



Universidad de Jaén

Escuela Politécnica Superior de Jaén

MyDrive: aplicación web para gestión unificada de almacenamiento en la nube.

Autor: Jesús Manzano Álvarez

Grado: Ingeniería Informática

Director: José Ramón Balsas Almagro
Departamento del director: Informática

Fecha: Septiembre, 2024

Licencia CC



CREA



Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Trabajo Fin de Grado titulado: MyDrive: aplicación web para gestión unificada de almacenamiento en la nube, que presenta Jesús Manzano Álvarez, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2024

El alumno:

Los tutores:

Jesús Manzano Álvarez

José Ramón Balsas Almagro

Agradecimientos

Este proyecto no habría sido posible sin el apoyo incondicional de mi familia, en especial mis padres, quienes siempre han estado a mi lado, no solo en la realización de este Trabajo de Fin de Grado, sino a lo largo de toda mi carrera universitaria. Su constante aliento, comprensión y sacrificio me han dado la fuerza necesaria para seguir adelante en los momentos más difíciles. Gracias por creer en mí cuando yo mismo lo dudaba.

También quiero agradecer a mis amigos, aquellos que han sido mis compañeros en este viaje. Con su apoyo, risas y compañía, han hecho que cada etapa del grado sea una experiencia inolvidable. Gracias por ser mi refugio y mi inspiración.

Por último, pero no menos importante, extendiendo mi gratitud a todos los profesores que me han formado como ingeniero informático. Su dedicación y pasión por la enseñanza me han moldeado no solo como profesional, sino también como persona. En especial, quiero agradecer a mi tutor, José Ramón Balsas Almagro, por su guía, paciencia y valiosos consejos durante la realización de este proyecto. Su apoyo ha sido fundamental para que este trabajo se realizara con éxito.

A todos vosotros, gracias por estar ahí.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Ingeniería del software
Mención (solo TFG)	Sin mención
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Jesús Manzano Álvarez
Fecha de asignación	22/03/2024
Descripción corta	En el panorama actual, la fragmentación en el uso de servicios de almacenamiento en la nube es un desafío, ya que los usuarios se ven obligados a gestionar sus archivos en múltiples plataformas. Para solucionar este problema, se ha desarrollado MyDrive, una aplicación web que permite a los usuarios centralizar la gestión de sus archivos en diferentes servicios de almacenamiento, eliminando las barreras de instalación y proporcionando una experiencia de uso unificada y accesible desde un simple navegador web.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
IEEE	Estilo de referencias y citas de IEEE (Institute of Electrical and Electronics Engineers)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
<i>Estas normas se han utilizado como base o referencia para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como proyecto la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.</i>	

Índice

Tabla de contenidos.....	8
Tabla de ilustraciones.....	8
Relación de tablas.....	9
1.- Introducción.....	11
1.1.- Motivación.....	11
1.2.- Objetivos.....	12
1.3.- Metodología.....	12
1.4.- Estructura del documento.....	13
2.- Análisis.....	15
2.1.- Análisis preliminar.....	15
2.1.1.- Estudio de tecnologías: tipo de aplicación.....	19
2.1.2.- Estudio de tecnologías: backend.....	20
2.1.3.- Estudio de tecnologías: frontend.....	22
2.2.- Propuesta de solución.....	25
2.3.- Historias de Usuario.....	25
2.4.- Planificación inicial de tareas.....	43
2.5.1.- Evolución de la planificación inicial.....	57
2.6.- Estudio de viabilidad.....	60
2.7.- Modelo de dominio.....	62
2.9.- Análisis de la interfaz.....	63
3.- Diseño.....	67
3.1.- Representación de la información.....	67
3.2.- Diagrama de clases.....	68
3.2.1.- Diagrama de clases del backend.....	69
3.2.2.- Diagrama de clases del frontend.....	72
3.3.- Diagramas de secuencia.....	77
3.3.1.- Conectar servicios de almacenamiento en la nube.....	79
3.3.2.- Obtener todo el contenido de una carpeta.....	81
3.3.3.- Subir archivos cifrados.....	83
3.3.4.- Descargar archivo cifrado.....	84
3.3.5.- Mover archivos entre nubes.....	86
3.4.- Diseño de la interfaz.....	87
3.4.1.- Wireframe de la interfaz gráfica del usuario.....	87
3.4.2.- Mapa web jerárquico.....	89
3.4.3.- Storyboards.....	90
3.4.3.1.- Conectar un servicio de almacenamiento en la nube.....	90
3.4.3.2.- Mover un archivo entre diferentes servicios de almacenamiento.....	91
3.4.3.2.- Subir un archivo cifrado.....	93
3.4.4.- Descripción de la API Rest.....	94
3.5.- Plan de pruebas.....	101
3.5.1.- Plan de pruebas de aceptación del usuario (UAT).....	102

4.- Implementación.....	105
4.1.- Arquitectura del sistema.....	105
4.2.- Arquitectura del software.....	106
4.1.1.- Arquitectura basada en capas.....	106
4.1.2.- Arquitectura de paquetes.....	110
4.2.- Detalles sobre implementación.....	112
4.2.1.- Implementación del frontend.....	113
4.2.2.- Implementación del backend.....	114
4.2.2.1.- Acceso a Google Drive.....	114
4.2.2.2.- Acceso a Dropbox.....	116
4.2.2.3.- Cifrado de archivos.....	120
5.- Pruebas.....	122
5.1.- Pruebas realizadas en el backend.....	122
5.1.1.- Pruebas unitarias realizadas al servicio de google drive.....	123
5.1.2.- Pruebas unitarias realizadas al servicio de dropbox.....	124
5.1.3.- Pruebas unitarias realizadas a la factoría.....	125
5.1.4.- Pruebas de integración realizadas.....	125
5.2.- Pruebas realizadas en el frontend.....	126
5.2.1.- Pruebas unitarias realizadas a FolderList.vue.....	127
5.2.2.- Pruebas unitarias realizadas a FileList.vue.....	128
5.3.- Pruebas con herramientas externas.....	129
6.- Conclusiones.....	131
6.1.- Mejoras y trabajos futuros.....	132
7.- Bibliografía.....	134
Apéndice I. Descripción de contenidos suministrados.....	135
Apéndice II. Manual de Instalación del sistema.....	136
II.1.- Configuración del backend.....	136
II.2.- Configuración del Frontend.....	137
Apéndice III. Manual de Usuario.....	139
III.1.- Inicio de sesión y autenticación.....	139
III.2.- Subida de Archivos.....	140
III.3.- Descarga de archivos.....	141
III.4.- Creación de una nueva carpeta.....	142
III.5.- Búsqueda de archivos y carpetas.....	143
III.6.- Apertura de Archivos.....	144
III.7.- Renombrar carpetas o archivos.....	145
III.8.- Mover archivos archivos o carpetas.....	146
III.9.- Ordenación de Archivos.....	147
III.10.- Periodo máximos de archivos recientes.....	148
III.11.- Mover archivos a la papelera de reciclaje.....	149
III.12.- Eliminar carpetas.....	149
III.13.- Eliminación Permanente de Archivos desde la Papelera.....	150
III.14.- Restaurar archivos.....	151
III.15.- Cerrar sesión en un servicio de almacenamiento en la nube.....	152

Tabla de contenidos

Tabla de ilustraciones

Ilustración 1. Servicios de almacenamiento en la nube más usados globalmente.....	16
Ilustración 2. MultCloud interfaz gráfica de usuario.....	17
Ilustración 3. Koofr interfaz gráfica de usuario.....	17
Ilustración 4. Odrive interfaz gráfica de usuario.....	18
Ilustración 5. Airexplorer interfaz gráfica de usuario.....	18
Ilustración 6. Frameworks backend más populares Enero 2023.....	22
Ilustración 7. Frameworks frontend más demandados 2023.....	24
Ilustración 8. Gráfico burn-down de horas estimadas - horas ideales.....	57
Ilustración 9. Gráfico burn-down de horas reales - horas estimadas.....	59
Ilustración 10. Modelo de dominio.....	62
Ilustración 11. Sketch a mano de la organización de contenidos de MyDrive.....	65
Ilustración 12. Representación de la información en base al modelo de dominio.....	67
Ilustración 13. Diagrama de clases de diseño del backend.....	70
Ilustración 14. Diagrama de clases de diseño del frontend.....	73
Ilustración 15. Diagrama de secuencia: Conectar servicio de almacenamiento en la nube... 79	
Ilustración 16. Diagrama de secuencia: Obtener todo el contenido de una carpeta.....	81
Ilustración 17. Diagrama de secuencia: Subir archivos cifrados.....	83
Ilustración 18. Diagrama de secuencia: Descargar archivo cifrado.....	84
Ilustración 19. Diagrama de secuencia: Mover archivos entre nubes.....	86
Ilustración 20. Diseño de la interfaz de usuario.....	88
Ilustración 21. Mapa web jerárquico.....	89
Ilustración 22. Documentación de la API con OpenAPI.....	96
Ilustración 23. Diagrama de arquitectura del sistema.....	105
Ilustración 24. Arquitectura software basada en capas.....	107
Ilustración 25. Diagrama de capas.....	109
Ilustración 26. Diagrama de paquetes.....	111
Ilustración 27. Nubes disponibles antes de iniciar sesión.....	139
Ilustración 28. Proceso de autenticación y autorización.....	140
Ilustración 29. Opción para subir archivos.....	140
Ilustración 30. Selección de archivos a subir.....	141
Ilustración 31. Descargar un único archivo.....	141
Ilustración 32. Descargar múltiples archivos.....	142
Ilustración 33. Descarga de un archivo cifrado.....	142
Ilustración 34. Selección para crear una carpeta.....	143
Ilustración 35. Nombre de la nueva carpeta.....	143
Ilustración 36. Introducción palabra para buscar.....	144
Ilustración 37. Aplicar filtros de búsqueda.....	144
Ilustración 38. Previsualizar un archivo.....	145

Ilustración 39. Opción para renombrar archivo.....	145
Ilustración 40. Renombrar carpeta.....	146
Ilustración 41. Mover archivos.....	146
Ilustración 42. Mover archivos.....	147
Ilustración 43. Ordenación de todo el contenido.....	148
Ilustración 44. Establecimiento de periodo máximo de archivos recientes.....	148
Ilustración 45. Opción para enviar a la papelera los archivos seleccionados.....	149
Ilustración 46. Opción para eliminar una carpeta.....	150
Ilustración 47. Eliminación de una carpeta.....	150
Ilustración 48. Eliminación de archivos.....	151
Ilustración 49. Restaurar archivos.....	152
Ilustración 50. Cerrar sesión en un servicio de almacenamiento en la nube.....	152

Relación de tablas

Tabla 1. HU01: Conectar servicios de almacenamiento en la nube.....	26
Tabla 2. HU02: Desconectar servicios de almacenamiento en la nube.....	27
Tabla 3. HU03: Obtener todo el contenido de una carpeta.....	27
Tabla 4. HU04: Navegación entre carpetas.....	28
Tabla 5. HU05: Visualización de archivos.....	28
Tabla 6. HU06: Creación de una carpeta.....	29
Tabla 7. HU07: Subir archivos.....	30
Tabla 8. HU08: Descargar archivos.....	31
Tabla 9. HU09: Mover carpetas o archivos.....	32
Tabla 10. HU10: Mover archivos entre nubes.....	33
Tabla 11. HU11: Renombrar archivo.....	34
Tabla 12. HU12: Mover a la papelera los archivos.....	35
Tabla 13. HU13: Eliminar carpetas.....	36
Tabla 14. HU14: Eliminar archivos.....	37
Tabla 15. HU15: Obtener todos los archivos de la papelera.....	37
Tabla 16. HU16: Vaciar papelera.....	38
Tabla 17. HU17: Restaurar archivos.....	38
Tabla 18. HU18: Obtener archivos recientes.....	39
Tabla 19. HU19: Buscar carpetas y archivos.....	39
Tabla 20. HU20: Ordenar carpetas y archivos.....	40
Tabla 21. HU21: Obtener información usuario.....	40
Tabla 22. HU22: Compartir archivos.....	41
Tabla 23. HU23: Crear carpeta espejo.....	42
Tabla 24. HU24: Búsqueda unificada.....	43
Tabla 25. Desglose en tareas y estimación de tiempos para HU01.....	44
Tabla 26. Desglose en tareas y estimación de tiempos para HU02.....	44

Tabla 27. Desglose en tareas y estimación de tiempos para HU03.....	44
Tabla 28. Desglose en tareas y estimación de tiempos para HU04.....	45
Tabla 29. Desglose en tareas y estimación de tiempos para HU05.....	45
Tabla 30. Desglose en tareas y estimación de tiempos para HU06.....	46
Tabla 31. Desglose en tareas y estimación de tiempos para HU07.....	46
Tabla 32. Desglose en tareas y estimación de tiempos para HU08.....	46
Tabla 33. Desglose en tareas y estimación de tiempos para HU09.....	47
Tabla 34. Desglose en tareas y estimación de tiempos para HU10.....	47
Tabla 35. Desglose en tareas y estimación de tiempos para HU11.....	48
Tabla 36. Desglose en tareas y estimación de tiempos para HU12.....	48
Tabla 37. Desglose en tareas y estimación de tiempos para HU13.....	49
Tabla 38. Desglose en tareas y estimación de tiempos para HU14.....	49
Tabla 39. Desglose en tareas y estimación de tiempos para HU15.....	49
Tabla 40. Desglose en tareas y estimación de tiempos para HU16.....	50
Tabla 41. Desglose en tareas y estimación de tiempos para HU17.....	50
Tabla 42. Desglose en tareas y estimación de tiempos para HU18.....	51
Tabla 43. Desglose en tareas y estimación de tiempos para HU19.....	51
Tabla 44. Desglose en tareas y estimación de tiempos para HU20.....	51
Tabla 45. Desglose en tareas y estimación de tiempos para HU21.....	52
Tabla 46. Desglose en tareas y estimación de tiempos para HU22.....	52
Tabla 47. Desglose en tareas y estimación de tiempos para HU23.....	52
Tabla 48. Desglose en tareas y estimación de tiempos para HU24.....	53
Tabla 49. Product backlog.....	54
Tabla 50. Estimación de costes.....	61
Tabla 51. Guía de los endpoints de la API Rest de MyDrive.....	100
Tabla 52. Tests unitarios para el servicio de Google Drive.....	123
Tabla 53. Tests unitarios para el servicio de Dropbox.....	125
Tabla 54. Tests unitarios para la factoría de la aplicación.....	125
Tabla 55. Tests de integración para el backend de la aplicación.....	126

1.- Introducción

En el panorama actual, donde la digitalización y la movilidad están cada vez más presentes en nuestras vidas cotidianas y en el ámbito laboral, la gestión eficiente de los archivos y documentos se ha convertido en una necesidad primordial. Con la llegada de la tecnología en la nube, el almacenamiento y la sincronización de archivos se ha vuelto más accesible y conveniente que nunca. Sin embargo, el incremento de múltiples servicios de almacenamiento en la nube ha dado lugar a una fragmentación en la manera en que las personas gestionan sus datos, lo que puede provocar una experiencia de usuario desagradable y poco eficiente.

Es en este contexto surge la necesidad de una solución que unifique la gestión de almacenamiento en la nube, proporcionando a los usuarios una plataforma centralizada desde la cual puedan acceder, organizar y administrar sus archivos almacenados en diferentes servicios de almacenamiento en la nube, de manera intuitiva y eficaz. Así, nace *MyDrive*, una aplicación web diseñada para ofrecer una gestión unificada de almacenamiento en la nube.

MyDrive está dirigida a una amplia gama de usuarios, desde estudiantes o trabajadores que tienen cuentas de almacenamiento tanto en el ámbito laboral como personal y desean sincronizar sus archivos entre diferentes perfiles. Además, empresas de todos los tamaños pueden beneficiarse de una solución que simplifique la gestión de datos en la nube, mejorando la comunicación entre sus trabajadores, aumentando la productividad y fomentando la colaboración entre equipos.

Esta aplicación tiene la idea de transformar la manera en que las personas y las organizaciones gestionan sus datos en línea, simplificando el acceso, la organización y la colaboración en el entorno digital actual.

1.1.- Motivación

En los últimos años, hemos sido testigos de un aumento significativo en el uso del almacenamiento en la nube y de herramientas en línea que no requieren instalación. Este incremento se debe, en gran parte, a la creciente digitalización de nuestras vidas y a la necesidad de acceder a nuestros archivos desde cualquier lugar y dispositivo. La conveniencia de poder almacenar, compartir y colaborar en documentos sin la necesidad de instalar software específico ha impulsado aún más esta tendencia, haciendo que las soluciones en la nube sean cada vez más populares y ampliamente adoptadas dando lugar a un sinfín de empresas que ofrecen servicios de almacenamiento en la nube.

Aunque existen diversas soluciones de almacenamiento en la nube como Google Drive, Dropbox, OneDrive, iCloud, etc. Muchas de ellas presentan limitaciones en

sus planes gratuitos o incluso en sus opciones de pago. Estas limitaciones pueden incluir restricciones en el espacio de almacenamiento, funcionalidades avanzadas solo disponibles en planes de pago, o dificultades para sincronizar y administrar archivos entre diferentes servicios de nube.

Esta fragmentación en las soluciones disponibles puede resultar en una experiencia de usuario incompleta y poco eficiente, especialmente para aquellos que gestionan archivos en múltiples servicios de almacenamiento. En este contexto surge la necesidad de una solución como *MyDrive*, que busca ofrecer una gestión unificada de almacenamiento en la nube, conectando algunos de los servicios más populares, proporcionando a los usuarios una experiencia integral y satisfactoria.

1.2.- Objetivos

Los objetivos de *MyDrive* se centran en abordar de manera efectiva las necesidades identificadas en el contexto planteado y en desarrollar una solución web para la gestión unificada de almacenamiento en la nube. Los objetivos específicos incluyen:

- Realizar un estudio de necesidades y soluciones para el contexto planteado
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas, principios de diseño y metodologías de desarrollo web.
- Seleccionar un entorno de desarrollo basado en tecnologías web e implementar un prototipo de aplicación web que satisfaga las necesidades que se hayan determinado.

1.3.- Metodología

Para el desarrollo del proyecto, se aplicará una metodología de desarrollo iterativo inspirada en Scrum [2], pero adaptada para un equipo de trabajo conformado únicamente por un miembro. Aunque la metodología se basará en los principios y prácticas de Scrum, se tendrá en cuenta que, dadas las circunstancias particulares de un equipo formado por un único miembro, el proceso resultante será en gran medida similar al desarrollo iterativo de una metodología ágil.

Dicha elección se debe a su capacidad para manejar proyectos de manera flexible y su adaptación continua, cuyos elementos son claves para el éxito de cualquier proyecto de desarrollo de software. En comparación con otras metodologías existentes, como el modelo en cascada o el desarrollo en espiral, Scrum ofrece una mayor flexibilidad y capacidad de respuesta a los cambios en los requisitos y prioridades del proyecto.

Aunque Scrum establece roles específicos, eventos formales, etc., en el contexto de un equipo formado por un único miembro, algunas de estas prácticas, como las reuniones diarias de seguimiento (*daily stand-ups*), pueden no ser aplicables, ya que el único miembro del equipo debe asumir múltiples roles y responsabilidades. Sin embargo, se fomentará la autoevaluación y la autocrítica como parte del proceso, lo que permitirá desempeñar eficazmente los diferentes roles que normalmente se asignarían en un equipo Scrum. De esta manera, se mantendrán, en cierta medida, los principios ágiles de Scrum.

En cuanto a la gestión de requisitos, se emplearán técnicas ágiles como la elaboración de historias de usuario y la descomposición en tareas. Esto permitirá una comprensión clara de los requisitos y una planificación efectiva de las actividades a realizar en cada sprint.

Se realizarán diferentes tipos de pruebas al final de la implementación de cada funcionalidad para comprobar que cumplen los criterios de aceptación correspondientes. Esto permitirá asegurar la calidad del código y la funcionalidad del software tal y como se haya especificado.

1.4.- Estructura del documento

El presente documento se ha organizado de manera que permita al lector comprender el proceso completo de desarrollo del proyecto. Comienza con un capítulo de Introducción, donde se expone la motivación que impulsó la realización del proyecto, los objetivos que se plantearon al inicio, la metodología empleada durante el desarrollo y una visión general de la estructura del documento. A continuación, en el capítulo de Análisis, se presenta un estudio detallado de las tecnologías que se consideraron para la implementación del proyecto, incluyendo tanto el *backend* como el *frontend*. Este capítulo también aborda la propuesta de solución, las historias de usuario que guiaron el desarrollo, la planificación inicial de tareas y su evolución, así como un estudio de viabilidad. Además, se incluye un análisis del modelo de dominio y de la interfaz de usuario, lo que permite establecer una base sólida para el diseño del sistema.

El capítulo de Diseño se centra en la representación de la información y en la arquitectura del sistema, proporcionando diagramas de clases y de secuencia que describen las interacciones entre los diferentes componentes del sistema. Asimismo, se presenta el diseño de la interfaz gráfica del usuario, complementado con wireframes, storyboards y una descripción detallada de la API Rest desarrollada. Posteriormente, en el capítulo de Implementación, se describen los aspectos técnicos de la implementación, tanto a nivel de la arquitectura del software como en lo referente a la implementación específica del *frontend* y *backend*. En este capítulo se detallan las tecnologías empleadas, las decisiones de diseño y cómo se

implementaron las principales funcionalidades, tales como el acceso a Google Drive y Dropbox, y el cifrado de archivos.

