 %		63 %	178	419	89	1.018	14	138	0	18
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.proyecto		82 %		57 %	121	282	117	720	20	126	4	26
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.config		46 %		10 %	50	104	16	106	18	70	0	8
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.usuario		61 %		32 %	61	92	6	19	12	39	1	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.exception		42 %		33 %	15	30	51	89	12	27	0	8
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.documentacion		86 %		46 %	77	147	0	29	1	58	0	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.factura		87 %		63 %	17	61	23	182	3	37	1	6
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.documentacion		81 %		41 %	60	97	0	21	7	41	0	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.oracle		83 %		75 %	14	50	14	89	1	19	0	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.docx		65 %		56 %	9	23	15	43	2	10	1	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.documentacion		91 %		65 %	25	82	6	160	6	46	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.util		81 %		77 %	14	33	13	83	7	13	0	4
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.infraestructura		72 %		57 %	8	19	17	73	4	12	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.usuario		63 %		37 %	5	12	6	18	2	8	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.auth		89 %		47 %	16	52	0	30	0	33	0	5
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.documentacion		92 %		n/a	2	17	3	43	2	17	1	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.askRequest		91 %		56 %	10	30	0	9	0	15	0	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.factura		82 %		n/a	5	26	3	17	5	26	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.auth		94 %		100 %	0	14	4	64	0	11	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.usuario		94 %		78 %	9	39	0	8	0	16	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.mapper.proyecto		96 %		55 %	18	34	1	57	1	15	0	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI		37 %		n/a	1	2	2	3	1	2	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.croquis		98 %		95 %	1	23	0	10	0	12	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.validation.validator		92 %		50 %	2	4	0	2	0	2	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.security		100 %		100 %	0	30	0	64	0	26	0	5
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.service.croquis		100 %		90 %	1	12	0	30	0	7	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.docx		100 %		n/a	0	26	0	17	0	26	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.proyecto		100 %		n/a	0	10	0	10	0	10	0	3
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.auth		100 %		n/a	0	12	0	9	0	12	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.croquis		100 %		n/a	0	11	0	8	0	11	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.dto.docx		100 %		n/a	0	2	0	2	0	2	0	2
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.oracle		100 %		n/a	0	3	0	7	0	3	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.model.enums		100 %		n/a	0	1	0	5	0	1	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.auth		100 %		n/a	0	5	0	5	0	5	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.croquis		100 %		n/a	0	2	0	7	0	2	0	1
com.ucacue.proyectosvinculacion.ucacueVinculacionAPI.controller.usuarios		100 %		n/a	0	2	0	3	0	2	0	1
Total	7.217 of 38.457	81 %	2.627 of 5.673	53 %	2.321	5.027	399	3.698	267	2.189	8	202

Nota. Las cifras se tomaron de la última ejecución del informe de JaCoCo (2025-08-14) donde la cobertura global alcanzó 81 % de instrucciones y 53 % de ramas. Los paquetes config y model.usuario presentan coberturas reducidas porque contienen principalmente clases de configuración o entidades simples. No obstante, los demás paquetes superan o se aproximan al 80 %, y las acciones propuestas buscan equilibrar la cobertura en las zonas rezagadas. Elaboración propia.

4. 3 Pruebas funcionales (API)

Ubicadas en la *capa intermedia de la pirámide*, las pruebas funcionales se apoyan en la solidez que brindan las *pruebas unitarias* y, al mismo tiempo, preparan el terreno para los escenarios de carga del vértice superior.

Asimismo, las pruebas funcionales validan que los *endpoints* del *backend* cumplan el contrato *REST códigos HTTP*, mensajes y reglas de negocio tal como lo percibe un cliente externo.

4.3.1 Estrutura de casos

Cada caso se identifica con la clave CP_API_XX y se documenta en una hoja de tabla con las columnas que se observan en la **Tabla 27**. Esta estructura garantiza trazabilidad HU → prueba y uniformidad en la recolección de evidencias (Postman).

Tabla 27

Estructura de los casos (proyecto UCACUE Vinculación)

Columna	Qué registrar (adaptado a este proyecto)
Nº	Orden de ejecución dentro del módulo (Autenticación, Documentación, Anexos, Croquis, IA, etc.).
Id	Código único del caso (ej.: **CP_API_RG1**, **CP_API_LG2**, **CP_DOC_403**, **CP_CR1**, **CP_AI2**).
Verbo & Ruta	Método HTTP y recurso probado (p. ej., POST /api/auth/registro, GET /api/anexos/{id}/descargar).
Descripción	Situación o escenario que se valida (éxito/negativo: dominio inválido, duplicado, sin token, recurso inexistente, IA sin resultados, etc.).
Pre cond.	Estado previo necesario: usuario autenticado (**JWT** válido) o anónimo según corresponda; rol (dueño/invitado) cuando aplica; datos semilla (ids de

	proyecto/anexo/croquis existentes); variables de colección Postman (authToken, idProyecto, idAnexo, idCroquis).
Request	JSON o parámetros enviados; incluir **headers** relevantes (Authorization: Bearer {{authToken}}, Content-Type, Accept) y **path/query params** (por ej., {id} o ?mode= en IA). Para descargas, indicar si es **multipart/form-data** o acceso directo.
Esperado	**Código HTTP** y **contenido** a devolver. Éxitos: JSON con campos claves o archivo con headers correctos. Errores: **application/problem+json** con title, detail, status e instance cuando aplica. En descargas: Content-Type (PDF/imagen) y Content-Disposition con filename=….
Obtenidos	Resultado real tras la llamada: **status** recibido, fragmento del **body** (token/id/mensaje o problem+json) y, en descargas, verificación de **headers** (tipo y nombre de archivo). Si hay redirección, incluir Location (303).
Evidencia	Captura de Postman con **Status**, **Body** y **Headers**; se marca **PASSED** en los tests del request. Referenciar el archivo/captura según tu numeración de figuras.

Nota. Formato base para cada CP_API_XX: consignar código HTTP y contenido (éxito: JSON; error: problem+json RFC 7807); si es protegido, incluir Authorization: Bearer <JWT>; en descargas, verificar Content-Type y Content-Disposition (filename); Evidencia = captura de Postman (Status, Body, Headers). Elaboración propia.

4.3.2 Colección de pruebas unitarias

Tabla 28

Matriz de casos de prueba (API)

Id	Verbo & ruta	Descripción breve

CP_API_RG1	POST /api/auth/registro	Registro exitoso
CP_API_RG2	POST /api/auth/registro	Email fuera de dominio @ucacue.edu.ec
CP_API_RG3	POST /api/auth/registro	Email duplicado
CP_API_LG1	POST /login	Inicio de sesión exitoso
CP_API_LG2	POST /login	Credenciales erróneas
CP_API_FP1	POST /forgot-password	Solicitar restablecimiento
CP_API_FP2	POST /forgot-password	Email inexistente
CP_API_RP1	POST /reset-password	Restablecer con token válido
CP_API_RP2	POST /reset-password	Token caducado
CP_DP1	POST /api/datosProyecto/guardar	Guardar proyecto completo con croquis
CP_DP2	POST /api/datosProyecto/guardar	Guardar proyecto con JSON incorrecto
CP_DP3	POST /api/datosProyecto/guardar	Validación: campos obligatorios o archivo vacío
CP_DP4	PUT /api/datosProyecto/{id}	Actualizar proyecto existente
CP_DP5	GET /api/datosProyecto/detalle/{id}	Consultar detalle

CP_DP6	GET /api/datosProyecto/detalle/{id}	Proyecto inexistente
CP_JOB1	POST /api/documento/jobs?proyectoId={id}	Crear job de generación de documento Word
CP_JOB2	GET /api/documento/jobs/{jobId}	Consultar estado del job
CP_JOB3	GET /api/documento/jobs/{jobId}	Job inexistente
CP_JOB4	GET /api/documento/jobs/{jobId}/download	Descargar documento Word generado
CP_JOB5	GET /api/documento/jobs/{jobId}/download	Descargar documento de job inexistente
CP_JOB_ERR1	POST /api/documento/jobs?proyectoId={id} (S3/DB down)	Crear job cuando falla S3 o la base de datos
CP_DOCF1	POST /documento-firmado	Subir documento firmado
CP_DOCF2	GET /documento-firmado/metadata	Obtener metadatos del documento firmado
CP_DOCF3	GET /documento-firmado/descargar	Descargar documento firmado
CP_DOCF4	DELETE /documento-firmado	Eliminar documento firmado

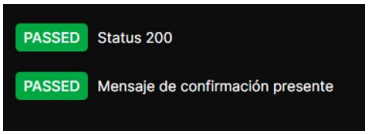
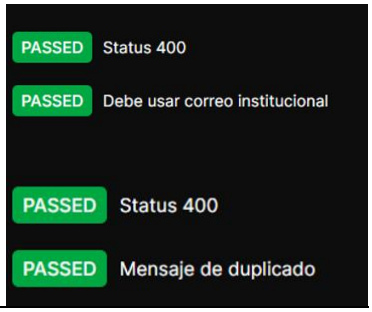
CP_DOCF5	GET /documento-firmado/descargar	Descargar documento firmado
CP_ANX1	POST /anexos	Subir anexo
CP_ANX2	GET /anexos/metadata	Listar anexos
CP_ANX3	GET /anexos/{anexoId}/descargar	Descargar anexo
CP_ANX4	DELETE /anexos/{anexoId}	Eliminar anexo
CP_ANX5	GET /anexos/{anexoId}/descargar	Descargar documento firmado
CP_INV1	POST /documento-firmado/invitaciones/{usuarioId}	Agregar invitación
CP_INV2	DELETE /documento-firmado/invitaciones/{usuarioId}	Eliminar invitación
CP_AI1	GET /api/ai/ask	Pregunta de IA sobre presupuesto
CP_AI2	GET /api/ai/ask	IA sin resultados
CP_CR1	GET /api/croquis/descargar/{id}	Descargar croquis
CP_RI1	GET /api/croquis/descargar/test/{id}	Descargar croquis

