Universidad Zaragoza

1542

Trabajo Fin de Grado

Integración de librerías CCSV para optimización y seguridad documental en proyectos de AMD Aragón en empresa

Integration of CCSV libraries for document optimization and security management in AMD projects Aragón in an enterprise

Autor

Clara Cerdán Torrubias

Director

Jesús Brosed Escario

Hiberus Tecnologías de la Información, S.L.

Ponente

Carlos Bobed Lisbona

Departamento de Informática e Ingeniería de Sistemas

Universidad de Zaragoza

ESCUELA DE INGENIERÍA Y ARQUITECTURA

ENERO - 2025

AGRADECIMIENTOS

Quiero expresar mi profunda gratitud a mi ponente, por su apoyo constante, su orientación llena de amabilidad y su dedicación como un docente ejemplar.

A mis mentores en la empresa, por su paciencia infinita y por invertir su tiempo en transmitirme con generosidad sus conocimientos y experiencia sobre el proyecto.

A mis padres y a mi hermana, porque sin ellos nunca habría tenido la oportunidad de estudiar aquello que amo ni de convertirme en la persona que soy hoy.

Al resto de mi familia, por su apoyo incondicional y por estar a mi lado en cada paso de este viaje, dándome fuerzas incluso cuando no creía en mí.

A mi pareja, por brindarme esperanza, por ser mi mayor fuente de ánimo y por apoyarme siempre a lo largo de este camino.

Y a mis amigos, por ser mi refugio de cordura y por estar a mi lado siempre que los he necesitado.

RESUMEN

En el contexto empresarial actual, donde la agilidad, eficiencia y capacidad de adaptación son esenciales, las empresas privadas deben tener una infraestructura tecnológica robusta que permita mejorar continuamente sus operaciones. En este proyecto, el objetivo ha sido integrar las librerías CCSV (Servicio de almacenamiento y verificación de documentos electrónicos) proporcionadas por el Sistema de Administración Electrónica (SAE) del Gobierno de Aragón en los procesos existentes de la empresa, lo que ha permitido mejorar las interacciones con la administración pública y optimizar la gestión de documentación electrónica.

El desarrollo de este proyecto ha requerido superar varios desafíos importantes que dificultaban el funcionamiento adecuado de los sistemas anteriores. Entre los principales obstáculos, destaca la falta de centralización en la gestión de documentos, lo que obligaba a realizar múltiples pasos y ralentizaba las operaciones y la obsolescencia de funciones clave, cuyas actualizaciones requerían una reingeniería compleja. Además, la ausencia de documentación completa complicaba tanto el mantenimiento del sistema como la incorporación de nuevos desarrolladores, lo que generaba dependencia de personas con experiencia previa en el sistema.

A lo largo de este documento, se detallan los pasos seguidos para realizar un análisis exhaustivo del sistema actual, adaptar las librerías CCSV al entorno tecnológico de la empresa e implementar una solución que optimice la gestión de la documentación electrónica. Se explica cómo se abordaron los problemas técnicos encontrados durante la integración, como la necesidad de unificar métodos de gestión de documentos y actualizar funciones obsoletas. El documento también incluye una revisión detallada de las herramientas y la arquitectura adoptada, un modelo de datos ajustado a las necesidades de integración y las pruebas realizadas para asegurar la fiabilidad del sistema. Finalmente, se incluyen las conclusiones finales y reflexiones personales sobre el proyecto.

Índice

Lista de Figuras

Listados

Capítulo 1

Introducción y objetivos

Este Trabajo Fin de Grado (TFG) aborda los retos de la integración del Sistema de Verificación Documental dentro de la administracion pública, particularmente en el Gobierno de Aragón, y se centra en la gestión del Código Seguro de Verificación (CSV). El objetivo es crear una Interfaz de Programación de Aplicaciones (API) unificada que pueda estandarizar y simplificar el proceso de integración de sistemas, lo que dará como resultado una interoperabilidad mejorada y un proceso de desarrollo más sencillo. Este proyecto no solo busca resolver problemáticas técnicas, sino también aumentar la consistencia en los sistemas relacionados con el CSV.

1.1. Contexto

En el desarrollo de soluciones tecnológicas es fundamental comprender el contexto en el que se encuentran los sistemas y procesos existentes, así como los desafíos y necesidades de integración que deben ser abordados. Este apartado presenta un marco de referencia que contextualiza el trabajo realizado, destacando la colaboración con la empresa Hiberus y la explicación del Código Seguro de Verificación (CSV), sobre el cual se centra el desarrollo de este Trabajo de Fin de Grado (TFG).

1.1.1. Colaboración con Hiberus

El proyecto se ha realizado en colaboración con Hiberus, una empresa tecnológica líder en España con la que he tenido la oportunidad de trabajar durante el desarrollo del proyecto. Debido a la relevancia de la empresa en el ámbito tecnológico y su papel clave en el enfoque de este TFG, resulta pertinente contextualizar quiénes son y cuáles son sus principales áreas de actuación.

Hiberus es una empresa de servicios tecnológicos y consultoría, especializada en soluciones innovadoras en áreas como transformación digital, desarrollo de software, integración de sistemas y gestión de datos. Fundada en Zaragoza, cuenta con presencia

nacional e internacional y trabaja con sectores variados como la administración pública, retail, turismo y banca, entre otros. Su enfoque está centrado en proporcionar soluciones personalizadas y escalables a empresas y organismos para mejorar su eficiencia, seguridad y competitividad.

La colaboración de Hiberus con administraciones públicas, como el Gobierno de Aragón, es especialmente destacable en proyectos relacionados con la optimización de sistemas de documentación y la implementación de soluciones basadas en el Códigos Seguro de Verificación. Esto refleja su compromiso por fortalecer la seguridad y la eficiencia en los procesos administrativos mediante el uso de tecnologías innovadoras.

Figura 1.1: Oficinas de Hiberus en Zaragoza

1.1.2. ¿Qué es el Código Seguro de Verificación?

El Código Seguro de Verificación (CSV) es un código único que acompaña a cada documento o expediente generado por el Gobierno de Aragón, el cual permite la verificación de su autenticidad y contenido. Este código es esencial para garantizar la integridad y autenticidad de los documentos electrónicos emitidos por las administraciones públicas. Normalmente, este código aparece impreso en los márgenes de los documentos.

El uso de este código permite a los usuarios verificar la validez de documentos electrónicos administrativos, comprobar sus firmas electrónicas, y descargar o consultar los documentos a través del servicio de verificación. De esta manera, se asegura que los documentos no hayan sido alterados y que las firmas asociadas sean legítimas.

1.2. Motivación del proyecto

La integración de la librería CCSV en los proyectos de Hiberus que colaboran con el Gobierno de Aragón ha demostrado ser un proceso difícil de estandarizar y lleno de obstáculos. Aunque esta herramienta es clave para desarrollar funcionalidades de verificación documental, la realidad es que su implementación varía ampliamente entre proyectos, lo que genera problemas tanto a nivel técnico como organizativo.

El principal reto es la escasez de documentación clara y detallada. Aunque el Gobierno de Aragón proporciona especificaciones y guías técnicas, estas son insuficientes para abordar todas las situaciones que surgen durante la integración. Los errores reportados suelen carecer de información suficiente para identificar sus causas, y la restricción en el acceso al código interno de las funciones impide que los desarrolladores puedan entender su funcionamiento en profundidad.

Por otro lado, la falta de un enfoque unificado entre proyectos agrava la situación. Cada equipo aborda la implementación de forma independiente, desarrollando soluciones a medida que no comparten un estándar común. Esta heterogeneidad resulta en disparidades significativas en la forma en que se manejan errores, validaciones y mejoras. En lugar de centralizar los esfuerzos, cada nuevo cambio o actualización debe repetirse en todos los proyectos afectados, duplicando el trabajo y encareciendo el mantenimiento.

Otro aspecto importante es el impacto que esta fragmentación tiene en los tiempos de desarrollo. Al no contar con una guía confiable ni con procesos definidos, los desarrolladores invierten un tiempo considerable en encontrar soluciones a través de ensayo y error. Esto no solo ralentiza los proyectos, sino que también incrementa la probabilidad de errores en producción, afectando la calidad general de las entregas.

En definitiva, los desafíos asociados con la integración del CCSV no solo ralentizan el desarrollo, sino que también encarecen los proyectos a largo plazo. Resolver estas dificultades es clave para garantizar una implementación más eficiente, reducir los costes de mantenimiento y ofrecer soluciones coherentes en todos los proyectos de Hiberus que desarrollan para el Gobierno de Aragón.

1.3. Alcance y objetivos

Como objetivo general, se plantea diseñar e implementar una API para facilitar la integración de sistemas de verificación documental basados en el Código Seguro de Verificación en administraciones públicas, con el propósito de simplificar procesos, aumentar la eficiencia y garantizar la consistencia en su adopción. La solución

propuesta estará diseñada para ser utilizada en el entorno de producción de la empresa, abarcando la mayoría de los proyectos que necesiten integrar la librería CCSV.

Como objetivos específicos se acaba planteando la siguiente lista:

- **Optimización de procesos**: identificar y resolver las ineficiencias actuales en la integración de librerías CCSV, enfocándose en simplificar su implementación y reducir los tiempos de desarrollo.
- **Mejora de la documentación**: crear y estandarizar una guía técnica completa que facilite a los desarrolladores la integración del CSV en diferentes proyectos, eliminando ambigüedades y reduciendo la curva de aprendizaje.
- **Homogeneización de librerías**: proporcionar una solución centralizada que unifique los servicios utilizados en proyectos relacionados con el CSV, permitiendo una implementación coherente en todos los entornos.
- **Estandarización en el manejo de errores**: definir y aplicar un enfoque uniforme para la gestión de errores y validaciones dentro de la API, evitando disparidades entre los proyectos y mejorando la experiencia del usuario final.
- **Evaluación comparativa**: realizar un análisis exhaustivo que contraste el sistema actual con el nuevo, evaluando métricas de eficiencia, usabilidad y coste para demostrar los beneficios de la solución propuesta.

1.4. Herramientas y tecnología de trabajo

Es importante destacar que el entorno y las herramientas utilizadas en este proyecto no han sido seleccionadas de manera personal, sino que han sido impuestas por las directrices y estándares de la empresa, Hiberus, y las necesidades del Gobierno de Aragón. El ecosistema tecnológico ya estaba definido, lo que ha requerido adaptar la configuración y desarrollo del proyecto a las bibliotecas y herramientas existentes.

Para el backend (todo el proyecto está desarrollado aquí), se ha utilizado **Eclipse IDE** [1] como entorno de desarrollo integrado (IDE), con **Java** [2] como lenguaje principal, ya que ambas herramientas son estándar en la empresa. Se utiliza Java para aprovechar sus características avanzadas y su amplio soporte en la industria de desarrollo de software. **Spring Boot** [3] ha sido el framework seleccionado para facilitar la creación de servicios backend empresariales robustos y escalables, mientras que **Project Lombok** [4] se emplea para reducir el código repetitivo, mejorando la

productividad al generar automáticamente constructores, getters y setters. **Spring Tools** [5] también se ha integrado en el IDE para facilitar el desarrollo con Spring, junto con **Spring Initializr** [6], que permite configurar rápidamente los proyectos Spring.

En cuanto a la gestión de dependencias y automatización de la construcción, se han utilizado **Apache Maven** [7] y **Apache Ant** [8], herramientas que permiten gestionar dependencias y automatizar tareas de compilación y despliegue. Además, se ha integrado **JUnit** [9] para pruebas unitarias, lo que automatiza el proceso de prueba, permitiendo identificar y corregir problemas durante el desarrollo.

Para la documentación y prueba de servicios API, se ha empleado **Swagger UI** (OpenAPI) [10], lo que facilita la validación y prueba de los servicios creados de manera eficiente.

En cuanto a la gestión de control de versiones, **SourceTree** [11] ha sido la herramienta principal utilizada, la cual simplifica la gestión de repositorios Git a través de una interfaz gráfica intuitiva, facilitando la colaboración entre equipos de desarrollo. Los repositorios se gestionan en **GitLab** [12], plataforma donde se almacena el código fuente de las aplicaciones y se gestionan las versiones.

En cuanto a las herramientas de productividad y colaboración, se han utilizado varias del ecosistema de Microsoft, como **Microsoft Word** [13] para la edición de la documentación, **Outlook** [14] como servicio de correo electrónico, **Teams** [15] para la mensajería instantánea, y **PowerPoint** [16] para la creación de presentaciones. Además, se ha empleado **Overleaf** [17], una plataforma en línea para la creación y edición de documentos en LaTeX, utilizando como referencia la guía "Learn LaTeX in 30 minutes" [18].

Adicionalmente, se ha utilizado **Diagrams.net** (draw.io) [19] para la creación de diagramas de flujo, UML, organigramas y otros esquemas, y **Notepad++** [20] como editor de texto avanzado para tareas puntuales. Para la seguridad, se ha usado **Palo Alto Global Protect** [21], una solución de acceso remoto que asegura la conexión a la red corporativa mediante encriptación y autenticación multifactorial.

1.5. Contenido del documento

A lo largo del documento, se presenta el desarrollo del proyecto desde el análisis de los requisitos hasta su implementación y validación. En primer lugar, se lleva a cabo un análisis detallado de los problemas identificados en sistemas existentes, que permite establecer los requisitos funcionales y no funcionales necesarios para diseñar una solución efectiva. Posteriormente, se aborda el diseño de la solución planteada,

destacando la estructura de la arquitectura utilizada y su alineación con principios de diseño reconocidos, lo que garantiza una integración eficiente y escalable.

En las siguientes secciones se describe la implementación técnica del proyecto, explicando cada componente desarrollado, desde la infraestructura hasta la lógica del sistema y el modelo de datos. También se detalla cómo se gestionaron aspectos clave como los errores y la interoperabilidad con diferentes entornos tecnológicos. Esta parte del trabajo incluye ejemplos prácticos de integración tanto en proyectos basados en Spring Boot como en aquellos que emplean otras tecnologías.

Finalmente, el documento examina las pruebas realizadas para asegurar la funcionalidad y robustez del sistema. Los resultados reflejan los beneficios obtenidos, mientras que las conclusiones recogen las principales lecciones aprendidas y abren la puerta a futuras mejoras y nuevas áreas de investigación. Además, en los anexos se proporciona información técnica complementaria que enriquece y completa la comprensión del proyecto.

Capítulo 2

Análisis de requisitos

El análisis de requisitos es una etapa clave en cualquier proyecto de desarrollo, ya que permite comprender a fondo qué se necesita solucionar y cómo hacerlo. En esta sección, se identifican los principales problemas que enfrentan los sistemas actuales, así como las expectativas que debe cumplir la solución planteada. Este proceso busca garantizar que el nuevo desarrollo no solo sea funcional, sino que también responda a las necesidades reales del entorno en el que se implementará.

2.1. Identificación de problemas en los proyectos actuales

En esta sección se analizan los principales problemas identificados en los proyectos actuales. Estos inconvenientes han sido determinantes en la ralentización de los procesos. A continuación, se describen las principales dificultades que afectan tanto a los desarrolladores como a los usuarios finales y que justifican la necesidad de la solución propuesta:

- **Falta de centralización para gestionar documentos**: los integradores se ven obligados a realizar múltiples pasos para llevar a cabo tareas como subir, descargar, modificar o eliminar documentos y expedientes debido a la falta de un servicio unificado, lo que dificulta y ralentiza considerablemente las operaciones.

- **Ausencia de estándares en los planes de pruebas**: los proyectos anteriores no cuentan con esquemas para validar las funcionalidades creadas, lo que dificulta la detección temprana de errores y compromete la calidad antes del despliegue.

- **Obsolescencia de funciones esenciales**: varias funciones que eran ampliamente utilizadas hasta hace poco han sido deprecadas sin una explicación detallada de los cambios. Esto ha obligado a los desarrolladores a realizar

actualizaciones a las nuevas alternativas, lo cual no siempre es un proceso directo. Esto ha obligado a los equipos de desarrollo a adaptarse a nuevas soluciones, lo que en ocasiones no resulta sencillo ni inmediato.

- **Documentación incompleta**: la falta de documentación detallada y consistente representa un gran obstáculo para la incorporación de nuevos desarrolladores y dificulta el mantenimiento y mejora del sistema.

- **Dependencia de desarrolladores con experiencia previa**: ante la ausencia de documentación e información adecuada, muchas veces la única solución viable es organizar reuniones con desarrolladores que ya hayan trabajado con estas integraciones. Sin embargo, estas reuniones a menudo resultan infructuosas, aumentando los tiempos y los recursos necesarios para resolver los problemas.

- **Errores no específicos por parte del SAE**: muchos errores reportados por el sistema no proporcionan información clara ni específica sobre el problema, lo que complica su resolución. Si bien algunos están documentados, otros carecen de detalles, lo que retrasa la identificación de la causa raíz y la implementación de una solución.

Estos inconvenientes hacen evidente la necesidad de implementar una solución que sea robusta, sostenible y bien estructurada. Detectar estas deficiencias es fundamental para poder establecer los requisitos funcionales, no funcionales y técnicos que guiarán el desarrollo del proyecto.

2.2. Definición de requisitos funcionales

En esta sección se especifican los requisitos funcionales de la solución, es decir, las capacidades y servicios que el sistema debe ofrecer a los usuarios. Estos requisitos establecen las operaciones y procesos que el sistema debe llevar a cabo y detallan las acciones específicas que se implementarán.

La captura de estos requisitos ha sido un proceso largo que ha requerido múltiples reuniones y consultas con el equipo de Hiberus. Durante este trabajo, se analizaron las operaciones más utilizadas, aquellas que estaban en desuso o próximas a quedar obsoletas, y las que resultaban irrelevantes en el contexto actual. Además, todas las observaciones y decisiones fueron coordinadas con mi director de proyecto para garantizar que los requisitos estuvieran en línea con los objetivos principales del desarrollo.

Las operaciones más utilizadas, y las que se decidieron desarrollar pueden verse reflejadas en el diagrama de casos de uso de la Figura 2.1

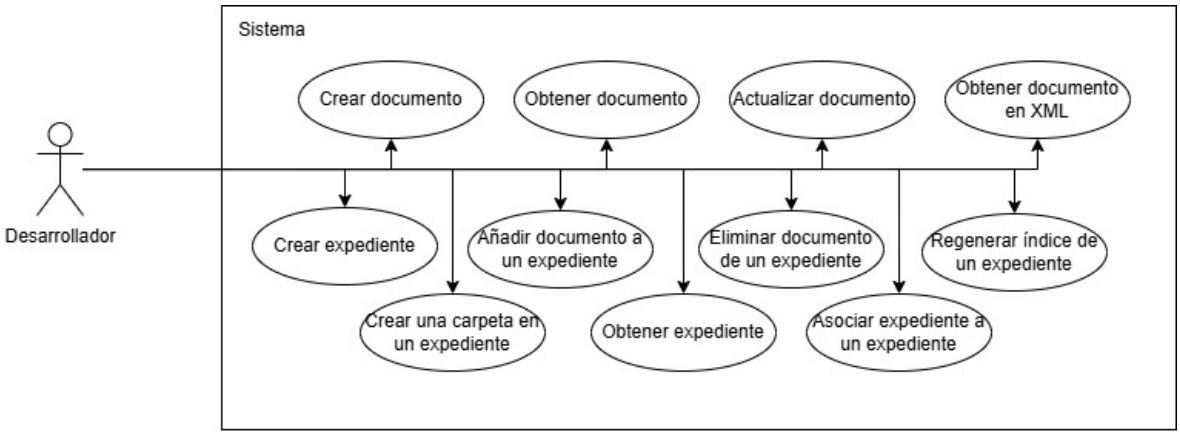

Figura 2.1: Diagrama de casos de uso

Los requisitos funcionales que se derivan de este análisis se muestran en la Tabla 2.1

Tabla 2.1: Requisitos funcionales

Nº	**Descripción**
RF-1	La aplicación deberá actualizar las bibliotecas ccsv_client y csv para garantizar la compatibilidad con la integración del cliente CCSV
RF-2	Se requerirá el desarrollo de una aplicación de pruebas que permita realizar peticiones al cliente CCSV para verificar la integración
RF-3	Se deberá desarrollar una función para inicializar los metadatos del documento antes de su creación
RF-4	Se deberá desarrollar una función para crear un documento
RF-5	Se deberá desarrollar una función para obtener un documento
RF-6	Se deberá desarrollar una función para obtener un documento en XML
RF-7	Se deberá desarrollar una función para actualizar un documento
RF-8	Se deberá desarrollar una función para crear un expediente
RF-9	Se deberá desarrollar una función para añadir documentos a un expediente
RF-10	Se deberá desarrollar una función para eliminar documentos de un expediente
RF-11	Se deberá desarrollar una función para regenerar el índice de un expediente
RF-12	Se deberá desarrollar una función crear una carpeta en un expediente
RF-13	Se deberá desarrollar una función para obtener un expediente
RF-14	Se deberá elaborar un plan de pruebas detallado que incluya casos de prueba para diversos escenarios

2.3. Definición de requisitos no funcionales

En esta sección se detallan los requisitos no funcionales, los cuales definen características esenciales que influyen en el desempeño global del sistema, pero no están directamente vinculadas a funcionalidades concretas. Estos requisitos abarcan criterios como calidad, seguridad, rendimiento y usabilidad, que son fundamentales para asegurar un sistema estable y una experiencia satisfactoria para los usuarios. Los requisitos no funcionales recogidos se presentan en la Tabla 2.2.

Tabla 2.2: Requisitos no funcionales

Nº	**Descripción**
RNF-1	La versión del JDK utilizada será la 1.8.0_291, garantizando compatibilidad y estabilidad en el desarrollo
RNF-2	El entorno de desarrollo empleado será Eclipse IDE for Enterprise Java and Web Developers, en su versión 2023-06 (4.28.0), optimizado para el desarrollo de aplicaciones empresariales en Java y tecnologías web
RNF-3	El proyecto deberá configurarse como un proyecto Maven, facilitando la gestión de dependencias y la construcción del sistema
RNF-4	La aplicación será desarrollada en Java utilizando el framework Spring Boot para aprovechar su configuración simplificada y su soporte para aplicaciones empresariales
RNF-5	Se mantendrá la estructura interna del proyecto según las convenciones establecidas, asegurando una organización coherente y de fácil mantenimiento
RNF-6	Deberá elaborarse un plan de pruebas de casos de uso detallado que cubra los aspectos funcionales y de integración

En resumen, este análisis de requisitos sienta las bases para el desarrollo de una solución integral que permita optimizar la gestión de documentos y expedientes, abordando los problemas identificados en los sistemas actuales. Con un enfoque en la funcionalidad, la eficiencia y la adaptabilidad, los requisitos definidos en este capítulo aseguran que el proyecto no solo cumpla con las expectativas funcionales, sino que también establezca estándares altos en términos de calidad y sostenibilidad a largo plazo. Esto garantiza que el sistema pueda evolucionar de forma robusta y escalable, adaptándose a las necesidades cambiantes de los usuarios y del entorno operativo.