El capítulo de Pruebas documenta las pruebas realizadas durante el desarrollo del proyecto, incluyendo pruebas unitarias, de integración y de aceptación del usuario, tanto en el *backend* como en el *frontend*, asegurando así la calidad y el correcto funcionamiento del sistema. En el capítulo de Conclusiones, se exponen las reflexiones finales sobre el proyecto, destacando los logros alcanzados y proponiendo posibles mejoras y trabajos futuros que podrían llevarse a cabo para extender y mejorar el sistema desarrollado. Finalmente, en la bibliografía, se recopilan las fuentes de información y referencias que fueron consultadas durante la realización del proyecto.

El documento concluye con una serie de apéndices que proporcionan información adicional relevante. Entre estos se incluyen una descripción de los contenidos suministrados, un manual de instalación del sistema y un manual de usuario que detalla de manera sencilla las funcionalidades implementadas en la aplicación. Estos apéndices están diseñados para servir de referencia tanto para la configuración inicial del sistema como para el uso diario de la aplicación.

2.- Análisis

2.1.- Análisis preliminar

MyDrive surge en un contexto marcado por la creciente popularidad y uso de servicios de almacenamiento en la nube. El concepto de almacenamiento en la nube tiene sus raíces en los años 60 con el trabajo de J. C. R. Licklider [4] y su visión de una red global para conectar a las personas y los datos desde cualquier lugar y en cualquier momento. Sin embargo, el concepto moderno de almacenamiento en la nube comenzó a materializarse en la década de 1980. En 1983, CompuServe ofreció a sus usuarios un pequeño espacio de disco para almacenar archivos, lo que puede considerarse uno de los primeros ejemplos de almacenamiento en la nube. En 1994, AT&T lanzó PersonaLink Services, una plataforma web para comunicación y gestión de contenidos, y la describió como una "nube electrónica" en sus anuncios.

No fue hasta la década de 2000 que los servicios de almacenamiento en la nube empezaron a popularizarse ampliamente. En 2005, Box introdujo su servicio de gestión de contenidos en la nube para empresas, y en 2006, Amazon Web Services lanzó Amazon S3, que ganó reconocimiento como proveedor de almacenamiento para servicios populares como SmugMug, Dropbox y Pinterest. Estos servicios ofrecían la posibilidad de almacenar archivos de forma remota, eliminando la necesidad de almacenar grandes cantidades de datos en discos duros locales.

A medida que la tecnología avanzaba, surgieron más alternativas para almacenar archivos en la nube, incluyendo Google Drive, Apple iCloud y Microsoft OneDrive. Así, la nube se convirtió en una herramienta esencial para acceder y compartir datos desde cualquier dispositivo con conexión a Internet, liberando a los usuarios de la preocupación por el almacenamiento físico de sus archivos y las copias de seguridad periódicas. No obstante, este modelo también implicaba que las empresas proveedoras de servicios en la nube tuvieran acceso potencial a los archivos almacenados en sus servidores.

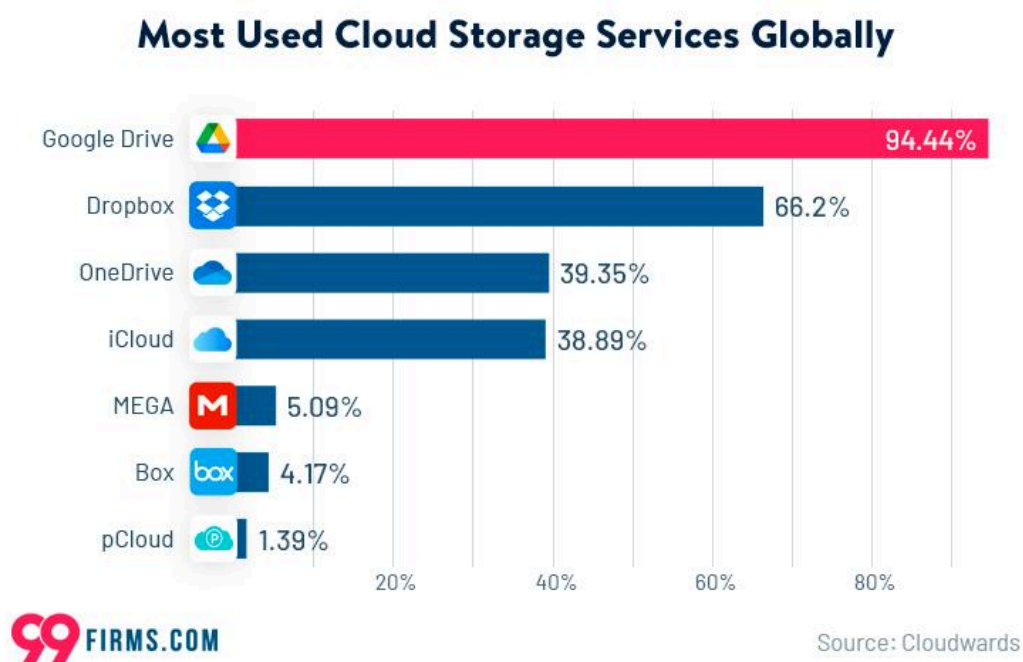


Ilustración 1. Servicios de almacenamiento en la nube más usados globalmente¹

A lo largo de los años, el uso de servicios de almacenamiento en la nube ha experimentado un crecimiento exponencial. Sin embargo, como vemos en la Ilustración 1, este aumento en la adopción de servicios de almacenamiento en la nube también ha dado lugar a una diversificación de aplicaciones que ofrecen este tipo de servicios.

Actualmente, existen diversas aplicaciones similares a *MyDrive*, como MultCloud, Koofr, Odrive, AirExplorer y rclone que buscan facilitar la gestión de múltiples servicios de almacenamiento en la nube desde una sola aplicación. Sin embargo, algunas de estas soluciones son aplicaciones nativas que requieren instalación y están diseñadas específicamente para diferentes sistemas operativos, lo que puede limitar su accesibilidad y uso en dispositivos diferentes.

MultCloud² se puede considerar el competidor más directo de *MyDrive*, debido a sus características muy similares. Esta aplicación destaca por su accesibilidad a través de una interfaz web que facilita la gestión de múltiples servicios de almacenamiento en la nube sin necesidad de instalación. Permite la transferencia de archivos entre servicios, cifrar y descifrar los archivos subidos, pero su versión gratuita tiene limitaciones en la velocidad de transferencia y tamaño de archivos.

¹ Fuente: <https://99firms.com/blog/cloud-computing-statistics>

² Fuente: <https://www.multcloud.com/>

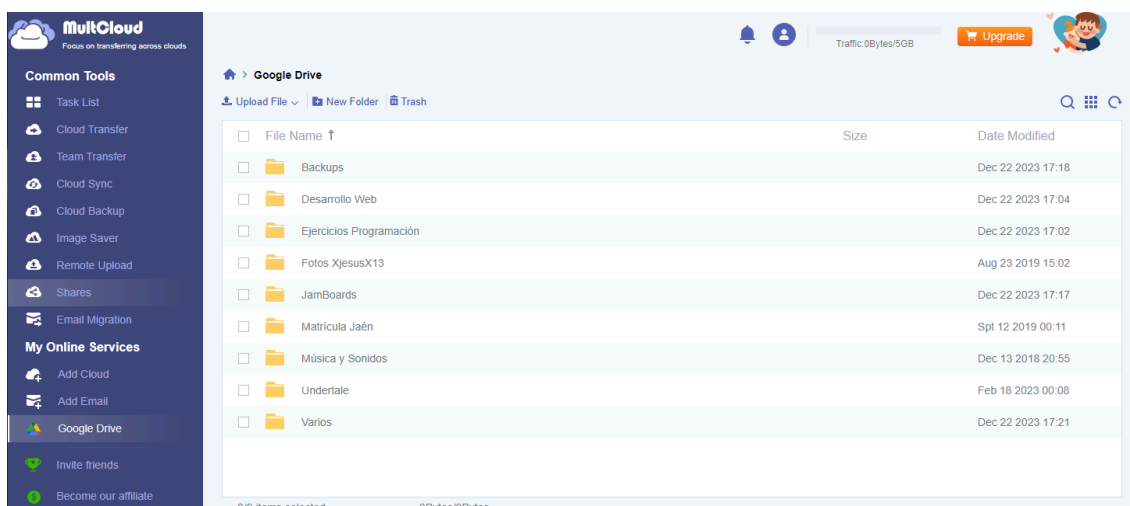


Ilustración 2. MultCloud interfaz gráfica de usuario

Koofr³ y **odrive**⁴ también son competidores significativos, aunque enfrentan críticas por la experiencia visual de sus interfaces web. Koofr ofrece características sólidas como el cifrado de archivos y funciones automáticas de backup, pero su integración con diversos servicios en la nube es menos extensa en comparación con otras opciones. Del mismo modo, odrive permite sincronización selectiva y está disponible en versión gratuita, pero algunas funciones avanzadas requieren la compra de la versión Premium y su experiencia visual en la web ha recibido críticas por ser menos intuitiva. Estas limitaciones podrían afectar la preferencia de usuarios que valoran tanto la funcionalidad como una experiencia de usuario fluida y atractiva.

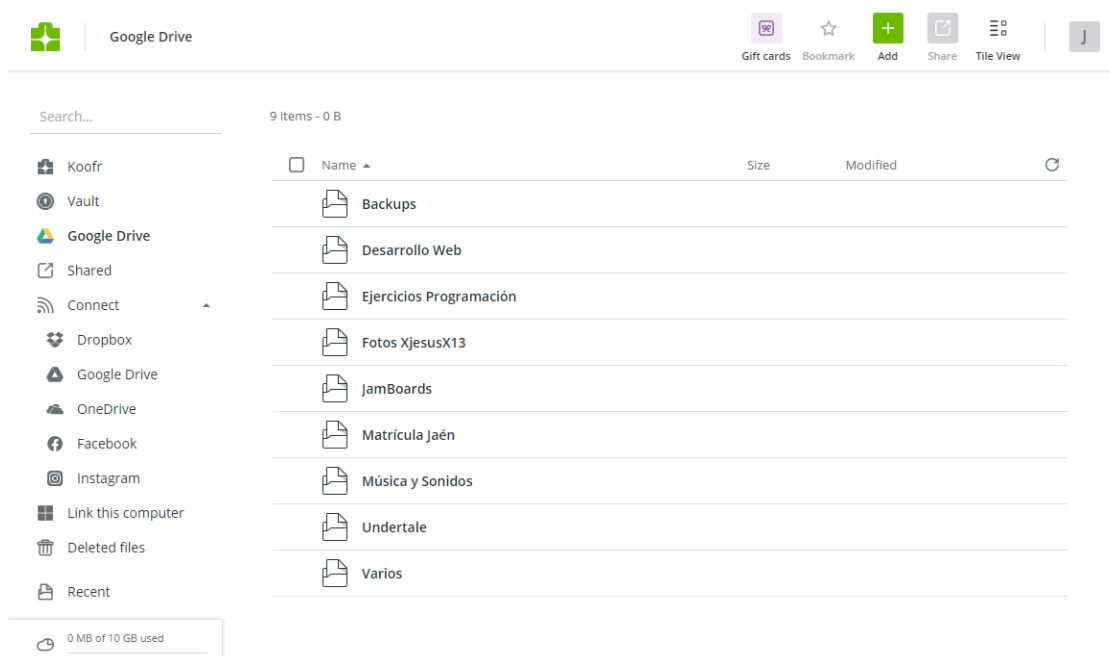


Ilustración 3. Koofr interfaz gráfica de usuario

³ Fuente: <https://koofr.eu/>

⁴ Fuente: <https://www.odrive.com/homepage5b>

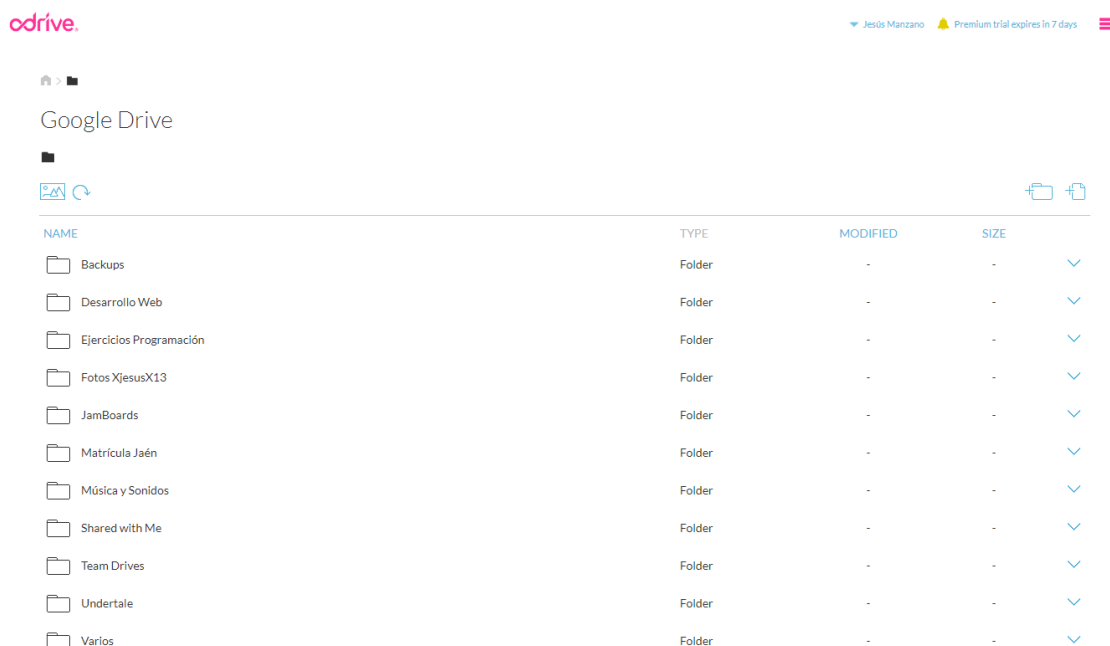


Ilustración 4. Odrive interfaz gráfica de usuario

AirExplorer⁵ ofrece una interfaz gráfica amigable y permite la sincronización directa entre servicios en la nube. Ofrece cifrado de archivos para mayor seguridad. Sin embargo, está disponible como aplicación de escritorio en Windows y macOS pero no para Linux. La versión gratuita tiene limitaciones en funcionalidades y cantidad de conexiones simultáneas.

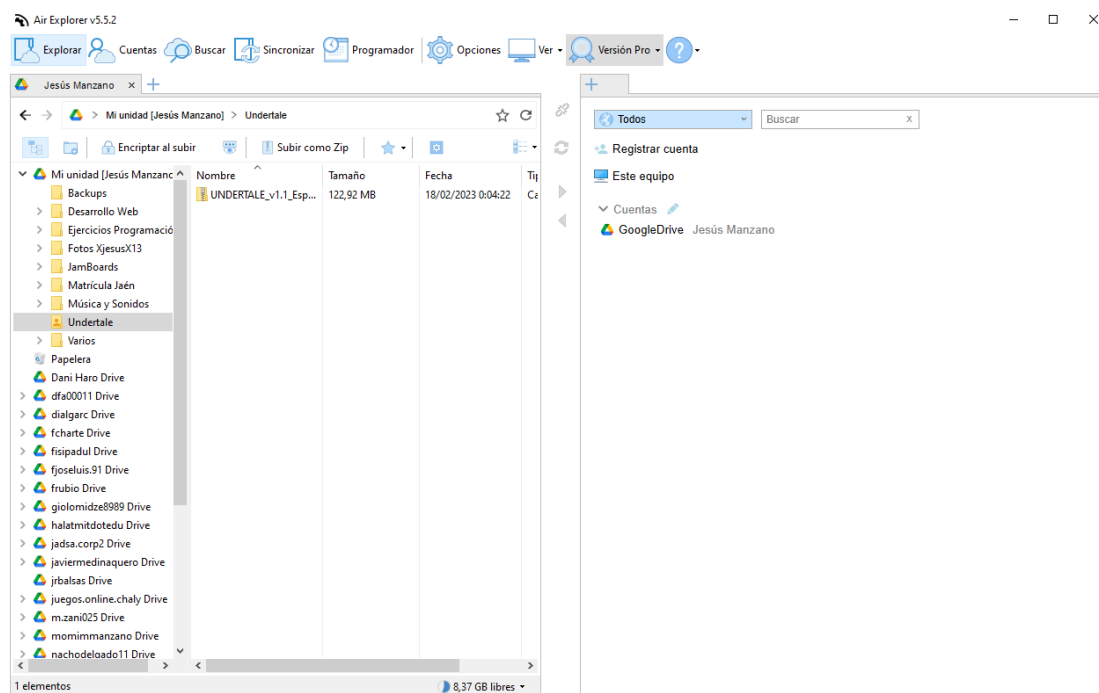


Ilustración 5. Airexplorer interfaz gráfica de usuario

⁵ Fuente: <https://www.airexplorer.net/es/>

rclone⁶, aunque es una herramienta de línea de comandos (CLI), es compatible con una amplia gama de servicios en la nube y sistemas operativos, incluyendo Windows, macOS y Linux. Ofrece cifrado de archivos y es totalmente gratuito y de código abierto, ideal para usuarios avanzados que prefieren flexibilidad y personalización. Requiere configuración manual y conocimientos de línea de comandos para su uso, lo que puede aumentar la curva de aprendizaje para usuarios no experimentados.

MyDrive busca encontrar un nicho de mercado ofreciendo una solución gratuita y basada en la web, eliminando las barreras de instalación y las limitaciones de dispositivos asociadas con las aplicaciones nativas. Además, *MyDrive* ofrecerá una interfaz gráfica visualmente atractiva, moderna y *responsive* que se adapte a diferentes dispositivos, proporcionando una experiencia de usuario fluida y consistente. Al igual que algunas de las soluciones antes mencionadas, *MyDrive* añadirá una capa extra de seguridad opcional al subir archivos a la nube, lo que proporcionará a los usuarios un mayor control sobre la privacidad y protección de sus datos.

2.1.1.- Estudio de tecnologías: tipo de aplicación

En el proceso de estudio de tecnologías para el desarrollo del proyecto *MyDrive*, se consideraron tres opciones principales: una aplicación web, una aplicación nativa y el uso de un CMS (Sistema de Gestión de Contenidos). Cada una de estas opciones tiene sus propias ventajas y desventajas en términos de accesibilidad, rendimiento, personalización y seguridad.

Una aplicación nativa ofrecería un rendimiento optimizado y una experiencia de usuario altamente integrada, diseñada específicamente para el sistema operativo en el que se ejecuta. Sin embargo, el principal desafío de desarrollar una aplicación nativa radica en la necesidad de crear y mantener versiones separadas para cada plataforma, como iOS, Android, Windows, MacOS, etc. Cada plataforma tiene su propio conjunto de herramientas de desarrollo y lenguajes de programación lo que requiere un esfuerzo adicional por parte de los desarrolladores para asegurar la coherencia y calidad en todas las versiones de la aplicación. Este proceso de desarrollo y mantenimiento múltiple no solo aumenta el costo y esfuerzo del proyecto, sino que también puede añadir complejidad y tiempo al ciclo de desarrollo.

Por otro lado, el uso de un CMS proporcionaría una solución rápida y económica para la gestión de contenido, permitiendo la creación y actualización de contenido de manera sencilla a través de una interfaz intuitiva. Sin embargo, los CMS pueden presentar limitaciones en términos de personalización y adaptación a funcionalidades específicas, al tratar de gestionar diferentes servicios de

⁶ Fuente: <https://rclone.org/>

almacenamiento en la nube, cada uno de ellos con sus características específicas. Además, decantarse por esta opción supondría estar sujetos a las políticas de privacidad y seguridad de la empresa que gestiona los servidores. Dependiendo de la configuración y las medidas de seguridad implementadas, existe la posibilidad de que los datos puedan ser accesibles por parte de la empresa que ofrece el CMS.

Finalmente, se optó por desarrollar una aplicación web personalizada para *MyDrive*. Una aplicación web ofrece accesibilidad universal a través de un navegador web en cualquier dispositivo con conexión a Internet, eliminando la necesidad de desarrollar y mantener múltiples versiones para diferentes plataformas, evitando paralelamente la necesidad de que los usuarios deban instalar aplicaciones en estos dispositivos. Además, permite una mayor escalabilidad a lo largo del tiempo, permitiendo implementar nuevos servicios que vayan surgiendo en un futuro. Por otro lado, permite manejar de una manera más flexible muchos servicios en la nube heterogéneos entre sí. Por último, el control total sobre la seguridad y gestión de datos proporcionados por una aplicación web personalizada es fundamental para garantizar la protección de la información confidencial de los usuarios, aspecto prioritario en un servicio de gestión de almacenamiento en la nube como *MyDrive*.

Una vez se ha determinado el tipo de aplicación a desarrollar, la selección de las tecnologías adecuadas tanto para el *backend* como para el *frontend* se convierte en un paso crucial. Este proceso implica realizar un análisis de las opciones disponibles, evaluando cuidadosamente las ventajas y desventajas de cada tecnología para justificar la elección final.