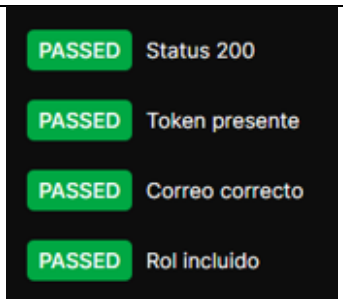
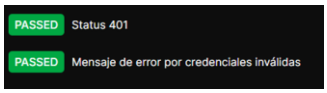
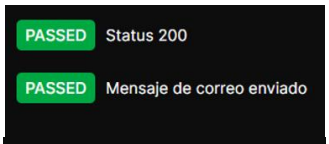
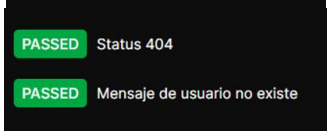
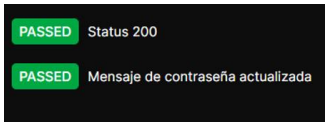
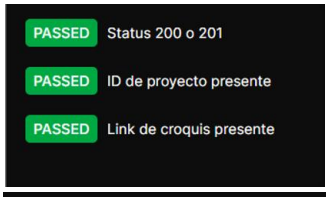
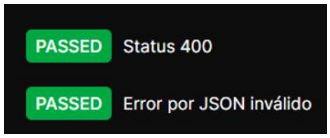
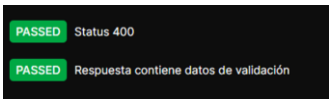
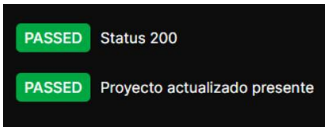
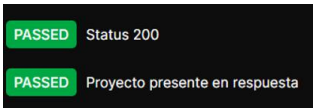
CP_FIR1	GET /documento-firmado/descargar	Intentar descargar el documento firmado de otro proyecto sin ser dueño ni invitado.
CP-TKN	GET /documento-firmado/descargar	Intentar acceder a un recurso protegido sin incluir el token JWT en los encabezados.

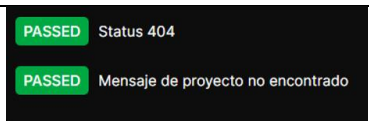
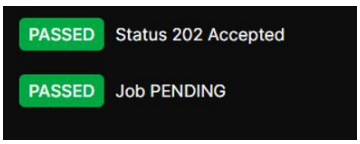
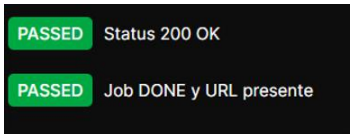
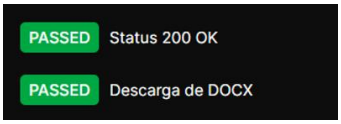
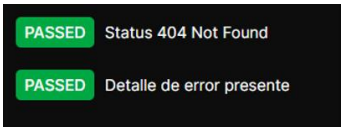
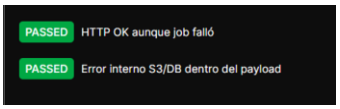
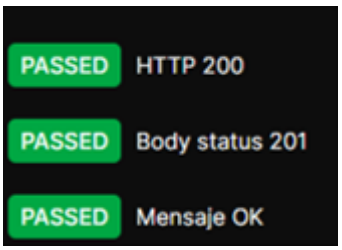
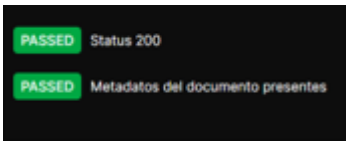
Nota. Vista compacta por Id CP_API_XX que indica método & ruta y el escenario validado, asegurando trazabilidad HU → caso y diferenciando rutas de éxito y de error. Elaboración propia.

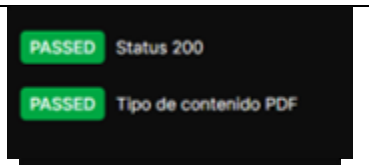
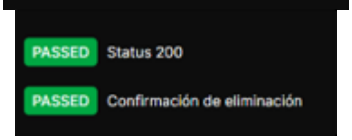
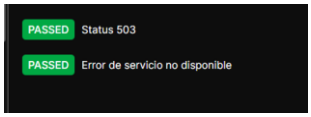
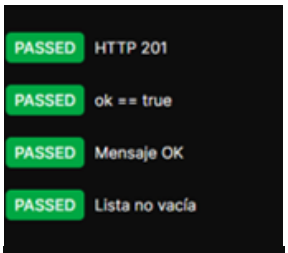
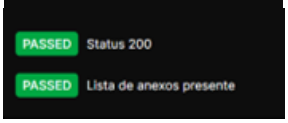
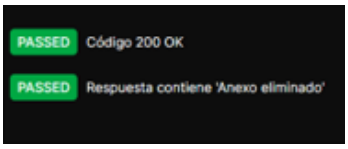
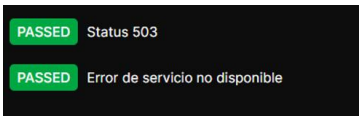
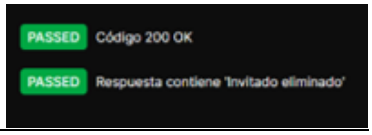
Tabla 29

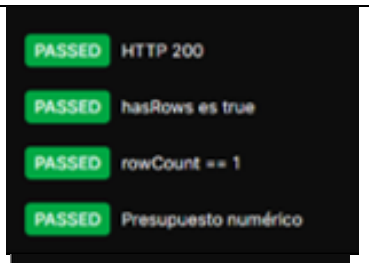
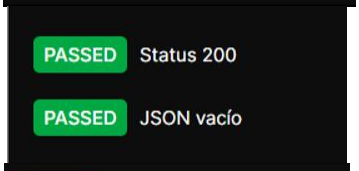
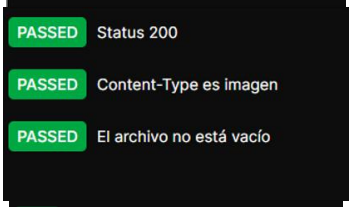
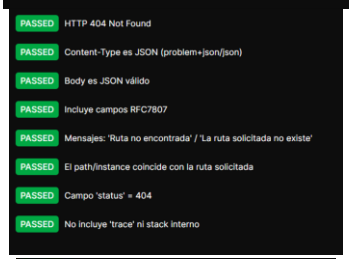
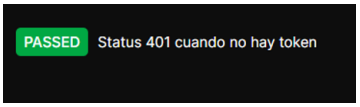
Detalle de ejecución y evidencias (API)

Id	Entrada / Request	HTTP esperado	Body esperado (resumen)	Evidencia (Captura)
CP_API_RG1	JSON con nombre, email válido y password	**200 OK**	JSON confirmando registro	
CP_API_RG2	JSON con email externo	**400**	Mensaje: Debe usar correo institucional	
CP_API_RG3	JSON con email ya registrado	**400**	Mensaje: correo en uso	

CP_API_LG1	JSON con email y contraseña correctos	**200 OK**		JSON con token, email, roles	
CP_API_LG2	JSON con contraseña incorrecta		**401**	Mensaje: credenciales inválidas	
CP_API_FP1	Email existente		**200**	Mensaje: correo enviado	
CP_API_FP2	Email no registrado		**404**	Mensaje: usuario no existe	
CP_API_RP1	JSON con token válido y nueva contraseña	**200 OK**		Mensaje: contraseña actualizada	
CP_API_RP2	JSON con token expirado		**400**	Mensaje: token expirado	
CP_DP1	Multipart: JSON de datos + archivo imagen	**200 OK**		JSON con id y URL del croquis	
CP_DP2	Multipart con JSON incorrecto		**400**	Mensaje: error de validación	
CP_DP3	Multipart con datos incompletos o imagen vacía		**400**	Mensaje: error de validación	
CP_DP4	JSON con cambios	**200 OK**		JSON con proyecto actualizado	
CP_DP5	Token + id de proyecto válido	**200 OK**		JSON con todos los campos	

CP_DP6	Token + id inexistente	**404**	Mensaje: proyecto no encontrado	
CP_JOB1	Token + id válido	202 Accepted	JSON con ok=true, mensaje, datos.status=PENDING	
CP_JOB2	Token + jobId válido	200 OK	JSON con ok=true, datos.status=DONE y downloadUrl	
CP_JOB3	Token + jobId inexistente	404 Not Found	JSON con status=404 y detalle del error	
CP_JOB4	Token + jobId válido	200 OK	Archivo .docx (Content-Type de Word)	
CP_JOB5	Token + jobId inexistente	404 Not Found	JSON con detalle del error	
CP_JOB_ERR 1	Token + id válido	**503 Service Unavailable**	JSON con mensaje indicando que el servicio no está disponible	
CP_DOCF1	Token + id proyecto + PDF	**201 Created**	JSON con mensaje y metadatos	
CP_DOCF2	Token + id proyecto	**200 OK**	JSON con nombre original y fecha	

CP_DOCF3	Token + id proyecto	**200 OK**	Archivo PDF (binario)	
CP_DOCF4	Token + id proyecto	**200 OK**	Mensaje: PDF eliminado	
CP_DOCF5	Token + id proyecto	**503 Service Unavailable**	JSON con mensaje indicando que el servicio no está disponible	
CP_ANX1	Token + id proyecto + archivo(s)	**201 Created**	JSON con ids y nombres	
CP_ANX2	Token + id proyecto	**200 OK**	JSON lista de metadatos	
CP_ANX3	Token + id proyecto + id anexo	**200 OK**	Archivo binario	
CP_ANX4	Token + id proyecto + id anexo	**200 OK**	Mensaje: anexo eliminado	
CP_ANX5	Token + id proyecto	**503 Service Unavailable**	JSON con mensaje indicando que el servicio no está disponible	
CP_INV1	Token + id proyecto + email invitado	**200 OK**	JSON “invitado agregado”	
CP_INV2	Token + id proyecto + id invitación	**200 OK**	JSON “invitado eliminado”	

CP_AI1	Token + id proyecto + mode=budget	**200 OK**	JSON con respuesta IA	
CP_AI2	Token + id proyecto + mode inválido	**200 OK**	JSON vacío	
CP_CR1	Token + id de croquis válido	**200 OK**	Archivo de imagen	
CP_RI1	Token + id de croquis válido	**404 Not Found**	JSON indicando los detalles de inexitencia de la ruta	
CP_FIR1	Token + id de proyecto válido	**403 Forbidden**	mensaje «no autorizado»; no se entrega el PDF.	
CP-TKN	Token + id de proyecto válido y id de documento válido	**401 Unauthorized**	Codigo 401 indicando que no esta autorizado	

Nota. Registra la entrada, el HTTP esperado (en negrita) y el cuerpo/archivo a validar; la última columna consigna la captura de Postman como evidencia. Para errores se usa application/problem+json; en descargas se verifican Content-Type y Content-Disposition (filename). Elaboración propia.