Capítulo 3

Diseño

En este apartado se detallará exclusivamente la **arquitectura del proyecto** para la integración de CCSV, así como la del contexto en el que se encuentra, describiendo la **infraestructura del entorno** en el que se desarrolla esta integración.

La infraestructura de los proyectos que utilizarán esta integración está detallada en el anexo C. Aunque leer este anexo proporciona una comprensión más completa del proyecto en su totalidad, no es esencial para el propósito de este capítulo, por lo que no se ampliará aquí.

Para facilitar la comprensión de lo que se explicará en los siguientes apartados, la Figura 3.1 muestra las dos secciones que serán desarrolladas y detalladas más adelante.

3.1. Infraestructura del entorno

Para comprender cómo se organiza la implementación de la librería que se está desarrollando, es importante situarla en el contexto general del proyecto. En esta empresa existen diversas librerías que están siendo desarrolladas, entre ellas la que corresponde al módulo de CCSV que es la que se detalla en este proyecto. Esta librería forma parte de un sistema más amplio, compuesto por múltiples módulos que proporcionan funcionalidades diferentes.

En la Figura 3.2 se puede ver cómo está diseñada la infraestructura para desarrollar todas estas integraciones. Está organizado en varios módulos que trabajan conjuntamente para ofrecer funcionalidades específicas.

A continuación, se detalla el propósito y la jerarquía de cada módulo:

- **AMD SAE Internal**: este módulo agrupa una variedad de submódulos, cada uno diseñado para implementar diversas integraciones con los múltiples servicios que proporciona SAE, el servicio de CCSV entre ellos. Existen dos tipos de submódulos:

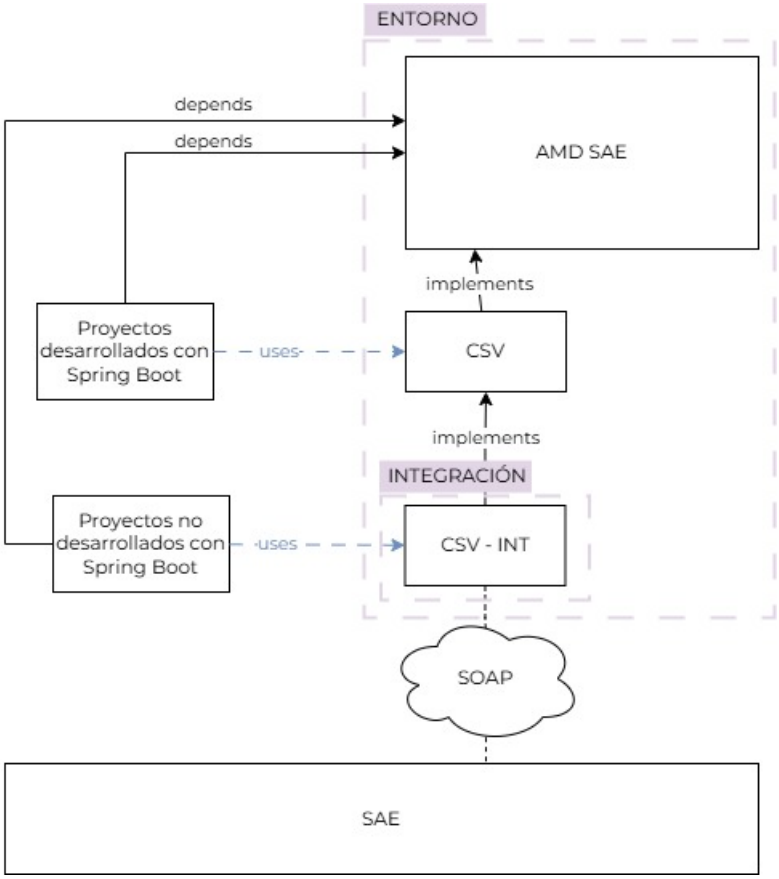

Figura 3.1: Diagrama de despliegue

- Submódulos externos (como CCSV): funcionan principalmente como interfaces que pueden ser implementadas por proyectos externos que utilicen Spring Boot. Permiten extender la funcionalidad y facilitar la integración con otros sistemas.
- Submódulos internos (como CCSV-INT): los submódulos internos alojan toda la lógica empresarial y los servicios de la API. Aquí reside la mayor parte de la funcionalidad crítica del sistema.

Además de CCSV y CCSV-INT, este módulo contiene otros submódulos encargados de diversas integraciones, como el módulo encargado del Sistema de Gestión de Avisos (SGA-INT y SGA), o el encargado del Servicio de Identificación de Usuarios (SIU-INT y SIU), entre muchos otros. Cada uno de estos submódulos gestiona integraciones específicas, cumpliendo con los requerimientos de diferentes áreas del sistema y ampliando las capacidades de interacción con plataformas externas, lo que permite mantener un flujo de trabajo modular, escalable y flexible.

La separación entre módulos internos y externos asegura un diseño modular y

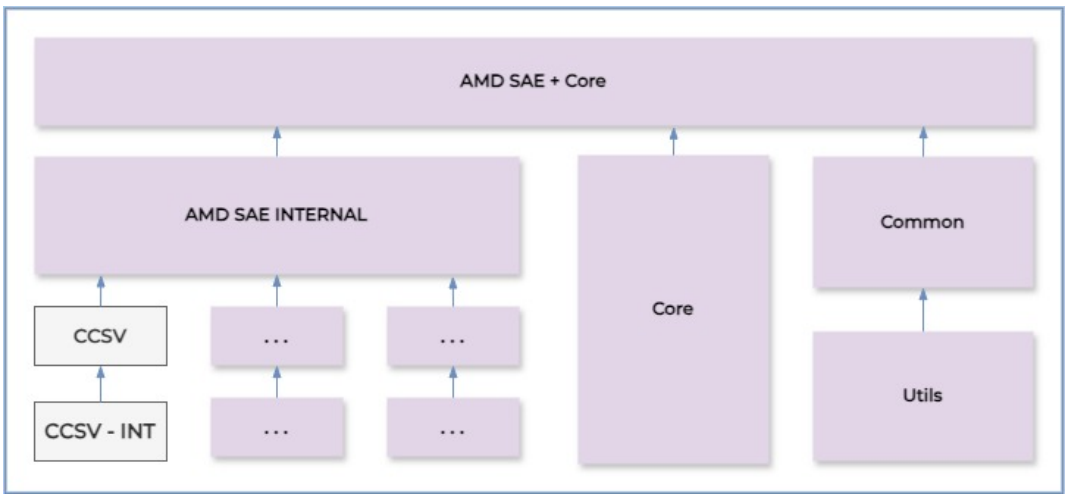

Figura 3.2: Infraestructura de módulos para integradores

flexible, donde la lógica interna se protege mientras se proporciona un acceso controlado a servicios esenciales.

El uso de este módulo se detalla con mayor profundidad en la Sección 4.4, donde se analiza cómo cada componente interactúa y se integra en el contexto general del sistema.

– **Core**: es el núcleo del sistema y cumple un rol crítico en la arquitectura. Este se encarga de:

  - Manejo de la autenticación: es responsable de gestionar las credenciales, la verificación de identidades y las autorizaciones.
  - Protección de datos y recursos: ofrece una capa robusta de seguridad para asegurar que solo los usuarios y sistemas autorizados puedan acceder a los recursos sensibles.

  Al centralizar estos aspectos, el módulo Core actúa como una base sólida sobre la cual se construyen otros módulos del sistema.

– **Common**: actúa como un conjunto de utilidades y funcionalidades compartidas que respaldan los demás módulos.

  - Funcionalidades transversales: proporciona herramientas reutilizables, como validadores, formatos de datos, y abstracciones para tareas comunes.
  - Lógica de negocio auxiliar: simplifica el desarrollo y reduce la duplicación de código al ofrecer soluciones genéricas para problemas frecuentes. Gracias a este módulo, los desarrolladores pueden enfocarse en implementar la lógica específica de cada módulo sin preocuparse por reinventar funciones básicas.

3.2. Arquitectura del proyecto

La integración de la librería CCSV tiene como objetivo encapsular la lógica interna de una aplicación legada para que el resto de sistemas o aplicaciones puedan interactuar con ella de manera sencilla y estandarizada. Esta integración se basa en la creación de una API que sirve como puente entre el sistema antiguo y los clientes o consumidores modernos. Los principales pasos y componentes del diseño son los siguientes:

1. **Encapsulación del sistema antiguo**: se ha aislado toda la lógica de negocio y complejidad de la aplicación legada en una nueva capa de abstracción. Esto asegura que los clientes no interactúen directamente con el sistema antiguo.

2. **Creación de una API moderna**: a través de esta API, los usuarios solo necesitan conocer las operaciones actuales que pueden realizar, sin preocuparse por cómo están implementadas o por la complejidad de los datos internos. La API actúa como la única entrada/salida del sistema.

3. **Estándares actuales**: se ha diseñado la API para cumplir con estándares modernos de diseño REST, asegurando su compatibilidad y extensibilidad a largo plazo.

4. **Transformación de datos**: en los casos donde el formato de los datos del sistema antiguo no es adecuado para las necesidades actuales, la API se encarga de transformar los datos, haciéndolos comprensibles y utilizables para los consumidores.

5. **Aislamiento para futuras modificaciones**: la lógica de la API permite implementar cambios en los servicios modernos sin tocar el código del sistema antiguo, promoviendo un bajo acoplamiento.

Como resultado de los objetivos planteados, se optó por desarrollar la integración utilizando el patrón *façade*.

3.2.1. Patrón *Façade*

El diseño planteado sigue los principios del **patrón *Façade*** o fachada en español, lo que significa que se ha construido un punto único y claro de acceso al sistema para los usuarios. Sin embargo, este diseño no solo se limita a simplificar, sino que también transforma la manera en la que se consume el sistema antiguo, adaptándolo a un entorno más moderno. El patrón se puede observar en la Figura 3.3.

Los puntos clave de la relación entre el diseño del proyecto y el patrón *Façade* son:

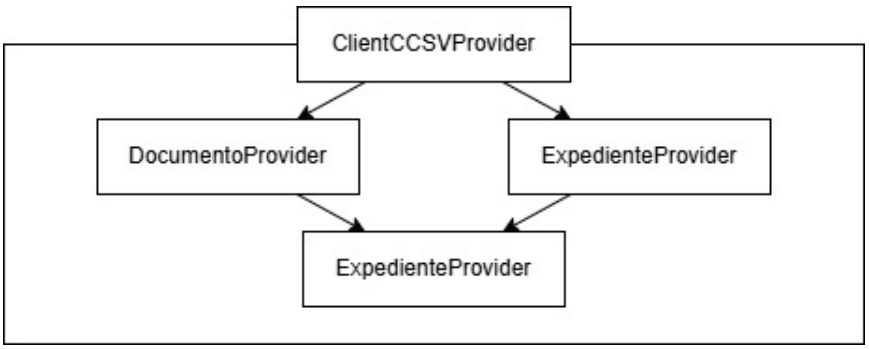

Figura 3.3: Patrón *Façade* el caso de la integración de CCSV

1. **Principios compartidos**: el patrón Facade oculta la complejidad de un sistema subyacente, proporcionando una interfaz sencilla y clara. Mi integración hace exactamente eso, encapsulando toda la lógica antigua en una API moderna.

2. **Separación de responsabilidades**: igual que en Facade, en este diseño se separa la lógica interna del sistema antiguo de los consumidores modernos. Esto permite que los usuarios interactúen únicamente con las operaciones expuestas, sin necesidad de conocer los detalles de cómo funciona el legado.

3. **Extensibilidad**: mientras el patrón Facade permite añadir nuevas funcionalidades a través de su interfaz simplificada, en mi integración se da un paso más al proporcionar herramientas para la transformación y validación de datos en la API.

4. **Simplificación de uso**: para los clientes (aplicaciones que consumen la API), solo existen los puntos de acceso que necesitan, reduciendo la carga cognitiva de tener que comprender o adaptar la complejidad del legado.

5. **Modernización**: aunque el patrón Facade en sí mismo no busca modernizar, mi diseño aprovecha la interfaz simplificada para hacer que el sistema sea compatible con tecnologías actuales, lo cual lo diferencia y lo mejora en este aspecto.

Se decidió seguir un enfoque basado en el patrón Facade porque encajaba perfectamente con los objetivos del proyecto. Este patrón permite encapsular sistemas complejos detrás de una fachada fácil de usar, y mi integración necesitaba ofrecer simplicidad a la hora de interactuar con la aplicación antigua. Además, esta estrategia ha permitido evitar problemas derivados de exponer directamente una arquitectura desactualizada, mientras se establece una base sólida para futuras ampliaciones y mantenimiento.

Capítulo 4

Implementación

En este capítulo se describe el proceso de desarrollo e implementación del sistema propuesto, incluyendo las decisiones técnicas, la configuración del modelo de datos, y la integración de las diferentes partes de la aplicación. Se detallan las tecnologías empleadas, la estructura del código y los desafíos enfrentados, explicando cómo se aseguraron tanto la funcionalidad como la compatibilidad del sistema con las especificaciones definidas previamente. Además, se aborda la creación de tipos de datos personalizados diseñados para optimizar la interacción con SAE y simplificar estructuras complejas, junto con ejemplos prácticos implementados en proyectos basados en Spring Boot y en entornos que no emplean este framework.

4.1. Implementación de la arquitectura

La infraestructura del sistema se basa en varias clases que interactúan con servicios web externos, utilizando SOAP[1] como protocolo de comunicación y el framework Apache CXF para gestionar dicha interacción. En este apartado se describen los detalles específicos de la implementación de los servicios relacionados con la gestión de documentos y expedientes.

El núcleo de esta interacción lo constituye una clase denominada ***ClienteCCSVProviderImpl***, que actúa como un catálogo de servicios y centraliza las operaciones. Esta clase distribuye la lógica de sus funciones entre ***DocumentProviderImpl*** y ***ExpedienteProviderImpl***, dependiendo de la operación que se deba realizar. A su vez, ambas clases (*DocumentProviderImpl* y *ExpedienteProviderImpl*) gestionan las llamadas a los servicios de SAE mediante la clase ***PeticionesSaeProviderImpl***.

Además, existe también la clase ***CCSVExternalProviderImpl***, diseñada

[1] SOAP (Simple Object Access Protocol) es un protocolo de comunicación estándar basado en XML que permite el intercambio de información estructurada en redes distribuidas. Es ampliamente utilizado para integrar aplicaciones en entornos heterogéneos.

específicamente para permitir que los proyectos desarrollados con Spring Boot integren de forma eficiente la funcionalidad CCSV. Cabe destacar que, a diferencia de las demás clases que residen en el módulo **CCSV-INT**, *CCSVExternalProviderImpl* se encuentra en el módulo superior de **CCSV**, fuera del módulo CCSV-INT donde se aloja la lógica central del proyecto. Debido a su ubicación, esta clase no aparece en el diagrama de paquetes, ya que dicho diagrama solo representa la estructura interna del módulo CCSV-INT.

Para proporcionar una mejor comprensión de la estructura general del proyecto, se incluye el diagrama de paquetes que se muestra en la Figura 4.1. Este diagrama refleja cómo se organizan y relacionan los distintos componentes dentro del módulo **CCSV-INT**.

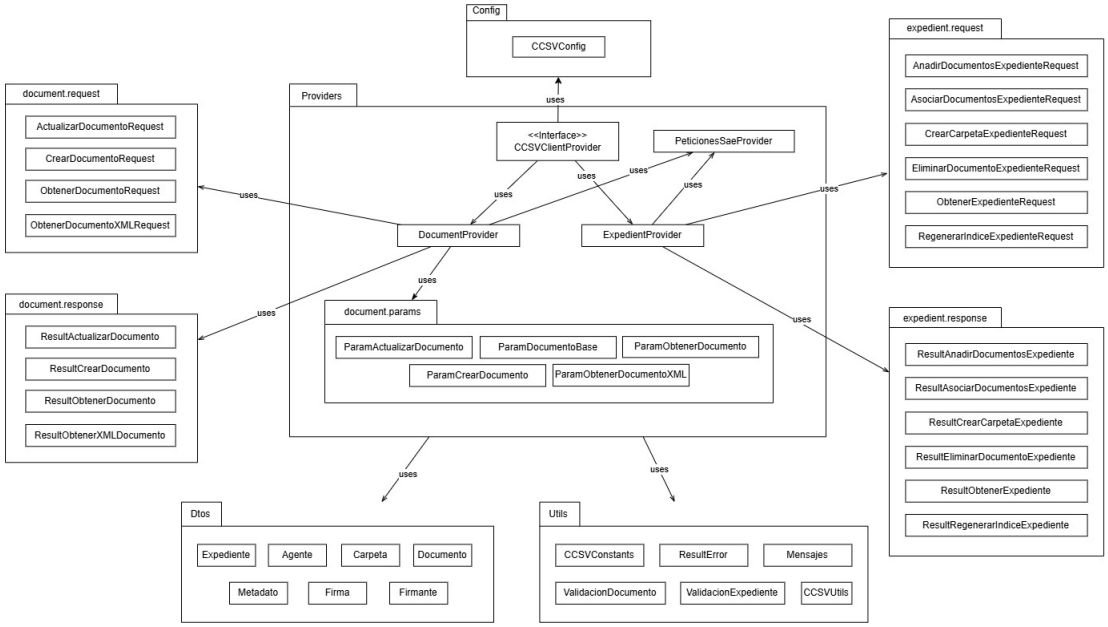

Figura 4.1: Diagrama de paquetes / clases del módulo CSV-INT

Tomando como referencia la Figura 4.1, se llevará a cabo una descripción detallada de las clases más relevantes y su configuración, siendo estos elementos fundamentales para el adecuado funcionamiento de la integración propuesta.

Para facilitar la comprensión, las clases a describir se han organizado en dos grupos: clases internas (módulo CSV-INT) y clases externas (módulo CSV). Las clases auxiliares ubicadas en los paquetes `Dtos` y `Utils` se detallarán en el Anexo D.3, mientras que las clases en los paquetes `request`, `params` y `response` se describirán más adelante en la Sección 4.3.

4.1.1. Clases internas

A continuación, se describen las clases destacadas del módulo interno, que incluyen `ClientCCSVProviderImpl`, `DocumentProviderImpl` y `ExpedientProviderImpl`.

ClientCCSVProviderImpl

La clase *ClientCCSVProviderImpl* es el punto de entrada. Esta clase recibe un objeto de configuración *CCSVProviderConfig*, el cual contiene parámetros esenciales como las URLs de los servicios, credenciales y otras configuraciones necesarias. Dentro de su constructor, se realiza la configuración de los clientes para ambos servicios:

- **Cliente de documentos**

  Se utiliza el *ClientProxyFactoryBean* de Apache CXF, especificando la interfaz *IDocumentMetadataSignatureService* para interactuar con el servicio de documentos, llamado casi de la misma manera *DocumentMetadataSignatureService*. Se configura el WSDL[2] y la URL[3] del servicio, asegurando al mismo tiempo un mapeo automático entre los objetos Java y los datos XML mediante el uso de *AegisDatabinding*[4]. Además, se habilita el mecanismo *MTOM*[5], lo cual optimiza el intercambio de datos binarios, como archivos, siendo crucial para los servicios que gestionan documentos e imágenes.

- **Cliente de expedientes**

  De manera similar a la configuración del cliente de documentos, se utiliza la interfaz *IAdministrativeFileService* para interactuar con el servicio de expedientes, que se también se llama parecido, ***AdministrativeFileService***. Se lleva a cabo la configuración de la URL base, el WSDL y se habilita *MTOM* para optimizar la transferencia de datos.

Al final de la configuración de cada cliente, se crean los proxies correspondientes que permiten que otras clases interactúen con los servicios a través de estas interfaces. Para más detalle, se puede acceder al Anexo D.1.1

DocumentoProviderImpl

[2] Web Services Description Language, es un formato estándar en XML para describir los servicios web.
[3] Uniform Resource Locator, que se utiliza para especificar la dirección de los recursos en la web.
[4] Herramienta para mapeo automático entre objetos Java y datos XML en Apache CXF.
[5] Message Transmission Optimization Mechanism, mecanismo para optimizar el intercambio de datos binarios a través de servicios web.

Una vez que el cliente de documentos ha sido configurado en la clase *ClientCCSVProviderImpl*, se pasa a la clase *DocumentProviderImpl*, que se encarga de gestionar la interacción con el servicio de documentos de forma más detallada. Su constructor toma el servicio *iDocumentMetadataSignatureService* y el objeto *CCSVProviderConfig* y comienza a propagarlos a lo largo de las clases de la aplicación.

Dentro de esta clase, se configura un proveedor de peticiones *PeticionesSaeProviderImpl*, encargado de las operaciones relacionadas con la solicitud y gestión de los documentos dentro del servicio. Para más detalle, consultar el Anexo D.1.2.

ExpedientProviderImpl

De manera análoga a la clase *DocumentProviderImpl*, la clase *ExpedientProviderImpl* gestiona el servicio de expedientes, pero aquí también se incorpora la funcionalidad del servicio de documentos. En esta clase, se configura igualmente el proveedor de peticiones *PeticionesSaeProviderImpl*, que se ocupa de las operaciones de ambos servicios, documentales y de expedientes, centralizando así el manejo de las funciones requeridas por el sistema. Para más detalle, se puede acceder al Anexo D.1.3.

Finalmente, la clase ***PeticionesSaeProviderImpl*** ya ha sido nombrada en varios apartados de este documento, y su funcionamiento ha sido descrito de manera general. Para una descripción más detallada de su implementación, se recomienda consultar el Anexo D.1.4.

4.1.2. Clases externas

En cuanto a las clases externas, solo se describirá la siguiente ya que es la única relevante para el entendimiento del proyecto.

CCSVExternalProvider

La clase *CCSVExternalProvider* es clave para integrar los servicios de CCSV en aplicaciones Spring Boot. Configurada con la anotación `@Configuration`, centraliza la creación de un cliente preparado para gestionar documentos y expedientes, utilizando los parámetros definidos en la clase *AmmSaeCCSVConfig*. Esta configuración, basada en `@ConfigurationProperties`, extrae automáticamente los valores del archivo de propiedades de la aplicación, como URLs y códigos de procedimiento, asignándolos a

los campos correspondientes.