2.1.2- Estudio de tecnologías: backend

La selección de las tecnologías adecuadas para el desarrollo del *backend* de *MyDrive* implica considerar una variedad enorme de tecnologías disponibles en el mercado. A nivel general, existen varios lenguajes de programación y frameworks populares utilizados en el desarrollo de aplicaciones web. Entre los lenguajes de programación más comunes se encuentran:

- **PHP:** PHP es un lenguaje de programación del lado del servidor. Se utiliza para crear aplicaciones web dinámicas y gestionar bases de datos. Es ampliamente utilizado en sistemas de gestión de contenido (CMS) como WordPress.
- **Python:** Python es un lenguaje de programación interpretado y de alto nivel, apreciado por su simplicidad y legibilidad. Es popular en el desarrollo web, especialmente para prototipado rápido y desarrollo ágil.

- **JavaScript:** JavaScript es el lenguaje de programación fundamental para el desarrollo web del lado del cliente. Con la llegada de Node.js, también se utiliza ampliamente en el desarrollo del lado del servidor.
- **Java:** Java es un lenguaje de programación versátil y robusto, conocido por su portabilidad y su amplia adopción en la industria [5]. Ofrece una sintaxis clara y es adecuado para aplicaciones empresariales que requieren escalabilidad y seguridad. Es un lenguaje de programación de código abierto y está disponible bajo la Licencia Pública General de GNU (GNU GPL).

Para el desarrollo del *backend* de *MyDrive* se ha escogido Java. Esta elección se basa en una combinación de razones técnicas y experiencia personal. Por un lado, Java es un lenguaje orientado a objetos, lo que resulta muy deseable para modelar de forma clara y eficiente elementos clave de la aplicación, como carpetas y archivos. Este enfoque de programación permite representar estos conceptos como clases y objetos, facilitando su manejo y haciendo que el código sea más mantenible, lo cual sería mucho más complejo de implementar en otros lenguajes que no sigan este paradigma. Java ofrece características técnicas adicionales como robustez y escalabilidad, además de una amplia adopción en aplicaciones empresariales. Mi experiencia previa con Java también ha sido un factor determinante, ya que me permite desarrollar la aplicación de manera más eficiente y efectiva, asegurando así la calidad y el éxito del proyecto.

Desarrollar *MyDrive* únicamente con Java, sin el uso de un *framework*, sería una tarea ardua y laboriosa. Si bien Java es un lenguaje poderoso y versátil, carece de muchas de las características y funcionalidades que ofrecen los *frameworks* modernos. La configuración manual de aspectos como la gestión de dependencias, la configuración del servidor y la implementación de patrones de diseño comunes, como el modelo-vista-controlador o MVC [1], requeriría un tiempo considerable y aumentaría la complejidad del proyecto.

Most Popular Backend Frameworks

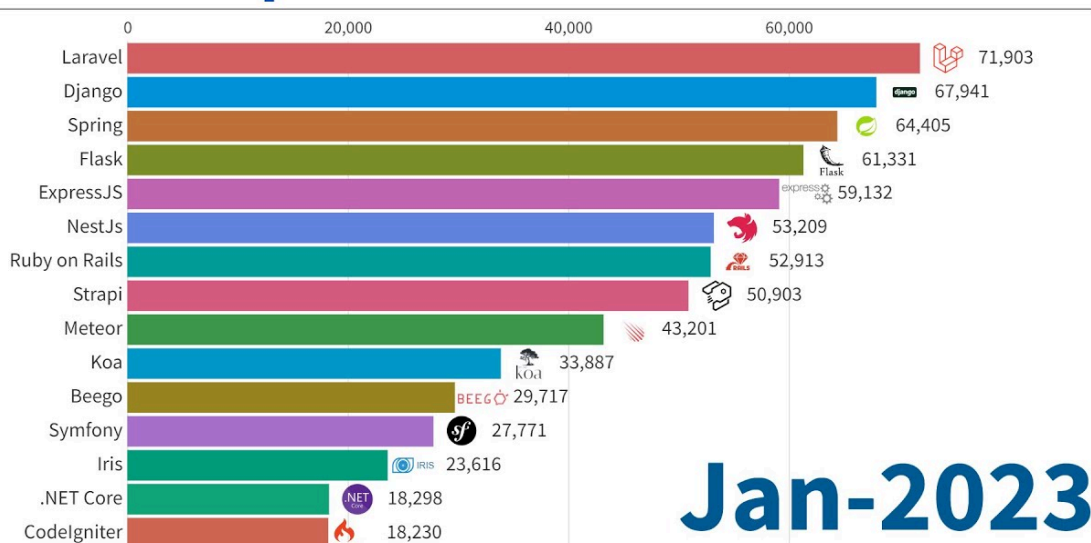


Ilustración 6. Frameworks backend más populares Enero 2023⁷

Para desarrollar *MyDrive* se ha escogido Spring [6], que se trata de un *framework* de desarrollo de aplicaciones Java que facilita la creación de aplicaciones web robustas y escalables. Ofrece una amplia gama de características integradas, como soporte para la creación de servicios web RESTful, seguridad, gestión de dependencias y pruebas unitarias, lo que acelera el desarrollo y mejora la calidad del software. Además, existe Spring Boot una herramienta extendida de Spring que te permite crear un prediseño de estructura de aplicaciones para facilitar tu desarrollo.

2.1.3- Estudio de tecnologías: frontend

La elección de tecnologías para el desarrollo del *frontend* de *MyDrive* es crucial para garantizar una interfaz de usuario intuitiva, responsiva y eficiente. Al igual que con el *backend*, existen diversas tecnologías y *frameworks* disponibles para el desarrollo *frontend*. A continuación, se presenta un análisis de algunas de las opciones más comunes.

En el caso de los lenguajes utilizados para el desarrollo web, existen HTML (*HyperText Markup Language*) y CSS (*Cascading Style Sheets*) que son las piedras angulares del desarrollo web. HTML se utiliza para estructurar el contenido de la web, mientras que CSS se encarga del diseño y la disposición visual. Aunque fundamentales, por sí solos no son suficientes para crear aplicaciones web interactivas y dinámicas. JavaScript, por otro lado, es el lenguaje de programación principal para el desarrollo *frontend*, permitiendo la creación de páginas web interactivas. Es un lenguaje versátil y ampliamente soportado en todos los

⁷ Fuente: <https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/>

navegadores modernos, lo que lo convierte en una opción esencial para el desarrollo web.

Cuando se utiliza Java para el desarrollo del *backend* de una aplicación web, existen múltiples opciones para el desarrollo del *frontend*. La elección del enfoque y la tecnología adecuada depende de factores, como los requisitos de la aplicación. Se pueden encontrar dos escenarios principales: el renderizado del lado del servidor y el renderizado del lado del cliente.

El renderizado en el servidor [3] (*Server-Side Rendering* - SSR) es una técnica donde las páginas web se generan en el servidor y se envían completamente renderizadas al cliente. Esto mejora el rendimiento inicial de la carga de la página y es beneficioso para el SEO (*Search Engine Optimization*), ya que los motores de búsqueda pueden indexar el contenido fácilmente. Sin embargo, SSR también presenta ciertas desventajas. En comparación con el renderizado en el cliente, puede aumentar la carga en el servidor, ya que éste debe procesar y generar las vistas para cada solicitud. Esto puede resultar en tiempos de respuesta más largos si el servidor está bajo mucha carga. Algunas de las tecnologías que podemos encontrar dentro del ecosistema Java son:

- **JSP (JavaServer Pages):** JSP permite la generación de contenido HTML dinámico en el servidor antes de enviar la respuesta al cliente. Es adecuado para aplicaciones donde el SEO es crucial y para sitios que no requieren mucha interactividad.
- **Thymeleaf:** Thymeleaf es un motor de plantillas moderno que se integra bien con Spring Boot, permitiendo la creación de vistas dinámicas en el servidor. Es una alternativa más amigable y moderna a JSP.
- **JSF (JavaServer Faces):** JSF es un framework Java para construir interfaces de usuario basadas en componentes, que permite el renderizado en el servidor. Es útil para aplicaciones empresariales con interfaces complejas y componentes reutilizables.

El Renderizado en el Cliente [3] (*CSR, Client-Side Rendering*) es un enfoque en el desarrollo de aplicaciones web donde el contenido de la página se genera en el navegador del usuario, utilizando JavaScript. En lugar de recibir una página HTML completamente formada desde el servidor, el navegador descarga una página básica y luego utiliza JavaScript para completar el contenido dinámicamente. Este enfoque se ha vuelto popular con el auge de los *frameworks* de JavaScript, que permiten una mejor organización del código y posibilitan desarrollar aplicaciones consistentes y escalables. Además, ayudan a manejar el estado de la aplicación, la comunicación con el *backend*, y el renderizado eficiente de la interfaz de usuario. Algunos de los frameworks JavaScript más usados son:

- **Angular:** Desarrollado por Google, Angular es un framework de JavaScript robusto y ampliamente utilizado. Ofrece una arquitectura basada en componentes, *two-way data binding*, y una variedad de herramientas integradas para el desarrollo de aplicaciones web complejas. Sin embargo, su curva de aprendizaje puede ser muy alta, y su configuración inicial puede ser complicada.
- **React:** Creado por Facebook, React es una librería de JavaScript enfocada en la construcción de interfaces de usuario. Utiliza un enfoque basado en componentes y un virtual DOM para mejorar el rendimiento. React es conocido por su flexibilidad y capacidad para integrarse con otras librerías y *frameworks*. Sin embargo, esta flexibilidad también puede resultar en una mayor complejidad en la gestión del estado y la configuración.
- **Vue.js:** Vue.js es un framework progresivo de JavaScript que se ha ganado una gran popularidad por su simplicidad y facilidad de integración. Ofrece una arquitectura basada en componentes similar a la de React y Angular, pero con una curva de aprendizaje más suave y una configuración inicial más sencilla. Vue.js es flexible y puede ser utilizado tanto para proyectos pequeños como grandes, lo que lo hace adecuado para una amplia gama de aplicaciones.

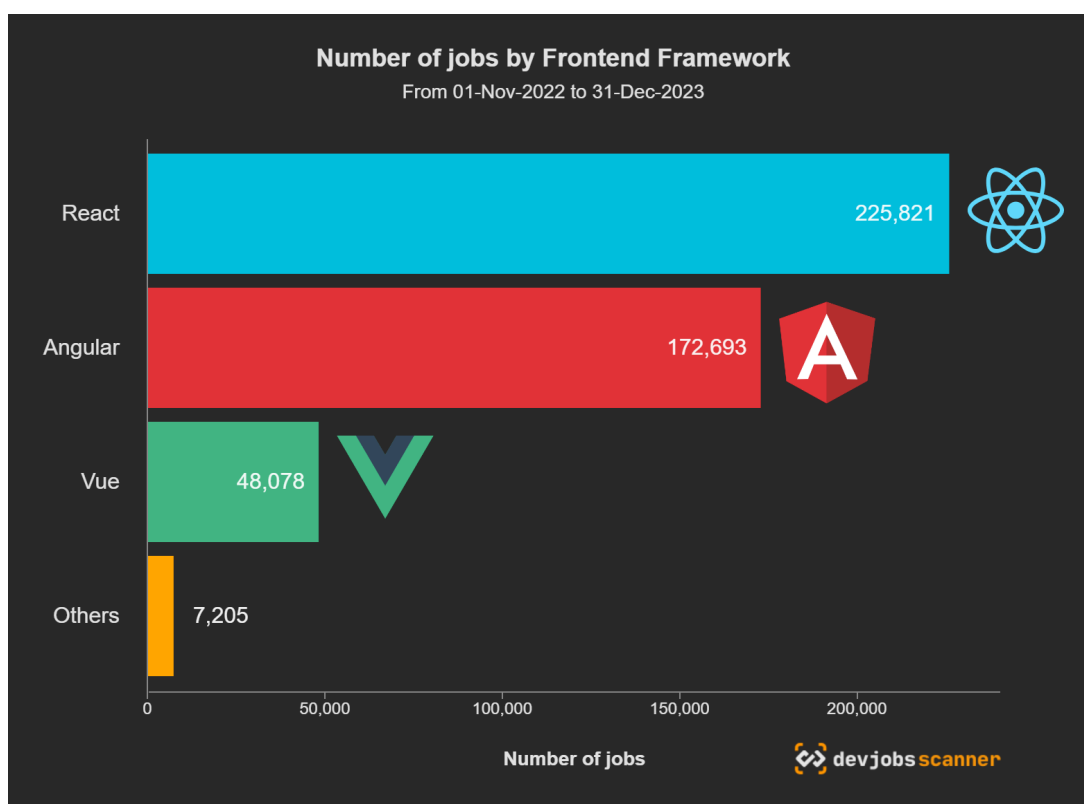


Ilustración 7. Frameworks frontend más demandados 2023⁸

⁸ Fuente: <https://www.devjobsscanner.com/blog/the-most-demanded-frontend-frameworks/>

Para el desarrollo del *frontend* de *MyDrive*, se ha elegido un escenario de tipo CSR, ya que ofrece, una mayor interactividad, y una separación de responsabilidades, usando como *framework* Vue.js [7], debido a su curva de aprendizaje más suave, lo cual es especialmente beneficioso dado que prácticamente no había utilizado este *framework* anteriormente. Además, Vue.js ofrece características técnicas destacadas como su arquitectura basada en componentes, flexibilidad para integrarse con otras herramientas, y un rendimiento eficiente gracias al uso del virtual DOM. La simplicidad en la configuración inicial y una comunidad activa que proporciona recursos y soporte adicionales también han sido factores clave en esta elección.

2.2.- Propuesta de solución

MyDrive implica el desarrollo de una aplicación web avanzada que permita a los usuarios gestionar sus archivos almacenados en múltiples servicios de almacenamiento en la nube desde una única interfaz. Este sistema ofrecerá una serie de funcionalidades clave dentro de cada servicio y entre diferentes servicios de almacenamiento en la nube. Este proyecto tendrá a su vez dos subproyectos formados por el *backend* y el *frontend*.

El *backend* se encargará de manejar la lógica de negocio y las interacciones con diferentes APIs de servicios en la nube como Google Drive y Dropbox. Esto incluye la autenticación de usuarios, la gestión de tokens de acceso, y la realización de operaciones tipo CRUD. Para este subproyecto, como he comentado, se ha elegido Java junto con el *framework* Spring Boot. Esta combinación proporciona una base robusta y escalable para la aplicación, permitiendo gestionar eficientemente las operaciones del servidor y las interacciones con los servicios de almacenamiento en la nube.

El *frontend* se encargará de renderizar la interfaz de usuario y de realizar operaciones en el cliente, además de enviar las peticiones al *backend* para ejecutar las diferentes funcionalidades disponibles, proporcionando una experiencia de usuario fluida y eficiente. Para este subproyecto, como he comentado, se ha escogido Vue.js, que permite la creación de componentes reutilizables y una gestión eficiente del estado de la aplicación, mejorando así la mantenibilidad y escalabilidad del sistema.

2.3.- Historias de Usuario

Una vez escogidas las tecnologías que vamos a emplear, ahora llega el momento de redactar las diversas historias de usuario que guiarán el desarrollo del proyecto siguiendo una metodología ágil como Scrum, aunque ligeramente adaptada para un equipo formado por un único miembro. Cada historia de usuario se detalla minuciosamente para asegurar una comprensión clara de los objetivos a alcanzar.

Además, se incluyen los criterios de aceptación que permitirán validar la correcta implementación de cada funcionalidad.

Cada historia de usuario se priorizará en categorías de muy alta, alta, media, baja y muy baja, según su importancia y urgencia en el proyecto. Esta priorización ayudará a enfocar los esfuerzos en las funcionalidades más críticas y asegurar que el valor más significativo se entregue primero.

Por otro lado, la estimación de tiempo que habrá que dedicar para desarrollar cada historia de usuario es la suma de tareas necesarias para completar cada historia de usuario. Este desglose de tareas está explicado en profundidad en el siguiente apartado, pero se indicará en este para tener recogidas todas las historias de usuario con todos sus datos establecidos, a modo de guía.

Historia de usuario		Nº 01
Nombre:	Conectar servicios de almacenamiento en la nube	
Descripción:	Como usuario, quiero identificarme en la aplicación usando mis cuentas de otros servicios de almacenamiento en la nube, para poder acceder y gestionar mis archivos de manera centralizada desde una única interfaz.	
Criterios de aceptación:	1. El usuario debe poder seleccionar el servicio de almacenamiento en la nube que desea conectar. 2. La aplicación debe redirigir al usuario a la página de autenticación del servicio seleccionado. 3. Tras una autenticación exitosa, la aplicación debe mostrar un mensaje de confirmación y permitir el acceso a los archivos del usuario. 4. En caso de error durante la autenticación, la aplicación deberá mostrar un mensaje de error al usuario.	
Estimación: 19h		Prioridad: Muy alta

Tabla 1. HU01: Conectar servicios de almacenamiento en la nube

Historia de usuario		Nº 02
Nombre:	Desconectar servicios de almacenamiento en la nube	
Descripción:	Como usuario, quiero cerrar sesión en un servicio de almacenamiento en la nube concreto, para proteger mi información personal y evitar accesos no autorizados.	
Criterios de aceptación:	1. El usuario debe poder cerrar la sesión en el servicio de almacenamiento en la nube que desea desconectar. 2. Tras una autenticación exitosa, la aplicación debe mostrar de nuevo la opción para volver a conectar dicho servicio. 3. En caso de error durante el cierre de sesión, la aplicación deberá mostrar un mensaje de error al usuario.	
Estimación: 12.5h		Prioridad: Muy alta

Tabla 2. HU02: Desconectar servicios de almacenamiento en la nube

Historia de usuario		Nº 03
Nombre:	Obtener todo el contenido de una carpeta	
Descripción:	Como usuario, quiero obtener las carpetas y archivos dentro de una carpeta específica de cada servicio en la nube vinculado, para poder visualizar todo el contenido dentro de una carpeta.	
Criterios de aceptación:	1. El usuario tras conectar algún servicio en la nube deberá poder visualizar el contenido de la carpeta raíz de manera automática. 2. La interfaz debe mostrar los nombres de las carpetas y archivos, así como cualquier otra información relevante. 3. En caso de error durante la obtención de archivos, la aplicación deberá redirigir a una página de error.	
Estimación: 17h		Prioridad: Muy alta

Tabla 3. HU03: Obtener todo el contenido de una carpeta

Historia de usuario		Nº 04
Nombre:	Navegación entre carpetas	
Descripción:	Como usuario, quiero poder abrir una carpeta, para obtener todo su contenido y así de forma recursiva dentro de cada servicio en la nube vinculado, permitiendo la navegación entre directorios.	
Criterios de aceptación:	1. Se debe proporcionar alguna opción de navegación para permitir al usuario moverse entre las carpetas y subcarpetas dentro de la interfaz. 2. La aplicación debe mostrar la ruta de carpetas desde el directorio raíz hasta el actual, permitiendo acceder directamente a una de estas. 3. En caso de error durante la obtención de la ruta de carpetas tras acceder a una carpeta, la aplicación deberá únicamente el directorio raíz y mostrar un error informativo al usuario.	
Estimación: 13h		Prioridad: Alta

Tabla 4. HU04: Navegación entre carpetas

Historia de usuario		Nº 05
Nombre:	Visualización de archivos	
Descripción:	Como usuario, quiero poder abrir un archivo dentro de cada servicio en la nube vinculado, para previsualizar su contenido en el navegador sin necesidad de descargarlo.	
Criterios de aceptación:	1. Se debe proporcionar alguna opción para abrir un archivo para que el usuario pueda previsualizar su contenido. 2. Dicha opción deberá redirigir al usuario a la página oficial del servicio en la nube indicado para ver la previsualización del archivo. 3. En caso de que dicha opción esté limitada, se deberá mostrar un mensaje informativo al usuario.	
Estimación: 8h		Prioridad: Media

Tabla 5. HU05: Visualización de archivos

Historia de usuario		Nº 06
Nombre:	Creación de una carpeta	
Descripción:	Como usuario, quiero poder crear una subcarpeta dentro de una carpeta padre dentro de cada servicio en la nube vinculado, para organizar mejor mis archivos y mantener una estructura ordenada.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para crear una nueva subcarpeta dentro de la carpeta padre indicada. 2. El usuario debe poder asignar un nombre a la nueva subcarpeta al momento de su creación. 3. La nueva subcarpeta debe aparecer inmediatamente en la lista de carpetas dentro de la carpeta padre. 4. La aplicación debe mostrar un mensaje de confirmación una vez que la subcarpeta haya sido creada con éxito. 5. La nueva subcarpeta debe ser accesible y funcional, al igual que el resto de carpetas del sistema. 6. En caso de error durante la creación de la subcarpeta, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 13h		Prioridad: Alta

Tabla 6. HU06: Creación de una carpeta

Historia de usuario		Nº 07
Nombre:	Subir archivos	
Descripción:	Como usuario, quiero poder subir múltiples archivos dentro de una carpeta padre dentro de cada servicio en la nube vinculado, para organizar mejor mis archivos y mantener una estructura ordenada.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para subir los archivos seleccionados dentro de la carpeta padre indicada. 2. La aplicación debe permitir soltar los archivos arrastrados para subir estos archivos dentro de la carpeta padre indicada. 3. La aplicación debe permitir al usuario introducir una contraseña para cifrar los archivos antes de subirlos a la nube. 4. Los archivos deben aparecer inmediatamente en la lista de archivos dentro de la carpeta padre. 5. La aplicación debe mostrar un mensaje de confirmación una vez que los archivos hayan sido subidos con éxito. 6. Los nuevos archivos deben ser funcionales, al igual que el resto de archivos del sistema. 7. En caso de error durante la subida de archivos, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 27h		Prioridad: Alta