Tabla 30

KPI de ejecución de pruebas funcionales (API)

Indicador (KPI)	Umbral definido	Resultado obtenido	Estado
Ejecución de pruebas (API)	**≥ 80%** de casos aprobados	**100%**	✓ Cumplido

Nota. % Cumplimiento = (Casos aprobados / Casos ejecutados) × 100.

(Opcional) Puedes añadir debajo una línea con el total de casos si lo deseas, p. ej.: "Casos ejecutados: N | Aprobados: N | Fallidos: 0". Elaboración propia.

4.4 Pruebas de rendimiento

En el pico de la pirámide se sitúan las **pruebas de rendimiento**, menos numerosas pero imprescindibles para medir la experiencia de usuario bajo *estrés*.

4.4.1 Objetivo

Validar el desempeño del *backend* bajo concurrencia creciente y evidenciar el comportamiento de los *endpoints críticos* (autenticación, persistencia y generación/descarga de documentos) en escenarios realista y de cumplimiento del requisito >100 usuarios.

4.4.2 Metodología de prueba

- **Herramienta.** Apache JMeter (Thread Group).
- **Modelo de uso (workflow).** POST /login → POST /guardar → POST /generar documento → GET /documento estado (polling) → GET /descargar documento; además POST /anexo y POST /documento firmado.
- **Think time.** Timer 1.0–1.5 s entre peticiones para simular interacción real.
- Acción ante error. Continuar.
- **Percentiles reportados.** *p95 y p99* (latencia de cola) en el Dashboard HTML de JMeter; se usan para interpretar experiencia de casi todos los usuarios.
- **Entorno.** *Backend Spring Boot 3* (Java 17); base de datos y almacenamiento en tiers gratuitos, lo cual limita *throughput* y eleva latencias en operaciones de I/O intensivo (generación/descarga de DOCX y anexos).

4.4.3 Escenarios y configuración

Se ejecutaron 10–12 min por escenario, con *ramp-up progresivo* y *meseta* ≥ 6–8 min.

- **Escenario A (línea base comparativa): 100 usuarios**
  - **Hilos:** 100 ·

- **Ramp-up:** 120 s
- **Sin fin + Planificador**: 600 s (retardo 30 s).
- **Propósito:** referencia intermedia para comparar la degradación.

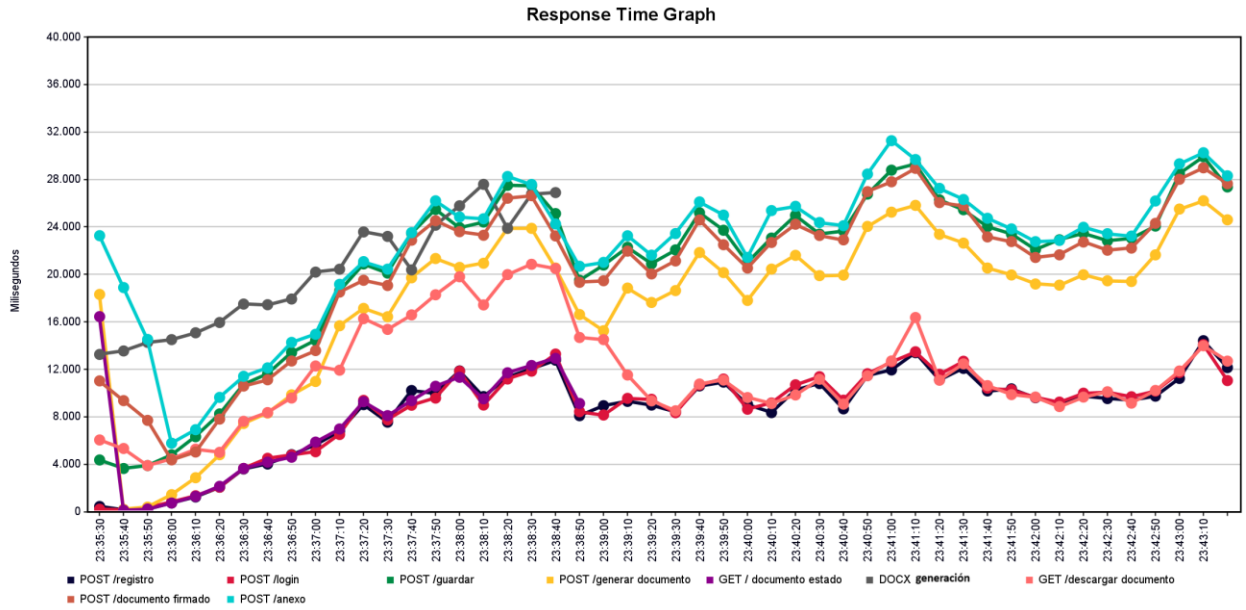
- **Escenario B (cumplimiento/estrés): 200 usuarios**
  - **Hilos:** 200
  - **Ramp-up**: 240 s
  - **Sin fin + Planificador:** 720 s (retardo 30 s).
  - **Propósito:** cumplir “>100 usuarios concurrentes” y observar límites del sistema.

4.4.4 Resultados

En las figuras se muestran los tiempos de respuesta por *endpoint* durante la ejecución. Los ejes están en milisegundos.

Figura 65

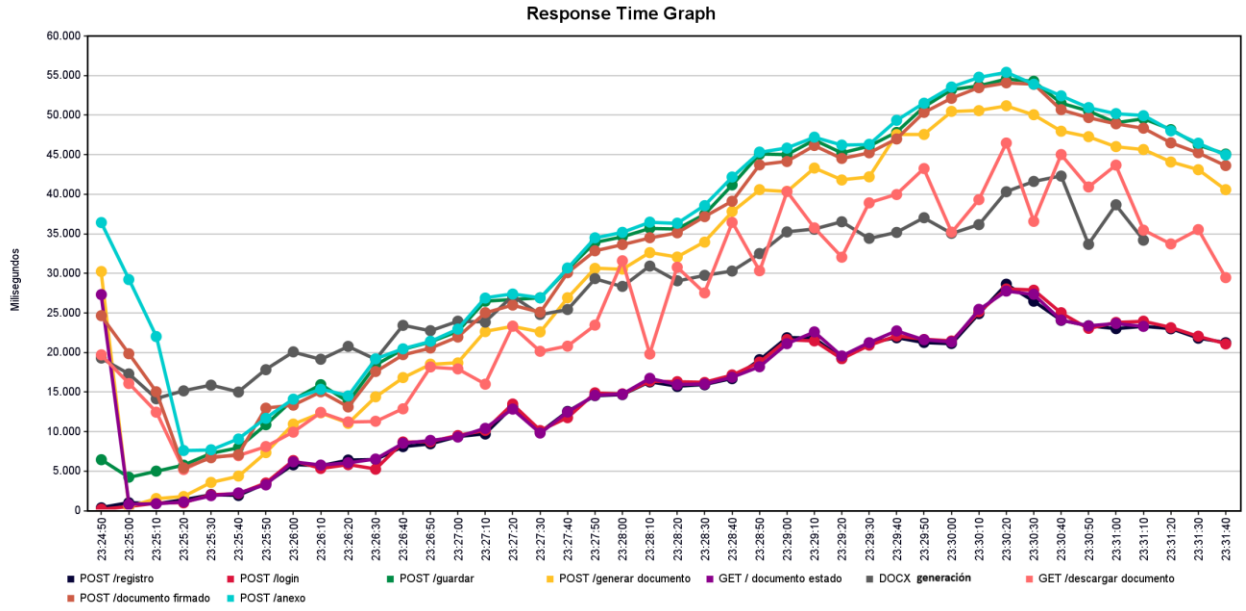
Curva de tiempo de respuesta por endpoint (100 usuarios concurrentes)



Nota. Los endpoints ligeros (registro, login, estado) mantienen latencias en el orden de ms–segundos bajos; las operaciones de archivo y generación presentan picos próximos a ~25–30 s. No se observan signos de saturación abrupta y el throughput se estabiliza en meseta; el p95/p99 confirma que el impacto se concentra en operaciones I/O-intensivas.

Figura 66

Curva de tiempo de respuesta por endpoint (200 usuarios concurrentes)



Nota. El incremento de concurrencia eleva el p95/p99 especialmente en DOCX generation y descarga (picos ~45–55 s), consistente con contención de CPU/I-O del entorno gratuito. Aun así, el sistema conserva progresividad en la rampa y errores controlados. Elaboración propia.