El cliente *ClientCCSVProviderImpl* permite gestionar los servicios de CCSV de manera sencilla y uniforme en todo el sistema. Más detalles sobre esta implementación pueden consultarse en el Anexo D.1.5.

4.2. Flujo de integración

El proceso de integración con CCSV se estructura en varias fases diseñadas para garantizar la correcta ejecución de las operaciones, desde la invocación inicial del catálogo de métodos hasta la comunicación final con SAE. Estas etapas aseguran la validación, la construcción adecuada de las solicitudes, y la interacción eficiente con los servicios proporcionados. Además, en el anexo D.2 se incluyen los diagramas de secuencia que ilustran el flujo detallado de la integración.

Las fases del flujo son las siguientes:

1. **Uso del catálogo de operaciones**

   ClientCCSVProvider actúa como el punto de entrada, ofreciendo métodos como crearDocumento, obtenerDocumento y actualizarDocumento, entre otros. Cada uno de estos métodos se implementa invocando las correspondientes clases proveedoras (DocumentProvider y ExpedientProvider), que contienen la lógica específica de cada operación.

2. **Lógica de las operaciones**

   En esta fase se validan los parámetros de entrada, se construyen las solicitudes necesarias con los objetos correspondientes y se maneja cualquier preprocesamiento previo a la comunicación con SAE. Asegura que las operaciones estén correctamente configuradas y las entradas sean válidas antes de realizar las llamadas externas.

3. **Comunicación con SAE**

   Una vez que las operaciones han sido validadas y configuradas, la interacción con el sistema CCSV subyacente se realiza mediante la clase PeticionesSae. Este componente maneja la comunicación directa con los servicios externos, enviando solicitudes y recibiendo las respuestas correspondientes.

4.3. Modelo de datos

En el contexto de integración con SAE, cada operación del sistema requiere y devuelve tipos de datos que encapsulan toda la información necesaria para su ejecución.

Estos tipos, generalmente definidos en inglés por SAE, son empleados en el intercambio de datos entre los servicios de SAE y el sistema integrado. Para adaptarse mejor a los requisitos específicos del proyecto, se han diseñado tipos de datos personalizados, cuya nomenclatura se encuentra en español y que simplifican las estructuras proporcionadas por SAE.

La distinción entre estos tipos de datos es importante:

- Nombres en inglés: representan modelos nativos proporcionados por SAE.
- Nombres en español: tipos creados internamente en el proyecto.

SAE, como plataforma de administración electrónica, ofrece servicios que abarcan múltiples áreas funcionales. Sin embargo, este proyecto se centra exclusivamente en su módulo de CSV, específicamente en operaciones vinculadas con la creación, actualización y manejo de documentos y expedientes.

Para cada operación en SAE, se establecen parámetros de entrada (*Params*), que contienen los datos requeridos para llevar a cabo la operación, y resultados de salida (*Results*), que encapsulan la información proporcionada tras su ejecución.

A modo de ejemplo, la operación encargada de crear un documento es ***createDocument***. Esta operación requiere como entrada un objeto del tipo ***ParamCreateDocument***, el cual incluye información como el documento a almacenar y el identificador de la carpeta destino. Como resultado, la operación devuelve un objeto del tipo ***ResultCreateDocument***, que contiene, entre otros, el identificador único del documento creado en el sistema.

Debido a la complejidad de los modelos proporcionados por SAE y la falta de documentación exhaustiva, en este proyecto se han diseñado nuevos tipos de datos simplificados. Estos nuevos modelos mejoran la claridad, hacen más manejables las estructuras de datos y se adaptan mejor a los requerimientos específicos del sistema.

Estos nuevos tipos de datos se crearon a partir de la adaptación de los modelos originales proporcionados por SAE. Durante este proceso, se analizaron las estructuras iniciales para identificar los elementos esenciales que debían mantenerse, descartando atributos innecesarios, obsoletos o mal documentados. El objetivo principal de estas adaptaciones fue optimizar la flexibilidad y simplificar la gestión de los datos, sin comprometer la funcionalidad ni la integridad de las operaciones implementadas.

Para cada operación del **módulo de documentos** se definieron tres tipos de datos: **Parámetros (Params)**, con la información necesaria para ejecutar la operación; **Resultados (Result)**, con la información devuelta tras su ejecución; y **Petición (Request)**, que es una encapsulación de Params, conteniendo únicamente los atributos esenciales necesarios para realizar las operaciones. Así, lo que el usuario debe enviar

se limita a los elementos más básicos, evitando la complejidad de estructuras más detalladas.

En el **módulo de expedientes**, con operaciones de naturaleza más sencilla, se crearon únicamente dos tipos de datos: **Petición (Request)**, que incluye los atributos necesarios para la solicitud, y **Respuesta (Result)**, que contiene la información devuelta al completar la operación. En este caso, no se definieron parámetros debido a la simplicidad de los atributos requeridos.

4.4. Uso de la biblioteca

Para demostrar la flexibilidad de la librería y su facilidad de integración, se han desarrollado dos proyectos de ejemplo: uno utilizando Spring Boot y otro sin dicho framework. Estos ejemplos tienen como objetivo verificar el funcionamiento de la librería en diferentes contextos y servir como referencia para futuras implementaciones.

4.4.1. Proyectos desarrollados con Spring Boot

La integración con proyectos basados en Spring Boot sigue una estructura clara y estandarizada. El proyecto define una configuración inicial que incluye los paquetes necesarios para que la librería sea detectada y utilizada de manera adecuada por el contenedor de Spring. Un ejemplo típico de clase principal en un proyecto Spring Boot es el del Listado 4.1.

Listado 4.1: Constructor de la clase DemoApplication

```
@SpringBootApplication(scanBasePackages = { "es.aragon.core.sae.ccsv",
    "com.example.demo" })

public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Esta configuración inicial permite que Spring detecte las configuraciones de la librería ubicadas en los paquetes de es.aragon.core.sae.ccsv, asegurando que todos los beans necesarios para interactuar con la misma se encuentren disponibles en el contexto de la aplicación.

Para interactuar con la librería, los proyectos Spring Boot definen controladores que encapsulan las llamadas a las funcionalidades de la misma. Estos controladores utilizan la inyección de dependencias a través de la anotación @Autowired, lo que permite acceder directamente a instancias del cliente ClientCCSVProvider, proporcionadas por

la clase CCSVExternalProvider descrita anteriormente.

Estos controladores implementan endpoints que reciben peticiones desde el exterior y utilizan un cliente proporcionado por la librería para invocar las funciones correspondientes. Gracias al uso de la inyección de dependencias de Spring, el cliente de la librería, configurado previamente, se encuentra listo para ser utilizado en los controladores sin necesidad de configuraciones adicionales. Cada método de los controladores valida las entradas proporcionadas por el usuario, construye las solicitudes necesarias utilizando los DTOs definidos por la librería y, posteriormente, obtiene y retorna los resultados de manera estructurada.

Listado 4.2: Endpoint de operación Obtener documento

```
@RequestMapping("/document")
@RestController
public class ControllerDocument {

    @Autowired
    private ClientCCSVProvider clientCCSV;

    @GetMapping("/obtenerDocumento")
    public ResponseEntity<CCSVResponseDTO<ResultObtenerDocumentoDTO>>
        obtenerDocumento(
            @RequestParam(required = false) String csv,
            @RequestParam(required = false) String id) throws CCSVException {

        // [...]

        ObtenerDocumentoRequestDto request =
            ObtenerDocumentoRequestDto.builder().id(id).csv(csv).build();

        return ResponseEntity.ok(clientCCSV.obtenerDocumento(request));
    }
}
```

En el Listado 4.2, el controlador define un endpoint llamado `/obtenerDocumento` que recibe solicitudes HTTP GET. Este endpoint valida los parámetros de entrada y construye una solicitud utilizando los DTO[6] de la librería, que posteriormente se envía al cliente `ClientCCSVProvider`. La respuesta del cliente se devuelve al solicitante de manera estructurada.

La implementación de este ejemplo destaca la sencillez y la modularidad del diseño de la librería, al ofrecer una solución eficaz que puede ser fácilmente integrada en un ecosistema basado en Spring Boot, permitiendo un manejo eficiente de documentos y

[6] DTO (Data Transfer Object): un objeto que se utiliza para transferir datos entre procesos, capas de una aplicación o aplicaciones diferentes. Los DTOs simplifican y agrupan datos, mejorando el rendimiento y desacoplando la lógica de negocio de la comunicación.

expedientes.

4.4.2. Proyectos no desarrollados Spring Boot

Además de la integración en aplicaciones basadas en Spring Boot, la librería está diseñada para su uso en proyectos sin dependencia de este framework, lo que demuestra su flexibilidad y capacidad de adaptarse a diferentes entornos.

La configuración en este tipo de proyectos se lleva a cabo manualmente, creando una instancia de configuración (CCSVProviderConfig) en la que se especifican los parámetros clave necesarios para interactuar con los servicios de la librería. Esto incluye detalles como URLs para los servicios de documentos y expedientes, códigos de organismos, nombres de procedimientos y otros valores relevantes según el caso de uso.

Una vez configurada esta instancia, se inicializa el cliente principal de la librería (ClientCCSVProvider), que actúa como punto de acceso a las funcionalidades. Este cliente permite invocar las distintas operaciones ofrecidas, desde la gestión de documentos hasta la interacción con expedientes, de manera directa y sin necesidad de configuraciones adicionales.

Un ejemplo representativo (Listado 4.3) de cómo se lleva a cabo esta integración incluye la construcción del objeto CCSVProviderConfig utilizando un patrón builder, donde se asignan todos los parámetros específicos del entorno en cuestión. A continuación, se crea una instancia de ClientCCSVProvider, que queda lista para ejecutar las operaciones necesarias de forma inmediata.

Listado 4.3: Configuración de ejemplo del cliente

```
public static void main() {

      CCSVProviderConfig config = CCSVProviderConfig.builder()
                .appCode(Constantes.APP_CODE)
                .appName(Constantes.APP_NAME)
                // Demas variables...
                .build();

    ClientCCSVProvider clientCCSV = new ClientCCSVProviderImpl(config);