Tabla 7. HU07: Subir archivos

Historia de usuario		Nº 08
Nombre:	Descargar archivos	
Descripción:	Como usuario, quiero poder descargar múltiples archivos de una carpeta dentro de cada servicio en la nube vinculado, para tener acceso fuera de línea a estos.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para descargar los archivos seleccionados dentro de una carpeta. 2. Si se están descargando archivos previamente cifrados por la aplicación, ésta deberá mostrar un apartado para introducir la contraseña correspondiente y descifrarlos antes de iniciar la descarga. 3. Los archivos deben aparecer inmediatamente en el apartado de descargas del navegador del usuario. 4. La aplicación debe mostrar un mensaje de confirmación una vez que los archivos hayan sido descargados con éxito. 5. En caso de error durante la descarga de archivos, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 24.5h		Prioridad: Alta

Tabla 8: HU08: Descargar archivos

Historia de usuario		Nº 09
Nombre:	Mover carpetas o archivos	
Descripción:	Como usuario, quiero poder mover múltiples carpetas o archivos dentro de una carpeta a otra dentro de cada servicio vinculado, para organizar mejor mis archivos y facilitar su gestión.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para mover las carpetas o archivos seleccionados dentro de una carpeta. 2. La aplicación debe mostrar un listado de carpetas, para navegar entre ellas hasta que el usuario indique la carpeta final. 3. Las carpetas o archivos deben desaparecer inmediatamente del listado de carpetas o archivos de la carpeta origen. 4. Las carpetas con todo su contenido o archivos, deben aparecer inmediatamente en el listado de carpetas o archivos de la carpeta destino. 5. La aplicación debe mostrar un mensaje de confirmación una vez que las carpetas o archivos hayan sido movidos con éxito. 6. En caso de error durante la movida de carpetas o archivos, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 13.5h		Prioridad: Alta

Tabla 9. HU09: Mover carpetas o archivos

Historia de usuario		Nº 10
Nombre:	Mover archivos entre nubes	
Descripción:	Como usuario, quiero poder mover múltiples archivos dentro de una carpeta a otra en un servicio diferente vinculado, para transferir fácilmente archivos de una nube a otra.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para mover los archivos seleccionados dentro de una carpeta. 2. La aplicación debe mostrar disponibles los servicios en la nube vinculados por el usuario y este deberá poder seleccionar uno de ellos. 3. La aplicación debe mostrar un listado de carpetas, para navegar entre ellas hasta que el usuario indique la carpeta final, dentro del servicio destino. 4. Los archivos deben copiarse y aparecer en el listado de archivos de la carpeta destino. 5. La aplicación debe mostrar un mensaje de confirmación una vez que los archivos hayan sido movidos con éxito. 6. En caso de error durante la movida de archivos, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 15h		Prioridad: Alta

Tabla 10. HU10: Mover archivos entre nubes

Historia de usuario		Nº 11
Nombre:	Renombrar archivo	
Descripción:	Como usuario, quiero poder renombrar un archivo en específico dentro de cada servicio vinculado, para organizar mejor mis archivos y facilitar su gestión.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para renombrar un archivo en concreto. 2. El nombre del archivo debe de verse reflejado de manera inmediata. 3. La aplicación debe mostrar un mensaje de confirmación una vez que el archivo haya sido renombrado con éxito. 4. En caso de error durante el renombrado del archivo, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 7h		Prioridad: Baja

Tabla 11. HU11: Renombrar archivo

Historia de usuario		Nº 12
Nombre:	Mover a la papelera los archivos	
Descripción:	Como usuario, quiero poder mover múltiples archivos de una carpeta a la papelera dentro de cada servicio vinculado, para deshacerme de aquellos archivos innecesarios, sin eliminarlos por completo.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para mover los archivos seleccionados dentro de una carpeta a la papelera. 2. Los archivos deben desaparecer inmediatamente del listado de archivos de la carpeta origen. 3. Los archivos deben aparecer inmediatamente en el listado de archivos de la papelera. 4. La aplicación debe mostrar un mensaje de confirmación una vez que los archivos hayan sido movidos a la papelera con éxito. 5. En caso de error durante la movida de archivos, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 14h		Prioridad: Media

Tabla 12. HU12: Mover a la papelera los archivos

Historia de usuario		Nº 13
Nombre:	Eliminar carpetas	
Descripción:	Como usuario, quiero poder eliminar de forma permanente múltiples carpetas de una carpeta dentro de cada servicio vinculado, para organizar mejor mis archivos y deshacerme por completo de aquellos innecesarios.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para eliminar las carpetas seleccionadas dentro de una carpeta. 2. La aplicación debe mostrar de forma visible una opción de confirmación, indicando que se eliminarán dichos archivos de manera permanente. 3. Las carpetas deben desaparecer inmediatamente del listado de carpetas de la carpeta padre, junto con todos sus archivos y subcarpetas contenidas. 4. La aplicación debe mostrar un mensaje de confirmación una vez que las carpetas hayan sido eliminadas con éxito. 5. En caso de error durante la eliminación, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 14h		Prioridad: Media

Tabla 13. HU13: Eliminar carpetas

Historia de usuario		Nº 14
Nombre:	Eliminar archivos	
Descripción:	Como usuario, quiero poder eliminar de forma permanente múltiples archivos de una carpeta dentro de cada servicio vinculado, para organizar mejor mis archivos y deshacerme por completo de aquellos innecesarios.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para eliminar definitivamente los archivos previamente enviados a la papelera. 2. Los archivos deben desaparecer inmediatamente del listado de archivos de la papelera. 3. La aplicación debe mostrar un mensaje de confirmación una vez que los archivos hayan sido eliminados con éxito. 4. En caso de error durante la eliminación, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 14h		Prioridad: Media

Tabla 14. HU14: Eliminar archivos

Historia de usuario		Nº 15
Nombre:	Obtener todos los archivos de la papelera	
Descripción:	Como usuario, quiero poder obtener todos los archivos que haya enviado a la papelera dentro de cada servicio vinculado, para poder visualizarlos y en consecuencia gestionarlos a mi antojo.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible para acceder a la papelera. 2. La aplicación debe mostrar una lista con todos los archivos eliminados. Dichos archivos deberán mostrar opciones diferentes relacionadas con su restauración y eliminación. 3. En caso de error durante el acceso a la papelera, la aplicación debe redirigir a una página de error.	
Estimación: 17h		Prioridad: Alta

Tabla 15. HU15: Obtener todos los archivos de la papelera

Historia de usuario		Nº 16
Nombre:	Vaciar papelera	
Descripción:	Como usuario, quiero poder eliminar todos los archivos que haya enviado a la papelera dentro de cada servicio vinculado, para liberar espacio de almacenamiento y mantener mis archivos organizados.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible para vaciar la papelera. 2. Los archivos deben desaparecer inmediatamente de la papelera, quedando vacía la lista de archivos de esta. 3. En caso de error durante el borrado de archivos, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 12h		Prioridad: Baja

Tabla 16. HU16: Vaciar papelera

Historia de usuario		Nº 17
Nombre:	Restaurar archivos	
Descripción:	Como usuario, quiero poder restaurar múltiples archivos de la lista de archivos de la papelera dentro de cada servicio vinculado, para recuperar archivos importantes y volver a utilizarlos en su ubicación original.	
Criterios de aceptación:	1. La aplicación una vez en la papelera debe proporcionar una opción visible y accesible para restaurar los archivos seleccionados. 2. Los archivos deben desaparecer inmediatamente de la lista de archivos de la papelera. 3. Cada uno de los archivos restaurados debe aparecer en la ruta original, antes de que fuera enviado a la papelera. 4. La aplicación debe mostrar un mensaje de confirmación una vez que los archivos hayan sido restaurados con éxito. 5. En caso de error durante la restauración, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 11h		Prioridad: Baja

Tabla 17. HU17: Restaurar archivos

Historia de usuario		Nº 18
Nombre:	Obtener archivos recientes	
Descripción:	Como usuario, quiero poder visualizar una lista de archivos recientes, para acceder rápidamente a archivos importantes recientemente utilizados.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para obtener los archivos más recientes. 2. La aplicación obtendrá los archivos dentro de un periodo reciente. Este periodo se debe poder modificar para ampliar o disminuir el rango. 3. En caso de error durante la obtención de archivos recientes, la aplicación deberá redirigir a una página de error.	
Estimación: 19h		Prioridad: Media

Tabla 18. HU18: Obtener archivos recientes

Historia de usuario		Nº 19
Nombre:	Buscar carpetas y archivos	
Descripción:	Como usuario, quiero poder buscar carpetas y archivos en base a un nombre dentro de cada servicio vinculado, para localizar rápidamente el contenido que necesito sin tener que navegar manualmente por todas mis carpetas.	
Criterios de aceptación:	1. La aplicación debe proporcionar una barra de búsqueda, para obtener los archivos que coinciden con dicho nombre. Se debe poder indicar si la búsqueda es a nivel general o dentro de una carpeta específica. 2. La aplicación obtendrá la lista de carpetas y de archivos, que coinciden con el nombre indicado. 3. Las carpetas y archivos obtenidos deben ser totalmente funcionales. 4. En caso de error durante la búsqueda, la aplicación debe redirigir a una página de error.	
Estimación: 23.5h		Prioridad: Alta

Tabla 19. HU19: Buscar carpetas y archivos

Historia de usuario		Nº 20
Nombre:	Ordenar carpetas y archivos	
Descripción:	Como usuario, quiero poder ordenar carpetas y archivos en base a un criterio de una carpeta dentro de cada servicio vinculado, para localizar rápidamente el contenido que necesito sin tener que recorrer todo el contenido de dicha carpeta.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para ordenar las carpetas y archivos dentro de una carpeta. Se debe poder ordenar por nombre (por defecto), fecha y tamaño. 2. La aplicación debe modificar la ordenación de la lista de carpetas y de archivos, según el criterio indicado.	
Estimación: 9h		Prioridad: Baja

Tabla 20. HU20: Ordenar carpetas y archivos

Historia de usuario		Nº 21
Nombre:	Obtener información usuario	
Descripción:	Como usuario, quiero poder visualizar mi nombre y mi foto de perfil dentro de cada servicio vinculado, para confirmar fácilmente que estoy utilizando la cuenta correcta y tener una experiencia personalizada.	
Criterios de aceptación:	1. La aplicación debe mostrar la foto de perfil del usuario junto con su nombre en la nube que esté usando. 2. En caso de no poder obtener dicha información, se debe mostrar una foto por defecto y como nombre 'usuario'.	
Estimación: 15h		Prioridad: Media

Tabla 21. HU21: Obtener información usuario

Historia de usuario		Nº 22
Nombre:	Compartir archivos	
Descripción:	Como usuario, quiero poder compartir archivos almacenados en un servicio de almacenamiento en la nube con otros usuarios, para facilitar la colaboración y el intercambio de información.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para poder compartir archivos en un servicio vinculado. 2. La aplicación deberá generar un enlace que mostrará al usuario para que envíe a los usuario con los que quiera compartir dicho archivo. 3. Los archivos solo deben tener permisos de visualización, para ver el contenido. 4. En caso de error durante la obtención del enlace para compartir, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 8h		Prioridad: Muy baja

Tabla 22: HU22: Compartir archivos

Historia de usuario		Nº 23
Nombre:	Crear carpeta espejo	
Descripción:	Como usuario, quiero poder crear una carpeta espejo, para que todo lo que suba a esta carpeta se replique automáticamente en otra carpeta en un servicio de almacenamiento en la nube diferente.	
Criterios de aceptación:	1. La aplicación debe proporcionar una opción visible y accesible para crear una nueva subcarpeta dentro de la carpeta padre indicada. 2. El usuario debe poder asignar un nombre a la nueva subcarpeta al momento de su creación, además de indicar en qué servicio en la nube deberá replicarse. 3. La nueva subcarpeta debe aparecer inmediatamente en la lista de carpetas dentro de la carpeta padre en el servicio principal. En el servicio donde se deba crear la carpeta espejo, se crearán todas las carpetas padres o intermedias hasta llegar a la carpeta final. 4. La aplicación debe mostrar un mensaje de confirmación una vez que la carpeta espejo haya sido creada con éxito. 5. La carpeta espejo debe cumplir que todas las carpetas o archivos subidos deben de replicarse en la otra carpeta. 6. En caso de error durante la creación de la carpeta espejo, la aplicación debe mostrar un mensaje de error informativo.	
Estimación: 13.5h		Prioridad: Muy baja

Tabla 23. HU23: Crear carpeta espejo

Historia de usuario		Nº 24
Nombre:	Búsqueda unificada	
Descripción:	Como usuario, quiero poder buscar carpetas y archivos en base a un nombre en todos los servicios vinculados, para localizar rápidamente el contenido que necesito sin tener que navegar manualmente por todas mis carpetas de todos los servicios.	
Criterios de aceptación:	1. La aplicación debe proporcionar una barra de búsqueda, para obtener los archivos que coinciden con dicho nombre. 2. La aplicación obtendrá una lista de carpetas y de archivos, que coinciden con el nombre indicado por cada servicio en la nube vinculado. 3. Las carpetas y archivos obtenidos deben ser totalmente funcionales. 4. En caso de error durante la búsqueda, la aplicación debe redirigir a una página de error.	
Estimación: 12.5h		Prioridad: Muy baja

Tabla 24. HU24: Búsqueda unificada

2.4.- Planificación inicial de tareas

En este apartado, se detalla la planificación inicial de las tareas necesarias para el desarrollo del proyecto *MyDrive*. El proceso implica descomponer las historias de usuario, previamente redactadas, en tareas más pequeñas y manejables, que constituirán la pila de producto (product backlog). A partir de esta pila, se propondrán iteraciones necesarias (sprints) y se asignarán tareas.

ID	Nombre historia de usuario	Tareas	Estima. (horas)
HU 01	Conectar servicios de almacenamiento en la nube.	1. Diseño interfaz de usuario para la selección del servicio.	1h
		2. Implementación frontend para la selección y redirección de autenticación.	6h
		3. Integración con las APIs de autenticación de servicios en la nube.	7h

		4. Desarrollo del backend para manejar tokens y autenticación.	4h
		5. Pruebas y validación del flujo de autenticación y mensajes de error.	1h
		Horas totales:	

Tabla 25. Desglose en tareas y estimación de tiempos para HU01

HU 02	Desconectar servicios de almacenamiento en la nube.	1. Diseño de la interfaz de usuario para la desconexión de servicios.	0.5h
		2. Implementación del frontend para cerrar sesión y gestionar estados.	6h
		3. Desarrollo del backend para eliminar tokens y manejar desconexión.	5h
		4. Pruebas y validación del cierre de sesión y mensajes de error.	1h
Horas totales:			12.5h

Tabla 26. Desglose en tareas y estimación de tiempos para HU02

HU 03	Obtener todo el contenido de una carpeta.	1. Diseño de la interfaz de usuario para la visualización de carpetas.	2h
		2. Implementación del frontend para mostrar el listado de carpetas y archivos.	6h
		3. Integración con las APIs de los servicios para listar carpetas y archivos.	5h
		4. Desarrollo del backend para recuperar, transformar y enviar datos al frontend.	2h
		5. Pruebas y validación de la obtención y visualización de carpetas y archivos.	2h
Horas totales:			17h

Tabla 27. Desglose en tareas y estimación de tiempos para HU03

HU 04	Navegación entre carpetas.	1. Diseño de la interfaz de usuario para mostrar la ruta de carpetas.	1h
		2. Implementación del frontend para navegar y actualizar vistas.	5h
		3. Integración con las APIs de los servicios para obtener contenido de subcarpetas.	4h
		4. Desarrollo del backend para manejar peticiones de navegación.	2h
		5. Pruebas y validación de la navegación y carga de contenido.	1h
Horas totales:			13h

Tabla 28. Desglose en tareas y estimación de tiempos para HU04

HU 05	Visualización de archivos.	1. Diseño de la interfaz de usuario para la visualización de archivos.	0.5h
		2. Implementación del frontend para mostrar una previsualización de los archivos.	2h
		3. Integración con las APIs de los servicios para acceder a los archivos.	3h
		4. Desarrollo del backend para gestionar peticiones de visualización.	2h
		5. Pruebas y validación de la visualización de diferentes tipos de archivos.	0.5h
Horas totales:			8h

Tabla 29. Desglose en tareas y estimación de tiempos para HU05

HU 06	Creación de una carpeta.	1. Diseño de la interfaz de usuario para la creación de carpetas.	1h
		2. Implementación del frontend para gestionar la creación de carpetas.	4h
		3. Integración con las APIs de los servicios para crear carpetas.	4h
		4. Desarrollo del backend para manejar las peticiones de creación.	3h

		5. Pruebas y validación de la creación de carpetas y mensajes de confirmación.	1h
Horas totales:			13h

Tabla 30. Desglose en tareas y estimación de tiempos para HU06

HU 07	Subir archivos.	1. Diseño de la interfaz de usuario para la subida de archivos.	2h	
		2. Implementación del frontend para gestionar la selección y subida de archivos, cifrandolos en base a una contraseña, si el usuario lo desea.	9h	
		3. Integración con las APIs de los servicios para la carga de archivos.	4h	
		4. Desarrollo del backend para manejar las peticiones de subida y aplicar los algoritmos de cifrado necesarios si se indica.	10h	
		5. Pruebas y validación de la subida de archivos y gestión de errores.	2h	
			Horas totales:	27h

Tabla 31. Desglose en tareas y estimación de tiempos para HU07

HU 08	Descargar archivos.	1. Diseño de la interfaz de usuario para la descarga de archivos.	1.5h
		2. Implementación del frontend para gestionar la selección y descarga de archivos, solicitando la contraseña con la que se cifró, en caso de estar cifrados.	7h
		3. Integración con las APIs de los servicios para la descarga de archivos.	4h
		4. Desarrollo del backend para manejar las peticiones de descarga, descifrando los archivos en caso necesario.	10h
		5. Pruebas y validación de la descarga de archivos y gestión de errores.	2h
Horas totales:			24.5h

Tabla 32. Desglose en tareas y estimación de tiempos para HU08

HU 09	Mover carpetas o archivos.	1. Diseño de la interfaz de usuario para mover archivos o carpetas.	2h
		2. Implementación del frontend para mover carpetas o archivos. En caso de ser carpetas, se deben poder mover entre nubes.	4h
		3. Integración con las APIs de los servicios para mover carpetas o archivos dentro de una misma nube o archivos entre distintas nubes.	3h
		4. Desarrollo backend para manejar las peticiones de movimiento.	3h
		5. Pruebas y validación del movimiento dentro de una nube y entre diferentes, además de gestión de errores.	1.5h
Horas totales:			13.5h

Tabla 33. Desglose en tareas y estimación de tiempos para HU09

HU 10	Mover archivos entre nubes.	1. Diseño de la interfaz de usuario para mover archivos o carpetas.	1h	
		2. Implementación del frontend para mover carpetas o archivos. En caso de ser carpetas, se deben poder mover entre nubes.	5h	
		3. Integración con las APIs de los servicios para mover carpetas o archivos dentro de una misma nube o archivos entre distintas nubes.	4h	
		4. Desarrollo backend para manejar las peticiones de movimiento.	3.5h	
		5. Pruebas y validación del movimiento dentro de una nube y entre diferentes, además de gestión de errores.	1.5h	
			Horas totales:	15h

Tabla 34. Desglose en tareas y estimación de tiempos para HU10

HU 11	Renombrar archivo.	1. Diseño de la interfaz de usuario para renombrar archivos.	0.5h	
		2. Implementación del frontend para gestionar la selección y renombrar dicho archivo.	3h	
		3. Integración con las APIs de los servicios para modificar parámetros como el nombre de los archivos.	2h	
		4. Desarrollo del backend para manejar las peticiones de renombrar.	1h	
		5. Pruebas y validación del renombrar y gestión de errores.	0.5h	
			Horas totales:	7h

Tabla 35. Desglose en tareas y estimación de tiempos para HU11

HU 12	Mover a la papelera los archivos.	1. Diseño de la interfaz de usuario para la gestión de papelera.	1h
		2. Implementación del frontend para mover archivos a la papelera.	6h
		3. Integración con las APIs de los servicios para mover archivos a la papelera.	4h
		4. Desarrollo del backend para manejar las peticiones de papelera.	2h
		5. Pruebas y validación de mover archivos a la papelera y gestión de errores.	1h
Horas totales:			14h

Tabla 36. Desglose en tareas y estimación de tiempos para HU12

HU 13	Eliminar carpetas.	1. Diseño de la interfaz de usuario para la eliminación de carpetas.	1h
		2. Implementación del frontend para gestionar la selección y eliminación definitiva.	6h
		3. Integración con las APIs de los servicios para eliminar carpetas y sus subcarpetas y archivos de manera definitiva.	4h
		4. Desarrollo del backend para manejar las	2h

		peticiones de eliminación.	
		5. Pruebas y validación de la eliminación de carpetas y gestión de errores.	1h
Horas totales:			14h

Tabla 37. Desglose en tareas y estimación de tiempos para HU13

HU 14	Eliminar archivos.	1. Diseño de la interfaz de usuario para la eliminación de archivos.	1h	
		2. Implementación del frontend para gestionar la selección y eliminación definitiva.	6h	
		3. Integración con las APIs de los servicios para eliminar carpetas y sus subcarpetas y archivos de manera definitiva.	4h	
		4. Desarrollo del backend para manejar las peticiones de eliminación.	2h	
		5. Pruebas y validación de la eliminación de carpetas y gestión de errores.	1h	
			Horas totales:	14h