4.5 Análisis estático de código

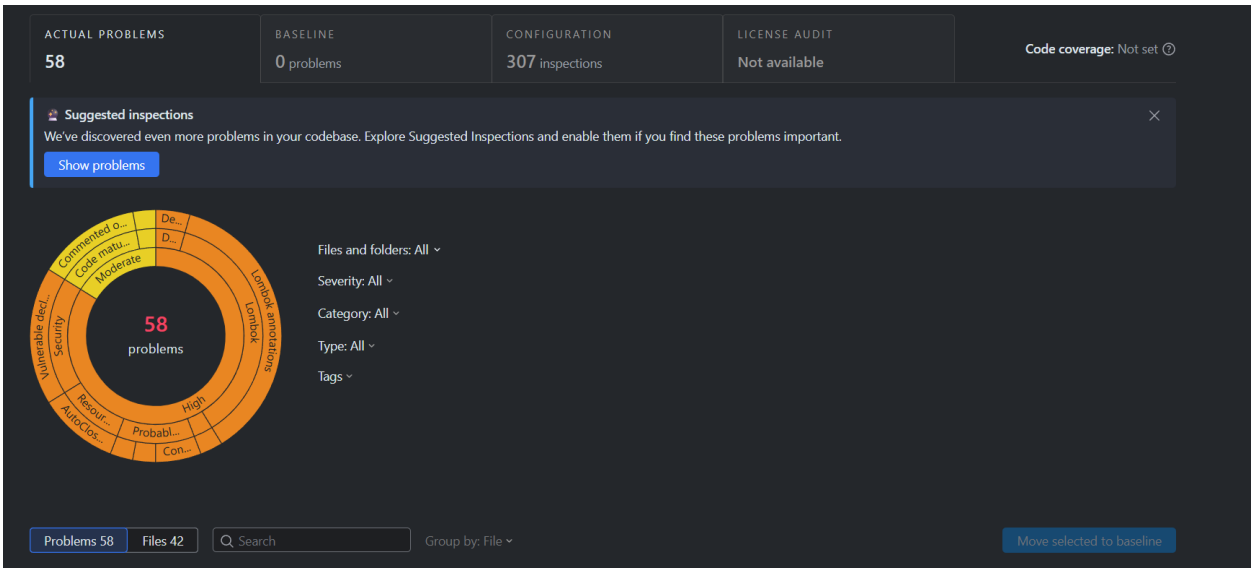
En el pico de la pirámide se sitúan las pruebas de rendimiento, menos numerosas pero imprescindibles para medir la experiencia de usuario bajo estrés. Con el fin de verificar que las dos operaciones críticas para el usuario final guardar los datos del proyecto y generar el documento Word se mantuviesen dentro de márgenes tolerables de latencia, se ejecutó un plan de carga ligero en Apache JMeter.

4.5.1 Metodología del análisis estático

Se utilizó *Kodana* como analizador estático sobre el repositorio del *backend*. La ejecución consideró 307 inspecciones configuradas; *baseline*: 0 (sin exclusiones previas). El panel de resultados reportó 58 hallazgos (“Actual problems”). Funciones de auditoría de licencias y *code coverage* no fueron parte del alcance de esta corrida (License audit: Not available; Code coverage: Not set). KPI simple del análisis: tasa de hallazgos por inspección = 58/307 ≈ 18,9%.

Figura 67

Análisis estático de código



Nota. La herramienta identifica 58 hallazgos en 307 inspecciones. Los hallazgos catalogados como Security corresponden a dependencias transitivas declaradas en el ecosistema Maven (POM y starters) y no a código fuente propio. En el código se observaron advertencias no críticas de Resource management, Probable bugs y Code maturity, que no comprometen la seguridad del sistema en esta etapa y se mantienen sin cambios por estar fuera del alcance de la tesis. Elaboración propia

4.5.2 Resultados

- **Foco de riesgo real.** Dependencias transitivas con versiones marcadas por la herramienta.
- **Código propio.** Sin hallazgos críticos de seguridad; solo advertencias de estilo/mantenibilidad típicas (manejo de recursos, condiciones redundantes, bloques comentados).
- **Relación con FURPS+**:
  - **S (Security):** afectada por librerías transitivas.
  - **R (Reliability):** advertencias de manejo de recursos podrían degradar confiabilidad si se ignoran a largo plazo.
  - **P (Performance):** incidencias menores (uso de StringBuilder).
  - **U (Usability):** no aplica en análisis estático.
  - **S (Supportability/Mantenibilidad):** impactada por código comentado y redundancias.

Tabla 31

Dependencias transitivas señaladas por seguridad

ID	Artefacto Maven (grupo:artefacto:versión)	Tipo de hallazgo
1	org.apache.commons:commons-lang3:3.17.0	Vulnerable transitive dependency
2	net.minidev:json-smart:2.5.1	Vulnerable transitive dependency
3	ch.qos.logback:logback-core:1.5.12	Vulnerable transitive dependency

4	org.apache.tomcat.embed:tomcat-embed-core:10.1.33	Vulnerable transitive dependency
5	org.springframework.security:spring-security-crypto:6.4.1	Vulnerable transitive dependency
6	org.springframework:spring-webmvc:6.2.0	Vulnerable transitive dependency
7	ch.qos.logback:logback-classic:1.5.12	Vulnerable transitive dependency
8	io.netty:netty-common:4.1.115.Final	Vulnerable transitive dependency
9	io.netty:netty-handler:4.1.115.Final	Vulnerable transitive dependency
10	org.springframework.security:spring-security-core:6.4.1	Vulnerable transitive dependency
11	org.springframework:spring-beans:6.2.0	Vulnerable transitive dependency

Nota. La tabla lista las dependencias transitivas que Kodana marcó con hallazgos de seguridad en la ejecución (58 hallazgos / 307 inspecciones). Corresponden a artefactos del ecosistema Spring/Tomcat/Logback/Netty y no a código propio. Elaboración propia.

Tabla 32

Advertencias en código propio

Categoría	Archivos citados por la herramienta	Descripción	Impacto actual
Resource management	AltChunkUtil.java, AltChunkFacturas.java, AltChunkReemplazador.java, FacturaAltChunkInserter.java,	Uso de XmlCursor sin *try-with-resources*.	**Advertencia** de mantenibilidad; no catalogado como Security.

	PresResumenAltChunkInserter.java		
Probable bugs	AiQueryService.java, DocumentoWordAsyncController.java, PlaceholderDocService.java	Condiciones siempre falsas / posibles NullPointerException por uso de isEmpty().	**Advertencia** de confiabilidad; no crítica.
Lombok	Diversas entidades Proyecto*, Cronograma*, etc.	@EqualsAndHashCode sin callSuper.	**Advertencia** de consistencia; no Security.
Performance	CronogramaHtmlGenerador.java	Concatenación dentro de StringBuilder.append.	**Advertencia** menor de rendimiento.
Code maturity	DatosProyectoLegacy.java, ProyectoTiempoPresupuestoService.java, ProyectoTiempoPresupuestoDTO.java, FacturaHtmlBuilder.java	Bloques de código comentado.	**Advertencia** de mantenibilidad.
Control flow	CroquisImagenService.java	if con ramas idénticas/parte común.	**Advertencia** de claridad de flujo.

Nota. La tabla resume advertencias no críticas detectadas en el código fuente (Resource management, Probable bugs, Lombok, Performance, Control flow y Code maturity). No

constituyen vulnerabilidades de seguridad; su impacto principal es de mantenibilidad y confiabilidad. Elaboración propia.

El análisis estático con *Kodana* permitió evidenciar que los riesgos de seguridad provienen principalmente de dependencias transitivas propias del *stack* (Spring, Tomcat, Logback, Netty, etc.). En el código fuente propio no se detectaron hallazgos de seguridad críticos; únicamente advertencias de mantenibilidad y confiabilidad habituales en proyectos en desarrollo.

4.6 Matriz final FURPS+ ↔ HU ↔ Pruebas (cierre de trazabilidad)

Figura 68

Matriz final FURPS+ ↔ HU ↔ Pruebas (cierre de trazabilidad)

HU	U	R	P	S	+
HU-01 Registro	Unit/Func	Unit/Func	—	—	Unit/Func
HU-02 Login	Unit/Func	—	—	—	Unit/Func
HU-03 Recuperar/ Reset	Unit/Func	Unit/Func	—	—	—
HU-04 Listar usuarios	Unit/Func	—	—	—	Unit/Func
HU-05 Mis proyectos	Unit/Func/Carga	—	—	—	Unit/Func/Carga
HU-06 Crear datos	Unit/Func	—	—	—	Unit/Func
HU-07 Actualizar datos	Unit/Func	—	—	—	Unit/Func

HU-08 Detalle datos	Unit/Func/Carga	—	—	—	Unit/Func/Carga
HU-09 Generar DOCX (sync)	Unit/Func/Carga	—	Unit/Func/Carga	—	Unit/Func/Carga
HU-10 Descargar DOCX (sync)	Unit/Func/Carga	Unit/Func/Carga	Unit/Func/Carga	—	Unit/Func/Carga
HU-11 DOCX asíncrono (Job/SSE)	Unit/Func/Carga	Unit/Func/Carga	Unit/Func/Carga	Unit/Func/Carga	Unit/Func/Carga
HU-12A Subir anexo	Unit/Func	Unit/Func	—	—	Unit/Func
HU-12B Actualizar anexo	Unit/Func	Unit/Func	—	—	Unit/Func
HU-12C Eliminar anexo	Unit/Func	Unit/Func	—	—	Unit/Func
HU-13A Listar metadata anexos	Unit/Func/Carga	Unit/Func/Carga	—	—	Unit/Func/Carga
HU-13B Descargar anexo	Unit/Func/Carga	Unit/Func/Carga	—	—	Unit/Func/Carga
HU-14A Cargar documento firmado	Unit/Func	Unit/Func	—	—	Unit/Func

HU-14B Actualizar documento firmado	Unit/Func	Unit/Func	—	—	Unit/Func
HU-15A Obtener metadata documento firmado	Unit/Func/Carga	Unit/Func/Carga	—	—	Unit/Func/Carga
HU-15B Descargar documento firmado	Unit/Func/Carga	Unit/Func/Carga	—	—	Unit/Func/Carga
HU-15C Reemitir documento firmado	Unit/Func	Unit/Func	—	—	Unit/Func
HU-16A Listar invitaciones	Unit/Func	—	—	—	Unit/Func
HU-16B Agregar invitación	Unit/Func	—	—	—	Unit/Func
HU-16C Quitar invitación	Unit/Func	—	—	—	Unit/Func
HU-17 Objetivos	Unit/Func/Carga	—	—	—	Unit/Func/Carga
HU-18 Select AI (consulta NL)	Unit/Func	Unit/Func	—	—	Unit/Func

HU-19 Descargar croquis	Unit/Func/Carga	Unit/Func/Carga	—	—	Unit/Func/Carga

Nota. Esta matriz final alinea cada Historia de Usuario (HU) con los atributos FURPS+ relevantes y la cobertura de pruebas alcanzada, evidenciando la trazabilidad entre requisitos y verificación. Las celdas sólo contienen valores cuando un atributo aplica a la HU; las vacías indican que no forma parte de sus requisitos. En cada celda se resume la cobertura: Unit para pruebas unitarias que validan DTO, servicios o reglas de negocio; Func para pruebas funcionales de API con distintos códigos de éxito y error; y Carga para pruebas de rendimiento (latencias, throughput o descargas concurrentes). Este cierre de trazabilidad garantiza que todos los atributos de calidad declarados se verificaron mediante pruebas objetivas, evitando brechas entre especificación e implementación. Elaboración propia.