    // Codigo continua

}
```

Este diseño resalta la capacidad de la librería para operar de manera desacoplada de un framework específico, permitiendo que desarrolladores trabajen en contextos diversos sin mayores complicaciones. Gracias a esta implementación, proyectos independientes y aquellos con restricciones específicas pueden aprovechar las ventajas

que brinda la librería, manteniendo un equilibrio entre simplicidad y funcionalidad avanzada.

4.5. Pruebas y validación

En esta sección se detallan las distintas metodologías de prueba implementadas para asegurar el correcto funcionamiento del sistema. Se emplearon pruebas unitarias, basadas en casos de uso y de validación de endpoints para verificar la fiabilidad y robustez del sistema en diversas situaciones. A continuación se describe cada tipo de prueba y se presentan los resultados obtenidos.

4.5.1. Pruebas unitarias

Las pruebas unitarias validan de manera aislada el funcionamiento de componentes como funciones o clases. Para esto, se desarrollaron pruebas con JUnit, cubriendo todas las operaciones del catálogo de la clase `ClientCCSVProviderImpl` sin distinción de relevancia. Se utilizaron dependencias simuladas para simular escenarios y verificar las interacciones, incluyendo casos como la creación, obtención y actualización de documentos. Todas las pruebas fueron exitosas, confirmando que las operaciones funcionan correctamente bajo diversas condiciones. Los detalles de las pruebas están en el Anexo E.1.

4.5.2. Pruebas de integración basadas en casos de uso

Las pruebas basadas en casos de uso validan la correcta ejecución del sistema desde una perspectiva de negocio. Cada caso se descompone en pasos detallados que incluyen la creación de objetos, ejecución de acciones y validación de los resultados obtenidos. Estas pruebas garantizan la coherencia y la integridad del sistema en su conjunto. Los resultados y detalles de las pruebas están disponibles en el Anexo E.2.

4.5.3. Pruebas basadas en casos de uso

La validación de endpoints garantiza que las APIs respondan correctamente a entradas válidas e inválidas. Se realizó utilizando un proyecto con Spring Boot, y se exploraron los endpoints mediante Swagger. Se verificaron operaciones principales como la creación y actualización de recursos, además de manejar errores excepcionales. Los detalles de las pruebas están en el Anexo E.3.

Capítulo 5

Conclusiones y trabajo futuro

El desarrollo de este proyecto ha sido un viaje enriquecedor que no solo ha implicado un importante desafío técnico, sino también un gran aprendizaje en diversos aspectos de la ingeniería y la vida profesional. En este capítulo, realizaré una reflexión sobre el alcance alcanzado durante el desarrollo, junto con una evaluación de lo aprendido, las conclusiones derivadas del trabajo realizado, y algunas propuestas para futuras líneas de mejora que podrían seguir enriqueciendo el sistema desarrollado. Este proyecto no solo cierra un ciclo de desarrollo, sino que también abre las puertas a futuras oportunidades de expansión que mejorarán aún más los procesos de la empresa.

En el Anexo B, se presenta un análisis detallado del tiempo invertido en el desarrollo del proyecto. Se incluyen el diagrama de Gantt que muestra la planificación temporal de las actividades y una tabla con las horas dedicadas a cada tarea específica. Este apartado proporciona una visión clara de la dedicación de recursos y tiempo en las distintas fases del proyecto.

5.1. Conclusiones acerca del proyecto

El proyecto desarrollado representa un gran esfuerzo para modernizar los procesos relacionados con el entorno SAE. Durante su desarrollo, me enfoqué en implementar las operaciones más comunes y demandadas por los integradores, asegurando que fueran funcionales, fiables y adaptadas a los casos de uso habituales. Además, se diseñó un marco modular que no solo cumple con los requisitos actuales, sino que también deja abierto un amplio margen para futuras mejoras y ampliaciones.

Uno de los principales logros de este proyecto es la simplificación de los procesos, al reducir la complejidad de las operaciones anteriores. Además, la mantenibilidad del sistema se ha incrementado, ya que las modificaciones y actualizaciones futuras podrán realizarse de forma centralizada, evitando la necesidad de realizar ajustes repetidos en múltiples proyectos.

Además del desarrollo funcional, se llevaron a cabo diversas pruebas para garantizar la robustez y el correcto funcionamiento del sistema. Se realizaron pruebas unitarias para validar cada componente por separado, asegurando que cumplieran con los requisitos esperados. También se desarrollaron pruebas basadas en casos de uso reales para simular el comportamiento del sistema en escenarios cotidianos, lo que permitió identificar posibles mejoras y afinar la experiencia del usuario. Finalmente, se ejecutaron pruebas de validación de los endpoints, comprobando que las comunicaciones entre el sistema y sus dependencias externas fueran seguras, confiables y cumplieran con los estándares de calidad definidos.

Entonces, para resumir, este proyecto ha sido un hito importante en la modernización de los procesos en el entorno SAE. La simplificación de operaciones, la mejora en la mantenibilidad del sistema y la implementación de pruebas exhaustivas aseguran que el sistema esté bien preparado para su uso y expansión. Gracias a su arquitectura modular, el proyecto no solo satisface las necesidades actuales, sino que también facilita futuras adaptaciones, mejorando la eficiencia operativa y la experiencia de usuario. El resultado es una solución robusta que simplifica el trabajo para los integradores y proporciona una base sólida para seguir avanzando.

5.1.1. Comparativa de uso antiguo y nuevo

El uso del sistema anterior, caracterizado por su complejidad y limitada adaptabilidad, contrastaba significativamente con el enfoque del nuevo proyecto. El sistema desarrollado ofrece una interfaz más clara y operativa, facilitando tareas cotidianas que antes requerían un mayor esfuerzo manual o técnico. Además, la base modular proporciona una estructura más comprensible para los futuros desarrolladores, permitiendo una curva de aprendizaje menos pronunciada.

Un aspecto clave del nuevo sistema es su mejor mantenibilidad. Gracias a su arquitectura centralizada, cualquier cambio o mejora se puede realizar directamente en este proyecto sin necesidad de modificar cada integración de forma individual. Esto elimina la necesidad de realizar ajustes proyecto por proyecto, optimizando el tiempo y los recursos necesarios para actualizaciones o correcciones. Estas mejoras no solo aumentan la eficiencia del equipo, sino que también aseguran una mayor adaptabilidad para atender necesidades específicas del entorno empresarial.

5.2. Evaluación personal

Trabajar en este proyecto ha sido tanto un desafío técnico como una experiencia de crecimiento personal y profesional. Desde el inicio, enfrenté la dificultad de integrarme

en un entorno empresarial completamente nuevo mientras lidiaba con un sistema legado, cuya arquitectura y funcionamiento no solo eran desconocidos, sino también, en muchos casos, obsoletos y poco documentados. Adaptarme a esta realidad fue un proceso tedioso que requirió paciencia, investigación constante y la capacidad de encontrar soluciones creativas frente a problemas inesperados.

En el aspecto técnico, este proyecto me ha permitido desarrollar habilidades cruciales como ingeniera informática. He profundizado en la comprensión de sistemas heredados y aprendido a adaptar mi enfoque para optimizar y modernizar funcionalidades dentro de las limitaciones existentes. También he ganado experiencia en la implementación de soluciones más eficientes y sostenibles que abren la puerta a mejoras futuras.

Sin embargo, además de estar desarrollando el TFG, he estado trabajando en proyectos de la empresa, lo que ha hecho que esta experiencia en un entorno profesional haya sido transformadora a nivel personal. Aprendí a gestionar el estrés que implica enfrentar tareas complejas con plazos ajustados, a colaborar efectivamente dentro de un equipo y a asumir responsabilidad en decisiones clave del proyecto. Además, trabajar en un entorno real me permitió entender cómo el trabajo técnico afecta directamente las operaciones empresariales, dándome una visión más amplia de mi rol como profesional.

Uno de los aspectos más gratificantes de este proyecto es saber que lo desarrollado no solo tiene valor técnico, sino que también será utilizado activamente, y no de manera ocasional. Esto aporta un gran sentido de satisfacción, ya que me da la seguridad de que mis esfuerzos tienen un impacto real y positivo en la operación del sistema, contribuyendo a la mejora continua de los procesos empresariales.

En conclusión, aunque el proceso ha tenido momentos de frustración, el aprendizaje obtenido supera con creces las dificultades encontradas. No solo he crecido como ingeniera, mejorando mi capacidad de análisis, adaptación y resolución de problemas, sino también como persona, adquiriendo habilidades que me serán valiosas en mi desarrollo profesional y personal a largo plazo.

5.3. Propuestas de mejora y líneas de investigación futura

El proyecto que he desarrollado proporciona una base sólida para futuras mejoras y expansiones. Las funciones implementadas cubren las operaciones más utilizadas en el entorno SAE, enfocándose en la funcionalidad más común que los integradores de la empresa necesitan. Aunque estas operaciones son eficaces en la mayoría de los casos, podrían beneficiarse de mayor sofisticación o de opciones adicionales para adaptarse a

situaciones más específicas. Además de perfeccionar las funciones ya creadas, se pueden incorporar muchas más operaciones que SAE ofrece, ya que he desarrollado solo una fracción de las disponibles. Existen aún muchas más operaciones por explorar que enriquecerían el sistema.

Es importante mencionar que, aunque me he centrado en la integración con archivos CSV, SAE no se limita a esta opción. Existen múltiples integraciones con otras librerías y sistemas que no he implementado, pero que pueden ser fácilmente incorporadas sobre la base creada. Estas integraciones adicionales permitirían ampliar la versatilidad del sistema, ofreciendo nuevas formas de interacción con otras fuentes de datos.

En conclusión, el proyecto no solo proporciona una herramienta funcional, sino también un punto de partida estratégico. Permite mejorar las funciones actuales, desarrollar nuevas operaciones dentro de la integración de CSV y servir como inspiración para implementar otras integraciones más complejas. El marco establecido ofrece una base sólida y adaptable para seguir ampliando y mejorando este sistema según las necesidades futuras.

Bibliografía

[1] Eclipse Foundation. Eclipse ide: Entorno de desarrollo integrado para múltiples lenguajes. https://www.eclipse.org/ide/.

[2] Oracle. Java: Lenguaje de programación multiplataforma. https://www.oracle.com/java/.

[3] Spring Boot: Framework para desarrollo de aplicaciones en Java. https://spring.io/projects/spring-boot.

[4] Project Lombok: Biblioteca para reducir código repetitivo en Java. https://projectlombok.org/.

[5] Spring. Spring tools: Conjunto de herramientas para desarrollo con Spring. https://spring.io/tools.

[6] Spring Initializr: Generador de proyectos para el ecosistema spring. https://start.spring.io/.

[7] Apache maven: Herramienta para la gestión de dependencias y construcción de proyectos. https://maven.apache.org/.

[8] Apache ant: Herramienta de automatización de compilación. https://ant.apache.org/.

[9] JUnit Team. Junit: Framework para pruebas unitarias en java. https://junit.org/.

[10] Swagger UI (OpenAPI): Herramienta de documentación y prueba de api. https://swagger.io/tools/swagger-ui/.

[11] Atlassian. Sourcetree: Cliente git con interfaz gráfica intuitiva. https://www.sourcetreeapp.com/.

[12] GitLab Inc. Gitlab: Plataforma de control de versiones y devops. https://about.gitlab.com/.

[13] Microsoft. Microsoft word: Herramienta para edición de documentos. https://www.microsoft.com/word/.

[14] Microsoft. Microsoft outlook: Servicio de correo electrónico. https://www.microsoft.com/outlook/.

[15] Microsoft. Microsoft teams: Plataforma de mensajería instantánea y colaboración. https://www.microsoft.com/teams/.

[16] Microsoft. Microsoft powerpoint: Herramienta para creación de presentaciones. https://www.microsoft.com/powerpoint/.

[17] Overleaf: Plataforma en línea para creación y edición de documentos en latex. https://www.overleaf.com/.

[18] Overleaf. Learn in 30 minutes. https://es.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes, 2025. Disponible en https://es.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes.

[19] Diagrams.net (draw.io): Herramienta para creación de diagramas. https://www.diagrams.net/.

[20] Notepad++: Editor de texto avanzado para múltiples lenguajes. https://notepad-plus-plus.org/.

[21] Palo Alto Networks. Global protect: Solución de acceso remoto seguro. https://www.paloaltonetworks.com/globalprotect.

[22] Servicios para la gestión de documentos - sae. https://paega2.atlassian.net/wiki/spaces/AreaUsuariosIntegradores/pages/3379397180/Servicios+para+la+gesti+n+de+documentos.

[23] Servicios para la gestión de expedientes - sae. https://paega2.atlassian.net/wiki/spaces/AreaUsuariosIntegradores/pages/3379364261/Servicios+para+la+gesti+n+de+expedientes.

[24] Miguel Ángel Latre, José Merseguer, and Javier Nogueras Iso. Apuntes de la asignatura ingeniería del software. https://moodle.unizar.es/. Accedido desde Moodle.

[25] Francisco Javier Fabra Caro. Apuntes de la asignatura sistemas y tecnologías web. https://moodle.unizar.es/. Accedido desde Moodle.

[26] Sergio Ilarri Artigas. Apuntes de la asignatura bases de datos 2. https://moodle.unizar.es/. Accedido desde Moodle.

[27] Taquel Trillo Lado and Carlos Tellería Orriols. Apuntes de la asignatura sistemas de información. https://moodle.unizar.es/. Accedido desde Moodle.

[28] F. Javier Zarazaga Soria and Rubén Béjar. Apuntes de la asignatura proyecto software. https://moodle.unizar.es/. Accedido desde Moodle.

[29] Eduardo Mena Nieto. Apuntes de la asignatura sistemas legados. https://webdiis.unizar.es/asignaturas/SL/. Accedido desde WebDIIS de la Universidad de Zaragoza.

Anexos

Anexos A

Diccionario de datos

Este Anexo proporciona definiciones y explicaciones de los términos clave utilizados a lo largo del documento. Este es de gran ayuda para comprender el contexto y los conceptos relacionados con el Trabajo de Fin de Grado.

1. SAE: Sistema de Atención Electrónica utilizado en el Gobierno de Aragón para la gestión y tramitación de documentos y expedientes electrónicos relacionados con procedimientos administrativos. Su propósito es proporcionar una infraestructura digital que facilite la gestión de la documentación y los trámites administrativos de forma eficiente y accesible.

2. CSV: Código Seguro de Verificación. Es un código único que identifica a cada documento o expediente generado por el Gobierno de Aragón. Permite verificar el contenido, autenticidad de las firmas y la integridad de los documentos almacenados en el gestor documental, generalmente visible en los márgenes del documento.

3. CCSV: Servicio de almacenamiento y verificación de Documentos electrónicos. Es un sistema de consulta de documentos mediante CSV es una aplicación que permite a los ciudadanos y empleados públicos la consulta de los documentos pertenecientes a trámites o expedientes en los que están involucrados.

4. Documento: unidad de información en formato digital que contiene datos estructurados, como archivos de texto, imágenes, audios, entre otros. En el contexto del SAE, es la información que se gestiona, visualiza y procesa en el sistema. En este caso, almacena la información de los documentos electrónicos, tales como identificadores (entre ellos el CSV), tipos, nombres, formatos, y contenido en el sistema.

5. Expediente: conjunto de documentos y otros materiales asociados a un trámite administrativo o proceso específico. En el SAE, el expediente integra todos

los archivos que forman parte de un caso o procedimiento. En el sistema, almacena la información relacionada con el expediente, incluyendo identificadores (como el CSV), el tipo, el estado, la fecha de apertura y cierre, entre otros datos relacionados. El expediente tiene documentos asignados, así como carpetas opcionales que agrupan dichos documentos.

6. AMD Aragón: Administración Electrónica de Aragón, plataforma institucional encargada de gestionar digitalmente los procedimientos administrativos. Agrupa diversos servicios como el SAE, para optimizar los trámites y la interacción entre los ciudadanos y la administración pública.

7. Sistema legado: sistema informático que ha quedado obsoleto pero que sigue siendo utilizado por el usuario y no se quiere o no se puede reemplazar o actualizar de forma sencilla.

Anexos B

Planificación

Se diseñó un Diagrama de Gantt, como se observa en la Figura B.1, para mostrar de forma clara el cronograma del proyecto en relación con sus etapas principales y los meses de trabajo. Este diagrama permite visualizar las fases clave y las tareas organizadas en el tiempo, teniendo en cuenta las fechas establecidas para el inicio y fin del proyecto, así como la estimación del tiempo requerido para cada actividad.

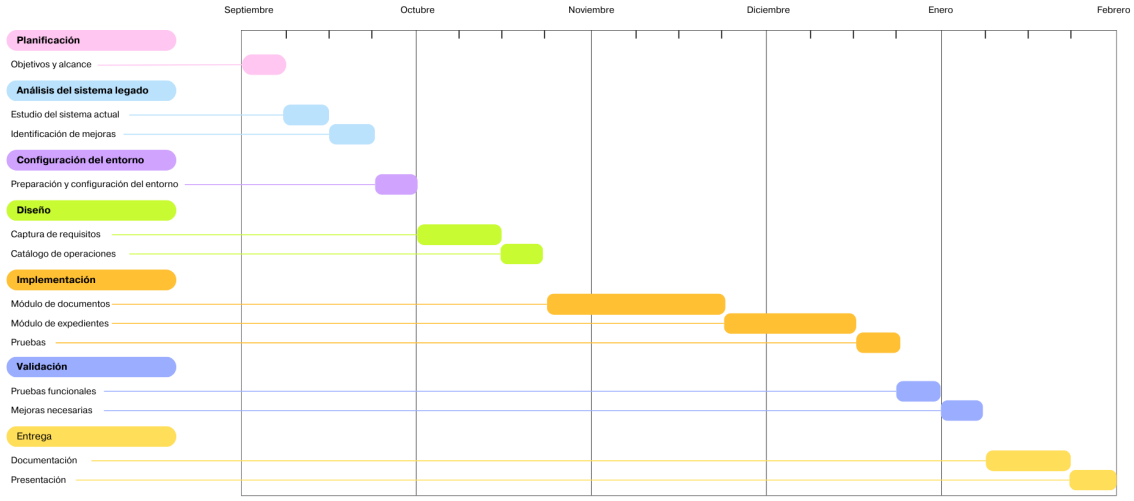

Figura B.1: Diagrama de Gantt

A diferencia del diagrama de Gantt mostrado en la Figura B.1, que presenta una estimación del tiempo necesario para el desarrollo del proyecto, la tabla de la Figura B.1 proporciona un detalle de las horas reales dedicadas a cada tarea y subtarea. Aunque las tareas y subtareas de esta tabla coinciden con las del diagrama de Gantt, hay diferencias notables entre ambas representaciones.

Realizar la estimación de horas fue un desafío, ya que era una de las primeras

veces que se abordaba este tipo de proyecto, lo que hizo complicado calcular con exactitud las horas. Además, debido a la naturaleza de las primeras estimaciones y la falta de experiencia, la precisión de estas fue limitada, como se puede evidenciar en la comparación entre lo previsto y lo realmente invertido en cada actividad.

La principal diferencia que se observa entre la estimación y la realidad es que se subestimó el tiempo necesario para la documentación, mientras que, en la práctica, este proceso resultó ser más extenso de lo previsto. Un desafío importante del proyecto fue la falta de documentación adecuada sobre la librería que se deseaba integrar. Para mitigar este problema, decidí dedicarme más a la documentación, con el objetivo de facilitar el trabajo del siguiente desarrollador que se encargue de este proyecto, para que no se enfrente a la misma incertidumbre con la que me encontré. Aunque la documentación final no es excesivamente detallada, tuve que ir adaptándola a medida que avanzaba, realizando modificaciones en ella conforme introducía cambios en el código.

Otro aspecto notable es que se destinó más tiempo de lo estimado para la planificación y el análisis de requisitos, aunque el proyecto ya estaba bastante definido cuando me proporcionaron el tema. Esto quiere decir que, a pesar de haber estimado más tiempo para estas fases, realmente no necesité tanto, ya que gran parte de las decisiones y detalles ya estaban establecidos. El tiempo ahorrado en estas fases lo aproveché para concentrarme más en la documentación, dando prioridad a este aspecto del proyecto dado los desafíos encontrados.

Tabla B.1: Horas reales dedicadas a cada tarea del proyecto

Tarea	**Subtarea**	**Horas reales**	**Total por tarea**
Planificación	Objetivos y alcance	19	19
Análisis del sistema legado	Estudio del sistema actual	15	21
	Identificación de mejoras	6	
Configuración del entorno	Preparación y configuración del entorno	9	9
Diseño	Captura de requisitos	10	21
	Catálogo de operaciones	11	
Implementación	Módulo de documentos	70	148
	Módulo de expedientes	49	
	Pruebas	10	
Validación	Pruebas funcionales	7	17
	Mejoras necesarias	10	
Entrega	Documentación	76	91
	Presentación	15	
	Total de horas	**326**	

Anexos C

Arquitectura de un proyecto para el Gobierno de Aragón

Los proyectos desarrollados para el Gobierno de Aragón emplean diversas tecnologías para desarrollar tanto el backend como el frontend. El backend está construido con *Java* y *Servlets*, utilizados para implementar la lógica de negocio y gestionar la conexión con la base de datos. Por otra parte, el *frontend* se desarrolla en *Angular*, siguiendo una estructura basada en módulos, componentes y servicios. Sin embargo, esta última parte no está directamente relacionada con el desarrollo de este proyecto.

En los proyectos desarrollados para el Gobierno de Aragón, se emplea el patrón Modelo-Vista-Controlador (MVC) como arquitectura base (reflejado en la Figura C.1). Este patrón separa claramente las responsabilidades de la aplicación en tres componentes principales:

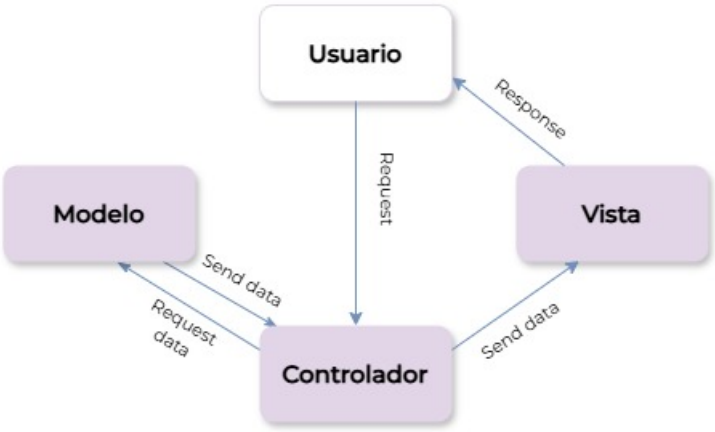

Figura C.1: Patrón MVC

– **Modelo**: maneja los datos y la lógica de negocio de la aplicación. Representa la estructura de los datos y es responsable de la interacción con la base de datos.

– **Vista**: controla la presentación de los datos al usuario. Es el encargado de mostrar interfaces amigables y dinámicas para facilitar la interacción con la aplicación.

– **Controlador**: actúa como intermediario entre el modelo y la vista. Recibe las entradas del usuario, las procesa, y determina qué se debe mostrar en la vista o cómo actualizar el modelo.

Para profundizar más, se puede observar el funcionamiento de un proyecto diseñado para el Gobierno de Aragón en la figura C.2.

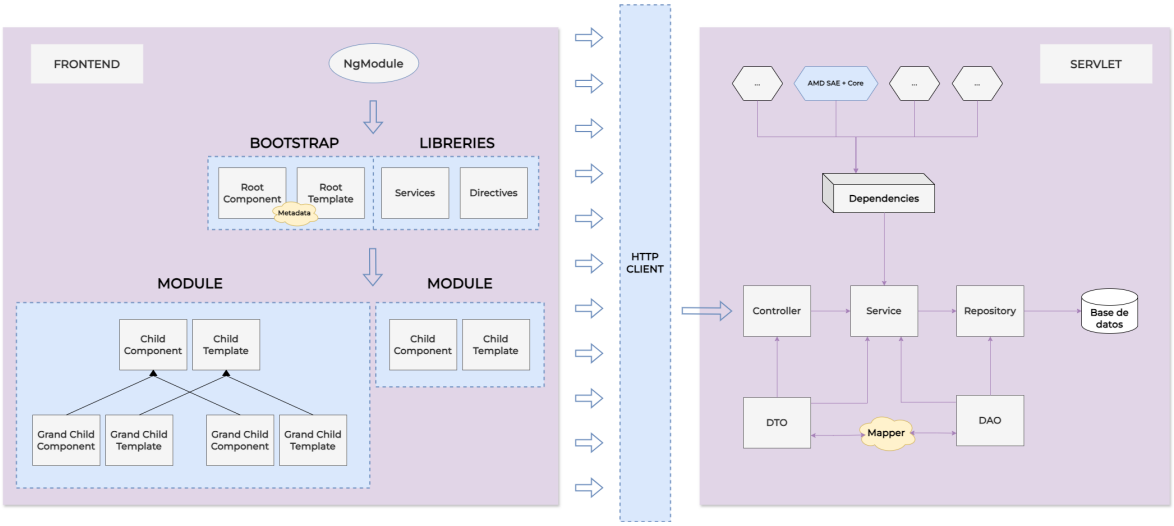

Figura C.2: Diseño de los programas del Gobierno de Aragón

C.1. Componentes del frontend

En la parte frontal, se utiliza Angular, un framework de desarrollo web altamente eficiente que organiza las aplicaciones en módulos, componentes y servicios. Cada elemento cumple funciones específicas para mantener la estructura y la claridad del código. A continuación, se explican los principales componentes del proyecto Angular:

1. **Ng Module**: es el núcleo organizativo de Angular. Agrupa componentes, servicios, directivas y otros recursos relacionados. Además, proporciona el contexto necesario para la inyección de dependencias y facilita la configuración de las rutas en la aplicación.

2. **Bootstrap**: un marco de diseño basado en CSS y JavaScript que ayuda a crear interfaces de usuario responsivas y visualmente atractivas. Su integración asegura que la aplicación funcione y se vea correctamente en dispositivos de distintos tamaños.

3. **Librerías**: estas incluyen bibliotecas de terceros o personalizadas, utilizadas para funciones específicas como la representación de gráficos, autenticación de usuarios o manipulación avanzada de datos. Aportan flexibilidad y reducen la complejidad del desarrollo.

4. **Root Component**: es el componente principal que actúa como punto de entrada de la interfaz de usuario. Este componente se carga en el DOM y contiene referencias a otros componentes secundarios.

5. **Metadata**: Angular utiliza metadatos, como decoradores, para definir configuraciones adicionales en componentes, módulos y servicios. Por ejemplo, rutas para la navegación entre vistas o información para facilitar la carga de dependencias.

6. **Root Module**: este es el módulo principal de la aplicación que define configuraciones globales y las dependencias esenciales. Sirve como un mapa para conectar y gestionar los diferentes módulos que forman parte del sistema.

7. **Services**: los servicios son componentes reutilizables que manejan la lógica de negocio, la comunicación con el servidor y otras tareas críticas. Normalmente, los métodos que contienen las peticiones HTTP se definen aquí.

8. **Directives**: directivas que permiten modificar el comportamiento o el estilo de elementos HTML dentro de las plantillas. Pueden ser estructurales (añadir/eliminar elementos del DOM) o de atributo (modificar propiedades del DOM).

C.2. Componentes del backend

En la parte trasera, los proyectos hacen uso de Servlets para procesar las peticiones HTTP y garantizar el correcto flujo de datos entre la aplicación y los usuarios. Estos componentes son esenciales para orquestar la lógica de negocio y la persistencia de datos:

1. **Controlador (Controller)**: Se encarga de recibir las peticiones HTTP entrantes desde el cliente, validar la información proporcionada y delegar las operaciones al servicio correspondiente.

2. **Servicio (Service)**: Implementa la lógica de negocio de la aplicación. Aquí se procesan datos, se aplican reglas de negocio y se coordinan las operaciones en la base de datos o con otros sistemas externos.

3. **Repositorio (Repository)**: Define las operaciones necesarias para interactuar con la base de datos. Implementa métodos para consultar, actualizar o eliminar información almacenada.

4. **DAO (Data Access Object)**: Este patrón de diseño proporciona una capa de abstracción adicional para la interacción con la base de datos. Ofrece métodos genéricos para realizar operaciones CRUD (crear, leer, actualizar, eliminar) y otras consultas específicas.

5. **DTO (Data Transfer Object)**: Los DTO se utilizan para transportar datos entre las distintas capas de la aplicación. Facilitan el traslado de información estructurada de forma eficiente y segura.

6. **Mapper**: Traduce entre objetos de las capas de negocio (DTO) y las entidades de la base de datos. Esta conversión permite mantener separada la lógica de negocio de la lógica de acceso a datos.

7. **Base de Datos (Database)**: Es el sistema donde se almacenan todos los datos persistentes. En estos proyectos suele utilizarse una base de datos relacional, que organiza los datos de manera estructurada para facilitar su consulta y manipulación.

8. **Dependencias (Dependencies)**: Incluyen bibliotecas, frameworks y otros recursos que el backend necesita para cumplir con sus funcionalidades. Por ejemplo, bibliotecas para manejar el protocolo HTTP, validación de datos o integración con otras APIs.

Anexos D

Detalles de implementación

Este anexo se adentra en los detalles del código implementado, centrándose en destacar aquellas partes que sobresalen por su relevancia y complejidad, ofreciendo una visión profunda de los aspectos más significativos del desarrollo.

D.1. Infraestructura

A continuación se presenta el código implementado relacionado con la infraestructura y su configuración.

D.1.1. ClientCCSV

La configuración de la clase *ClientCCSVProviderImpl* permite conectar con dos servicios externos, el de documentos y el de expedientes, a través de proxies que utilizan *SOAP* (Simple Object Access Protocol). Esto se lleva a cabo mediante el uso de *Apache CXF*, un framework que facilita la interacción con servicios web.

Esta clase recibe un objeto de configuración *CCSVProviderConfig* como parámetro en su constructor. Este contiene las URLs base de los servicios de documentos y expedientes, junto con otros posibles parámetros.

Para configurar el cliente del servicio de documentos, se utiliza una instancia de *ClientProxyFactoryBean*, que actúa como una fábrica para crear clientes que interactúan con servicios web. En primer lugar, se define la interfaz que implementará la clase cliente, estableciendo que esta utilizará la interfaz *IDocumentMetadataSignatureService*, la cual contiene los métodos disponibles para el servicio de documentos. A continuación, se especifican las URLs necesarias, configurando tanto la dirección base del servicio mediante *documentExpedientProviderConfig.getUrlDocument()* como la ubicación del archivo *WSDL* añadiendo *?wsdl*, el cual proporciona la descripción detallada de las operaciones soportadas por el servicio. Posteriormente, se configura el mapeo de objetos Java a

XML mediante la clase *AegisDatabinding*, lo que permite una conversión automática y eficiente entre estos formatos, adaptada a las necesidades de las operaciones *SOAP*. También se habilita el mecanismo *MTOM* (*Message Transmission Optimization Mechanism*) para optimizar la transferencia de datos binarios, especialmente útil en el manejo de archivos grandes como imágenes y documentos. Por último, se procede a la creación del cliente proxy, denominado *iDocumentMetadataSignatureService*, el cual se utiliza como punto de acceso para interactuar directamente con el servicio y realizar las operaciones requeridas. Tal como se ve en el código del apartado **??**.

Para la configuración del cliente del servicio de expedientes, el proceso es bastante similar al de los documentos. Se sigue el mismo procedimiento de creación del *ClientProxyFactoryBean*, definición de la interfaz a implementar, y configuración de las URLs correspondientes, que en este caso se obtienen de *documentExpedientProviderConfig.getUrlExpedient()*. La principal diferencia radica en la interfaz utilizada para el servicio de expedientes, que en este caso es *IAdministrativeFileService*, en lugar de la interfaz de documentos. Al igual que en el caso anterior, se configura el mapeo de objetos Java a XML mediante *AegisDatabinding* y se habilita *MTOM* para optimizar la transferencia de datos binarios. Una vez configurado todo, se crea un cliente proxy denominado *iAdministrativeFileService*, que permite interactuar con el servicio de expedientes de la misma manera que se hace con el servicio de documentos.

Listado D.1: Constructor de la clase ClientCCSVProviderImpl

```
public ClientCCSVProviderImpl(CCSVProviderConfig documentExpedientProviderConfig) {
      /** Cliente CCSV Documentos **/
      ClientProxyFactoryBean proxyFactoryDoc = new ClientProxyFactoryBean();
      // Establece la interfaz que define los metodos del servicio
      proxyFactoryDoc.setServiceClass(IDocumentMetadataSignatureService.class);
      // Configura la URL base del servicio para documentos
      proxyFactoryDoc.setAddress(documentExpedientProviderConfig.getUrlDocument());
      // Configura la URL del archivo WSDL para describir el servicio
      proxyFactoryDoc.setWsdlURL(documentExpedientProviderConfig.getUrlDocument() + "?wsdl
      // Configura el metodo de binding (mapeo entre objetos Java y XML)
      proxyFactoryDoc.getServiceFactory().setDataBinding(new AegisDatabinding());
      // Activa MTOM para optimizar la transferencia de datos binarios
      HashMap<String, Object> properties = new HashMap<String, Object>();
      properties.put("mtom-enabled", "true");
      proxyFactoryDoc.setProperties(properties);

      // Crea el cliente proxy para interactuar con el servicio de documentos
      IDocumentMetadataSignatureService iDocumentMetadataSignatureService = (IDocumentMeta
            .create();

      /** Cliente CCSV Expedientes **/
```

```
        ClientProxyFactoryBean proxyFactoryExp = new ClientProxyFactoryBean();
        // Establece la interfaz que define los metodos del servicio
        proxyFactoryExp.setServiceClass(IAdministrativeFileService.class);
        // Configura la URL base del servicio para documentos
        proxyFactoryExp.setAddress(documentExpedientProviderConfig.getUrlExpedient());
        // Configura la URL del archivo WSDL para describir el servicio
        proxyFactoryExp.setWsdlURL(documentExpedientProviderConfig.getUrlExpedient() + "?wsd
        // Configura el metodo de binding (mapeo entre objetos Java y XML)
        proxyFactoryExp.getServiceFactory().setDataBinding(new AegisDatabinding());
        // Activa MTOM para optimizar la transferencia de datos binarios
        HashMap<String, Object> properties2 = new HashMap<String, Object>();
        properties2.put("mtom-enabled", "true");
        proxyFactoryExp.setProperties(properties2);

        // Crea el cliente proxy para interactuar con el servicio de expedientes
        IAdministrativeFileService iAdministrativeFileService = (IAdministrativeFileService)