Tabla 38. Desglose en tareas y estimación de tiempos para HU14

HU 15	Obtener todos los archivos de la papelera.	1. Diseño de la interfaz de usuario para la visualización de papelera.	2h
		2. Implementación del frontend para listar archivos en la papelera.	6h
		3. Integración con las APIs de los servicios para obtener archivos de la papelera.	5h
		4. Desarrollo del backend para manejar las peticiones de obtener archivos enviados a la papelera.	2h
		5. Pruebas y validación de la obtención y visualización de archivos en la papelera y gestión de errores.	2h
Horas totales:			17h

Tabla 39. Desglose en tareas y estimación de tiempos para HU15

HU 16	Vaciar papelera.	1. Diseño de la interfaz de usuario para vaciar la papelera.	1h
		2. Implementación del frontend para gestionar la acción de vaciado.	4h
		3. Integración con las APIs de los servicios para eliminar de manera definitiva todos los archivos previamente enviados a la papelera.	4h
		4. Desarrollo del backend para manejar las peticiones de vaciado.	2h
		5. Pruebas y validación de la acción de vaciado y gestión de errores.	1h
Horas totales:			12h

Tabla 40. Desglose en tareas y estimación de tiempos para HU16

HU 17	Restaurar archivos.	1. Diseño de la interfaz de usuario para restaurar archivos.	1h
		2. Implementación del frontend para gestionar la selección y restauración de archivos de la papelera.	3h
		3. Integración con las APIs de los servicios para restaurar archivos a su ubicación original.	4h
		4. Desarrollo del backend para manejar las peticiones de restauración.	2h
		5. Pruebas y validación de la restauración y gestión de errores.	1h
Horas totales:			11h

Tabla 41. Desglose en tareas y estimación de tiempos para HU17

HU 18	Obtener archivos recientes.	1. Diseño de la interfaz de usuario para la visualización de archivos recientes.	2h
		2. Implementación del frontend para listar archivos recientes, permitiendo cambiar el periodo máximo (1 día, 1 sem., etc.).	8h
		3. Integración con las APIs de los servicios para obtener archivos recientes, según el periodo establecido.	5h

		4. Desarrollo del backend para manejar las peticiones de archivos recientes.	2h
		5. Pruebas y validación de la obtención y visualización de archivos recientes.	2h
			Horas totales:

Tabla 42. Desglose en tareas y estimación de tiempos para HU18

HU 19	Buscar carpetas y archivos.	1. Diseño de la interfaz de usuario para la búsqueda de archivos y carpetas dentro de la carpeta actual o a nivel global.	2.5h
		2. Implementación del frontend para gestionar la búsqueda a nivel de carpeta o a nivel global.	8h
		3. Integración con las APIs de los servicios para buscar archivos y carpetas.	8h
		4. Desarrollo del backend para manejar las peticiones de búsqueda.	2h
		5. Pruebas y validación de la búsqueda y visualización de resultados, además de gestión de errores.	3h
Horas totales:			23.5h

Tabla 43. Desglose en tareas y estimación de tiempos para HU19

HU 20	Ordenar carpetas y archivos.	1. Diseño de la interfaz de usuario para la ordenación de archivos y carpetas.	1h
		2. Implementación del frontend para gestionar la ordenación en base al alfabeto, más recientes o tamaño.	8h
			Horas totales:

Tabla 44. Desglose en tareas y estimación de tiempos para HU20

HU 21	Obtener información usuario.	1. Diseño de la interfaz de usuario para la visualización de información del usuario.	2h
		2. Implementación del frontend para mostrar la información del usuario como su nombre y su foto de perfil.	6h

		3. Integración con las APIs de los servicios para obtener información del usuario.	4h
		4. Desarrollo del backend para manejar las peticiones de información del usuario.	2h
		5. Pruebas y validación de la obtención y visualización de información del usuario.	1h
		Horas totales:	

Tabla 45. Desglose en tareas y estimación de tiempos para HU21

HU 22	Compartir archivos.	1. Diseño de la interfaz de usuario para compartir archivos con otros usuarios.	1h	
		2. Implementación del frontend para obtener el enlace que hace referencia al archivo.	2h	
		3. Integración con las APIs de los servicios para generar y obtener el enlace al archivo.	2.5h	
		4. Desarrollo del backend para manejar las peticiones para compartir un archivo.	1.5h	
		5. Pruebas y validación de la obtención y acceso al archivo del usuario.	1h	
			Horas totales:	8h

Tabla 46. Desglose en tareas y estimación de tiempos para HU22

HU 23	Crear carpeta espejo.	1. Diseño de la interfaz de usuario para la creación de una carpeta espejo.	1h
		2. Implementación del frontend para indicar en qué servicio deberá replicarse la carpeta.	4h
		3. Integración con las APIs de los servicios para crear carpetas en cada uno de ellos.	5h
		4. Desarrollo del backend para manejar las peticiones de creación de carpetas espejo.	2h
		5. Pruebas y validación de la creación y réplica de las carpetas y todo su contenido subido.	1.5h
Horas totales:			13.5h

Tabla 47. Desglose en tareas y estimación de tiempos para HU23

HU 24	Búsqueda unificada.	1. Diseño de la interfaz de usuario para la búsqueda de archivos y carpetas en todos los servicios vinculados.	1.5h	
		2. Implementación del frontend para gestionar la búsqueda a nivel de todos los servicios.	4h	
		3. Integración con las APIs de los servicios para buscar archivos y carpetas.	4h	
		4. Desarrollo del backend para manejar las peticiones de búsqueda.	2h	
		5. Pruebas y validación de la búsqueda y visualización de resultados, además de gestión de errores.	1h	
			Horas totales:	12.5h

Tabla 48. Desglose en tareas y estimación de tiempos para HU24

A continuación, se llevará a cabo la creación del *product backlog*, donde se listarán y ordenarán todas las historias de usuario necesarias para el desarrollo del proyecto. Este nos permitirá tener una visión clara de las funcionalidades que se deben implementar y el orden en el que se deben abordar, permitiéndonos crear una planificación detallada de todos los sprints necesarios para completar el proyecto.

Product Backlog			
Prioridad	Historia de usuario	Duración (horas)	Implementada
Muy alta	HU01	19h	SI 
Muy alta	HU02	12.5h	SI 
Muy alta	HU03	17h	SI 
Alta	HU04	13h	SI 
Alta	HU06	13h	SI 
Alta	HU07	27h	SI 
Alta	HU08	24.5h	SI 
Alta	HU09	13.5h	SI 
Alta	HU10	15h	SI 
Alta	HU15	17h	SI 

Alta	HU19	23.5h	SI ✓
Media	HU05	8h	SI ✓
Media	HU12	14h	SI ✓
Media	HU13	14h	SI ✓
Media	HU14	14h	SI ✓
Media	HU18	19h	SI ✓
Media	HU21	15h	SI ✓
Baja	HU11	7h	SI ✓
Baja	HU16	12h	SI ✓
Baja	HU27	11h	SI ✓
Baja	HU20	9h	SI ✓
Muy baja	HU22	8h	NO ✗
Muy baja	HU23	13.5h	NO ✗
Muy baja	HU24	12.5h	NO ✗

Tabla 49. Product backlog

Una vez obtenido el *product backlog* y desglosadas las historias de usuario en tareas más pequeñas y manejables, se procederá a la asignación de estas tareas a cada sprint. Sin embargo, debido a la cantidad de tareas involucradas en total, se ha decidido que en cada sprint se nombrará y se trabajará por historias de usuario, no por tareas individuales dentro de cada historia, dando por hecho que en cada historia nombrada habrá que dedicar el tiempo necesario a cada una de las tareas que la envuelven. Estas tareas se deberán asignar por orden de prioridad, es decir, aquellas tareas con mayor prioridad se realizarán antes y las que tengan menor prioridad se realizarán posteriormente, tal y como se han indicado en el *product backlog*. Finalmente, aquellas historias de usuario que se vayan completando se irán indicando en la última columna de la tabla 49.

Sprint 1 (semanas 1 y 2) - 31.5h

- HU01: Conectar servicios de almacenamiento en la nube (19h)
- HU02: Desconectar servicios de almacenamiento en la nube (12.5h)

Sprint 2 (semanas 3 y 4) - 30h

- HU03: Obtener todo el contenido de una carpeta (17h)
- HU04: Navegación entre carpetas (13h)

Sprint 3 (semanas 5 y 6) - 38h

- HU06: Creación de una carpeta (13h)
- HU07: Subir archivos (Primeras cuatro tareas 25h)

Sprint 4 (semanas 7 y 8) - 26.5h

- HU07: Subir archivos (Última tarea, pruebas 2h)
- HU08: Descargar archivos (24.5h)

Sprint 5 (semanas 9 y 10) - 36.5h

- HU09: Mover carpetas o archivos (13.5h)
- HU10: Mover archivos entre nubes (15h)
- HU15: Obtener todos los archivos de la papelera (Primeras dos tareas 8h)

Sprint 6 (semanas 11 y 12) - 32.5h

- HU15: Obtener todos los archivos de la papelera (Últimas tres tareas 9h)
- HU19: Buscar carpetas y archivos (23.5h)

Sprint 7 (semanas 13 y 14) - 27h

- HU18: Obtener archivos recientes (19h)
- HU05: Visualización de archivos (8h)

Sprint 8 (semanas 15 y 16)- 28h

- HU12: Mover a la papelera los archivos (14h)
- HU13: Eliminar carpetas (14h)

Sprint 9 (semanas 17 y 18) - 29h

- HU14: Eliminar archivos (14h)
- HU21: Obtener información usuario (15h)

Sprint 10 (semanas 19 y 20) - 28h

- HU20: Ordenar carpetas y archivos (9h)
- HU16: Vaciar papelera (12h)
- HU11: Renombrar archivo (7h)

Sprint 11 (semana 21 y 22) - 24.5h

- HU17: Restaurar archivos (11h)
- HU23: Crear carpeta espejo (13.5h)

Sprint 12 (Semana 23 y 24) - 20.5h

- HU24: Búsqueda unificada (12.5h)
- HU22: Compartir archivos (8h)

El proyecto tiene una duración total estimada de 24 semanas, lo que equivale a exactamente 6 meses. Esta planificación se ha dividido en 12 *sprints*, con una duración típica de dos semanas por *sprint*. Cada *sprint* ha sido cuidadosamente diseñado para incluir historias de usuario de alta prioridad, haciendo uso del *product backlog* y reduciéndose conforme se iban completando las historias de usuario más prioritarias, manteniendo una entrega continua de valor. El total de horas estimadas para completar todas las tareas, y en consecuencia todo el proyecto es de 352 horas.

Para visualizar el progreso del proyecto a lo largo de las 24 semanas voy a crear un gráfico *burn-down*. Este gráfico refleja nuestra metodología ágil, dividida en 12 *sprints* (eje x), permitiéndonos monitorear nuestro avance de manera continua y ajustar nuestras estrategias según sea necesario con respecto al número de horas que nos quedan restantes para finalizar el proyecto (eje y).

En este gráfico voy a crear una línea de horas restantes estimadas para completar el proyecto, basadas en nuestra planificación inicial de 352 horas. Esta línea nos proporcionará una visión clara de cómo debería de progresar el proyecto según lo estimado.

La otra línea en el gráfico será la de horas restantes ideales. Esta línea mostrará el ritmo teórico al que se debería de haber dividido cada *sprint*, pero debido al ajuste real de algunas tareas ha provocado que no se puedan completar de manera regular, variando ligeramente el ritmo de la estimación ideal.

Gráfico burn-down: horas estimadas - ideales



Ilustración 8. Gráfico burn-down de horas estimadas - horas ideales

Si comparamos ambas líneas del gráfico, se puede observar que la línea azul de horas restantes estimadas sigue de cerca a la línea roja de horas restantes ideales, lo que indica que la planificación del progreso del proyecto ha sido consistente con lo que habría sido una planificación ideal.

Sin embargo, a partir del sprint 2 hasta la finalización del proyecto, la línea azul de horas restantes estimadas se encuentra por debajo de la línea roja de horas restantes ideales. Esto indica que se deberá dedicar más tiempo y esfuerzo en esos *sprints* con respecto a lo que habría sido una planificación ideal.

2.5.1- Evolución de la planificación inicial

Durante el desarrollo del proyecto, se experimentaron variaciones significativas con respecto a la planificación inicial que provocaron algunos cambios en la ejecución del proyecto. Estos cambios incluyen desajustes por exceso o por defecto en las estimaciones iniciales de tiempo en la duración de las tareas, problemas imprevistos, y ajustes en las historias de usuario y requisitos.

Una de las mayores desviaciones se presentó en la historia de usuario relacionada con la navegación entre carpetas (HU04). Inicialmente, se estimó que esta historia de usuario requeriría 13 horas para completarla. Sin embargo, durante la implementación, se descubrió que Google Drive no almacenaba la ruta completa hasta una carpeta en base a su ID. Esto nos obligó a desarrollar una solución que consultara a cada carpeta "padre" hasta llegar al directorio raíz, incrementando

significativamente el esfuerzo requerido. Este imprevisto aumentó el tiempo de desarrollo del lado del *backend* de la HU04 a 22 horas, generando un desajuste de 9 horas adicionales.

Por otro lado, encontramos que la tarea de desconectar servicios de almacenamiento en la nube (HU02) fue más sencilla de lo previsto. Inicialmente se estimaron 12.5 horas para completar esta historia de usuario. Sin embargo, la implementación real se completó en 8.5 horas, ahorrando 4 horas respecto a la estimación inicial. Este ahorro de tiempo se debió a la simplicidad con la que se ha implementado del lado del servidor debido a que este elimina toda la información de la sesión de usuario, permitiendo la desconexión de los servicios en la nube y la eliminación de tokens de autenticación.

A la hora de implementar las opciones para gestionar archivos dentro de la papelera, como visualizar archivos, restaurar o eliminar definitivamente (HU05, HU17, HU14), encontramos restricciones específicas con Dropbox. Este servicio no permitía dichas opciones o requería una cuenta de pago para empresas. Como resultado, se redujo el tiempo estimado para estas historias de usuario de 8, 11, 14 horas a 7, 8, 11 horas ya que no se pudieron implementar las funcionalidades completas, ahorrando 7 horas entre las tres historias de usuario. En su lugar, desarrollamos una solución para informar al usuario, que dichas opciones no están disponibles directamente desde *MyDrive* y que deben realizarse manualmente en la nube correspondiente. Este cambio se dejó implementado para ser utilizado en alguna futura nube que no permitiera alguna de las funciones comentadas en las historias de usuario. Esto simplificó el desarrollo y redujo el esfuerzo necesario, pero también limitó las funcionalidades ofrecidas por la aplicación para los usuarios de Dropbox.

La funcionalidad de búsqueda de carpetas y archivos (HU19) también requirió más tiempo de lo estimado inicialmente. Originalmente, se planificaron 23.5 horas para esta tarea. Sin embargo, se tuvo que adaptar para permitir la búsqueda no solo en la vista principal, sino también en las vistas de archivos recientes y la papelera. Esta ampliación del alcance incrementó el tiempo necesario a 25.5 horas, aumentando 2 horas con respecto a la estimación inicial, ya que se necesitó desarrollar y validar la funcionalidad en diferentes contextos y asegurar que la experiencia del usuario fuera consistente en todas las vistas.

Otra de las desviaciones más significativas se debió a la exclusión de aquellas historias de usuario con prioridad muy baja que no se implementaron, lo que influyó en la planificación final. Debido a cuestiones de tiempo, ya que el proyecto tenía una duración de 352 horas distribuidas en 6 meses y 12 sprints, según la planificación inicial, había que considerar que parte de este tiempo también debía dedicarse a tareas adicionales no relacionadas con el desarrollo del código del proyecto, como la elaboración de informes y documentación. Se decidió excluir las historias de usuario compartir archivos (HU22), crear carpeta espejo (HU23) y búsqueda unificada

(HU24). Estas historias quedaron fuera del alcance del proyecto debido a su baja prioridad. Esto provocó un ahorro de 34 horas, entre las 3 historias de usuario.

Por tanto, aunque la duración estimada del proyecto fue de 24 semanas (6 meses) y 352 horas, debido a los ajustes y variaciones en la implementación de ciertas historias de usuario y el descarte de otras, el tiempo dedicado a cada tarea varió. Al realizar el balance general entre horas ahorradas y horas adicionales, resultó en una duración total del proyecto que se redujo significativamente con respecto a la planificación inicial, con un total de 318 horas. Esto provocó que ya no tuviera sentido tener 12 sprints de 2 semanas cada uno, sino que se redujo a 11 sprints, siendo el último de una semana, debido a su corta duración de 11 horas, es por ello que el proyecto se completó en 5 meses y 1 semana. Para ver la evolución real con respecto a la estimada, se ha diseñado otro gráfico burn-down comparativo.

Gráfico burn-down: Horas reales - estimadas

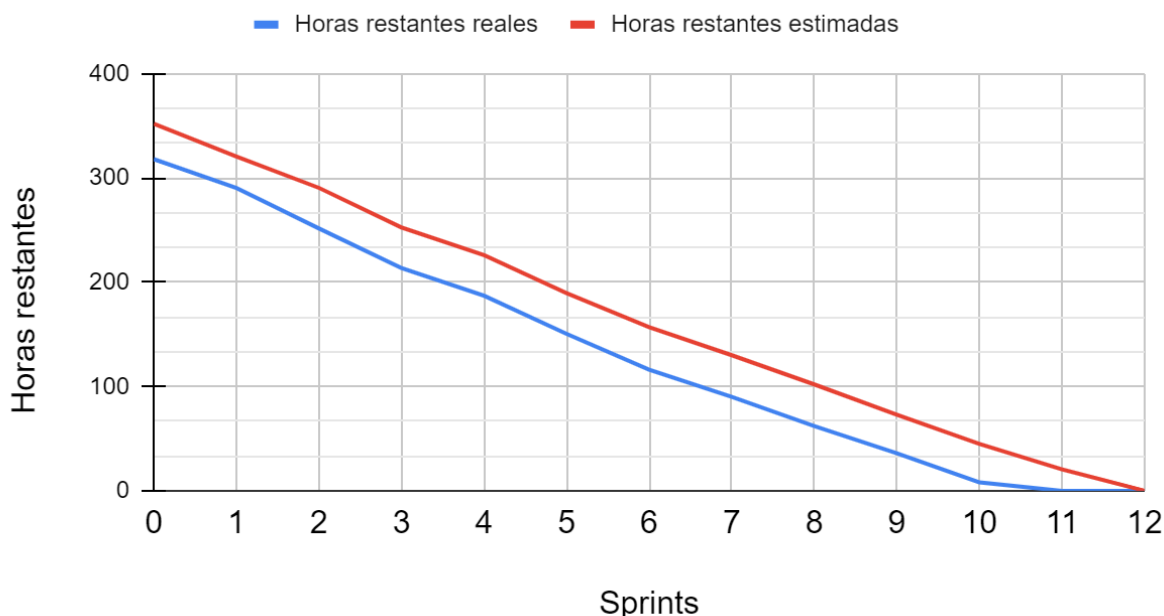


Ilustración 9. Gráfico burn-down de horas reales - horas estimadas

El gráfico burn-down comparando las horas reales con las horas estimadas revela información interesante sobre el desarrollo del proyecto. Aunque al final las horas reales utilizadas fueron menos que las horas estimadas, ambas líneas siguen trayectorias bastante paralelas. Esto indica que, a pesar de haber recortado la duración del proyecto, se ha mantenido un ritmo de trabajo muy cercano al planificado inicialmente. Las ligeras desviaciones observadas son naturales en cualquier proyecto ágil y reflejan la flexibilidad y la capacidad de ajuste frente a los imprevistos y cambios en el alcance.

2.6.- Estudio de viabilidad

Para llevar a cabo un análisis de viabilidad del proyecto considerando que el equipo está formado por un único miembro, establecemos que solo existe un analista/programador que se encargará de desarrollar el proyecto por completo. Se estimarán los costes asociados con el desarrollo del sistema de la siguiente manera:

1. Gastos materiales

- a. Equipos:** Para el desarrollo del proyecto, se requiere un equipo de desarrollo personal, que incluye una computadora de escritorio de medio/alto rendimiento con su respectivo monitor, teclado, ratón, etc. El coste total estimado de este equipo es de 1,200€. Considerando una vida útil del equipo de 5 años (60 meses), el coste mensual se prorratea de la siguiente manera:

$$\text{Coste Mensual} = \frac{1200 \text{ €}}{60 \text{ meses}} = 20 \text{ €}$$

Dado que la duración del proyecto es de 6 meses, el coste del equipo durante este periodo es de 120 €.