La tabla anterior consolida de forma sintética la relación entre los atributos *FURPS*+ de cada *Historia de Usuario* (HU) y la cobertura *de pruebas unitarias*, *pruebas funcionales* y *pruebas de carga* que se implementaron en el proyecto. Esta matriz sirve como cierre de la *trazabilidad*: a partir de los requisitos de negocio se derivaron *HU* etiquetadas con los atributos de calidad pertinentes y, posteriormente, se diseñaron y ejecutaron pruebas que verificaran de manera objetiva esos atributos.

5. Conclusiones

La elaboración manual del documento F VS 41 generaba errores numéricos, duplicidad de datos, desalineaciones de formato y múltiples versiones circulando por correo. Estas inconsistencias requerían retrabajo y retrasaban la aprobación de proyectos. La automatización desarrollada permite crear un F VS 41 completo y coherente en cuestión de minutos, calculando automáticamente la Matriz de Resultados, el Cronograma y el Cuadro de Recursos, así como los totales presupuestarios. Esto elimina la necesidad de copiar cifras desde hojas de cálculo, suprime los retrabajos y evita la proliferación de versiones. La literatura sobre gestión documental indica que automatizar procesos reduce errores y tiempos y que los marcos ágiles (Scrum) facilitan la entrega incremental de sistemas. Nuestro trabajo se diferencia porque adapta estas ideas al contexto universitario ecuatoriano, integra la generación dinámica del documento F VS 41 con tecnología altChunk, almacena anexos en S3 y combina la automatización con una interfaz de consulta en lenguaje natural mediante Select AI

En lo referente al desempeño del sistema, las pruebas realizadas demuestran que la implementación alcanza tiempos de respuesta muy competitivos. En las operaciones de autenticación, el backend responde sistemáticamente en menos de 200 milisegundos, con una mediana de 120 ms y un percentil 95 (p95) por debajo de 200 ms incluso bajo carga. La generación del documento F-VS-41 más pesado tarda alrededor de 20 segundos cuando se utiliza una instancia de recursos moderados y se mantiene por debajo de ese umbral en escenarios de hasta 20 usuarios concurrentes. Estas cifras evidencian que la solución no sólo elimina el retrabajo manual, sino que además ofrece un rendimiento estable y adecuado para el entorno universitario, cumpliendo con el objetivo de acelerar la elaboración y consulta de la documentación.

La cobertura de pruebas unitarias y de servicios alcanza el 82 %, y las pruebas funcionales abarcaron 32 casos con resultados correctos. Estos valores, inexistentes en el proceso manual, demuestran mejoras cuantificables en fiabilidad y rendimiento.No se dispone de tiempos exactos del proceso manual; sin embargo, pasar de horas de edición manual a segundos o minutos de generación automática implica una reducción de orden de magnitud en el tiempo de elaboración y la eliminación prácticamente total de errores de transcripción.

En la evaluación frente a FURPS+, el sistema cumple los criterios establecidos. En funcionalidad, el backend satisface todos los requisitos definidos: autentica usuarios, gestiona proyectos y anexos, genera el F-VS-41 dinámicamente y permite consultas en lenguaje natural. En usabilidad, aunque aún no existe una interfaz gráfica, la API REST sigue estándares claros y la incorporación de Select AI facilita la consulta sin necesidad de SQL, mejorando la usabilidad para futuros usuarios. En fiabilidad, la arquitectura en capas y las pruebas unitarias, de integración y de carga, con una cobertura del 82 %, proporcionan robustez; los errores se manejan de forma uniforme (RFC-7807) y la persistencia en Oracle y S3 garantiza integridad y disponibilidad de datos. En rendimiento, los tiempos de respuesta (<200 ms en autenticación y ~20 s en generación de documentos) cumplen con el SLA interno y muestran que el sistema escala al menos a 20 usuarios concurrentes sin cuellos de botella. En soportabilidad y “+”, la separación Controller–Service–Repository, los principios SOLID, la documentación y el uso de Docker facilitan mantenimiento y evolución; los aspectos “+” incluyen seguridad (JWT y control de roles), escalabilidad (almacenamiento en la nube y preparación para procesamiento asíncrono) y portabilidad (despliegue en contenedores).

En líneas de trabajo futuro, se prioriza desarrollar una interfaz web y un circuito de aprobación para que docentes y administradores creen y editen proyectos, visualicen anexos y

aprueben etapas del flujo, incorporando firma digital y trazabilidad de versiones; automatizar la confección de anexos y formularios del protocolo F-VS-41 reutilizando plantillas y datos; escalar la generación de documentos mediante colas y workers o migrar a microservicios para mejorar rendimiento bajo mayores cargas y habilitar despliegues independientes; aplicar análisis de datos y minería cuando exista histórico suficiente, con el fin de identificar áreas de mayor impacto, detectar brechas, analizar cumplimiento de cronogramas y presupuestos y apoyar decisiones estratégicas; e integrar con otras plataformas universitarias (portal académico, planificación financiera, RR. HH.) para evitar duplicidades y potenciar la trazabilidad.

Como reflexión final, la automatización de la gestión de proyectos de vinculación en la Universidad Católica de Cuenca demuestra que es posible reemplazar un procedimiento manual y propenso a errores por un flujo digital que genera documentos completos y consistentes en pocos minutos. El cumplimiento de objetivos funcionales, la baja latencia y la alta cobertura de pruebas evidencian la madurez del sistema. Aun así, su carácter de backend limita el acceso para usuarios finales y abre oportunidades para enriquecer la solución con firma electrónica, aprobación en línea, generación completa de anexos y analítica avanzada. Con estas mejoras, la institución podría contar con un ecosistema de gestión de proyectos de vinculación completamente digital, trazable y basado en datos, mejorando la transparencia, reduciendo tiempos y potenciando el impacto social.

Referencias

Abdelfattah, A. S., Cordes, K. E., Medina, A., & Cerny, T. (20 de January de 2025). Semantic Dependency in Microservice Architecture: A Framework for Definition and Detection. *Semantic Dependency in Microservice Architecture: A Framework for Definition and Detection*. arXiv. doi:10.48550/arXiv.2501.11787

Achachlouei, M. A., Patil, O., Joshi, T., & Nair, V. N. (2021). Document Automation Architectures and Technologies: A Survey. doi:https://doi.org/10.48550/arXiv.2109.11603

Anwar, A. (2014). Rational Unified Process.

Arregocés, I., Camargo Martínez, N., Díaz Hernández, J., & Ariza Coronado, M. (18 de July de 2022). Integración de Scrum y RUP para el desarrollo de software de planes turísticos basado en preferencias de usuario. *RIINN, 10*. doi:10.21897/rii.2974

Bautista, A. (2014). *Traducción de consultas de Lenguaje Natural Español a SQL que involucran agrupamiento.* Master's thesis, Instituto Tecnológico de Ciudad Madero.

Bautista-Villegas, E. (25 de January de 2022). Metodologías agiles XP y Scrum, empleadas para el desarrollo de páginas web, bajo MVC, con lenguaje PHP y framework Laravel. *Rev. Amaz. Digit., 1*, e168. doi:10.55873/rad.v1i1.168

Beltrán Espinoza, E. M., & Ortega Changa, D. B. (2024). Desarrollo de un sistema inteligente de gestión documental basado en inteligencia artificial para la Aldea Niños Cristo Rey. *Desarrollo de un sistema inteligente de gestión documental basado en inteligencia artificial para la Aldea Niños Cristo Rey*.

Cabral, R., Kalinowski, M., Baldassarre, M. T., Villamizar, H., Escovedo, T., & Lopes, H. (14 de April de 2024). Investigating the Impact of SOLID Design Principles on Machine Learning

Code Understanding. *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI* (págs. 7–17). Lisbon Portugal: ACM. doi:10.1145/3644815.3644957

De Padua, G. B., & Shang, W. (September de 2017). Revisiting Exception Handling Practices with Exception Flow Analysis. *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (págs. 11–20). Shanghai: IEEE. doi:10.1109/SCAM.2017.16

Gani, D. H., Kadir, I. K., Masrek, M. N., & Rahman, A. A. (6 de May de 2024). An Evaluation of Electronic Document Management System Functionalities and Effectiveness in Malaysia., (págs. Dayangku Horiah Awang–620). doi:10.15405/epsbs.2024.05.50

Grady, R. B. (1992). *Practical software metrics for project management and process improvement.* USA: Prentice-Hall, Inc.

İRen, E., İRen, G., & Kantarci, A. (2021). Data Driven Software Testing with Selenium Apache POI Tool.