        // this.peticionesSaeProvider = new
        // PeticionesSaeProviderImpl(documentProviderConfig, clienteCcsv);
        this.documentProvider = new DocumentProviderImpl(documentExpedientProviderConfig,
                iDocumentMetadataSignatureService);
        this.expedientProvider = new ExpedientProviderImpl(documentExpedientProviderConfig,
                iDocumentMetadataSignatureService);
    }
```

Luego se definieron las funciones, organizándolas en dos grupos según su tipo: una parte para los documentos y otra para los expedientes. Para facilitar la gestión de estas funciones, se crearon dos proveedores específicos: uno para manejar los documentos y otro para los expedientes, de modo que cada uno se encargara de las operaciones correspondientes a su tipo de entidad.

D.1.2. Documento

La clase *DocumentProviderImpl* está diseñada para integrarse con el servicio de gestión de documentos, y su configuración comienza desde el constructor. Este constructor requiere un objeto del tipo *CCSVProviderConfig*, el cual contiene información crucial sobre la entidad que está utilizando la librería, además de las credenciales necesarias para acceder a *SAE*, entre otros datos importantes. Toda esta información se transmite a través del *ClientCCSVProvider*, que actúa como el único punto de conexión con el usuario de la librería. Al ser el punto inicial de la integración, esta información se va propagando entre clases, llegando hasta su uso en la última parte del flujo.

Un elemento central en la integración es el servicio *iDocumentMetadataSignatureService*, que es responsable de ejecutar las funciones

puras de *SAE*. Este servicio también es configurado en *ClientCCSVProvider* y se pasa entre clases hasta llegar a *PeticionesSae*, que es donde realmente se emplea. Esta decisión de configurar el servicio en la clase cliente y luego transferirlo entre clases permite evitar la transmisión directa de todos los detalles de configuración del servicio. De este modo, se logra encapsular el servicio para que sea accesible cuando sea necesario, sin necesidad de pasar todos los parámetros de configuración en cada paso del flujo, simplificando el proceso.

Finalmente, se crea un objeto de *PeticionesSae*, en el cual se encuentran las funciones que llaman directamente al servicio *SAE*. Este objeto también recibe la configuración necesaria y el servicio de documentos, completando el ciclo de integración. Por último, se capturan y asignan los valores pertinentes de la configuración, permitiendo a la clase *DocumentProviderImpl* estar completamente preparada para funcionar dentro del contexto de la integración.

Listado D.2: Constructor de la clase DocumentCCSVProviderImpl

```
public ClientCCSVProviderImpl(CCSVProviderConfig documentExpedientProviderConfig) {
     public DocumentProviderImpl(CCSVProviderConfig documentProviderConfig,
        IDocumentMetadataSignatureService iDocumentMetadataSignatureService) {
     log.info("DocumentProviderImpl() - Hi");

     this.iDocumentMetadataSignatureService = iDocumentMetadataSignatureService;

     this.peticionesSaeProvider = new PeticionesSaeProviderImpl(documentProviderConfig, i

     this.codigoOrganismo = documentProviderConfig.getOrganismCode();
     this.nombreAportadorInteresado = documentProviderConfig.getInterestedContributorName
     this.nifAportadorInteresado = documentProviderConfig.getNifInterestedContributor();
     this.organismoProductorNombre = documentProviderConfig.getOrganismProducerName();
     this.procedimientoNombre = documentProviderConfig.getProcedureName();
     this.procedimientoNumero = documentProviderConfig.getProcedureCode();
     this.appCode = documentProviderConfig.getAppCode();
     this.appName = documentProviderConfig.getAppName();
     this.suffixAppCCSV = documentProviderConfig.getSuffixAppCCSV();

  }
```

Una vez realizada la configuración, se procede a implementar la lógica correspondiente a las funciones necesarias.

D.1.3. Expediente

En el caso del proveedor de expedientes, la configuración sigue una lógica muy similar a la del proveedor de documentos. La diferencia radica en que la clase *ExpedientProviderImpl* no solo utiliza su propio servicio, el

iAdministrativeFileService, sino que también hace uso del servicio de documentos, *iDocumentMetadataSignatureService*. Al igual que en el caso anterior, el constructor de esta clase recibe un objeto de tipo *CCSVProviderConfig* que contiene los datos de configuración necesarios, como el código del organismo, el nombre del aportador interesado, y otros parámetros fundamentales para la conexión con el sistema.

En este caso, también se configura un objeto de *PeticionesSaeProviderImpl*, el cual se pasa la configuración y los dos servicios necesarios: *iDocumentMetadataSignatureService* y *iAdministrativeFileService*. Este objeto es el que maneja las funciones que interactúan directamente con el servicio *SAE*, de manera que se centralizan en un único lugar las operaciones que requieren ambos servicios, el de expedientes y el de documentos.

Al igual que en el *DocumentProviderImpl*, los detalles de la configuración del servicio se propagan entre las clases sin necesidad de pasar todos los parámetros repetidamente. Además, la clase *ExpedientProviderImpl* captura los valores necesarios de la configuración proporcionada por el objeto *CCSVProviderConfig*, lo cual permite una integración ordenada y eficiente.

Listado D.3: Constructor de la clase ExpedientCCSVProviderImpl

```
public ExpedientProviderImpl(CCSVProviderConfig expedientProviderConfig,
        IAdministrativeFileService iAdministrativeFileService,
        IDocumentMetadataSignatureService iDocumentMetadataSignatureService) {
    log.info("ExpedientProviderImpl() - Hi");

    this.iAdministrativeFileService = iAdministrativeFileService;
    this.iDocumentMetadataSignatureService = iDocumentMetadataSignatureService;

    this.peticionesSaeProvider = new PeticionesSaeProviderImpl(expedientProviderConfig,
         iDocumentMetadataSignatureService, iAdministrativeFileService);

    this.codigoOrganismo = expedientProviderConfig.getOrganismCode();
    this.nombreAportadorInteresado = expedientProviderConfig.getInterestedContributorNam
    this.nifAportadorInteresado = expedientProviderConfig.getNifInterestedContributor();
    this.organismoProductorNombre = expedientProviderConfig.getOrganismProducerName();
    this.procedimientoNombre = expedientProviderConfig.getProcedureName();
    this.procedimientoNumero = expedientProviderConfig.getProcedureCode();
    this.appCode = expedientProviderConfig.getAppCode();
    this.appName = expedientProviderConfig.getAppName();
    this.suffixAppCCSV = expedientProviderConfig.getSuffixAppCCSV();
}
```

D.1.4. PeticionesSae

La configuración de la clase *PeticionesSaeProviderImpl* es bastante simple. En su constructor se reciben tres parámetros: un objeto del tipo *CCSVProviderConfig*, que contiene la configuración necesaria para el acceso a los servicios; el servicio *IDocumentMetadataSignatureService*, utilizado para interactuar con los documentos; y el servicio *IAdministrativeFileService*, utilizado para interactuar con los expedientes. Dentro del constructor, se asignan estos parámetros a las variables de instancia correspondientes: *documentProviderConfig*, *clienteDocumentoCCSV* y *clienteExpedienteCCSV*. Esto permite que la clase *PeticionesSaeProviderImpl* tenga acceso a toda la información necesaria y a los servicios que serán utilizados en sus funciones. La configuración es básica, pero crucial, ya que permite que los diferentes componentes interactúen entre sí de manera adecuada para el funcionamiento de la librería.

Listado D.4: Constructor de la clase PeticionesSaeCCSVProviderImpl

```
protected PeticionesSaeProviderImpl(CCSVProviderConfig documentProviderConfig,
        IDocumentMetadataSignatureService clienteDocumentoCCSV, IAdministrativeFileServic
    log.info("PeticionesSae() - Hi");
    this.documentProviderConfig = documentProviderConfig;
    this.clienteDocumentoCCSV = clienteDocumentoCCSV;
    this.clienteExpedienteCCSV = clienteExpedienteCCSV;
  }
```

D.1.5. CCSVExternal

La clase `CCSVExternalProvider` actúa como un puente entre los proyectos desarrollados con Spring Boot y los servicios proporcionados por SAE. Configurada con la anotación `@Configuration`, esta clase crea e inyecta automáticamente un cliente (`ClientCCSVProvider`) que proporciona un punto de acceso centralizado y eficiente para interactuar con los servicios de documentos y expedientes. Gracias a esta configuración, los proyectos que utilicen Spring Boot pueden conectarse de manera sencilla y coherente con los servicios de SAE, sin necesidad de gestionar manualmente los parámetros de conexión.

Para obtener la configuración necesaria, *CCSVExternalProvider* se basa en la clase *AmmSaeCCSVConfig*, que contiene todos los parámetros esenciales como las URLs de los servicios de documentos y expedientes, los códigos de procedimiento, y otros detalles relevantes. Esta configuración se obtiene automáticamente del archivo de propiedades (como `application.properties` o `application.yml`) mediante la anotación `@ConfigurationProperties(prefix = ''ccsv'')`, lo que permite a Spring

asignar de forma automática los valores del archivo de configuración a las propiedades de *AmmSaeCCSVConfig*.

La clase *AmmSaeCCSVConfig* mapea los valores como las URLs y los códigos de procedimiento desde el archivo de configuración a los campos correspondientes. Por ejemplo, cuando se define en el archivo de propiedades una línea como `ccsv.urlDocument=http://url-documento.com`, Spring asigna el valor `http://url-documento.com` al campo *urlDocument* de la clase *AmmSaeCCSVConfig*. Una vez que *AmmSaeCCSVConfig* tiene los parámetros configurados, *CCSVExternalProvider* utiliza esta configuración para crear una instancia de *ClientCCSVProviderImpl*, un cliente que permite gestionar los documentos y expedientes de manera adecuada.

Este enfoque centraliza la configuración y facilita la reutilización del cliente en diferentes partes del sistema sin necesidad de gestionar manualmente los parámetros de configuración en cada clase que los requiere. Con esta solución, *CCSVExternalProvider* asegura una integración coherente y eficiente con los servicios de CCSV, al mismo tiempo que minimiza la complejidad de configurar repetidamente los mismos parámetros a lo largo del sistema. El código correspondiente a lo explicado en este apartado se puede consultar en el Anexo D.1.5.

Listado D.5: Constructor de la clase CCSVExternalProvider

```
@Configuration
@ComponentScan(basePackageClasses = {CCSVProviderConfig.class,
    AmmSaeCCSVConfig.class})
@Slf4j
public class CCSVExternalProvider {

   @Bean
    public ClientCCSVProvider getClientCCSVProvider
        (AmmSaeCCSVConfig ammSaeCCSVConfig) {
        log.debug("[ {} ] es.aragon.core.sae.ccsv.providers.CCSVExternalProvider =>
            Inicio getClientCCSVProvider", LocalDateTime.now());

        // Configuracion del CCSVProviderConfig con constantes especificas
        CCSVProviderConfig config = CCSVProviderConfig.builder()
                .appCode(ammSaeCCSVConfig.getAppCode())
                .appName(ammSaeCCSVConfig.getAppName())
                .urlDocument(ammSaeCCSVConfig.getUrlDocument())
                .urlExpedient(ammSaeCCSVConfig.getUrlExpedient())
                .suffixAppCCSV(ammSaeCCSVConfig.getSuffixAppCCSV())
                .getDocumentAdv(ammSaeCCSVConfig.isGetDocumentAdv())
                .organismCode(ammSaeCCSVConfig.getOrganismCode())
                .interestedContributorName(ammSaeCCSVConfig
                    .getInterestedContributorName())
                .nifInterestedContributor(ammSaeCCSVConfig
```

```
                .getNifInterestedContributor())
            .organismProducerName(ammSaeCCSVConfig
                .getOrganismProducerName())
            .procedureName(ammSaeCCSVConfig.getProcedureName())
            .procedureCode(ammSaeCCSVConfig.getProcedureCode())
            .build();

        // Crear instancia de ClientCCSVProvider utilizando el
        //  config
        return new ClientCCSVProviderImpl(config);
    }