- b. Acceso a internet:** Para el desarrollo del proyecto, es crucial contar con acceso a internet de buena velocidad para conectar con los diferentes servicios en la nube, obteniendo todos los datos necesarios. El coste mensual estimado es de 30 €, debido a que dicho servicio se paga mensualmente y el desarrollo durará 6 meses, suma todo un total de 180 €.
- c. Licencias de software:** Aunque se utilizarán algunas herramientas de software libre, se incluirán costos de licencias adicionales, como una licencia anual de IntelliJ IDEA Ultimate, ya que será el IDE que usará para desarrollar *MyDrive*. El uso de software profesional como IntelliJ IDEA mejora la eficiencia y la calidad del desarrollo. El coste mensual es de 17€, sumando 6 meses al igual que en el caso anterior, tenemos un coste total de 102€.
- d. Gastos de luz:** Para tener el equipo encendido durante las 352 horas de desarrollo estimadas, se calculará el costo de la electricidad, sabiendo que el equipo en el que se va a trabajar consume en promedio 250 Wh y el router con el que nos permitirá proveerá acceso a internet es de según la ficha técnica es de 6.3 Wh. Esto hace un total de 256,3 Wh, que suponiendo un promedio del kWh en España

actualmente de 0,0901 €/kWh, podemos obtener que el precio total será aproximadamente:

$$\text{Precio total} = 352h \times 0,2563kWh \times 0,0901€/kWh = 8,13€$$

2. Nóminas

- a. Analista / Programador:** Basándonos en el convenio colectivo de Telefónica y otras fuentes como Indeed, el sueldo base por hora de un analista/programador se estima en 17,5€/h. Sabiendo que se trabajarán 352 horas durante los 6 meses de desarrollo, el costo será alrededor de 6160 €.

Una vez identificados todos los gastos necesarios para el desarrollo del proyecto, vamos a resumir la información en la siguiente tabla. Esta tabla nos permitirá visualizar de manera clara y concisa los diferentes tipos de gastos, sus detalles específicos y sus respectivos costes. A continuación, se presenta el resumen de los costes identificados anteriormente.

Tipos de gastos	Detalles	Coste
Gastos materiales	Equipos	120 €
	Acceso a internet	180 €
	Licencias de software	102 €
	Gastos de luz	8,13 €
Nóminas	Analista / programador	6160 €
Coste total:		6570,13 €

Tabla 50. Estimación de costes

En total, el coste de desarrollo estimado asciende a aproximadamente 6570,13 €, incluyendo equipos, acceso a internet, licencias de software, gastos de luz y nómina.

Es importante destacar que este proyecto se desarrollará como una aplicación open source, sin ánimo de lucro. El objetivo principal es contribuir a la comunidad de software libre proporcionando una herramienta útil y accesible para la gestión de almacenamiento en la nube. Los costos estimados representan la inversión necesaria para desarrollar y mantener el proyecto, pero no se busca obtener beneficios económicos directos. En lugar de ingresos, el éxito del proyecto se medirá en términos de adopción por parte de la comunidad, contribuciones de otros

desarrolladores y el impacto positivo en los usuarios finales. Debido a esto la viabilidad del proyecto se justifica por su potencial impacto y utilidad, más que por una rentabilidad económica directa.

2.7.- Modelo de dominio

El modelo de dominio es una representación conceptual del sistema que identifica las entidades principales y sus relaciones, proporcionando una visión estructurada de los datos y su interacción dentro del sistema. Este modelo es esencial para guiar la fase de diseño e implementación, asegurando que todos los elementos necesarios están presentes y correctamente interrelacionados.

En *MyDrive*, cada entidad y relación dentro del modelo de dominio responde a las entidades y asociaciones identificadas en la historias de usuario previamente redactadas. A continuación, se presenta una descripción de las entidades principales, sus relaciones y las justificaciones para su uso, asegurando una implementación coherente y alineada con los objetivos del proyecto. Más adelante en el apartado de diseño, se entrará más en detalle sobre qué atributos debe almacenar cada entidad además de algunas modificaciones adicionales.

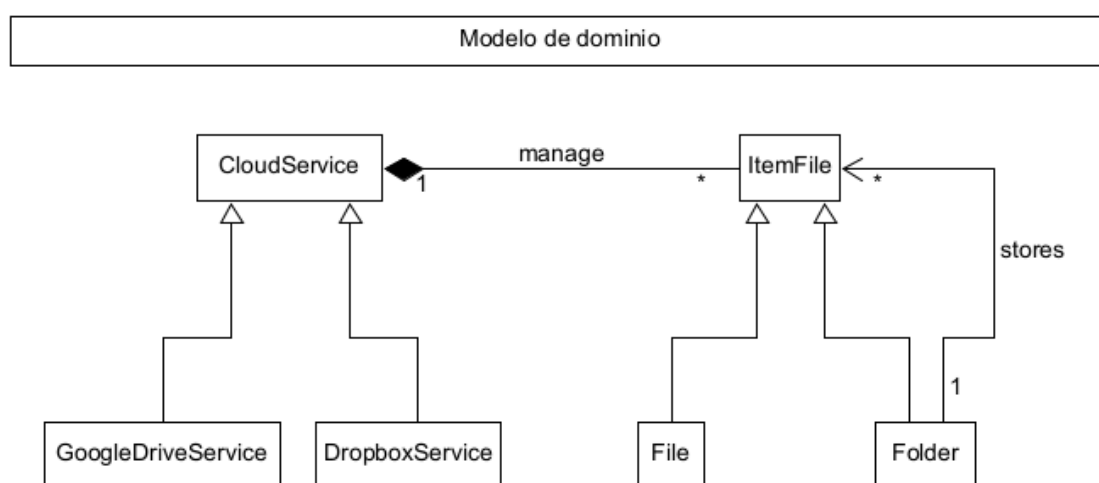


Ilustración 10. Modelo de dominio

En primer lugar, tenemos la entidad *CloudService*, que representa los servicios de almacenamiento en la nube que pueden ser conectados a *MyDrive*, como Google Drive o Dropbox. Esta relación de herencia proporciona una entidad general para cada uno de los servicios que se implementen, ofreciendo una solución común y transparente para el usuario. A su vez cada uno de estos servicios puede gestionar múltiples archivos, estableciendo una relación de uno a muchos con la entidad *ItemFile*. No tendría sentido un servicio en la nube que no gestionara archivos, es por esto que se ha creado una relación de tipo composición.

La entidad *ItemFile* representa un archivo o carpeta almacenado en un servicio de almacenamiento en la nube. Es importante tener en cuenta que estos archivos y carpetas están realmente almacenados en los propios servicios de la nube, los cuales contienen muchos datos acerca de estos.

En el caso de la subclase *File* que hereda de *ItemFile*, ésta representa específicamente un archivo almacenado en un servicio de almacenamiento en la nube. La subclase *File* contendrá atributos específicos a los archivos que son esenciales para la correcta gestión y visualización de estos, los cuales los trataremos en la fase de diseño en la representación de la información.

Por otro lado, la subclase *Folder* también hereda de *ItemFile* y representa una carpeta que puede contener otros archivos o subcarpetas, estableciendo una relación de uno a muchos con *ItemFile*. Esta diferenciación clara entre tipos de *ItemFile* permite una estructura jerárquica que facilita la navegación y gestión de archivos y carpetas.

De esta manera, reflejamos que un usuario, considerado fuera del modelo de dominio como un actor principal de la aplicación, podrá conectar diferentes *CloudService*, cada uno con sus implementaciones específicas, y a su vez cada uno de ellos, ofrecerá su sistema de archivos jerárquico comúnmente conocido. De esta manera se ofrece una estructura que soporta el desarrollo de las funcionalidades clave de la aplicación de manera efectiva y eficiente, alineándose con los objetivos del proyecto y las necesidades identificadas en las historias de usuario.

Hay que tener presente que la entidad *ItemFile* y sus subentidades, es decir, toda la parte relacionada con el almacenamiento de archivos y carpetas, son entidades que se deben manejar, pero no se almacenan en la parte del *backend* del servidor. En lugar de eso, estos archivos y carpetas son gestionados y almacenados por los propios servicios en la nube, lo que significa que *MyDrive* actúa como un intermediario que facilita la interacción del usuario con sus datos en diversas plataformas de almacenamiento en la nube sin duplicar el almacenamiento de dichos datos. A causa de esto, la implementación variará significativamente, ya que se requerirá una lógica específica para interactuar con las APIs de los diferentes servicios en la nube.

2.9.- Análisis de la interfaz

La interfaz de usuario de *MyDrive* debe ser intuitiva, accesible y *responsive*, permitiendo que la aplicación se vea y funcione adecuadamente en una variedad de dispositivos y tamaños de pantalla, para garantizar una experiencia de usuario positiva para todos ellos. En esta fase de análisis, es crucial identificar las

necesidades específicas de cada perfil de usuario y establecer los aspectos generales que se deben tener en cuenta al diseñar la interfaz. En el caso de *MyDrive* no podemos considerar que existen perfiles muy diferenciados como el de administrador, ni otros perfiles jerárquicos, ya que cada usuario es responsable de la gestión de sus propios archivos. Sin embargo, debemos tener presentes dos tipos de usuarios: usuarios generales y usuarios avanzados.

Para los usuarios generales, la interfaz debe proporcionar acceso fácil y rápido a sus archivos y servicios de almacenamiento en la nube. Necesitan una gestión centralizada de archivos, capacidad para subir, descargar, mover, renombrar y eliminar archivos, así como crear carpetas y organizarlas. La interfaz debe ser simple y directa, permitiendo realizar acciones comunes con pocos clics. La navegación debe ser intuitiva, con un diseño limpio y moderno, minimizando la curva de aprendizaje para el usuario general.

Por otro lado, los usuarios avanzados requieren funcionalidades más complejas, como el cifrado opcional para subir archivos y descifrado de estos archivos al descargarlos mediante una contraseña, además de mover archivos entre diferentes nubes conectadas simultáneamente. Para estos usuarios, la interfaz debe ofrecer opciones avanzadas sin complicar la experiencia para el usuario general, mediante menús desplegables o configuraciones establecidas por defecto que un usuario general en principio no modificará.

Para proporcionar una visión clara y concreta de cómo se organizarán los contenidos y funcionalidades de *MyDrive*, se ha creado un *sketch* hecho a mano de la interfaz de usuario. Este boceto ilustra la disposición intuitiva de los elementos, facilitando la navegación y el acceso a las distintas funciones de la aplicación.

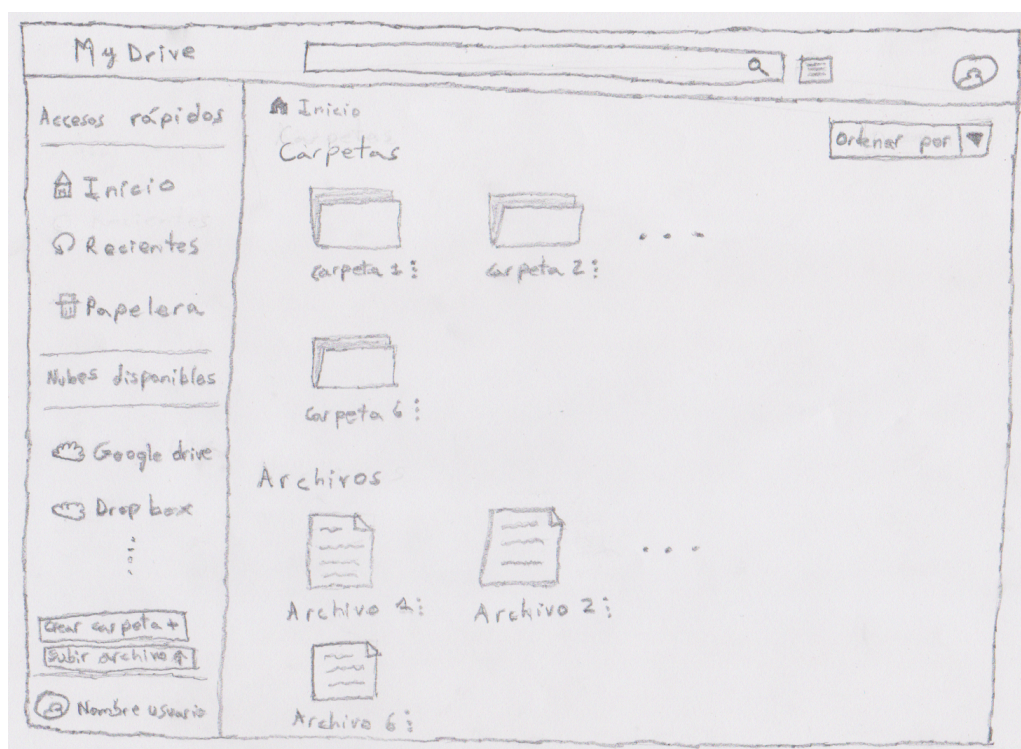


Ilustración 11. Sketch a mano de la organización de contenidos de MyDrive

El sketch de la interfaz de usuario de *MyDrive* muestra una organización clara y funcional de los elementos visuales. La interfaz se divide en tres secciones principales que facilitan la navegación y la gestión de archivos.

En primer lugar, la *sidebar* se encuentra a la izquierda de la pantalla. Esta barra lateral contiene accesos rápidos a las vistas principales de la aplicación, como "Inicio," "Recientes," y "Papelera." Además, se listan los servicios de almacenamiento en la nube disponibles, como Google Drive y Dropbox. Al final de la barra lateral, se incluyen botones para crear nuevas carpetas y subir archivos, junto con el nombre de usuario en la parte inferior. Esta organización permite un acceso rápido y sencillo a las funciones y servicios más utilizados.

En la parte superior de la interfaz se encuentra el *header*, que muestra el nombre de la aplicación *MyDrive*. A la derecha del nombre, se ubica una barra de búsqueda que facilita a los usuarios encontrar archivos y carpetas. Junto a la barra de búsqueda, hay un botón de configuración para acceder a opciones adicionales que se indican en las historias de usuario. A la derecha se encuentra la foto de perfil del usuario, para realizar acciones acerca de su sesión.

La sección principal de la interfaz, conocida como *main*, está ubicada en el centro y la derecha de la pantalla. Aquí es donde se muestran todas las carpetas y archivos obtenidos. Cada carpeta y archivo tiene sus propias opciones accesibles desde esta vista, permitiendo a los usuarios gestionar su contenido de manera eficiente y organizada. Esta estructura asegura que los usuarios puedan navegar y manejar sus

archivos de forma intuitiva. Además, en la parte superior aparece la ruta actual, además de otras opciones como ordenar archivos.

Cabe mencionar que esta es la estructura y organización que se mantendrá en cada una de las vistas, sin embargo el contenido del apartado del main irá cambiando según la vista en la que nos encontremos. Estos aspectos serán discutidos y detallados en profundidad durante la fase de diseño, asegurando que cada elemento de la interfaz esté optimizado para ofrecer la mejor experiencia posible al usuario.

Además, es fundamental que la interfaz de usuario de *MyDrive* informe de manera clara sobre los resultados de las acciones realizadas, mostrando mensajes de confirmación en caso de éxito y mensajes de error cuando algo no funcione como se esperaba o incluso redirigiendo a una página de error. Esto no solo mejora la transparencia y confianza en el sistema, sino que también ayuda a los usuarios a comprender qué ha sucedido y qué pasos seguir a continuación, asegurando una experiencia de usuario más fluida y satisfactoria.

3.- Diseño

3.1.- Representación de la información

En este apartado, partiremos del modelo de dominio previamente definido, pero ahora nos enfocaremos en detallar todos aquellos atributos y entidades más específicas que son cruciales para la implementación. Este nivel de detalle es esencial para garantizar que el diseño técnico esté perfectamente alineado con los requisitos del proyecto, facilitando una implementación eficiente y efectiva.

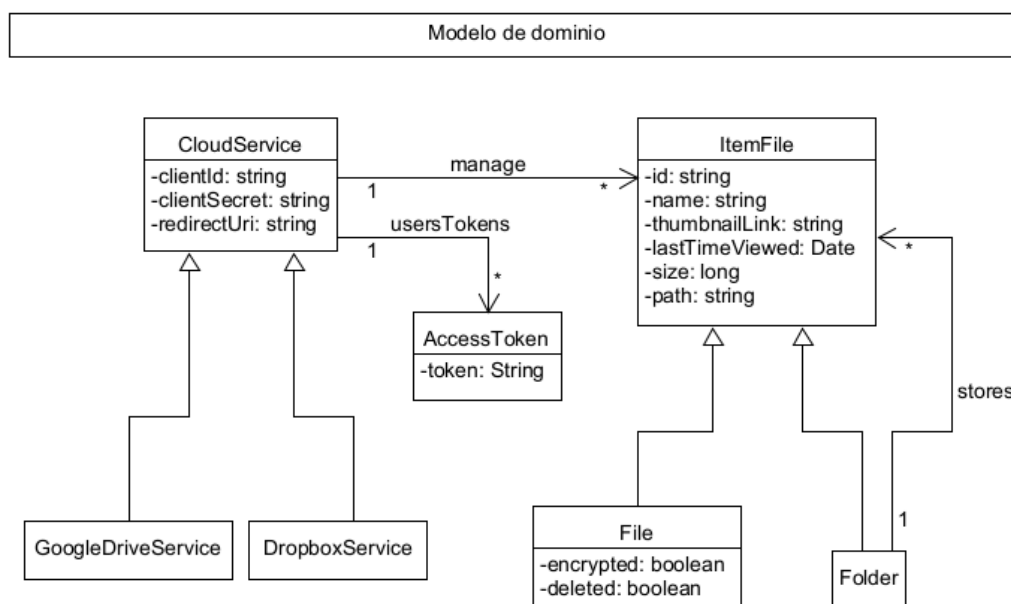


Ilustración 12. Representación de la información en base al modelo de dominio

La entidad *CloudService* tiene atributos como *clientId*, *clientSecret* y *redirectUri*. Estos atributos son necesarios para implementar protocolos como OAuth2, permitiendo al usuario autorizar a *MyDrive* a acceder a sus cuentas en cada uno de los servicios de almacenamiento.

La entidad *AccessToken*, con su atributo *token*, se trata de una nueva entidad añadida que permite gestionar los *tokens* de acceso de los usuarios. Estos *tokens* son esenciales para mantener las sesiones autenticadas con los servicios de almacenamiento en la nube sin necesidad de que el usuario introduzca sus credenciales repetidamente. La relación entre *CloudService* y *AccessToken* es de uno a muchos, reflejando que un servicio de almacenamiento en la nube puede tener múltiples *tokens* de acceso asociados a diferentes usuarios, permitiendo manejar múltiples sesiones de distintos usuarios.

La entidad *ItemFile*, tiene atributos como *id*, *name*, *thumbnailLink*, *lastTimeViewed*, *size* y *path*. Atributos como el *id* o el *path* son indispensables para poder hacer referencia a ellos de manera inequívoca, además del resto de atributos para dotar a

la parte visual de un mejor aspecto. Hay que tener en cuenta, que estos son los atributos que se consideran importantes para desarrollar la aplicación, pero como hemos dicho, estos archivos están almacenados realmente en los propios servicios en la nube y estos almacenan muchos más datos de los que se muestran en el modelo de dominio.

En el caso de la subclase *File* que hereda de *ItemFile*, podemos distinguir dos atributos adicionales que nos permiten indicar si el archivo que subimos está cifrado, para luego al descargarlo aplicar el descifrado o no, además de otro atributo para saber si está en la papelera o no. Este atributo no es necesario para la clase *Folder*, ya que en *MyDrive* no se contempla la posibilidad de enviar carpetas a la papelera, sino solo los archivos.

Por otro lado, para la subclase *Folder* no se considera necesario ningún atributo adicional aparte de los de la entidad *ItemFile*. Esta simplificación se debe a que las carpetas, a diferencia de los archivos, no requieren características específicas como cifrado o estado de papelera.

Es importante destacar que las entidades *ItemFile* y sus subentidades no guardan los archivos y carpetas en el *backend* del servidor de *MyDrive*. En lugar de eso, estos datos son administrados y almacenados por los propios servicios en la nube. *MyDrive* funciona como un intermediario, permitiendo a los usuarios interactuar con sus datos en diferentes plataformas de almacenamiento sin duplicar el almacenamiento de dichos datos. Esta arquitectura implica que la implementación necesitará una lógica especializada para interactuar con las APIs de los distintos servicios en la nube que se verá reflejado en el diagrama de clases de la fase de diseño.

3.2.- Diagrama de clases

El diagrama de clases es una representación fundamental en la fase de diseño, ya que proporciona una vista estructural de los elementos principales del sistema y sus relaciones. El diseño se centra en la creación de un sistema de almacenamiento en la nube que soporte múltiples proveedores, como Google Drive y Dropbox, asegurando una arquitectura modular y extensible.

En comparación con el modelo de dominio, el diagrama de clases incluye detalles adicionales sobre la implementación y las interacciones entre los componentes del sistema. Mientras que el modelo de dominio se enfoca en las entidades principales y sus relaciones de alto nivel, el diagrama de clases profundiza en los métodos y atributos específicos de cada clase, así como en las interfaces y excepciones que forman parte de la lógica de negocio.

Además, partiremos del modelo de dominio de la ilustración 12, pero aquí resolveremos toda la parte de aquellas entidades que están en los servicios en la nube y que no eran persistentes en nuestro sistema. Estas se modificarán por una comunicación que permita realizar todos los métodos necesarios y demás interacciones requeridas. En lugar de almacenar archivos localmente, se implementarán clases con sus métodos para manejar archivos directamente en los servicios de almacenamiento en la nube.

El diseño de clases se presentará por separado para el *backend* y el *frontend*, permitiendo una comprensión más clara y detallada de cada componente de la aplicación. Abordar estos diseños por separado es fundamental, ya que el *backend* y el *frontend* tienen responsabilidades y arquitecturas muy separadas, cada uno con sus propias particularidades y desafíos técnicos.

Es importante mencionar que los diagramas de clases que se presentan tanto para el *backend* como para el *frontend* son el resultado final de una serie de modificaciones aplicadas a lo largo del desarrollo del proyecto. Estas modificaciones surgieron de la necesidad de que el diseño final sea adaptable a futuras ampliaciones, mejoras en la estructura del proyecto y en la forma de gestionar ciertas operaciones.