Izurieta, C., Reimanis, D., O'Donoghue, E., Liyanage, K., Manzi Muneza, A. R., Whitaker, B., & Reinhold, A. M. (8 de October de 2024). A Generalized approach to the operationalization of Software Quality Models. *PeerJ Computer Science, 10*, e2357. doi:10.7717/peerj-cs.2357

Jara Valentín, C. P. (2023). La automatización de procesos administrativos como estrategia de mejoramiento de atención al usuario en una Institución Educativa pública. *La automatización de procesos administrativos como estrategia de mejoramiento de atención al usuario en una Institución Educativa pública*.

Jatnika, H., Rifai, M. F., & Napitupulu, L. T. (23 de April de 2023). Implementation of the Rational Unified Process (Rup) Method in Designing a Web-Based Certification Scheduling Application (Citation) on Itcc Itpln. *SLJIL, 5*, 452–458. doi:10.46799/syntax-idea.v5i4.2188

Jayatilleke, N., & Silva, N. d. (24 de July de 2025). Zero-shot OCR Accuracy of Low-Resourced Languages: A Comparative Analysis on Sinhala and Tamil. *Zero-shot OCR Accuracy of Low-Resourced Languages: A Comparative Analysis on Sinhala and Tamil*. arXiv. doi:10.48550/arXiv.2507.18264

Jones, M., Bradley, J., & Sakimura, N. (May de 2015). JSON Web Token (JWT). *JSON Web Token (JWT)*, RFC7519. doi:10.17487/RFC7519

Kang, H., Liu, G., Wang, Q., Meng, L., & Liu, J. (28 de November de 2023). Theory and Application of Zero Trust Security: A Brief Survey. *Entropy, 25*, 1595. doi:10.3390/e25121595

Khande, R., Rajapurkar, S., Barde, P., Balsara, H., & Datkhile, A. (6 de July de 2023). Data Security in AWS S3 Cloud Storage. *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (págs. 1–6). Delhi: IEEE. doi:10.1109/ICCCNT56998.2023.10306922

Lages, S., Confessor, A. O., da Silva Neto, P. C., de Faria, E. B., & de Figueiredo, J. M. (2024). Metodologias Ágeis aplicadas na automatização da Gestão Acadêmica de Escolas Técnicas do Estado de Mato Grosso. *Anais do XII Workshop de Computação Aplicada em Governo Eletrônico (WCGE 2024)* (págs. 269–271). Brasília, Brasil: Sociedade Brasileira de Computação. doi:10.5753/wcge.2024.2022

Lama, A. (8 de May de 2025). Monitoring and Logging Best Practices in Java Microservices for Effective Software Development. doi:10.5281/ZENODO.15362493

Loarte Cajamarca, B. G. (2022). Desarrollo de un backend para la gestión del sistema penitenciario del Ecuador. *ConcienciaDigital, 5*, 47–66. doi:10.33262/concienciadigital.v5i3.2.2319

Macías Cáceres, V. M., & Mindiola Alejandro, J. S. (2018). Desarrollo de un sistema de gestión de contenidos web utilizando 2 metodología Scrum para Educación Continua-ESPOL. *Desarrollo de un sistema de gestión de contenidos web utilizando 2 metodología Scrum para Educación Continua-ESPOL*.

Mohammadjafari, A., Maida, A. S., & Gottumukkala, R. (4 de February de 2025). From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems. *From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems*. arXiv. doi:10.48550/arXiv.2410.01066

Necula, S. (April de 2024). Exploring The Model-View-Controller (MVC) Architecture: A Broad Analysis of Market and Technological Applications. *Exploring The Model-View-Controller (MVC) Architecture: A Broad Analysis of Market and Technological Applications*. doi:10.20944/preprints202404.1860.v1

Nguyen, T. T., Jatowt, A., Coustaty, M., & Doucet, A. (31 de July de 2022). Survey of Post-OCR Processing Approaches. *ACM Comput. Surv., 54*, 1–37. doi:10.1145/3453476

Oluwafemi Oloruntoba. (28 de February de 2025). AI-Driven autonomous database management: Self-tuning, predictive query optimization, and intelligent indexing in enterprise it environments. *World J. Adv. Res. Rev., 25*, 1558–1580. doi:10.30574/wjarr.2025.25.2.0534

Parejo, J. A., Cabanillas, C., Estrada-Torres, B., García, J. M., Müller, C., & Resinas, M. (2023). Suite de pruebas auto-evaluable como examen de laboratorio: una aproximación pragmática con Spring Boot y GitHub.

Pcs, R., Nachiyappan, S., Ramakrishna, V., Senthil, R., Anwer, S., & Rk, K. (2021). Hybrid Model Using Scrum Methodology for Softwar Development System. *10*.

Roque, I. I. (2021). La Gestión Documental del Sistema de Gestión de Calidad. *19*.

Ruales Aldás, J. I. (2024). *Análisis, diseño y desarrollo de un sistema de automatización de la gestión documental en empresas a través de una interfaz gráfica intuitiva.* Master's thesis, Universidad Politécnica Salesiana (Quito).

Sánchez Villarreal, C. A. (2023). *Sistema informático de gestión académica bajo la metodología Scrum para mejorar la rentabilidad de la Institución Educativa Privada María Auxiliadora, Lima-2023.* Master's thesis, Universidad Nacional del Callao.

Sassa, A. C., Almeida, I. A., Pereira, T. N., & Oliveira, M. S. (2023). Scrum: A Systematic Literature Review. *IJACSA, 14*. doi:10.14569/IJACSA.2023.0140420

Schwaber, K., & Sutherland, J. (November de 2020). The Scrum Guide: The Definitive Guide to Scrum. *The Scrum Guide: The Definitive Guide to Scrum*.

Silva-Peñafiel, G. E., Morales-Guamán, K. P., Chalar-Suárez, J. P., & Rodríguez-Lirio, A. F. (2021). Implementación de un sistema mediante la metodología Scrum del proceso de Titulación en la Universidad Técnica de Cotopaxi Extensión La Maná. *Polo del Conocimiento, 6*, 188–215.

Suárez, Y. D., & Vázquez, T. O. (2021). Sistema de gestión documental para la Maestría en Gestión de Información de la UH. *15*.

Suhartono, D. T., & Indriyanti, A. D. (2025). Software Quality Evaluation Of MV5PAS Airport Authority Region III Using FURPS Model. 14–27. doi:https://doi.org/10.26740/jeisbi.v6i1.64722

Sun, P., & Kim, D.-K. (12 de May de 2022). Analyzing Impact of Dependency Injection on Software Maintainability. *Analyzing Impact of Dependency Injection on Software Maintainability*. arXiv. doi:10.48550/arXiv.2205.06381

Yungan Gualli, A. F., Morales Alarcón, C. H., Delgado Altamirano, J. E., & Espinoza Tinoco, L. M. (30 de December de 2019). Modelo FURPS para el análisis del rendimiento de frameworks JSF. *3C TIC, 8*, 65–83. doi:10.17993/3ctic.2019.84.65-83

Zalimben, S. (2022). Una pequeña guía de Scrum.

Anexos

Anexo A

Tabla33

Casos de uso.

Caso de uso #	Nombre	Actor participante	Flujo de eventos	Condición de entrada	Condición de salida	Escenarios
CU01	Iniciar sesión	Usuario (cliente que consume la API)	1. El Usuario envía una petición POS /api/auth/login con { "email", "password", "nombres" }. 2. El backend valida el formato de los datos. 3. Busca el email en la base de datos. 4. Verifica la contraseña mediante BCrypt. 5. Si la verificación es exitosa, genera un token JWT y responde con HTTP 200 y { "token", "usuario", "correo", "nombre "}. 6. El Usuario almacena el token para futuras peticiones autenticadas.	• El Usuario dispone de credenciales registradas. • No hay token válido en la petición inicial.	• Toke JWT emitido (éxito). • Error HTTP 401 con mensaje “Credenciales inválidas” (fallo).	A. Autenticación exitosa B. Credenciales incorrectas

CU02	Registrar usuario	Usuario (cliente que consume la API)	1. El Usuario envía una petición POS /api/auth/registro con un JSON con: { "nombres","email", "password" } 2. Comprueba si el correo ya está registrado. 3. Si el correo es único, encripta la contraseña con BCrypt. 4. Crea el nuevo registro en la base de datos con rol USER por defecto. 5. Devuelve HTTP 201 Created un mensaje "Usuario registrado exitosamente"n.	• El correo electrónico suministrado no debe existir en el sistema. • Todos los campos obligatorios vienen en el payload.	• Usuario persistido correctamente (escenario éxito). • Error HTTP adecuado (409 o 400) con detalle del problema (escenario fallo).	A. Registro exitoso B. Correo electrónico ya registrado

CU03	Recuperar contraseña	Usuario (cliente que consume la API)	**Fas A** – Solicitud del código 1. El Usuario envía POST /api/auth/forgot-password con { "email" }. 2. El backend verifica que el correo exista. 3. Genera un código de verificación (6 dígitos) con caducidad, lo guarda en DB y lo envía por correo al Usuario. 4. Devuelve HTTP 200 con mensaje “Código enviado”. **Fase B** Cambio de contraseña 5. El Usuario recibe el código e invoca POST /api/auth/reset-password con { "email", "code", "newPassword" }. 6. El backend comprueba que el código coincide y no ha expirado. 7. Encripta la nueva contraseña (BCrypt) y actualiza el registro. 8. Borra el código usado y responde HTTP 200 con “Contraseña actualizada”.	• El email debe pertenecer a un usuario registrado. • El código sólo puede usarse una vez antes de expirar.	• Contraseña cambiada exitosamente (escenario éxito). • Mensaje de error apropiado (escenario fallo).	A. Recuperación exitosa B. Correo no registrado