}
```

D.2. Flujo de integración

En esta sección se presentan los diagramas de secuencia correspondientes a todas las operaciones desarrolladas durante el proyecto. Estos diagramas son una herramienta clave para visualizar y comprender cómo interactúan los diferentes componentes del sistema en cada caso de uso.

Cada diagrama ilustra el flujo de mensajes entre las entidades del sistema, destacando las acciones que se llevan a cabo desde la invocación inicial hasta la finalización de la operación.

Crear documento

El diagrama de la Figura E.2 representa el flujo de la operación *Crear documento*, responsable de gestionar la creación de un nuevo documento en el sistema.

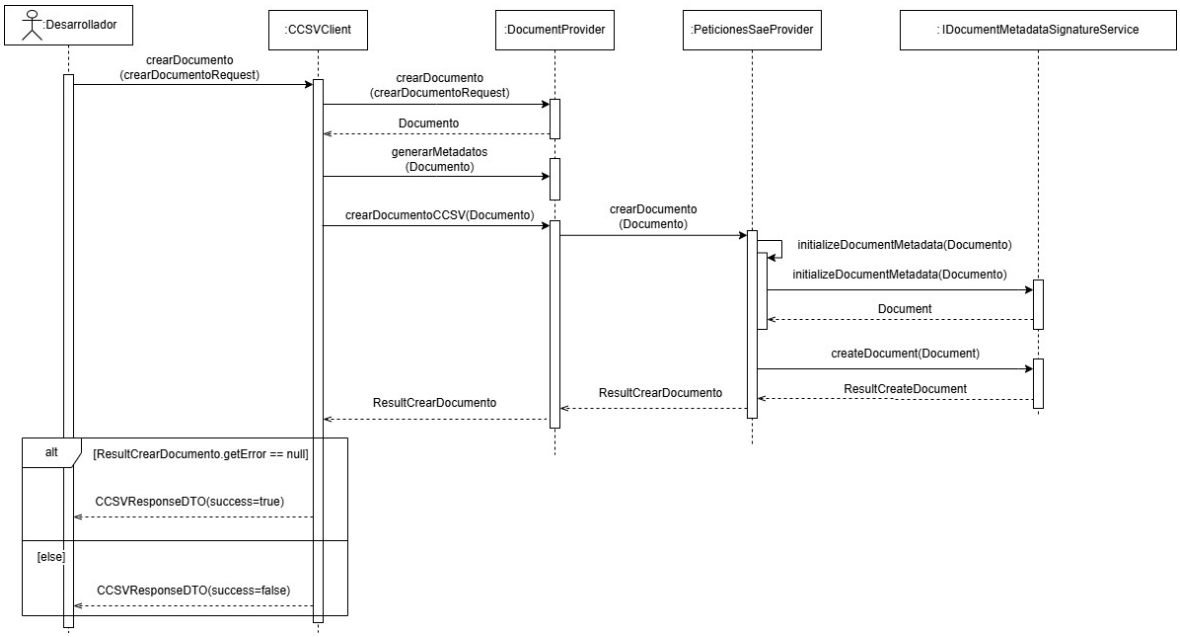

Figura D.1: Diagrama de secuencia de la operación *Crear operación*

Obtener documento

El diagrama de la Figura D.2 muestra el flujo de la operación *Obtener documento*, encargada de recuperar un documento específico del sistema.

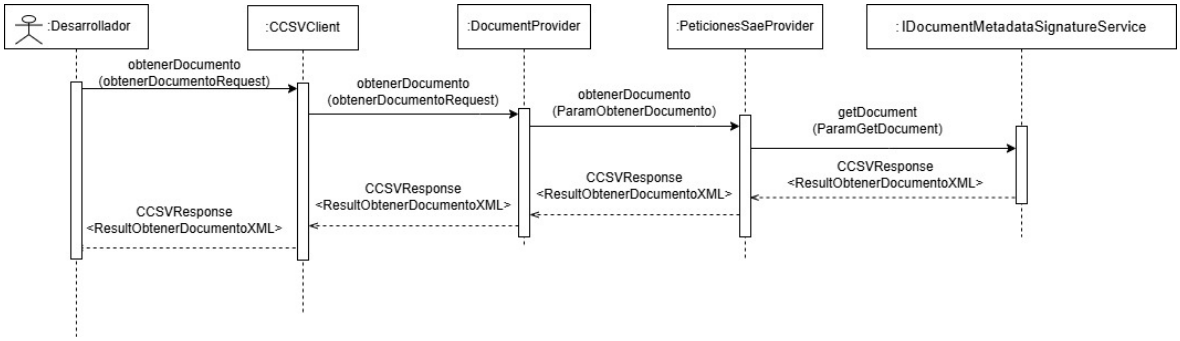

Figura D.2: Diagrama de secuencia de la operación *Obtener documento*

Obtener documento XML

El diagrama de la Figura D.3 describe el flujo de la operación *Obtener documento XML*, utilizada para extraer documentos en formato XML.

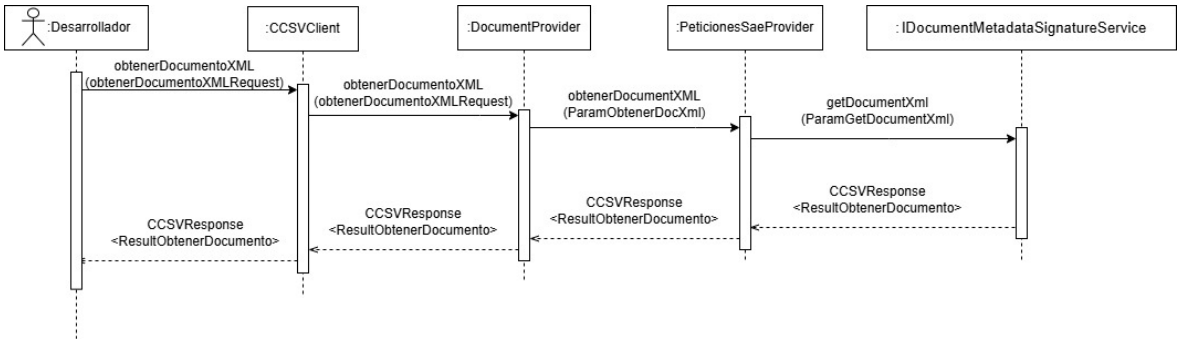

Figura D.3: Diagrama de secuencia de la operación *Obtener documento XML*

Actualizar documento

La Figura D.4 ilustra el flujo de la operación *Actualizar documento*, que actualiza los datos de un documento existente.

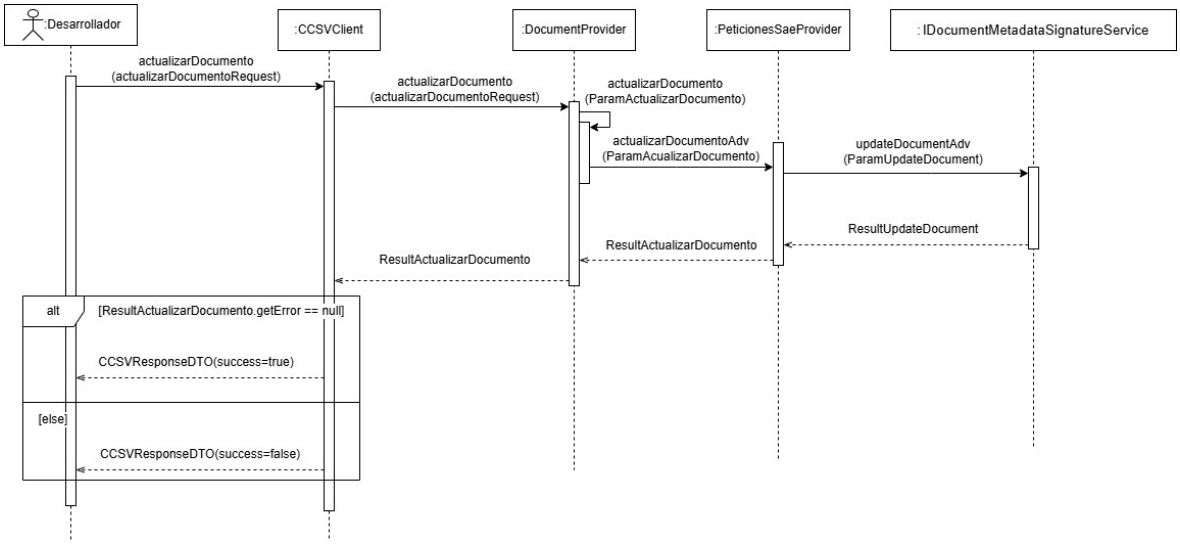

Figura D.4: Diagrama de secuencia de la operación *Actualizar documento*

Crear expediente

La operación *Crear expediente*, representada en la Figura D.5, genera un nuevo expediente en el sistema.

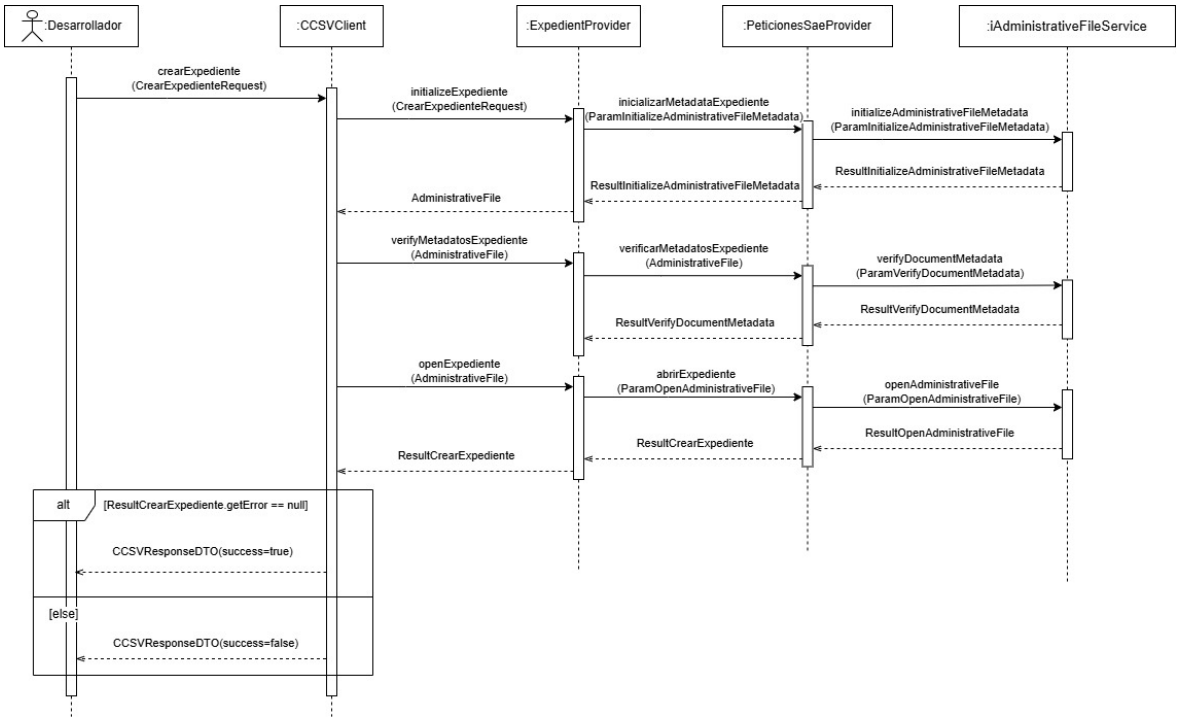

Figura D.5: Diagrama de secuencia de la operación *Crear expediente*

Añadir documentos al expediente

El diagrama en la Figura D.6 muestra el flujo de la operación *Añadir documentos al expediente*, que incorpora documentos a un expediente existente.

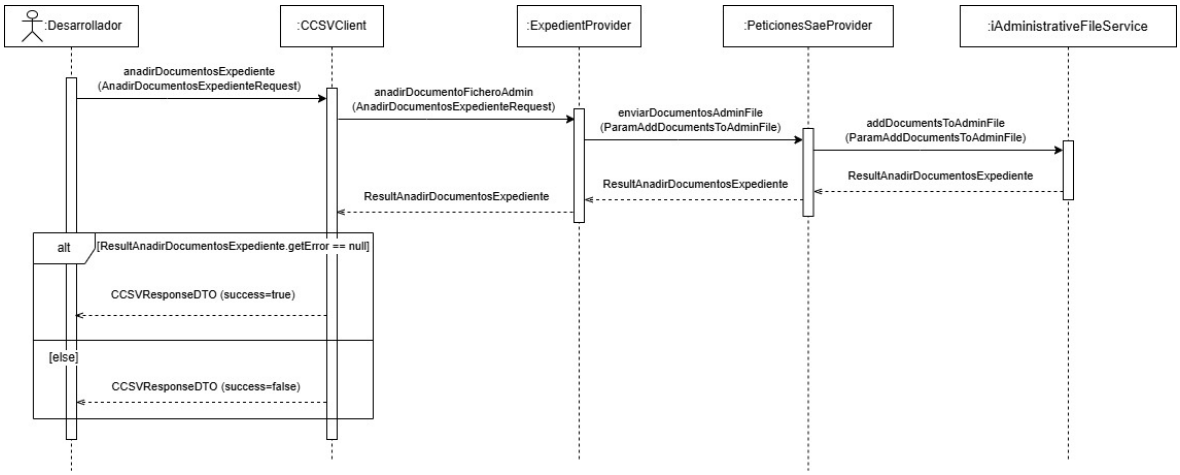

Figura D.6: Diagrama de secuencia de la operación *Añadir documentos al expediente*

Eliminar documentos del expediente

El flujo de la operación *Eliminar documentos del expediente* se presenta en la Figura D.7, mostrando cómo se eliminan documentos de un expediente.

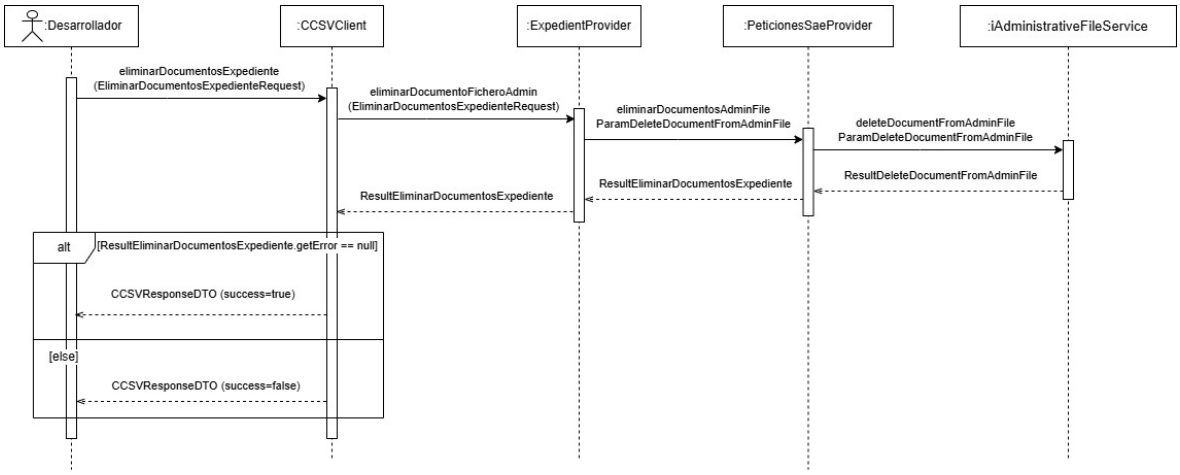

Figura D.7: Diagrama de secuencia de la operación *Eliminar documentos del expediente*

Regenerar índice del expediente

En la Figura D.8, se detalla el flujo de la operación *Regenerar índice del expediente*, que actualiza los índices de un expediente.

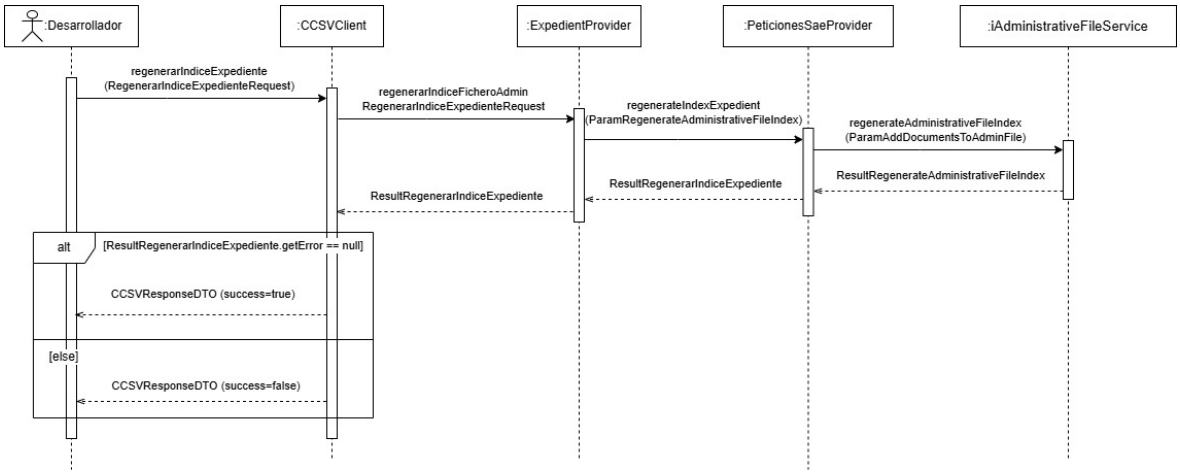

Figura D.8: Diagrama de secuencia de la operación *Regenerar índice del expediente*

Crear carpeta en expediente

El diagrama de la Figura D.9 representa el flujo de la operación *Crear carpeta en expediente*, que permite crear una nueva carpeta dentro de un expediente.

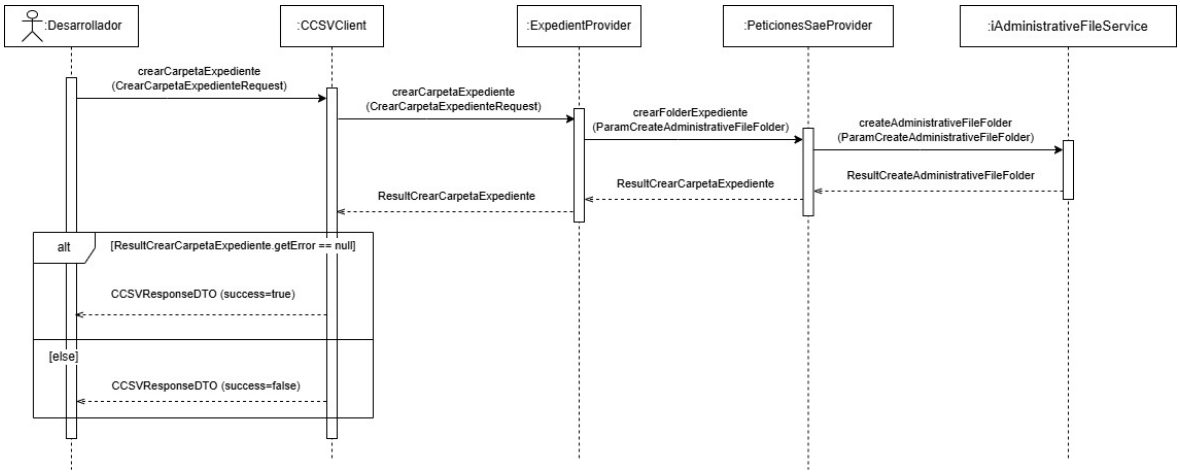

Figura D.9: Diagrama de secuencia de la operación *Crear carpeta en expediente*

Asociar un expediente a otro expediente

La operación *Asociar un expediente a otro expediente* está ilustrada en la Figura D.10, donde se visualiza cómo un expediente se relaciona con otro.

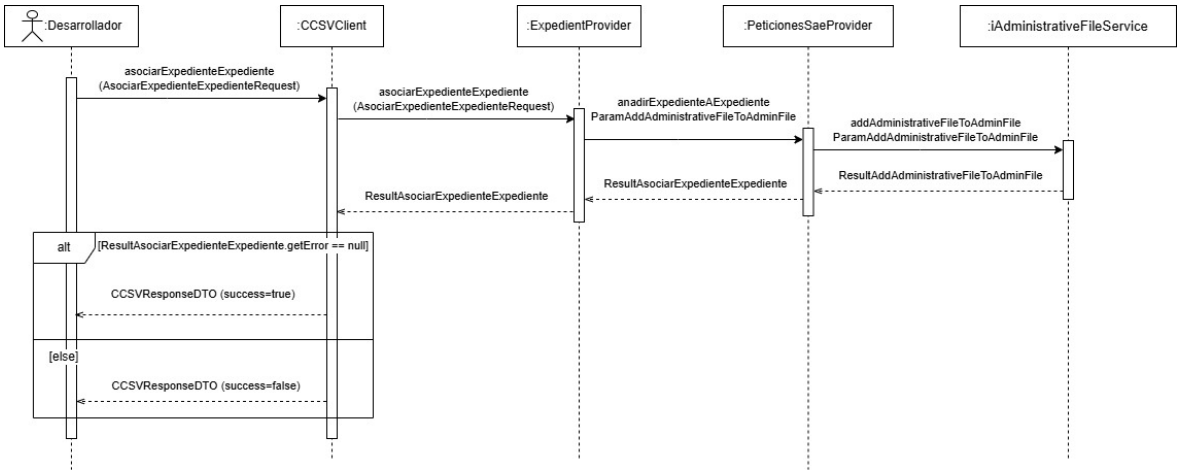

Figura D.10: Diagrama de secuencia de la operación *Asociar un expediente a otro expediente*

Obtener expediente

La Figura D.11 detalla el flujo de la operación *Obtener expediente*, encargada de recuperar la información de un expediente específico.

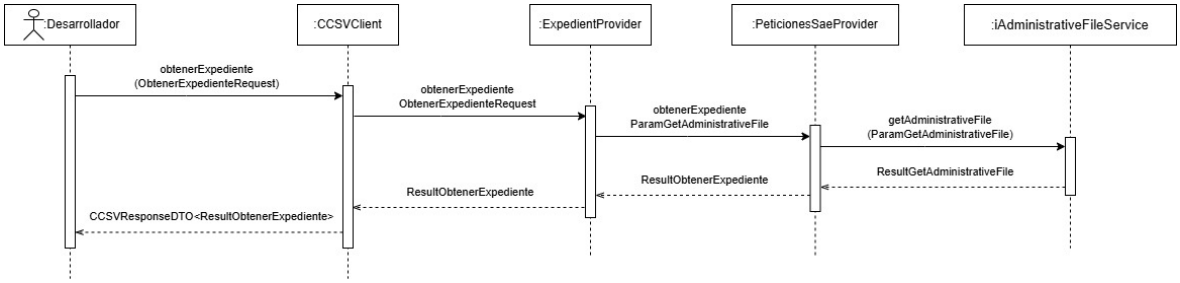

Figura D.11: Diagrama de secuencia de la operación *Obtener expediente*

D.3. Paquetes auxiliares

Con el fin de mantener limpio y organizado el flujo principal de la aplicación, se han creado dos paquetes auxiliares: utils y dtos. Cada uno de estos paquetes tiene un propósito específico y contribuye al correcto funcionamiento y escalabilidad del proyecto.

Paquete utils:

Este paquete se ha diseñado para almacenar todas las clases auxiliares que son necesarias a lo largo del flujo principal de la aplicación. Entre las clases más destacadas en este paquete se encuentran:

- **Constantes**: en esta clase se declaran valores constantes que permanecen inmutables, tales como metadatos, códigos de error, y otros parámetros que no cambian a lo largo del proyecto.

- **Utils**: Este conjunto de funciones auxiliares incluye métodos genéricos que facilitan tareas comunes, como conversiones de formato, cálculos y manipulaciones de datos. Estas funciones buscan reducir la repetición de código y hacer que el flujo sea más sencillo. Además, se integra con otros módulos del sistema, como el módulo Utils que proporciona herramientas adicionales de procesamiento y gestión de datos, favoreciendo la reutilización del código y la eficiencia en el proceso de integración.

- **Validaciones**: en esta clase se agrupan las validaciones necesarias para la ejecución de funciones clave dentro de la aplicación, garantizando que los

parámetros de entrada sean correctos y que el proceso se ejecute de manera adecuada.

Paquete dtos:

El paquete dtos alberga las declaraciones de los tipos de objetos que se utilizan a lo largo del proyecto. Estos objetos son esenciales para transportar datos de manera coherente y estructurada entre las distintas partes de la aplicación. Algunos de los DTOs importantes en este paquete son:

- **Documento**: representa un documento específico dentro del sistema de integración.
- **Expediente**: se utiliza para encapsular la información relacionada con un expediente administrativo o procesal.
- **Carpeta**: agrupa y organiza elementos o expedientes dentro de una estructura jerárquica.
- **Metadato**: representa información adicional o descriptiva asociada a los documentos o expedientes.
- **Agente**: se utiliza para encapsular información sobre los agentes involucrados en el proceso.
- **Error**: un objeto preparado para almacenar los detalles de cualquier error que se produzca durante las operaciones. Este DTO es utilizado especialmente cuando una función devuelve un error, permitiendo que se registre y gestione toda la información devuelta por SAE u otros sistemas externos.

El uso de estos paquetes auxiliares ayuda a mantener la modularidad, claridad y organización del código, lo que facilita tanto la comprensión como el mantenimiento del sistema en el largo plazo.

Anexos E

Pruebas y validaciones

En este capítulo se describen detalladamente las pruebas y validaciones realizadas durante el desarrollo del proyecto. Estas actividades han sido esenciales para garantizar la funcionalidad, estabilidad y fiabilidad de las operaciones implementadas. A través de diferentes tipos de pruebas, se ha evaluado el correcto comportamiento de las funciones desarrolladas y su integración en el sistema.

Las secciones siguientes cubren tres áreas clave: las pruebas unitarias diseñadas para evaluar cada operación de forma aislada, las pruebas de integración orientadas a casos de uso, y la validación de los endpoints mediante un entorno de pruebas creado específicamente para verificar su funcionamiento desde herramientas como Swagger.

E.1. Pruebas unitarias

Las pruebas unitarias son un tipo de prueba de software cuyo objetivo es validar de forma aislada y detallada el correcto funcionamiento de unidades individuales de código, como funciones, métodos o clases. Estas pruebas permiten identificar errores en etapas tempranas del desarrollo y aseguran que cada componente cumple con su propósito específico bajo diferentes escenarios.

Para validar la correcta implementación de las funcionalidades, se desarrollaron pruebas unitarias utilizando JUnit. Estas pruebas abarcan todas las operaciones del catálogo de la clase `ClientCCSVProviderImpl`, sin que exista una distinción en la relevancia de las mismas, ya que todas fueron consideradas igual de importantes para el sistema.

Durante el proceso, se simularon escenarios variados mediante el uso de dependencias simuladas, verificando que todas las interacciones y resultados de las operaciones fueran correctos. Los casos probados incluyen, entre otros, la creación, obtención y actualización de documentos, siempre asegurando que el sistema mantuviera su funcionalidad bajo diferentes condiciones.

E.1.1. Módulo de documentos

El módulo de documentos se encarga de realizar operaciones relacionadas con la creación, actualización, y obtención de documentos almacenados en el sistema.

Crear documento

La prueba unitaria del método *Crear documento* del Listado E.1 evalúa el proceso de creación y almacenamiento de un documento en el sistema. Verifica la correcta asignación de datos, la interacción con los servicios externos simulados mediante *mocks*, y la validación de la respuesta final.

Listado E.1: Prueba unitaria de la operación *Crear documento*

```
@Test
    public void testCrearDocumento() throws CCSVException {
        // Simulacion: Creacion del DTO de prueba con los datos necesarios
        CrearDocumentoRequestDto documentoSubir = new CrearDocumentoRequestDto();
        documentoSubir.setNombre("NombreEjemplo.pdf");

        // Simulacion: Leer archivo simulado y asignarlo al DTO
        File pdfFile =
            new File("C:\\Users\\ClaraCerdanTorrubias\\Downloads\\prueba 2.pdf");
        try (FileInputStream fis = new FileInputStream(pdfFile)) {
            byte[] pdfBytes = new byte[(int) pdfFile.length()];
            fis.read(pdfBytes);
            documentoSubir.setDocument(pdfBytes);
        } catch (Exception e) {
            fail("Error al leer el archivo PDF: " + e.getMessage());
        }

        // Simulacion: Crear el resultado esperado y el DTO a retornar por el mock
        DocumentDto documentDto = new DocumentDto();
        documentDto.setNombre("HOLA");
        documentDto.setContenido(documentoSubir.getDocument());

        ResultCrearDocumentoDTO result = new ResultCrearDocumentoDTO();
        result.setReturnCodigo("OK");

        // Configuracion de los mocks: Se simula la respuesta para los metodos
        llamados
        when(documentProvider
            .crearDocumentoDto(any(CrearDocumentoRequestDto.class)))
            .thenReturn(documentDto);
        when(documentProvider.crearDocumentoCCSV(any(DocumentDto.class)))
            .thenReturn(result);

        // Llamada: Se llama al matodo que estamos probando
        CCSVResponseDTO<String> responseCrear = clientCCSVProviderImpl
```

```
            .crearDocumento(documentoSubir);

        // Validaciones: Se verifican los resultados de la llamada
        assertNotNull(responseCrear);
        assertTrue(responseCrear.isSuccess());
        assertNull(responseCrear.getMessage());

        // Verificacion: Comprobamos que las interacciones con los mocks se
        // ejecutaron correctamente
        verify(documentProvider, times(1))
            .crearDocumentoDto(any(CrearDocumentoRequestDto.class));
        verify(documentProvider, times(1))
            .crearDocumentoCCSV(any(DocumentDto.class));
    }
```

Obtener documento

El método *Obtener documento* permite recuperar un documento almacenado en el sistema a partir de un identificador único, como un CSV. Esta operación asegura que los datos se devuelvan de manera correcta, incluyendo su contenido y metadatos asociados. Las pruebas (del Listado E.2) verifican la interacción adecuada con los servicios simulados y la consistencia de la respuesta entregada al usuario final.

Listado E.2: Prueba unitaria de la operación *Obtener documento*

```
@Test
    public void testObtenerDocumento() throws CCSVException {
        // Simulacion: Crear el DTO de peticion con un CSV de prueba
        ObtenerDocumentoRequestDto requestCsv =
                                    new ObtenerDocumentoRequestDto();
        requestCsv.setCsv("CSVRDOT44VOHU1MOGISS");

        // Simulacion: Crear el DTO de documento con el CSV
        // obtenido desde la peticion
        DocumentDto documentoDTO = new DocumentDto();
        documentoDTO.setNombre("DocumentoEjemplo.pdf");
        documentoDTO.setCsv(requestCsv.getCsv());
        documentoDTO.setContenido(new byte[]{1, 2, 3, 4});

        // Simulacion: Crear el resultado esperado con el DTO
        ResultObtenerDocumentoDTO result = new ResultObtenerDocumentoDTO();
        result.setDocumento(documentoDTO);

        // Simulacion: Crear la respuesta CCSVResponseDTO con exito y el
        // mensaje con el documento
        CCSVResponseDTO<ResultObtenerDocumentoDTO> responseGet =
            new CCSVResponseDTO<>();
        responseGet.setSuccess(true);
```

```
        responseGet.setMessage(result);

        // Configuracion de los mocks: Se simula la respuesta para la
        // llamada al metodo obtenerDocumento
        when(documentProvider
            .obtenerDocumentoDto(any(ObtenerDocumentoRequestDto.class)))
                .thenReturn(responseGet);

        // Llamada: Se llama al metodo bajo prueba
        CCSVResponseDTO<ResultObtenerDocumentoDTO> response =
            clientCCSVProviderImpl.obtenerDocumento(requestCsv);

        // Validaciones: Verificar los resultados de la llamada
        assertNotNull(response);
        assertTrue(response.isSuccess());
        assertNotNull(response.getMessage());
        assertEquals("CSVRDOT44VOHU1MOGISS",
            response.getMessage().getDocumento().getCsv());

        // Verificacion de interaccion con los mocks
        verify(documentProvider, times(1)).
            obtenerDocumentoDto(any(ObtenerDocumentoRequestDto.class));
    }
```

Actualizar documento

El método *Actualizar documento* se encarga de modificar los datos de un documento almacenado en el sistema, como su contenido, nombre o metadatos. Esta operación asegura que los cambios realizados son consistentes y cumplen con los requisitos del usuario. Las pruebas (del Listado E.3) validan que el sistema maneje correctamente la actualización y se integre de manera adecuada con las dependencias simuladas.

Listado E.3: Prueba unitaria de la operación *Actualizar documento*

```
@Test
    public void testActualizarDocumento() throws CCSVException {
        // Simulacion: Crear el documento a actualizar con los datos necesarios
        DocumentDto documentoToUpdate = new DocumentDto();
        documentoToUpdate.setCsv("CSVRDOT44VOHU1MOGISS");
        documentoToUpdate.setNombre("ACTUALIZADO");

        // Simulacion: Leer archivo PDF y asignarlo al documento
        File pdfFile =
            new File("C:\\Users\\ClaraCerdanTorrubias\\Desktop\\updatePrueba.pdf");
        try (FileInputStream fis = new FileInputStream(pdfFile)) {
            byte[] pdfBytes = new byte[(int) pdfFile.length()];
            fis.read(pdfBytes);
            documentoToUpdate.setContenido(pdfBytes);
```

```
        } catch (Exception e) {
            fail("Error al leer el archivo PDF: " + e.getMessage());
        }