3.2.1.- Diagrama de clases del backend

En el *backend*, se detalla la estructura y las interacciones entre las clases que manejan la lógica de negocio y la comunicación con los servicios en la nube. La misión principal del *backend* es gestionar la lógica de la aplicación y ofrecer a los posibles clientes una visión unificada de los diferentes servicios de almacenamiento en la nube. Para ello el *backend* ofrece una serie de operaciones comunes para el cliente mediante un *API REST*. Esta *API REST* permite a los usuarios interactuar con la aplicación de manera consistente y transparente, independientemente del proveedor de almacenamiento en la nube que utilicen.

Es importante tener presente que el diagrama de clases del *backend* mostrado se implementará con Spring Boot. Como resultado, muchas de estas clases formarán parte del contenedor de inversión de control (IoC) de Spring. Esto significa que Spring gestiona la creación, configuración y ciclo de vida de los objetos (*beans*) utilizados en la aplicación. En el diagrama presentado, no se mostrarán los detalles internos de la gestión de Spring para no complicarlo aún más, pero se indicará si se trata de un controlador, servicio, excepción o un POJO (*Plain Old Java Object*). Esta distinción es crucial para comprender cómo se estructuran y se comunican las diferentes partes del sistema dentro del ecosistema de nuestra aplicación, facilitando así una mayor claridad en la implementación y en el mantenimiento del código.

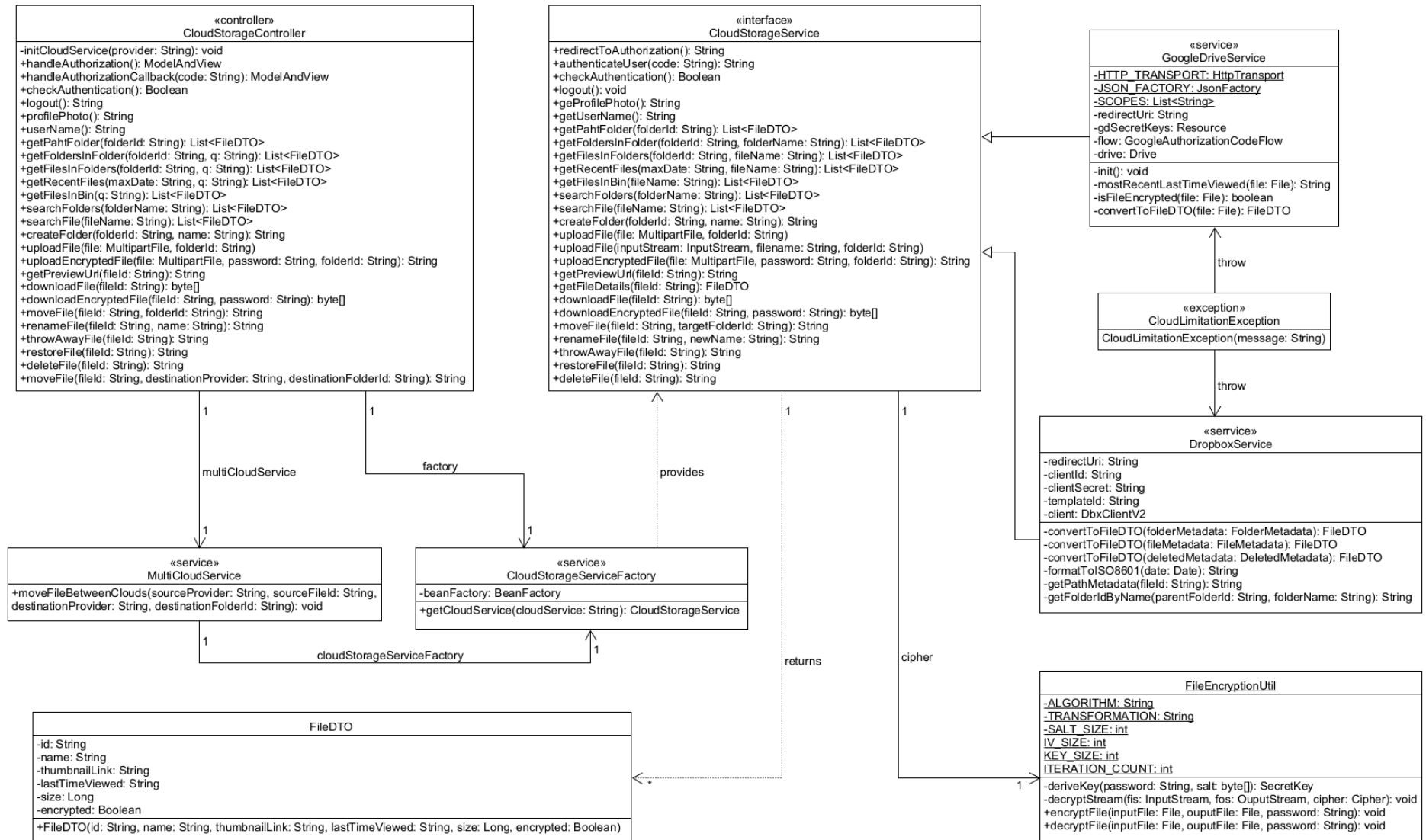


Ilustración 13. Diagrama de clases de diseño del backend

En primer lugar, tenemos la clase *CloudStorageController*, que maneja las solicitudes del usuario y actúa como un intermediario entre la interfaz de usuario y la lógica de negocio. Esta clase es esencial en el patrón MVC [1] (Model-View-Controller), asegurando que las interacciones del usuario se procesen y se traduzcan en acciones apropiadas sobre los datos. Incluye métodos para manejar la autenticación, subir y descargar archivos, crear y navegar entre carpetas, entre otras funcionalidades clave. Estas funcionalidades están distribuidas para mantener la separación de preocupaciones y facilitar la mantenibilidad del código.

La interfaz *CloudStorageService* define los métodos principales que deben ser implementados por cualquier servicio de almacenamiento en la nube. Estos métodos abarcan desde la autenticación y obtención de archivos y carpetas hasta la gestión de archivos, incluyendo su carga, descarga, cifrado y descifrado. Esta interfaz asegura que cualquier servicio de almacenamiento en la nube pueda ser integrado de manera uniforme y sencilla, siguiendo el patrón de diseño Estrategia [1]. Este patrón permite definir un conjunto de métodos, definiendo cada servicio y encapsular cada uno de ellos y hacerlos intercambiables, lo cual es útil en este caso para cambiar entre distintos proveedores de servicios en la nube sin alterar el código del cliente.

Las clases *GoogleDriveService* y *DropboxService* implementan la interfaz *CloudStorageService*, proporcionando la funcionalidad específica para Google Drive y Dropbox, respectivamente. Utilizan las APIs de sus correspondientes proveedores para realizar operaciones como autenticación, obtención y manipulación de archivos y carpetas. Al implementar la misma interfaz, garantizan que el controlador pueda interactuar con ellos de manera uniforme. Esto permite que *CloudStorageController* trate a ambos servicios de la misma manera, delegando las diferencias específicas del proveedor a las implementaciones concretas.

Para facilitar la creación de instancias específicas de servicios de almacenamiento, se utiliza la clase *CloudStorageServiceFactory*. Esta clase actúa como una fábrica que instancia objetos que implementan la interfaz *CloudStorageService*, permitiendo la creación de servicios de almacenamiento en la nube basados en el proveedor especificado. Esta clase es la encargada de indicar al controlador el servicio que se debe usar específicamente. Este patrón de diseño, conocido como *Factory* [1], facilita la extensibilidad del sistema, permitiendo la incorporación de nuevos proveedores de almacenamiento con facilidad. Al centralizar la lógica de creación, se garantiza que la instanciación de servicios específicos se maneje de manera controlada en tiempo de ejecución.

La clase *MultiCloudService* proporciona funcionalidad adicional para mover archivos entre diferentes servicios de almacenamiento en la nube. Esta clase depende de *CloudStorageServiceFactory* para obtener las instancias del servicio de almacenamiento origen y destino. Facilita la transferencia de archivos entre

diferentes proveedores, aumentando así la flexibilidad del sistema. Este enfoque asegura que los usuarios puedan operar en un entorno *multicloud* sin necesidad de interactuar directamente con los detalles de implementación de cada servicio.

En cuanto a la seguridad, la clase *FileEncryptionUtil* contiene métodos para el cifrado y descifrado de archivos, asegurando que los datos almacenados y transferidos estén protegidos. Esto es crucial para mantener la confidencialidad e integridad de los datos del usuario. La seguridad se implementa a nivel de archivo, proporcionando una capa adicional de protección además de la ofrecida por los propios servicios de almacenamiento en la nube.

El objeto de transferencia de datos *FileDTO* encapsula la información de los archivos, como el ID, nombre, enlace a la miniatura, tiempo de última visualización, tamaño y si está cifrado. Este objeto es utilizado por varios métodos en *CloudStorageService* y sus implementaciones para transferir datos de archivos entre las capas de la aplicación de manera consistente, llegando incluso este objeto hasta el lado del cliente. Al utilizar DTOs (*Data Transfer Objects*), se facilita el intercambio de información estructurada entre los diferentes componentes del sistema, reduciendo la cantidad de llamadas remotas y se disminuye el volumen de datos transferidos a solo aquellos esenciales para el desarrollo de la aplicación. Esto optimiza el rendimiento y la eficiencia del sistema, enfocándose únicamente en los datos relevantes y minimizando el tráfico innecesario.

Por último, se define la excepción personalizada *CloudLimitationException*, que se lanza cuando se encuentran limitaciones específicas del proveedor de almacenamiento en la nube, como restricciones de API o características no soportadas. Esta excepción permite manejar de manera adecuada las limitaciones inherentes de cada proveedor y proporcionar mensajes de error claros al usuario, mejorando la robustez y la experiencia del usuario.

3.2.2.- Diagrama de clases del frontend

En el *frontend*, se presenta el diagrama de clases del *frontend*, que aunque finalmente se implemente con componentes de Vue.js, se pueden distinguir características que siguen principios de diseño orientado a objetos. Cada componente en Vue.js representa una clase con sus propios métodos, atributos y eventos.

Estos componentes, se integran para formar la interfaz de usuario y cómo se comunican con el *backend* a través de peticiones HTTP a la API REST del *backend*. El uso de Vue.js facilita la creación de interfaces de usuario reactivas y dinámicas, permitiendo que los componentes actualicen la vista automáticamente cuando cambian los datos. Además, *Vuex* se utiliza para la gestión del estado de la

aplicación, asegurando que todos los componentes compartan un estado consistente y que cualquier cambio en los datos sea inmediatamente reflejado en la interfaz de usuario.

Gran parte de la estructura del diagrama de clases en el *frontend* viene determinada por las características del *framework* Vue.js y de las herramientas adicionales usadas como *Vuex* y *Vue Router*. De esta manera, se puede distinguir tanto la parte de aquellas vistas y componentes que se han tenido que desarrollar y la parte transparente que hace Vue.js para que todo funcione correctamente.

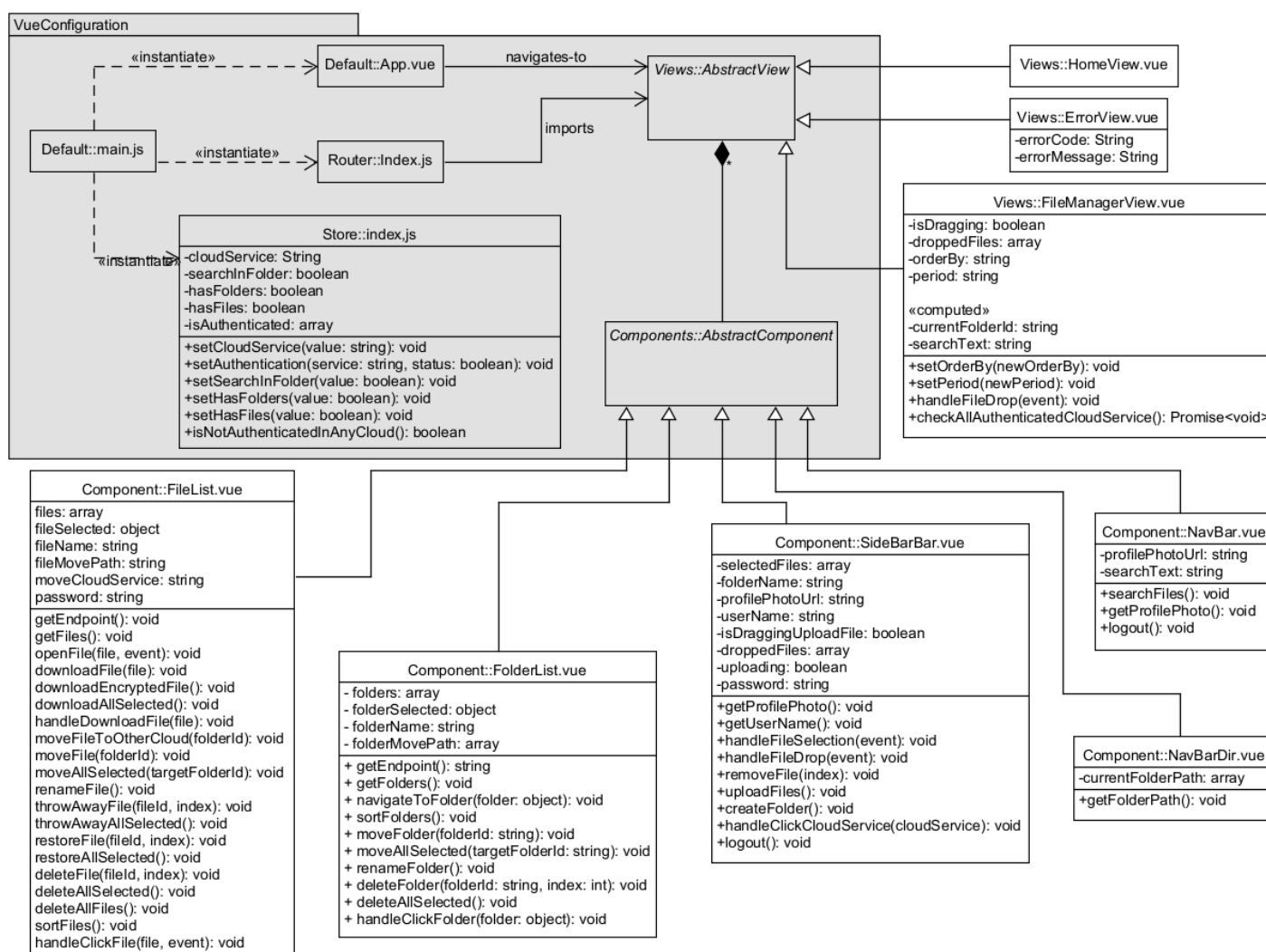


Ilustración 14. Diagrama de clases de diseño del frontend

Como podemos ver en el diagrama del *frontend*, tenemos un paquete *VueConfiguration* que se ha creado para dejar claro que esa parte es la encargada de gestionar la configuración global de la aplicación Vue.js. Este paquete incluye los archivos de entrada y configuración esenciales que permiten inicializar y configurar la aplicación, incluyendo el enrutamiento y el estado global. En primer lugar, se hablará de las clases del lado de *VueConfiguration*, dado que son aquellas clases

que manejan todo el estado de la aplicación y luego se hablará de las clases que manejan todas las vistas y componentes, que son aquellas que se encargan de mostrar los datos y funcionalidades correctamente.

La clase *Default::main.js* representa el archivo principal de entrada de la aplicación Vue.js. Su responsabilidad principal es iniciar la aplicación, configurar el *router* y el *store* de Vuex, y montar la aplicación en el DOM. Esta clase instancia *Default::App.vue*, *Router::index.js* y *Store::index.js*, asegurando que la aplicación esté correctamente configurada y lista para su ejecución. La justificación de esta clase radica en su papel fundamental en la inicialización y configuración de la aplicación, lo cual es esencial para el funcionamiento de cualquier aplicación Vue.js. La clase *Default::App.vue* es el componente principal de la aplicación Vue.js. Actúa como el contenedor principal que gestiona la estructura y navegación general de la interfaz de usuario. Este componente navega a *Views::AbstractView*, lo que significa que sirve como el punto de partida para las vistas de la aplicación. Esta clase, centraliza la gestión de la interfaz de usuario, proporcionando una base sólida sobre la cual se construyen y navegan las diferentes vistas de la aplicación.

La clase *Router::index.js* es responsable de la configuración del enrutador de Vue.js. Esta clase importa *Views::AbstractView* para definir las rutas y la navegación entre las diferentes vistas de la aplicación. La gestión adecuada del enrutamiento es crucial para cualquier aplicación de una sola página (SPA), ya que permite la navegación dinámica sin recargar la página completa. Por lo tanto, esta clase es esencial para manejar la estructura de navegación de la aplicación y mejorar la eficiencia de la misma.

La clase *Store::index.js* se encarga de la gestión del estado global de la aplicación utilizando Vuex. Esta clase define varios atributos como *cloudService*, *searchInFolder*, *hasFolders*, *hasFiles* e *isAuthenticated*, que representan el estado compartido entre diferentes componentes de la aplicación. Dentro de estos atributos los más relevantes son *cloudService*, que indica la nube que está usando el usuario y *isAuthenticated*, que se trata de un *array* con todas las nubes disponibles, indicando en cuáles ha iniciado sesión el usuario. El resto de parámetros son simplemente *flags* para el resto de clases para saber si realizar una operación o no. Además, incluye métodos para modificar los valores de cada uno de estos atributos a lo largo de la ejecución del programa. Esta clase centraliza la gestión del estado, permitiendo que los componentes compartan y actualicen información de manera coherente, lo cual es fundamental para la ejecución de toda la aplicación.

La clase *Views::AbstractView* actúa como la clase base para todas las vistas de la aplicación. Esta clase define una estructura y funcionalidades comunes que son heredadas por las vistas derivadas como *Views::HomeView.vue*, *Views::ErrorView.vue* y *Views::FileManagerView.vue*. Esta clase es parte de la gestión interna de Vue y se encarga de proporcionar una base común para todas las

vistas, siendo todo este proceso transparente para el desarrollador. De manera similar, la clase *Components::AbstractComponent* sirve como la clase base para todos los componentes, proporcionando funcionalidades y atributos comunes que son heredados por componentes específicos como *Component::FileList.vue*, *Component::FolderList.vue*, *Component::SideBar.vue*, *Component::NavBar.vue* y *Component::NavBarDir.vue*. Esta clase también es parte de la gestión interna de Vue y se encarga de proporcionar una base común para todos los componentes, haciendo que este proceso sea igualmente transparente para el desarrollador.

Dentro de las vistas que están a parte de la propia gestión de Vue, tenemos la clase *Views::HomeView.vue*, que representa la vista de la página de inicio de la aplicación. Aunque no tiene atributos específicos, su responsabilidad es mostrar el contenido inicial al usuario. Esta vista es crucial ya que ofrece la primera impresión de la aplicación y proporciona acceso a la vista principal de la aplicación.

La clase *Views::ErrorView.vue* maneja la visualización de mensajes de error personalizados. Incluye atributos como *errorCode* y *errorMessage* para proporcionar información detallada sobre el error ocurrido. Esta clase es importante para mejorar la experiencia del usuario, brindando mensajes de error claros y comprensibles cuando algo sale mal.

La clase *Views::FileManagerView.vue* representa la vista principal de la aplicación donde se gestiona la visualización y manipulación de archivos en la aplicación. Incluye atributos como *isDragging*, *droppedFiles*, *orderBy* y *period* para manejar el estado y las interacciones del usuario con los archivos. Los atributos *isDragging* y *droppedFile*, permiten conocer si el usuario ha arrastrado y soltado archivos que desee subir. En el caso *orderBy* indica el orden de los archivos en el que se deben mostrar. Por último, el atributo *period*, permite indicar el periodo de tiempo máximo de los archivos recientemente utilizados. Además, dicha clase posee todos los métodos necesarios para modificar estos atributos y tomar decisiones en consecuencia tanto en la parte visual como en la lógica.

Hay que tener en cuenta que aunque las vistas sean gestionadas de manera diferente a los componentes, estos deben ser incluidos en las vistas en las que se deben mostrar. En este caso aunque no esté reflejado en el diagrama todos los componentes se tratan de partes visuales de la propia vista *FileManager.vue*, en consecuencia todos estos componentes son hermanos e hijos de la misma vista padre, lo cuál es crucial tener en cuenta a la hora de la comunicación entre ellos. Por esta razón, los atributos *currentFolderId* y *searchText* son *computed* en el diagrama y se les pasan a los componentes hijos que los necesiten. Estos atributos computados permiten que los componentes hijos reciban información actualizada y reactiva sobre el estado del gestor de archivos, facilitando una comunicación eficiente y coherente entre todos los componentes involucrados.

Dentro de los componentes que nos podemos encontrar, está la clase *Component::NavBar.vue*, que proporciona una barra de navegación superior con funciones de búsqueda y gestión de perfil. Incluye atributos como *profilePhotoUrl* y *searchText*, en este último caso se lo comunica a la vista padre. Los métodos asociados, como *searchFiles*, *getProfilePhoto* y *logout*, permiten al usuario buscar carpetas y archivos, ver su foto de perfil dentro de cada servicio en la nube y cerrar sesión en dicho servicio. Esta clase es esencial para la navegación y la experiencia del usuario en la aplicación.