CU04	Gestionar proyecto CRUD	Usuario (cliente autenticado que consume la API)	1. El Usuario incluye su toke JWT en la cabecera Authorization: Bearer 2. Envía una petición HTTP según la operación deseada: • Crear → POST /api/datos-proyecto con JSON de proyecto. •Consultar → GET /api/projects /detalle/{id} o GET /api/proyectos/mis-proyectos para listado. • Actualizar → PUT /api/datos-proyecto/{id }con campos a modificar. • Eliminar → DELETE /api/datos -proyecto/{id} (borrado lógico). 3. El backend valida la autenticidad del token y la estructura de los datos. 4. Ejecuta la operación solicitada utilizando la capa Service + Repository. 5. Devuelve una respuesta HTTP adecuada (201, 200, 204, 404, 400, 403) con cuerpo JSON cuando corresponda.	• Usuario autenticado (token válido). • Para Crear/Actualizar, el campo obligatorio que se debe enviar es el tirulo del proyecto. • Para Actualizar/Eliminar/Consultar, el id debe existir.	• Proyecto creado/consultado/actualizado/eliminado con éxito. • Mensaje de error (validación, inexistencia, permiso).	A. Creación exitosa B. Actualización exitosa C. Eliminación exitosa D. Proyecto no encontrado E. Validación de datos fallida

CU05	Generar documento Word	Usuario (cliente autenticado)	1. El Usuario incluye su toke JWT y envía POST /api/documento/generar/{id}. 2. El backend busca el proyecto por {id} y verifica la propiedad/autorización. 3. Revisa que todos los campos del proyecto estén completos. 4. Si la validación es exitosa, ensambla una plantilla .docx mediante Apache POI + altChunk, reemplazando marcadores con los datos del proyecto. 5. Guarda el archivo generado en AWS S3 y registra la metadata en la BD. 6. Devuelve HTTP 201 Created con con un mesaje ("Documento generado exitosamente"). 7. (Relación <<extend>>) El Usuario puede ejecutar CU06 Descargar documento Word usando la URL o endpoint dedicado.	• Toke JWT válido. • Proyecto existente y autorizado. • Todos los campos requeridos del proyecto deben estar llenos (si falta alguno, se aborta la generación).	• Documento Word generado y almacenado. (escenario éxito) • Respuesta de error adecuada (escenario fallo).	A. Generación exitosa B. Campos obligatorios faltantes C. Proyecto no encontrado

CU06	Descargar documento Word	Usuario (cliente autenticado)	1. El Usuario incluye su toke JWT en la cabecera Authorization y realiza GET /api/datos-proyecto/descargat/{id}. 2. El backend valida el token y confirma que el proyecto {id} pertenece al Usuario. 3. Verifica que el documento Word haya sido generado previamente (precondición proveniente de CU05). 4. Transmite el archivo directamente, estableciendo Content-Disposition: attachment. 5. Devuelve HTTP 200 OK con el flujo binario del .docx. 6. El Usuario descarga y almacena el documento en su dispositivo.	• Toke JWT válido. • Documento Word existente y previamente generado para el proyecto. • Usuario autorizado sobre ese proyecto.	• Documento entregado al cliente (escenario éxito) o • Mensaje de error (escenario fallo).	A. Descarga exitosa B. Documento no encontrado C. Acceso no autorizado

CU07	Gestionar PDF de proyecto aprobado	• Usuario Creador (propietario del proyecto) • Usuario Invitado (participante con permisos sólo de lectura)	1. El Usuario incluye su toke JWT y accede a la ruta /api/proyectos/{id}documento-firmado. 2. El backend valida el token. 3. Verifica que el proyecto {id} exista. 4. Dependiendo del método HTTP: • Creador – POST/PUT: sube o reemplaza el PDF firmado (multipart/form-data). – DELETE: elimina la versión existente. • Creador o Invitado – GET: descarga el PDF. 5. El archivo se almacena en AWS S3; la metadata se actualiza en BD. 6. El backend devuelve el código HTTP apropiado (201, 200, 204, 403, 404, 409).	• Proyecto en estado APPROVED. • Toke JWT válido. • Para subir/eliminar: el Usuario debe ser Creador del proyecto.	• PDF creado/actualizado/eliminado o descargado correctamente o • Error HTTP con descripción.	A. Subida/actualización exitosa (Creador) B. Descarga exitosa (Creador o Invitado) C. Proyecto no aprobado D. Acceso no autorizado

CU08	Gestionar participantes del proyecto	Usuario Creador del proyecto	1. El Creador envía su toke JWT en la cabecera Authorization. 2. Para agregar un participante: • POST /api/projects/{id}/participantes con JSON { "email":}. 3. Para eliminar un participante: • DELETE /api/projects/{id}/participants/{participanteId}. 4. El backend valida: • Que el proyecto {id} exista y pertenezca al Creador. • Que el email corresponda a un usuario registrado (al agregar). • Que el participante exista en la lista (al eliminar). 5. Agrega o elimina el registro en la tabla de participaciones. 6. Devuelve el código HTTP adecuado (201, 204, 404, 409, 403).	• Toke JWT válido. • Usuario es el Creador del proyecto. • Proyecto existe.	• Participante agregado o eliminado correctamente o • Respuesta de error correspondiente.	A. Adición exitosa B. Eliminación exitosa C. Usuario ya invitado D. Participante no encontrado E. Acceso no autorizado

CU09	Gestionar anexos del proyecto	Usuario Creador del proyecto Usuario – Invitado (participante)	1. El Usuario envía su toke JWT en la cabecera Authorization. 2. Según la operación deseada utiliza el endpoint /api/proyectos/{projectId}/anexos : • POST (multipart) → subir anexo. • GET → listar anexos del proyecto. • GET /{attachmentId} → descargar un anexo concreto. • PUT /{attachmentId} (multipart) → reemplazar archivo. • DELETE /{attachmentId} → eliminar anexo. 3. El backend verifica: • Proyecto {projectId} existe y el Usuario está asociado (creador o invitado). 4. Opera sobre AWS S3 y actualiza la tabla anexos (URL, nombre, tipo, ownerId, timestamp). 5. Devuelve el HTTP apropiado (201, 200, 204, 400, 404, 401).	• Toke JWT válido. • Usuario pertenece al proyecto (como creador o invitado).	• Anexo creado, consultado, actualizado o eliminado correctamente o • Respuesta de error correspondiente.	A. Subida exitosa B. Descarga exitosa C. Eliminación exitosa D. Anexo no encontrado E. Acceso no autorizado

CU10	Consultar datos del proyecto en lenguaje natural	• Usuario Creador del proyecto • Usuario Invitado (participante)	1. El Usuario envía su toke JWT y realiza POST /api/ai/{projectId}/ai-query con un JSON: { "prompt": "¿Cuánto es el presupuesto aprobado?" } 2. El backend valida el token y confirma que el Usuario pertenece al proyecto (creador o invitado). 3. Construye la llamada a Oracle Select AI: • Establece el perfil DBMS_CLOUD_AI.SET_PROFILE con la clave LLM. • Llama a SELECT ai showsql :prompt FROM dual para obtener la traducción SQL y los datos. 4. Ejecuta la consulta SQL sobre la misma base de datos del proyecto. 5. Devuelve HTTP 200 OK con un JSON de resultados6. El Usuario muestra la información en su cliente.	• Toke JWT válido. • Usuario pertenece al proyecto. • Proyecto existe. • Prompt de texto no vacío.	• Resultados devueltos correctamente o • Mensaje de error correspondiente.	A. Consulta exitosa B. Sin resultados C. Acceso no autorizado

CU11	Listar proyectos creados	Usuario – Creador	1. El Usuario envía su toke JWT y realiza GET /api/proyectos/mis-proyectos. 2. El backend valida el token y extrae el userId del subject del JWT. 3. Consulta la tabla projects filtrando creator_id = :userId. 4. Devuelve HTTP 200 OK con un arreglo JSON de proyectos (id, título, estado, fecha de creación, etc.).	• Toke JWT válido. • El Usuario tiene al menos una sesión activa.	• Lista de proyectos creada exitosamente o • Error HTTP correspondiente.	A. Lista obtenida con proyectos B. Sin proyectos registrados C. Acceso no autorizado
CU12	Listar proyectos agregados	Usuario – Invitado (participante) *o* Usuario – Creador (cuando también figure como invitado en proyectos ajenos)	1. El Usuario envía su toke JWT e invoca GET /api/invitaciones/mis-proyectos. 2. El backend valida el token y extrae el userId del subject. 3. Devuelve HTTP 200 OK con un arreglo JSON que resume cada proyecto (id, título, estado, creador, fecha de invitación, etc.).	• Toke JWT válido. • El Usuario está asociado como participante a ≥ 0 proyectos.	• Lista devuelta correctamente o • Error HTTP correspondiente.	A. Lista obtenida con proyectos B. Sin proyectos agregados C. Acceso no autorizado

Nota. Elaboración propia.

Anexo B

Tabla 34

Especificación de escenarios.

Caso de uso	Escenario	Suposiciones	Flujo de eventos	Resultado
CU01	A. Autenticación exitosa	• Usuario registrado y activo.	1. POST /login con credenciales válidas.2. Backend responde HTTP 200 + JWT, nombre, correo, ROL.	Cliente recibe y guarda el token.Autenticación confirmada.
CU01	B. Credenciales incorrectas	• Email no existe o contraseña incorrecta.	1. POST /login con credenciales inválidas.2. Backend responde HTTP 401 + mensaje de error.	Cliente no recibe token.Acceso denegado.
CU02	A. Registro exitoso	• El correo no existe.• La contraseña cumple las reglas de seguridad.	1. POST /register con datos válidos.2. Backend crea el usuario.3. Devuelve HTTP 201 + { "userId", "message":"Usuario registrado exitosamente"}.	Usuario nuevo registrado; en la BD aparece el registro con rol CREADOR y contraseña encriptada.
CU02	B. Correo electrónico ya registrado	• El correo enviado coincide con uno existente.	1. POST /register con email duplicado.2. Backend detecta duplicidad.3. Devuelve HTTP 409 Conflict con { "error":El correo ya esta en uso" }.	Usuario no creado; el cliente recibe mensaje de conflicto y debe indicar otro correo.
CU03	A. Recuperación exitosa	• Email existe.• Código ingresado correcto y vigente.	1. POST /forgot-password (pasos 1-4).2. POST /reset-password (pasos 5-8).	Contraseña actualizada; el Usuario podrá iniciar sesión con la nueva clave.