        // Simulacion: Agregar metadatos
        HashMap<String, Object> metadatos = new HashMap<>();
        metadatos.put("dea_desc_idioma", "en");
        documentoToUpdate.setMetadatos(metadatos);

        // Simulacion: Crear la peticion de actualizacion con el
        // documento y traceData
        ActualizarDocumentoRequestDto documentoActualizar =
            new ActualizarDocumentoRequestDto();
        documentoActualizar.setDocumento(documentoToUpdate);

        TraceData traceData = new TraceData();
        traceData.setReason("Motivo de actualizacion: prueba");
        documentoActualizar.setTraceData(traceData);

        // Simulacion: Crear la respuesta esperada para la actualizacion
        ResultActualizarDocumentoDTO responseActualizar =
            new ResultActualizarDocumentoDTO();
        responseActualizar.setCsv("CSVRDOT44VOHU1MOGISS");

        // Configuracion de los mocks: Se simula la respuesta para la llamada
        // al metodo actualizarDocumentoDto
        when(documentProvider.
            actualizarDocumentoDto(any(ActualizarDocumentoRequestDto.class)))
                .thenReturn(responseActualizar);

        // Llamada: Se llama al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<String> response =
        clientCCSVProviderImpl.actualizarDocumento(documentoActualizar);

        // Validaciones: Se verifica el resultado de la llamada
        assertNotNull(response);
        assertTrue(response.isSuccess());
        assertEquals(documentoToUpdate.getCsv(), response.getMessage());

        // Verificacion: Comprobamos que las interacciones con los
        // mocks se ejecutaron correctamente
        verify(documentProvider, times(1)).
            actualizarDocumentoDto(any(ActualizarDocumentoRequestDto.class));
    }
```

Obtener documento XML

El método *Obtener documento XML* permite recuperar un documento almacenado en formato XML a partir de un identificador único. Este método asegura que el

contenido y los metadatos del documento XML se devuelvan correctamente. Las pruebas (del Listado E.4) verifican que la operación cumple con los requisitos establecidos y garantiza la correcta interacción con los servicios simulados.

Listado E.4: Prueba unitaria de la operación *Obtener documento XML*

```
@Test
    public void testObtenerDocumentoXML() throws CCSVException {
        // Simulacion: Crear el DTO de peticion con el CSV de prueba
        ObtenerDocumentoXMLRequestDto requestCsv =
            new ObtenerDocumentoXMLRequestDto();
        requestCsv.setCsv("CSVRDOT44VOHU1MOGISS"); // CSV de prueba

        // Simulacion: Crear el DTO de respuesta esperada con el resultado obtenido
        ResultObtenerDocumentoXMLDto resultObtenerDocumentoXMLDto =
            new ResultObtenerDocumentoXMLDto();
        resultObtenerDocumentoXMLDto.setId("12345"); // ID del documento

        // Crear una respuesta simulada con el resultado esperado
        CCSVResponseDTO<ResultObtenerDocumentoXMLDto> responseGetXML =
            new CCSVResponseDTO<>();
        responseGetXML.setSuccess(true);
        responseGetXML.setMessage(resultObtenerDocumentoXMLDto);

        // Configuracion de los mocks: Se simula la respuesta de
        // la llamada al metodo obtenerDocumentoXMLDto
        when(documentProvider
            .obtenerDocumentoXMLDto(any(ObtenerDocumentoXMLRequestDto.class)))
                .thenReturn(responseGetXML);

        // Llamada al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<ResultObtenerDocumentoXMLDto> response =
            clientCCSVProviderImpl.obtenerDocumentoXML(requestCsv);

        // Validaciones: Verificamos que la respuesta es correcta
        assertNotNull(response);
        assertTrue(response.isSuccess());
        assertNotNull(response.getMessage());
        assertEquals("12345", response.getMessage().getId());

        // Verificacion: Comprobamos que el metodo
            obtenerDocumentoXMLDto se llamo una vez
        verify(documentProvider, times(1)).
            obtenerDocumentoXMLDto(any(ObtenerDocumentoXMLRequestDto.class));
    }
```

E.1.2. Módulo de expedientes

En esta sección, se detallan las pruebas unitarias relacionadas con la gestión de expedientes. Este módulo permite crear, actualizar y gestionar los expedientes administrativos, asegurando la correcta interacción entre las distintas capas de la aplicación.

Crear expediente

El método *Crear expediente* permite inicializar un expediente administrativo a partir de un documento inicial, asociado a un identificador único. Este proceso asegura que los metadatos del expediente estén correctos y que el expediente se abra y almacene en el sistema con los datos indicados. Las pruebas (que se muestran en el Listado E.5) verifican que todas las operaciones se ejecutan correctamente.

Listado E.5: Prueba unitaria de la operación *Crear expediente*

```
@Test
    public void testCrearExpediente() {
        try {
            // Simulacion: Crear el DTO de peticion con los datos de prueba
            CrearExpedienteRequestDto crearExpedienteRequestDTO =
                new CrearExpedienteRequestDto();
            crearExpedienteRequestDTO.
                setCsvDocumentoInicial("CSV5E65BHH6GY1FYGELP");
            crearExpedienteRequestDTO.setNombreExpediente("Prueba");
            crearExpedienteRequestDTO.setNif("17456123G");
            crearExpedienteRequestDTO.setNumExpediente("1111111");

            // Simulacion: Crear los objetos AdministrativeFile para simular
            // los datos de respuesta
            AdministrativeFile initExpediente = new AdministrativeFile();
            initExpediente.setCsv("INIT123");

            AdministrativeFile datosApertura = new AdministrativeFile();
            datosApertura.setCsv("CSV5E65BHH6GY1FYGELP");

            // Simulacion: Configuracion de los mocks de los metodos del provider
            when(expedientProvider.
                initializeExpediente(crearExpedienteRequestDTO))
                    .thenReturn(initExpediente);
            doNothing().when(expedientProvider).
                verifyMetadatosExpediente(initExpediente,
                    crearExpedienteRequestDTO.getNif());
            when(expedientProvider.openExpediente(initExpediente,
                crearExpedienteRequestDTO.getCsvDocumentoInicial(),
                    crearExpedienteRequestDTO.getNif()))
                        .thenReturn(datosApertura);
```

```
            // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
            CCSVResponseDTO<String> response =
                clientCCSVProviderImpl.crearExpediente(crearExpedienteRequestDTO);

            // Validacion: Verificamos que la respuesta es correcta
            assertNotNull(response);
            assertTrue(response.isSuccess());
            assertEquals("CSV5E65BHH6GY1FYGELP", response.getMessage());
        } catch (Exception e) {
            fail("El test lanzo una excepcion inesperada: " + e.getMessage());
        }
    }
```

Añadir documentos a un expediente

La operación *Añadir documentos a un expediente* permite asociar múltiples documentos a un expediente administrativo existente. Para ello, es fundamental contar con un identificador único del expediente, así como con los identificadores únicos de cada documento que se desea añadir. Esta funcionalidad asegura que los documentos queden correctamente vinculados al expediente en cuestión. Las pruebas (que se muestran en el Listado E.6) verifican que todas las operaciones se ejecutan correctamente.

Listado E.6: Prueba unitaria de la operación *Añadir documentos a un expediente*

```
@Test
    public void testAnadirDocumentosExpedienteExitoso() throws Exception {
        // Simulacion: Configuracion del DTO de peticion con los datos de prueba
        AnadirDocumentosExpedienteRequestDto anadirDocumentosExpedienteRequestDTO =
            new AnadirDocumentosExpedienteRequestDto();
        String[] listaCsv = {"CSVL36O1YC2FW12YGELP", "CSVCV5LMV46FU1IOGELP"};
        anadirDocumentosExpedienteRequestDTO
            .setCcsvExpediente("CSVO33OKFM5GK1BOGELP");
        anadirDocumentosExpedienteRequestDTO.setListaCsv(listaCsv);

        // Simulacion: Crear el objeto de respuesta para el caso de exito
        ResultAnadirDocumentosExpedienteDTO resultadoExito =
            new ResultAnadirDocumentosExpedienteDTO();
        resultadoExito.setCsv("CSV_SUCCESS_RESPONSE");
        resultadoExito.setError(null);

        // Simulacion: Configuracion de los mocks del proveedor
        when(expedientProvider.
            anadirDocumentoFicheroAdmin(anadirDocumentosExpedienteRequestDTO))
                .thenReturn(resultadoExito);
```

```
        // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<String> responseExito = clientCCSVProviderImpl
                .anadirDocumentosExpediente(anadirDocumentosExpedienteRequestDTO);

        // Validacion: Verificamos que la respuesta es correcta
        assertNotNull(responseExito);
        assertTrue(responseExito.isSuccess());
        assertEquals("CSV_SUCCESS_RESPONSE", responseExito.getMessage());
    }
```

Eliminar documentos de un expediente

La operación *Eliminar documentos de un expediente* permite desvincular uno o más documentos previamente asociados a un expediente administrativo. Para llevar a cabo esta operación, se requiere identificar tanto el expediente como los documentos que se desean eliminar mediante sus identificadores únicos. Las pruebas (que se muestran en el Listado E.7) verifican que todas las operaciones se ejecutan correctamente.

Listado E.7: Prueba unitaria de la operación *Eliminar documentos de un expediente*

```
@Test
    public void testEliminarDocumentosExpedienteExitoso() throws Exception {
        // Simulacion: Configuracion del DTO de peticion con los datos de prueba
        EliminarDocumentosExpedienteRequestDto eliminarDocumentosExpedienteRequestDTO =
            new EliminarDocumentosExpedienteRequestDto();
        String[] listaEliminarCsv = {"CSVL36O1YC2FW12YGELP", "CSVCV5LMV46FU1IOGELP"};
        eliminarDocumentosExpedienteRequestDTO
            .setCcsvExpediente("CSVO33OKFM5GK1BOGELP");
        eliminarDocumentosExpedienteRequestDTO.setListaCsv(listaEliminarCsv);

        // Simulacion: Crear el objeto de respuesta para el caso de exito
        ResultEliminarDocumentosExpedienteDTO resultadoExito =
            new ResultEliminarDocumentosExpedienteDTO();
        resultadoExito.setCsv("CSV_DELETE_SUCCESS_RESPONSE");
        resultadoExito.setError(null); // No hay error

        // Simulacion: Configuracion de los mocks del proveedor
        when(expedientProvider.
            eliminarDocumentoFicheroAdmin(eliminarDocumentosExpedienteRequestDTO))
                .thenReturn(resultadoExito);

        // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<String> responseExito = clientCCSVProviderImpl
                .eliminarDocumentosExpediente(eliminarDocumentosExpedienteRequestDTO);

        // Validacion: Verificamos que la respuesta es correcta
```

```
        assertNotNull(responseExito);
        assertTrue(responseExito.isSuccess());
        assertEquals("CSV_DELETE_SUCCESS_RESPONSE", responseExito.getMessage());
    }
```

Regenerar índice de un expediente

La funcionalidad *Regenerar índice de un expediente* permite recalcular y actualizar el índice que organiza la documentación de un expediente específico. Este proceso es esencial cuando ocurren cambios significativos en los documentos asociados al expediente, ya que asegura que el índice refleje el estado actual de los documentos. La operación requiere identificar el expediente por su código único. Las pruebas (que se muestran en el Listado E.8) verifican que todas las operaciones se ejecutan correctamente.

Listado E.8: Prueba unitaria de la operación *Regenerar índice de un expediente*

```
@Test
    public void testRegenerarIndiceExpedienteExitoso() throws Exception {
        // Simulacion: Configuracion del DTO de peticion con los datos de prueba
        RegenerarIndiceExpedienteRequestDto regenerarIndiceExpedienteRequestDTO =
            new RegenerarIndiceExpedienteRequestDto();
        String csv = "CSV033OKFM5GK1BOGELP";
        regenerarIndiceExpedienteRequestDTO.setCcsvExpediente(csv);

        // Simulacion: Crear el objeto de respuesta con el resultado exitoso
        ResultRegenerarIndiceExpedienteDTO resultadoExito =
            new ResultRegenerarIndiceExpedienteDTO();
        resultadoExito.setIndex("INDEX_GENERATION_SUCCESSFUL");
        resultadoExito.setError(null);

        // Simulacion: Configuracion de los mocks del proveedor
        when(expedientProvider
            .regenerarIndiceFicheroAdmin(regenerarIndiceExpedienteRequestDTO))
                .thenReturn(resultadoExito);

        // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<String> responseExito = clientCCSVProviderImpl
                .regenerarIndiceExpediente(regenerarIndiceExpedienteRequestDTO);

        // Validacion: Verificamos que la respuesta es correcta
        assertNotNull(responseExito);
        assertTrue(responseExito.isSuccess());
        assertEquals("INDEX_GENERATION_SUCCESSFUL", responseExito.getMessage());
    }
```

Crear carpeta en un expediente

La operación *Crear carpeta en un expediente* permite crear una nueva carpeta dentro de un expediente administrativo, asociándole un código y un nombre específicos. Esta funcionalidad es útil para organizar y gestionar los documentos dentro del expediente. Para llevar a cabo esta operación, se debe proporcionar la información necesaria, como el código de expediente y los detalles de la nueva carpeta a crear. Las pruebas (que se muestran en el Listado E.9) verifican que todas las operaciones se ejecutan correctamente.

Listado E.9: Prueba unitaria de la operación *Crear carpeta en un expediente*

```
@Test
    public void testCrearCarpetaExpediente() throws Exception {
        // Simulacion: Configuracion del DTO de solicitud con los datos
        // de prueba
        CrearCarpetaExpedienteRequestDto crearCarpetaExpedienteRequestDTO =
            new CrearCarpetaExpedienteRequestDto();
        String csv = "CSV033OKFM5GK1B0GELP";
        String folderIdEsperado = "FOLDER123";
        crearCarpetaExpedienteRequestDTO.setCsvExpediente(csv);
        crearCarpetaExpedienteRequestDTO.setFolderCode("2");
        crearCarpetaExpedienteRequestDTO.setFolderName("PRUEBA 2");

        // Simulacion: Crear el objeto de respuesta con el resultado exitoso
        ResultCrearCarpetaExpedienteDTO resultadoExito =
            new ResultCrearCarpetaExpedienteDTO();
        resultadoExito.setFolderId(folderIdEsperado);
        resultadoExito.setError(null);

        // Simulacion: Configuracion de los mocks del proveedor
        when(expedientProvider
            .crearCarpetaExpediente(crearCarpetaExpedienteRequestDTO))
                .thenReturn(resultadoExito);

        // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<String> responseExito = clientCCSVProviderImpl
                .crearCarpetaExpediente(crearCarpetaExpedienteRequestDTO);

        // Validacion: Verificamos que la respuesta es correcta
        assertNotNull(responseExito);
        assertTrue(responseExito.isSuccess());
        assertEquals(folderIdEsperado, responseExito.getMessage());
    }
```

Asociar expediente a un expediente

La operación *Asociar expediente a un expediente* permite vincular un expediente existente a otro, formando una relación entre ambos. Esta operación es útil cuando es necesario agrupar o relacionar expedientes, facilitando su gestión y acceso. Para realizar la asociación, es necesario proporcionar el código del expediente principal y el de los expedientes que se van a asociar. Las pruebas (que se muestran en el Listado E.10) verifican que todas las operaciones se ejecutan correctamente.

Listado E.10: Prueba unitaria de la operación *Asociar expediente a un expediente*

```
@Test
    public void testAsociarExpedienteExpediente() throws Exception {
        // Simulacion: Configuracion del DTO de solicitud con los datos
        // de prueba
        AsociarDocumentoExpedienteRequestDTO asociarDocumentoExpedienteRequestDTO =
            new AsociarDocumentoExpedienteRequestDTO();
        String csv = "CSV033OKFM5GK1BOGELP";
        List<String> listaAsociar = Arrays.asList("CSVTJ5D3XU8F717YGELP");
        String csvAsociadoEsperado = "CSV033OKFM5GK1BOGELP";

        asociarDocumentoExpedienteRequestDTO.setCsvExpediente(csv);
        asociarDocumentoExpedienteRequestDTO.setListaCsv(listaAsociar);

        // Simulacion: Crear el objeto de respuesta con el resultado exitoso
        ResultAsociarDocumentoExpedienteDTO resultadoExito =
        new ResultAsociarDocumentoExpedienteDTO();
        resultadoExito.setCsv(csvAsociadoEsperado);
        resultadoExito.setError(null); // No hay errores

        // Simulacion: Configuracion de los mocks del proveedor
        when(expedientProvider
            .asociarExpedienteExpediente(asociarDocumentoExpedienteRequestDTO))
                .thenReturn(resultadoExito);

        // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
        CCSVResponseDTO<String> responseExito = clientCCSVProviderImpl
                .asociarExpedienteExpediente(asociarDocumentoExpedienteRequestDTO);

        // Validacion: Verificamos que la respuesta es correcta
        assertNotNull(responseExito);
        assertTrue(responseExito.isSuccess());
        assertEquals(csvAsociadoEsperado, responseExito.getMessage());
    }
```

Obtener expediente

La operación *Obtener expediente* permite recuperar los detalles de un expediente

a partir de su identificador único, representado por el código CSV. Esta operación es fundamental cuando se necesita acceder a la información completa de un expediente, como sus documentos, estado y otros detalles asociados. El proceso requiere enviar una solicitud con el código CSV del expediente que se desea obtener.

Listado E.11: Prueba unitaria de la operación *Obtener expediente*

```
@Test
   public void testObtenerExpediente() throws Exception {
       // Simulacion: Configuracion del DTO de entrada con el valor
       // del expediente CSV
       ObtenerExpedienteRequestDto obtenerExpedienteRequestDTO =
           new ObtenerExpedienteRequestDto();
       String csvExpediente = "CSV033OKFM5GK1BOGELP";
       obtenerExpedienteRequestDTO.setCsvExpediente(csvExpediente);

       // Simulacion: Configuracion del DTO del expediente con el
       // valor 'csv'
       ExpedienteDTO expedienteDTO = new ExpedienteDTO();
       expedienteDTO.setCsv(csvExpediente);

       // Simulacion: Configuracion del resultado esperado con el expediente
       // completo en el DTO
       ResultObtenerExpedienteDto resultadoExito =
           new ResultObtenerExpedienteDto();
       resultadoExito.setExpediente(expedienteDTO);

       // Mock del comportamiento del proveedor cuando se invoca obtenerExpediente
       when(expedientProvider.
           obtenerExpediente(obtenerExpedienteRequestDTO))
               .thenReturn(resultadoExito);

       // Ejecucion: Llamada al metodo bajo prueba en clientCCSVProviderImpl
       CCSVResponseDTO<ResultObtenerExpedienteDto> responseExito =
           clientCCSVProviderImpl.obtenerExpediente(obtenerExpedienteRequestDTO);