CU03	B. Correo no registrado	• El email no existe en la BD.	1. POST /forgot-password con correo inexistente.2. Backend devuelve HTTP 404 Not Found con { "error":"Email no registrado" }.	No se envía código; Usuario informado del error.
CU04	A. Creación exitosa	• Datos requeridos completos • id todavía no existe.	1. POST con JSON válido.2. Backend persiste y devuelve HTTP 201 con { "projectId": tituloProyecto,etc.}.	Proyecto registrado.
CU04	B. Actualización exitosa	• Proyecto existe y pertenece al Usuario.• Nuevo JSON cumple validaciones.	1. PUT con cambios.2. Backend actualiza y responde HTTP 200 con objeto actualizado.	Datos del proyecto modificados.
CU04	C. Eliminación exitosa	• Proyecto existe	1. DELETE 2. Backend marca registro como *deleted* y responde HTTP 204 (sin cuerpo).	Proyecto ya no aparece en listados activos.
CU04	D. Proyecto no encontrado	• id no corresponde a ningún registro.	1. GET/PUT/DELETE /datos-rpoyecto/{id} con ID inexistente. 2. Backend devuelve HTTP 404 Not Found con { "error":"Proyecto no encontrado" }.	Cliente informado; sin cambios en BD.
CU04	E. Validación de datos fallida	• JSON mal estrucuturado •Formato incorrecto.	1. POST o PUT con JSON inválido.2. Backend devuelve HTTP 400 Bad Request con detalles.	Proyecto no creado ni modificado.
CU05	A. Generación exitosa	• Proyecto existe.• Todos los campos requeridos completos.	Pasos 1-6 del flujo principal.	Backend responde HTTP 201 y un mensaje "Documento generado exitosamente"
CU05	B. Campos obligatorios faltantes	• Faltan uno o más campos críticos (p.ej. presupuesto).	1. POST /documentogenerar/. 2. Validación detecta incompletitud.3. Backend responde HTTP 422 Unprocessable Entity con { "error":"Required fields missing" }.	Documento no generado; Usuario informado del motivo.

CU05	C. Proyecto no encontrado	• ID inexistente o no pertenece al Usuario.	1. POST con {id} inexistente.2. Backend devuelve HTTP 404 Not Found con { "error":"Proyecto no econtrado" }.	Sin cambios en BD ni en S3.
CU06	A. Descarga exitosa	• Documento existe en S3.• Usuario es propietario.• Token válido.	1. GET /datos-proyecto/descargar.2 . Backend localiza documento y responde HTTP 200 con archivo.	Cliente recibe el .docx listo para guardar.
CU06	B. Documento no encontrado	• El Word no ha sido generado (CU05 no ejecutado) o ha sido eliminado.	1. GET /documento/generar/. 2. Backend no localiza archivo.3. Devuelve HTTP 404 Not Found con { "error":"Documento no econtrado" }.	Cliente informado; sin descarga.
CU06	C. Acceso no autorizado	• El proyecto pertenece a otro usuario o token inválido/ausente.	1. GET con token inválido o proyecto ajeno.2. Backend devuelve HTTP 403 Forbidden o 401 Unauthorized.	Archivo no entregado; seguridad garantizada.
CU07	A. Subida/actualización exitosa (Creador)	• Proyecto en APPROVED.• Usuario es Creador.• PDF válido (< 20 MB).	1. POST proyectos/{id}documento-firmado con archivo.2. Backend guarda en S3, actualiza BD.3. Responde HTTP 201 Created (o 200) con { "pdfUrl": … }.	PDF disponible para descargas; metadatos actualizados.
CU07	B. Descarga exitosa (Creador o Invitado)	• PDF previamente subido.• Usuario autenticado (rol cualquiera).	1. GET proyectos/{id}/documento-firmado/descargar. 2. Backend obtiene URL firmada o stream.3. Devuelve HTTP 200 OK con archivo.	Cliente recibe el PDF firmado.
CU07	D. Acceso no autorizado	• Usuario Invitado intenta POST/PUT/DELETE.• O token inválido.	1. Invitado hace POST /proyectos/{id}/documento-firmado. 2. Backend devuelve HTTP 403 Forbidden (o 401).	Sin cambios; seguridad mantenida.
CU08	A. Adición exitosa	• Email pertenece a un usuario existente.• Aún no está en el proyecto.	1. POST /api/proyectos/{idProyecto}/documento-firmado/invitaciones/{idUsuario}. 2. Backend crea vínculo.3. Devuelve	Nuevo participante agregado.

			HTTP 201 con mensaje "Usuaio invitado"	
CU08	B. Eliminación exitosa	• participante está asociado al proyecto.	1. DELETE /participants/{participantId}.2. Backend borra registro.3. Devuelve HTTP 204 No Content.	Participante removido.
CU08	C. Usuario ya invitado	• Email ya existe en la lista.	1. POST con email duplicado.2. Backend responde HTTP 409 Conflict con { "error":"Usuario ya invitado"}.	No se crea duplicado.
CU08	D. Participante no encontrado	• participanted no corresponde a ningún participante del proyecto.	1. DELETE con ID inexistente.2. Backend devuelve HTTP 404 Not Found.	Sin cambios en la BD.
CU08	E. Acceso no autorizado	• Token inválido o Usuario no es Creador.	1. POST/DELETE con token inválido o rol incorrecto.2. Backend responde HTTP 401 o 403.	Seguridad mantenida; sin modificaciones.
CU09	A. Subida exitosa	• Archivo válido (tamaño/tipo).• Usuario asociado al proyecto.	1. POST /documento-firmado con archivo.2. Backend guarda en S3 y BD.3. Devuelve HTTP 201 Created con { "attachmentId":…, "url":… }.	Anexo disponible; listado actualizado.
CU09	B. Descarga exitosa	• anexoId existe.• Usuario asociado.	1. GET {idProyecto}/document-firmado/metadata/{attachmentId}.2. Backend responde HTTP 200 OK con mensaje "Anexo eliminado".	Cliente descarga el archivo.
CU09	C. Eliminación exitosa	• anexoId• Usuario asociado.	1. DELETE /attachments/{attachmentId}.2. Backend borra en S3 y BD.3. Devuelve HTTP 204 No Content.	Anexo eliminado de forma permanente.

CU09	D. Anexo no encontrado	• anexoId no existe.	1. Cualquier método (GET/PUT/DELETE) con ID inválido.2. Backend devuelve HTTP 404 Not Found con { "error":"Anexo no eocntrado" }.	Sin cambios en BD/S3.
CU09	E. Acceso no autorizado	• Usuario no pertenece al proyecto o token inválido.	1. Operación con token inválido o usuario ajeno.2. Backend devuelve HTTP 401 Unauthorized o 403 Forbidden.	Seguridad garantizada; operación abortada.
CU10	A. Consulta exitosa	• Prompt bien formado.• Usuario pertenece al proyecto.	1. POST /ai-query con prompt.2. Backend genera SQL, ejecuta, devuelve HTTP 200 con filas.	Cliente recibe los datos solicitados.
CU10	B. Sin resultados	• Prompt correcto, pero la consulta devuelve 0 filas.	1. POST con prompt.2. SQL ejecutada sin filas.3. Backend responde HTTP 200 con "rows":[].	Usuario informado; sin datos que mostrar.
CU11	A. Lista con proyectos	• Usuario ha creado ≥ 1 proyecto.	1. GET /proyectos/mis-proyectos devuelve HTTP 200 con [{ "id":1, "tituloProyecto":, "ect."},].	Cliente muestra los proyectos.
CU11	B. Sin proyectos registrados	• Usuario no ha creado proyectos.	1. GET /proyectos/mis-proyectos. 2. Consulta devuelve 0 filas.3. Backend responde HTTP 200 con [].	Cliente informa “No hay proyectos”.
CU11	C. Acceso no autorizado	• Token ausente o inválido.	1. GET sin token o token expirado.2. Backend devuelve HTTP 401 Unauthorized.	Lista no entregada; seguridad mantenida.
CU12	A. Lista con proyectos	• Usuario figura como invitado en ≥ 1 proyecto.	1. GET /invitaciones/mis-proyectos. 2. Backend devuelve HTTP 200 con [{ "id":3, "tituloProyectp": Test, "etc".].	Cliente muestra la lista de proyectos donde participa.
CU12	B. Sin proyectos agregados	• El usuario aún no ha sido invitado a ningún proyecto.	1. GETinvitaciones/mis-proyectos. 2. Consulta devuelve 0 filas.3. Backend responde HTTP 200 con [].	Cliente informa “No participas en ningún proyecto”.
CU12	C. Acceso no autorizado	• Token ausente, inválido o expirado.	1. GET sin token o token inválido.2. Backend devuelve HTTP 401 Unauthorized.	Lista no entregada; seguridad garantizada.

Nota. Elaboración propia.

Universidad Católica de Cuenca

AUTORIZACIÓN DE PUBLICACIÓN EN EL REPOSITORIO INSTITUCIONAL

Kevin Oswaldo Guaicha Vásquez portador(a) de la cédula de ciudadanía Nº **0105886949**. En calidad de autor/a y titular de los derechos patrimoniales del trabajo de titulación **"Desarrollo del backend para un sistema de automatización de la gestión de proyectos de vinculación en la Universidad Católica de Cuenca utilizando metodologías ágiles e inteligencia artificial"** de conformidad a lo establecido en el artículo 114 Código Orgánico de la Economía Social de los Conocimientos, Creatividad e Innovación, reconozco a favor de la Universidad Católica de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos y no comerciales. Autorizo además a la Universidad Católica de Cuenca**,** para que realice la publicación de éste trabajo de titulación en el Repositorio Institucional de conformidad a lo dispuesto en el artículo 144 de la Ley Orgánica de Educación Superior.

Cuenca, **28 de septiembre de 2024**

F: .....

Kevin Oswaldo Guaicha Vásquez

C.I. **0105886949**