       // Validacion: Comprobamos que la respuesta obtenida es correcta
       assertNotNull(responseExito);
       assertTrue(responseExito.isSuccess());
       assertNotNull(responseExito.getMessage());
       assertNotNull(responseExito.getMessage().getExpediente());
       assertEquals(csvExpediente,
           responseExito.getMessage().getExpediente().getCsv());
   }
```

E.2. Pruebas basadas en casos de uso

En las pruebas basadas en casos de uso, el objetivo es verificar que el sistema funcione correctamente según las especificaciones y necesidades del usuario. Para ello, se describe en una tabla cada operación del sistema que se va a probar, detallando las acciones y el comportamiento esperado en cada paso. Luego, se compara el resultado real obtenido tras ejecutar la operación con el resultado esperado. Si todas las condiciones del caso de uso se cumplen correctamente, la prueba es exitosa; de lo contrario, se identifica y se corrige cualquier desviación o error en el sistema. Este enfoque asegura que el sistema cumpla con los requisitos funcionales especificados desde la perspectiva del usuario. Estas pruebas son especialmente populares en Hiberus, la empresa en la que ha sido desarrollado el proyecto.

En la Figura E.1 se prueban las operaciones correspondientes al módulo de documentos, mientras que en la Figura E.2 se realizan las pruebas de las operaciones del módulo de expedientes. Cada figura representa un conjunto de operaciones a través de las cuales se valida el comportamiento del sistema, asegurando que cumpla con los requisitos funcionales específicos de cada módulo.

REQ	Caso de uso	Pasos	Pasos detallados del caso de uso	Resultado esperado	Resultado obtenido
RF-4	Crear documento	1	Se crea una instancia del tipo CrearDocumentoRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el nombre y el documento que se quiere subir	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función crearDocumento de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-5	Obtener documento	1	Se crea una instancia del tipo ObtenerDocumentoRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del documento que estamos buscando	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función obtenerDocumento de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-7	Actualizar documento	1	Se crea una instancia del tipo ActualizarDocumentoRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el nuevo documento que se desea actualizar	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función actualizarDocumento de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-6	Obtener documento XML	1	Se crea una instancia del tipo ObtenerDocumentoXMLRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del documento que estamos buscando	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función obtenerDocumentXML de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK

Figura E.1: Pruebas basadas en casos de uso del módulo de documentos

REQ	Caso de uso	Pasos	Pasos detallados del caso de uso	Resultado esperado	Resultado obtenido
RF-8	Crear expediente	1	Se crea una instancia del tipo ObtenerDocumentoXMLRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del documento inicial del expediente, el nombre del expediente, el nif y el numero de expediente	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función crearExpediente de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-9	Añadir documentos a un expediente	1	Se crea una instancia del tipo AnadirDocumentosExpedienteRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del expediente y una lista con los csv de los documentos que se quieren añadir	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función anadirDocumentosExpediente de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-10	Desasociar documentos de un expediente	1	Se crea una instancia del tipo EliminarDocumentosExpedienteRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del expediente y una lista con los csv de los documentos que se quieren desasociar del expediente	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función eliminarDocumentosExpediente de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-12	Crear una carpeta en un expediente	1	Se crea una instancia del tipo CrearCarpetaExpedienteRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del expediente y el nombre y el código de la carpeta que se quiere crear	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función crearCarpetaExpediente de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-13	Obtener un expediente	1	Se crea una instancia del tipo CrearCarpetaExpedienteRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del expediente y el nombre y el código de la carpeta que se quiere crear	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función crearCarpetaExpediente de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK
RF-11	Regenerar índice de un expediente	1	Se crea una instancia del tipo RegenerarIndiceExpedienteRequestDto	Se ha creado una instancia del objeto	OK
		2	Se especifica en la nueva instancia el csv del expediente del que se quiere regenerar el índice	Se ha actualizado la instancia del objeto	OK
		3	Se llama a la función regenerarIndiceExpediente de la clase CCSVClientProvider	Se recibe una respuesta del tipo CCSVResponse con el atributo "success" con el valor a "true"	OK

Figura E.2: Pruebas basadas en casos de uso del módulo de expedientes

E.3. Validación de endpoints

La validación de endpoints es una parte importante en el desarrollo de servicios web, ya que asegura que cada función dentro de la API se ejecute correctamente y que las respuestas cumplan con los estándares establecidos.

En este caso, la validación de los endpoints se llevó a cabo utilizando un proyecto implementado con Spring Boot. Durante el proceso, se exploraron y documentaron las APIs mediante la herramienta Swagger, lo que permitió una interacción eficiente con cada endpoint y la realización de pruebas exhaustivas para todas las operaciones disponibles.

Las pruebas incluyeron la verificación de operaciones principales, como la creación, actualización, consulta de recursos. Cada respuesta fue analizada cuidadosamente, comparando los resultados obtenidos con los esperados para confirmar que el comportamiento de los endpoints era el deseado en todos los casos.

Crear documento

La validación del endpoint `crearDocumento` que se puede ver en la Figura E.3 asegura que el archivo recibido sea válido y obligatorio, verificando que no esté vacío. También valida que el campo `nombre` sea obligatorio y que el tipo de documento sea opcional.

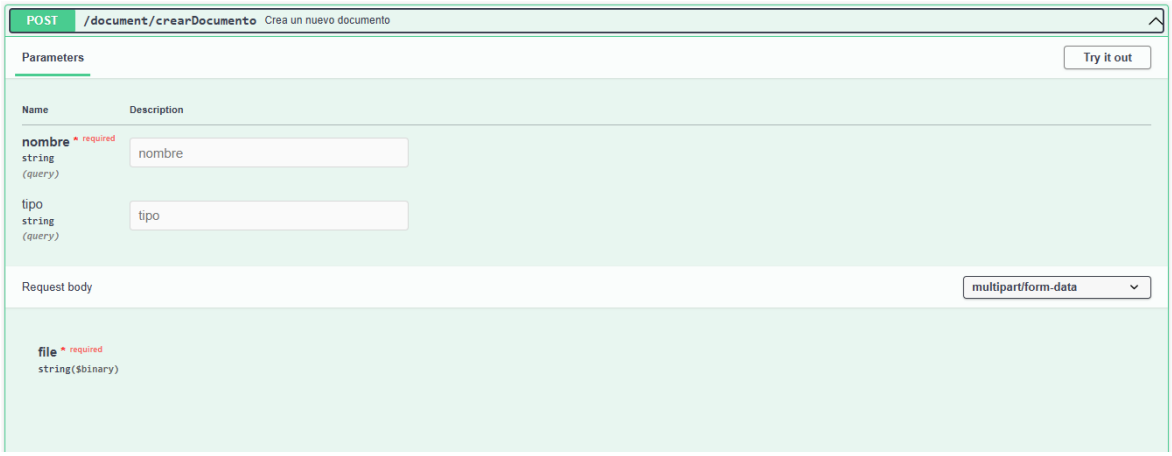

Figura E.3: Definición del endpoint para la creación de documentos en Swagger

Obtener documento

El endpoint `obtenerDocumento` (reflejado en la Figura E.4) permite recuperar un documento a partir de su identificador (`id`) o un código CSV asociado. Valida que al menos uno de estos parámetros sea proporcionado.

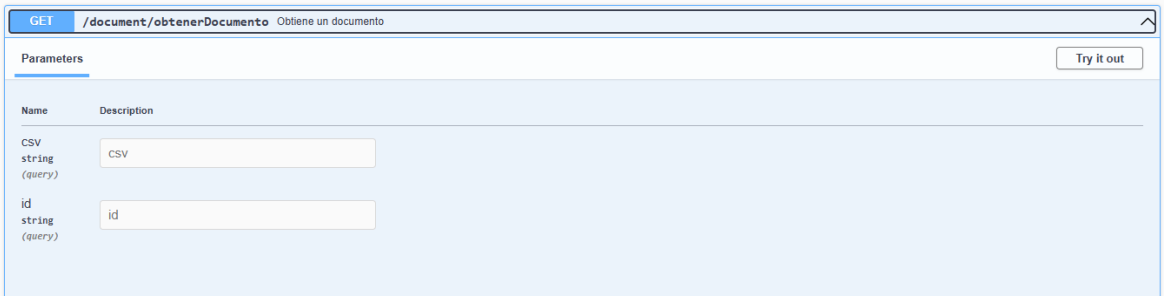

Figura E.4: Definición del endpoint para obtener un documento en Swagger

Actualizar documento

El endpoint `actualizarDocumento` permite modificar un documento existente. Para ello, se requiere el envío de un archivo (`file`) como parámetro obligatorio, mientras que otros parámetros como el código `csv`, el `nombre`, el `tipo`, y la `razón` son opcionales. El contenido del archivo se valida y se procesa antes de ser enviado para su actualización. La definición completa del endpoint se muestra en la Figura E.5.

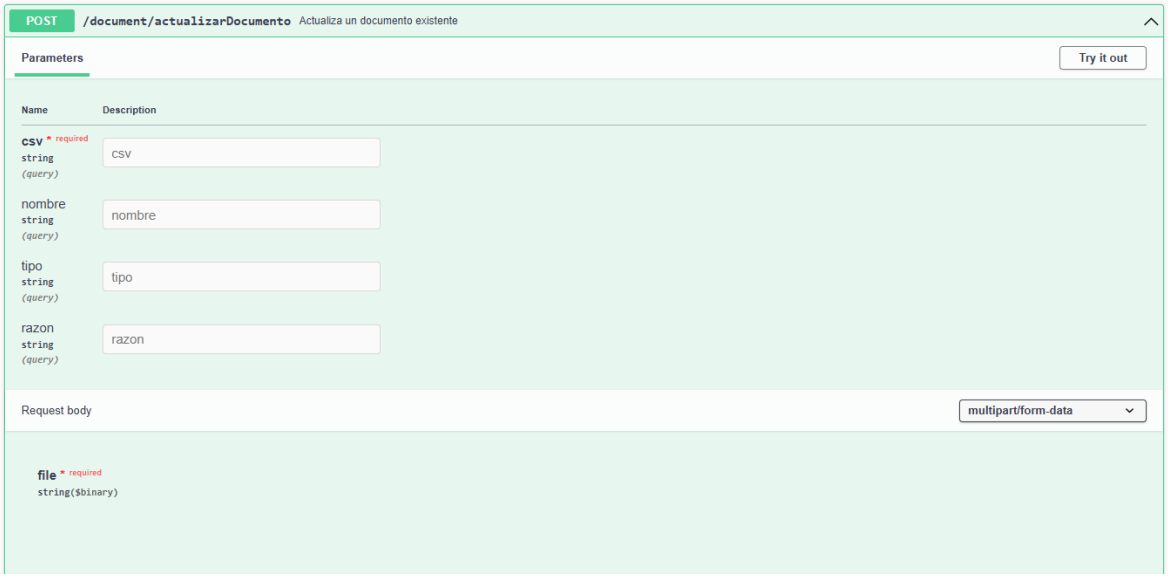

Figura E.5: Definición del endpoint para actualizar un documento en Swagger

Obtener documento XML

El endpoint `obtenerDocumentoXML` permite recuperar un documento en formato XML a partir de su identificador único (`id`) o su código CSV. Es necesario que se proporcione al menos uno de estos parámetros para que la solicitud sea válida. La definición detallada de este endpoint se presenta en la Figura E.6.

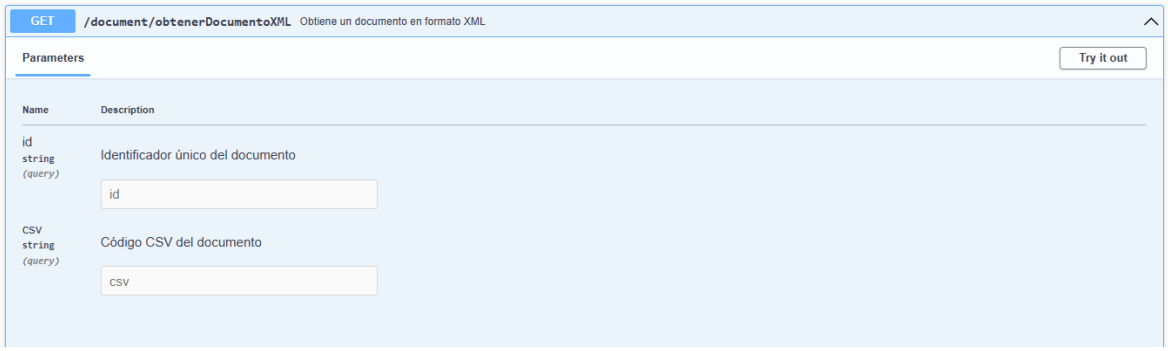

Figura E.6: Definición del endpoint para obtener un documento XML en Swagger

Crear expediente

El endpoint `crearExpediente` permite la creación de un nuevo expediente en el sistema. Para ello, se requiere proporcionar un `csvDocumentoInicial`, el `nif`, el `numExpediente`, y el `nombreExpediente`. Estos datos se procesan y se envían a la lógica de negocio para registrar el expediente. La definición detallada del endpoint se puede observar en la Figura E.7.

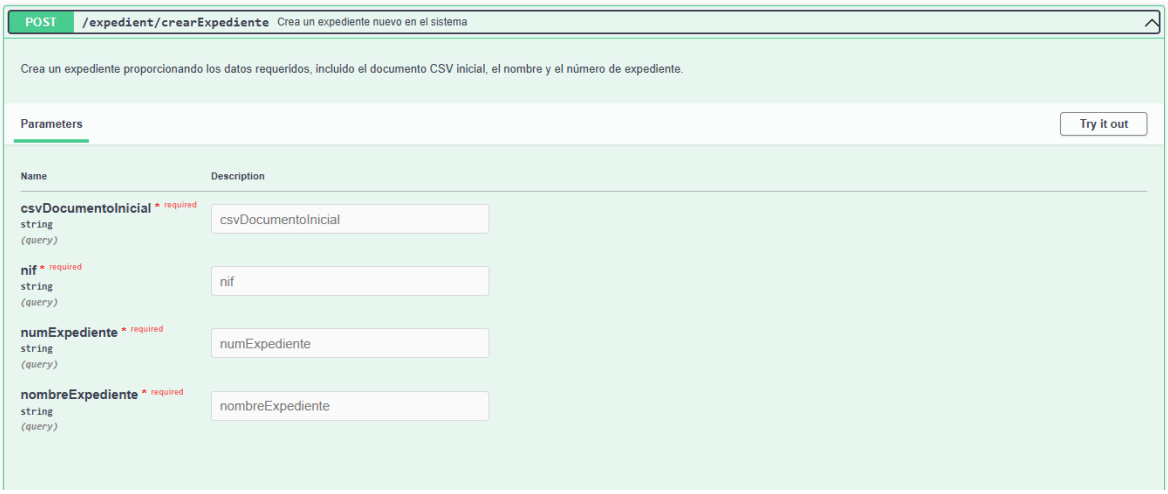

Figura E.7: Definición del endpoint para crear un expediente en Swagger

Añadir documentos a un expediente

El endpoint `anadirDocumentosExpediente` permite añadir uno o varios documentos a un expediente existente. Para ello, es necesario proporcionar el `ccsvExpediente` correspondiente al expediente, así como una lista de `ccsvDocumento` con los documentos a añadir. Opcionalmente, también se puede especificar el `idCarpeta` donde se añadirán los documentos. Este endpoint gestiona la validación de los datos y realiza la llamada al servicio correspondiente para actualizar el expediente.

La Figura E.8 muestra la definición completa del endpoint en Swagger.

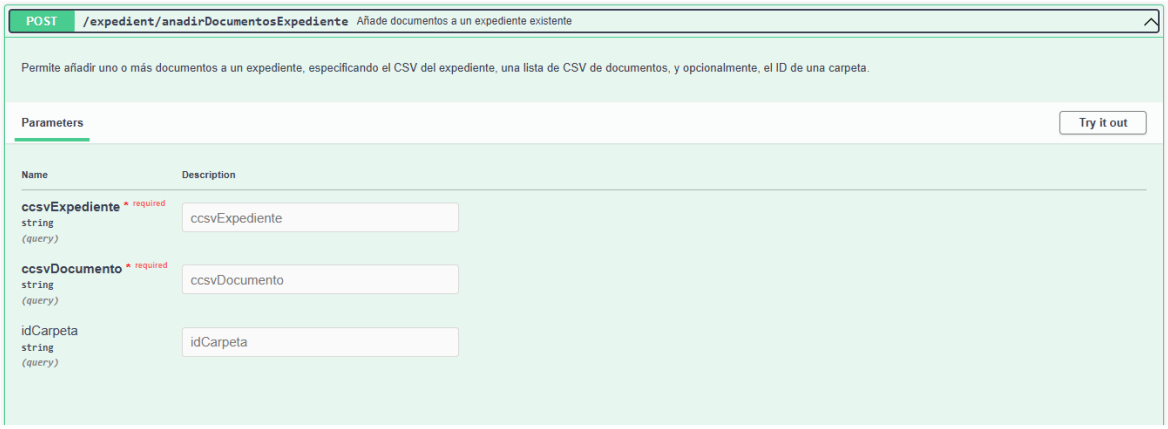

Figura E.8: Definición del endpoint para añadir documentos a un expediente en Swagger

Eliminar documentos de un expediente

El endpoint `documentosExpediente` permite eliminar documentos específicos asociados a un expediente existente. Para realizar esta operación, es necesario proporcionar el `ccsvExpediente`, que identifica al expediente, y una lista de

`ccsvDocumento`, que contiene los documentos a eliminar. Adicionalmente, se puede indicar el `idCarpeta` como parámetro opcional para especificar una carpeta concreta donde se encuentran los documentos. El endpoint valida los datos y realiza la operación llamando al servicio correspondiente.

La Figura E.9 muestra la definición detallada de este endpoint en Swagger.

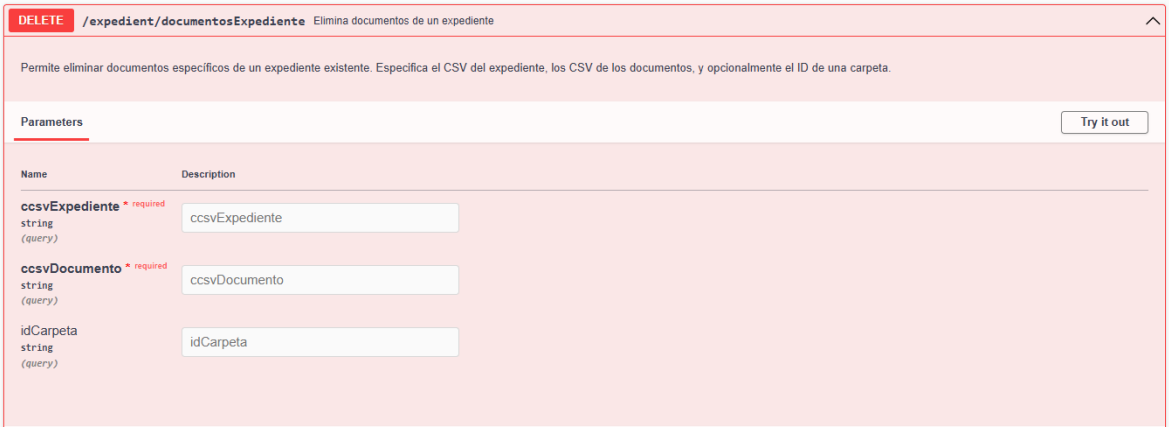

Figura E.9: Definición del endpoint para eliminar documentos de un expediente en Swagger

Regenerar índice de un expediente

El endpoint `/regenerarIndiceExpediente` proporciona la funcionalidad para regenerar el índice de un expediente, identificado de manera única mediante su `ccsvExpediente`. Este proceso se utiliza para actualizar o reorganizar la estructura de índices del expediente en el sistema, garantizando que refleje correctamente los documentos y carpetas asociadas.

La Figura E.10 muestra la definición del endpoint en Swagger, con los detalles de los parámetros requeridos, el formato de respuesta, y los posibles códigos de estado devueltos por la operación.

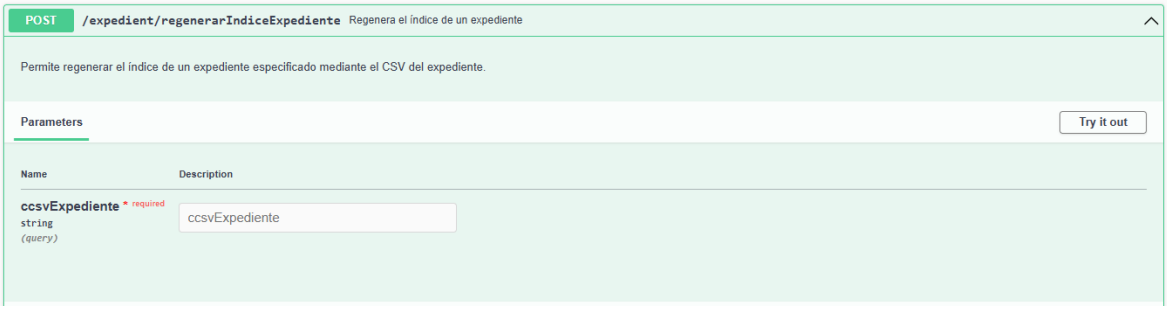

Figura E.10: Definición del endpoint para regenerar el índice de un expediente en Swagger

Crear una carpeta en un expediente

El endpoint `/crearCarpetaExpediente` permite añadir una nueva carpeta en un expediente especificado, utilizando el CSV del expediente, el nombre de la carpeta y su código. Esta operación es fundamental para mantener una estructura organizada de documentos dentro del sistema.

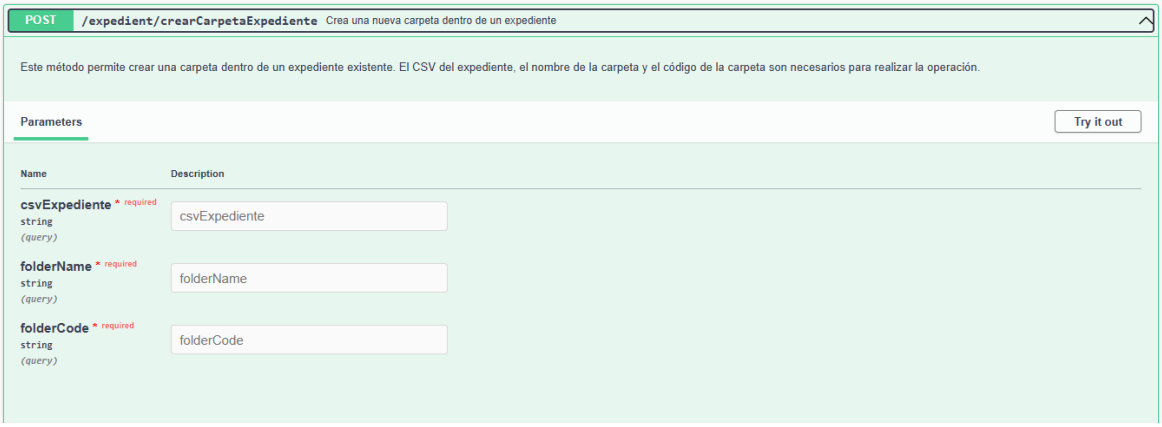

Figura E.11: Definición del endpoint para crear una carpeta en un expediente en Swagger

Asociar un expediente a un expediente

El endpoint `/asociarExpedienteExpediente` permite asociar un expediente principal a un subexpediente mediante los CSVs de ambos. La relación se establece de manera que uno de los expedientes queda como el expediente principal y el otro como subexpediente, con el objetivo de organizar y relacionar los expedientes entre sí.

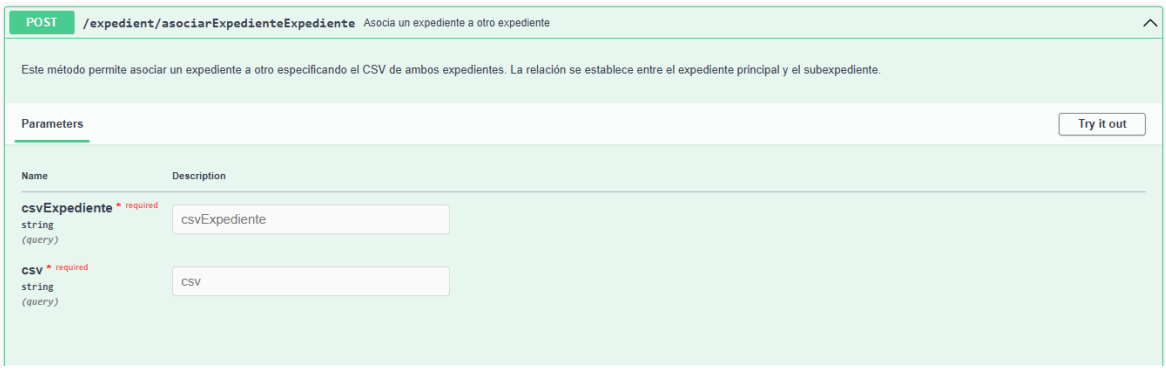

Figura E.12: Definición del endpoint para asociar un expediente a un expediente en Swagger

Obtener un expediente

El endpoint `/obtenerExpediente` permite obtener los detalles completos de

un expediente específico mediante su CSV. Esta operación se usa para consultar información sobre un expediente en el sistema y se responde con los detalles relevantes del mismo.

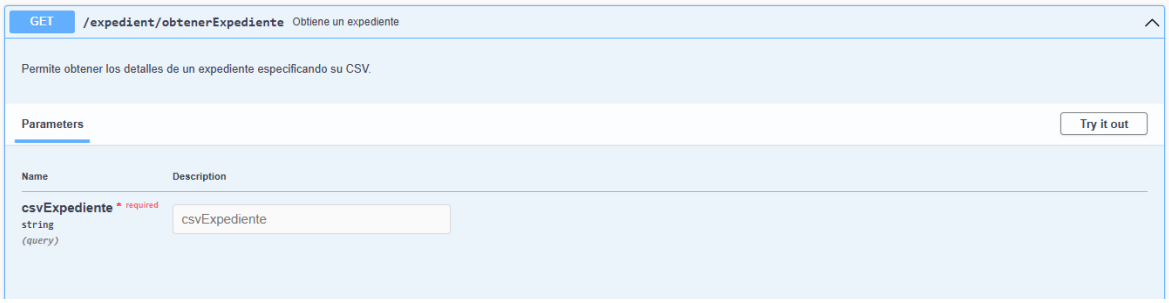

Figura E.13: Definición del endpoint para obtener un expediente en Swagger