La clase *Component::NavBarDir.vue* muestra la ruta actual de la carpeta, permitiendo navegar por ella a un directorio de los que la conforman. Incluye el atributo *currentFolderPath* y el método *getFolderPath* para obtener la ruta de la carpeta actual. Esta clase mejora la usabilidad de la aplicación al proporcionar una indicación clara de la ubicación actual del usuario dentro de la estructura de carpetas.

La clase *Component::SideBarBar.vue* proporciona una barra lateral para la gestión de archivos y carpetas. Algunos de los atributos más relevantes de este componente son *selectedFiles*, *folderName* y *password*. En el caso de *selectedFiles* indica los archivos seleccionados a los que se le va a aplicar una operación. *folderName* indica el nombre que se le va a aplicar cuando se va a crear una nueva carpeta. *password* indica la contraseña con la que se debe cifrar un archivo en caso de ser cifrado. Los métodos asociados, permiten seleccionar un servicio en la nube concreto y si la sesión no está iniciada, permitir el inicio de sesión, subir archivos, crear una nueva carpeta y obtener algunos datos sobre el usuario. Esta clase es importante para proporcionar acceso rápido a las funcionalidades más usuales de gestión de archivos y carpetas.

La clase *Component::FolderList.vue* gestiona la lista de carpetas y sus interacciones en la aplicación. Incluye atributos como *folders*, que representa la lista de carpetas obtenida, *folderSelected*, que indica la carpeta a la que se le va a aplicar alguna operación, *folderName*, que representa el nombre nuevo que se le va a aplicar y *folderMovePath*, que representa la ruta destino a la que se debe mover la carpeta. Los métodos asociados, permiten al usuario navegar entre carpetas y realizar diversas operaciones definidas en la fase de análisis. Esta clase es crucial para la organización y gestión de carpetas en la aplicación.

Por último, la clase *Component::FileList.vue* gestiona la lista de archivos y sus interacciones en la aplicación. Incluye atributos muy similares al componente *Component::FolderList.vue*, aunque añade un atributo *password* que indica la contraseña que se debe introducir para descargar un archivo cifrado. Los métodos asociados, permiten al usuario realizar diversas operaciones con los archivos dentro de un mismo servicio en la nube o entre varios. Esta clase es esencial para la gestión de archivos, uno de los componentes principales de la aplicación.

3.3.- Diagramas de secuencia

En este apartado, se presentan los diagramas de secuencia que hacen referencia a los aspectos más relevantes del proyecto. Los diagramas de secuencia son herramientas visuales que muestran cómo los objetos interactúan en un proceso específico, detallando el orden de los mensajes intercambiados entre ellos. El objetivo principal de estos diagramas es explicar al lector de la memoria o a la persona encargada de ampliar o mantener el proyecto en el futuro, cuál es el funcionamiento interno del diseño propuesto en el diagrama de clases.

Para lograr esto, solo se incluyen los diagramas de secuencia que corresponden a funcionalidades clave del proyecto. No es necesario mostrar todos los diagramas posibles, ya que si se escogen adecuadamente las funcionalidades y las historias de usuario representativas, el resto de las funcionalidades suelen ser similares en cuanto a su funcionamiento. Esta selección justifica por qué no se han desarrollado todos los diagramas posibles, permitiendo al lector comprender los aspectos más críticos del sistema sin sobrecargar la documentación.

Dado que nuestra aplicación dispone de una API REST, que se mostrará en el apartado de diseño de la interfaz, es conveniente desarrollar los diagramas de secuencia de las historias de usuario o casos de uso de manera que aparezcan tanto las clases de la parte cliente como las del servidor. Esto ayuda a entender cómo interactúan ambas partes, ofreciendo una visión completa del flujo de datos y procesos. Además, se han utilizado colores específicos para distinguir fácilmente en los diagramas las clases que pertenecen a cada parte, mejorando la claridad y la comprensión visual: el gris para los actores implicados, el azul para la parte del *frontend* y el naranja para la parte del *backend*.

Las funcionalidades claves que se desarrollarán en los diagramas de secuencia incluyen:

- **Conectar servicio de almacenamiento en la nube:** Representa el proceso en el que un usuario se autentica en un servicio de almacenamiento en la nube de los ofrecidos por la aplicación.
- **Obtener todo el contenido de una carpeta:** Muestra el proceso de obtención de todas las carpetas y archivos de una carpeta concreta, demostrando la obtención múltiple de diferentes archivos, que es similar en otras funcionalidades.
- **Subir archivos cifrados:** Detalla cómo se suben archivos cifrados al servicio de almacenamiento en la nube, añadiendo una capa extra de protección de datos.

- **Descargar archivo cifrado:** Muestra el proceso de descarga de un archivo cifrado desde el servicio de almacenamiento en la nube y su posterior descifrado.
- **Mover archivos entre nubes:** Explica cómo múltiples archivos se mueven de un servicio de almacenamiento en la nube a otro, demostrando esa unificación clave que ofrece la aplicación.

Estos diagramas no solo explican los flujos normales de operación, sino que también incluyen secuencias de mensajes alternativas que representan casos de error en las operaciones. Esto es fundamental para proporcionar una visión completa de cómo maneja el sistema tanto las situaciones esperadas como las excepciones, asegurando una robustez en el diseño y la implementación.

3.3.1.- Conectar servicios de almacenamiento en la nube

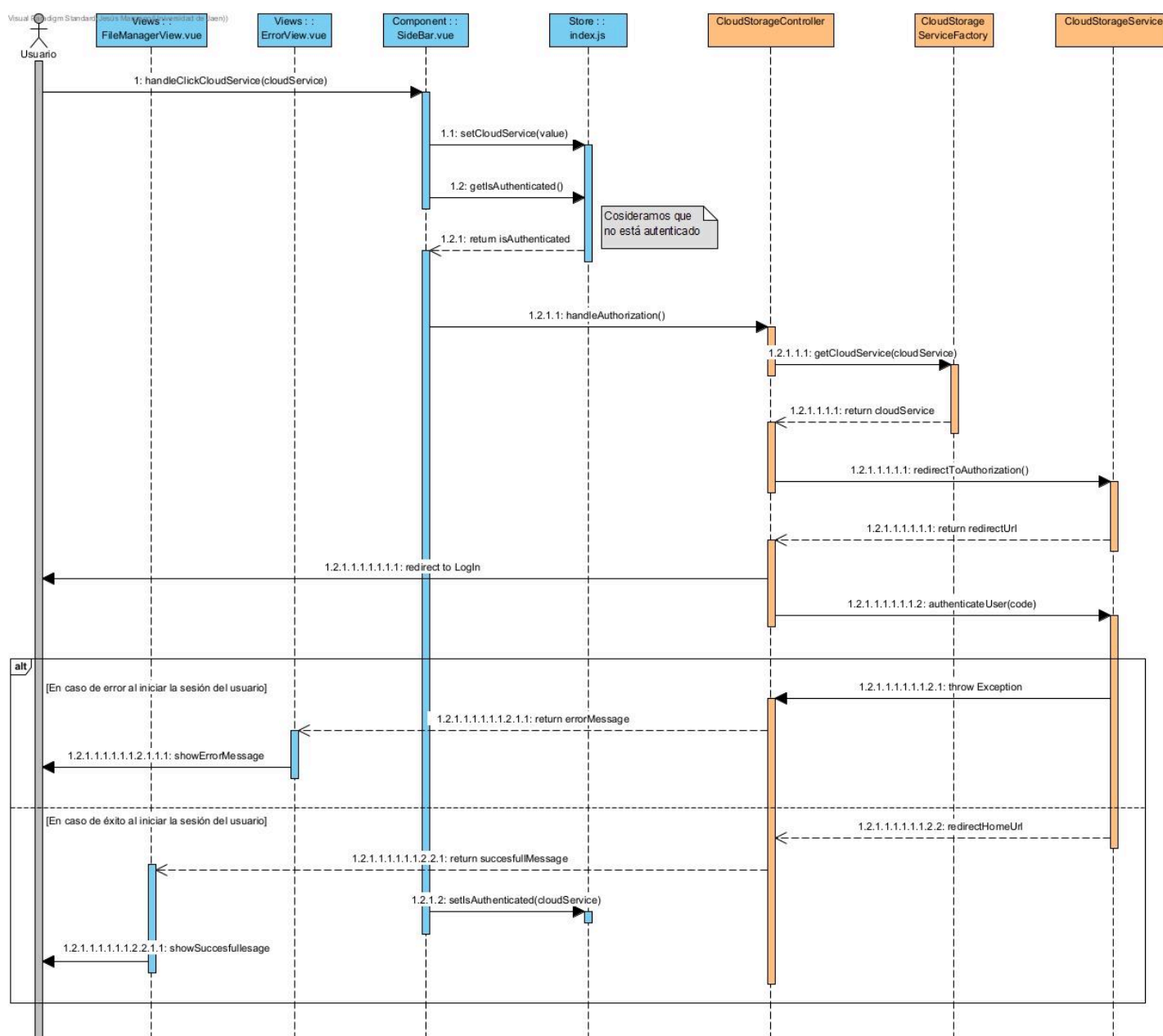


Ilustración 15. Diagrama de secuencia: Conectar servicio de almacenamiento en la nube

El flujo de eventos comienza cuando el usuario inicia el proceso de autenticación seleccionando un servicio de nube de la barra lateral. Esta acción desencadena el método *handleClickCloudService(cloudService)*, que llama a *setCloudService(value)* en el estado global (*Store::index.js*) para establecer el servicio de nube seleccionado. Posteriormente, se verifica el estado de autenticación del servicio seleccionado llamando a *getIsAuthenticated()*.

Si el servicio de nube no está autenticado, que es lo que se considera en este caso para demostrar todo el proceso por completo, el *frontend* llama al método *handleAuthorization()* del controlador del *backend*, conectando ambas partes. Este

método solicita al *CloudStorageController* la autorización necesaria para el servicio en la nube correspondiente. Aunque al llamar al *backend* no aparezca ningún atributo sobre el servicio en concreto al que quiere conectarse, dependiendo de la ruta a la que haga la petición se podrá obtener dicho servicio. Para ello el *CloudStorageController* utiliza la *CloudStorageServiceFactory* para obtener el servicio de nube correspondiente, y este proporciona la URL de redirección necesaria para iniciar el proceso del protocolo OAuth mediante el método *redirectToAuthorization()*.

El usuario es redirigido a la URL de autorización para autenticarse, y tras la autenticación, el servicio de nube retorna un código de autorización que será usado a través del método *authenticateUser(code)* para iniciar la sesión del usuario en dicho servicio. Dependiendo del resultado de la autenticación, se maneja el flujo de la siguiente manera: en caso de éxito, se retorna la url final y un mensaje de éxito (*return successfullMessage*), y se llama a *setIsAuthenticated(cloudService)* en el estado global para marcar el servicio como autenticado. En caso de error, se lanza una excepción en el *backend* y se retorna un mensaje de error al frontend (*return errorMessage*), y el usuario es redirigido a la vista *ErrorView.vue* gracias al *Router*, mostrando el mensaje de error al usuario.

3.3.2.- Obtener todo el contenido de una carpeta

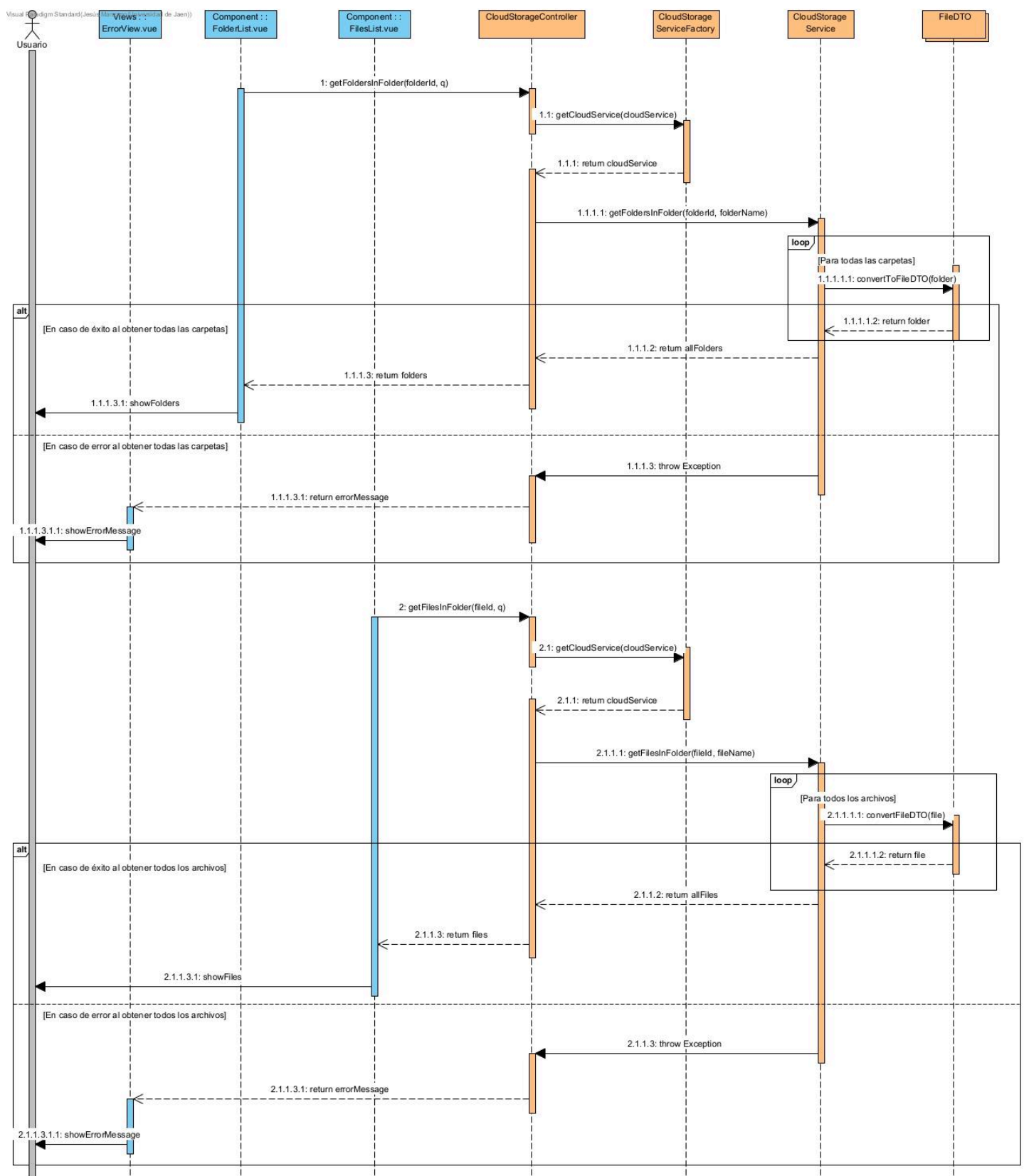


Ilustración 16. Diagrama de secuencia: Obtener todo el contenido de una carpeta

El flujo de eventos comienza cuando se carga el componente *FoldersList.vue* o *FilesList.vue*. Por defecto la carpeta inicial es *root*, solicitando obtener las carpetas y archivos contenidas en el directorio raíz. Aunque no se ha puesto al usuario como actor principal, realmente cuando se mueva a otra carpeta, al cambiar el *currentFolderId*, volverán a cargarse los componentes de la vista *FileManager.vue* que usan dicho parámetro, teniendo que volver a obtener dichas carpetas y archivos, osea que de alguna manera el usuario también desencadena dicha acción aunque no la inicie él explícitamente.

Cada componente hace su correspondiente petición a los endpoints del controlador. Para el caso de *FolderList.vue* llama a *getFoldersInFolder(folderId, folderName)* y *FilesList.vue* llama a *getFilesInFolder(folderId, fileName)*. El atributo *folderId* es el identificador de la carpeta que queremos obtener todo el contenido. *folderName* sirve para obtener sólo aquellos archivos o carpetas que cumplen con dicho nombre, el cuál es usado para la búsqueda de carpetas y archivos.

Al igual que en todas las peticiones al *backend*, se debe determinar a qué servicio se está refiriendo. Para ello se hace uso de la factoría y el método *getCloudService(cloudService)*. Una vez determinado el servicio, se llama al método *getFoldersInFolder(folderId, folderName)* para obtener las carpetas o a *getFilesInFolder(folderId, fileName)* para los archivos, que se encargarán de hacer la petición al servicio correspondiente. Una vez obtenidas todas las carpetas o archivos, se itera sobre ellos y se convierte cada uno en un objeto *FileDTO* usando el método *convertToFileDTO()*. Luego, el servicio de almacenamiento devuelve una lista de carpetas o archivos, que se mostrará en el componente correspondiente.

El diagrama también considera flujos alternativos para manejar errores durante el proceso de recuperación de carpetas y archivos. Si ocurre un error al obtener alguno de ellos, se lanza una excepción y redirige al usuario a la vista *ErrorView.vue* y muestra un mensaje de error indicativo.

3.3.3.- Subir archivos cifrados

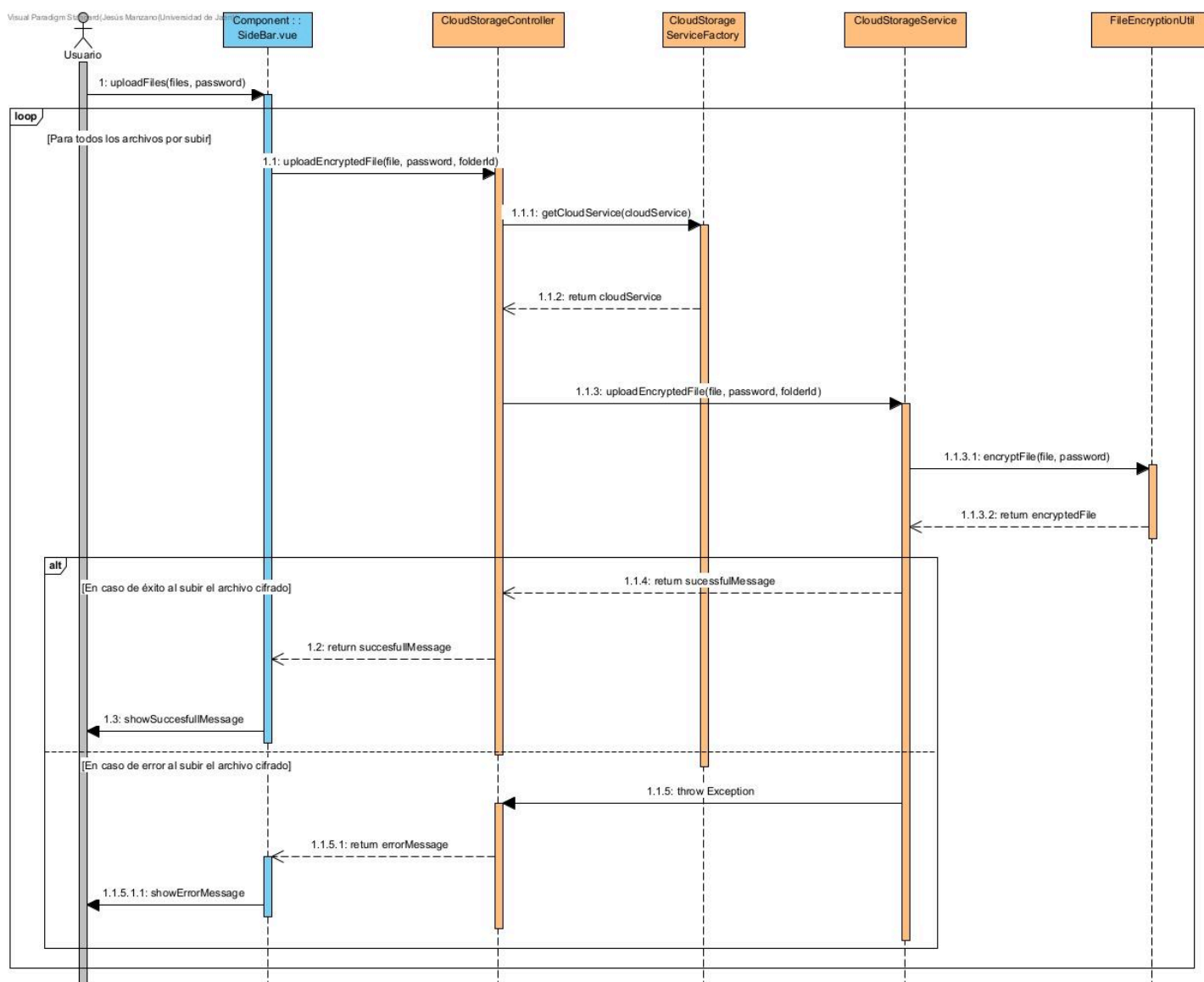


Ilustración 17. Diagrama de secuencia: Subir archivos cifrados

El flujo de eventos comienza cuando el usuario inicia la acción de subir archivos cifrados llamando al método *uploadFiles(files, password)* en *SideBar.vue*. El usuario selecciona los archivos que quiere subir (*files*) y la contraseña con la que se van a cifrar (*password*). Este método invoca *uploadEncryptedFile(file, password, folderId)* en *CloudStorageController*, que utiliza *getCloudService(cloudService)* para obtener el servicio de nube correspondiente a través de la *CloudStorageServiceFactory*.

Una vez que se obtiene el servicio de nube, se llama al método *uploadEncryptedFile(file, password, folderId)* en el servicio correspondiente. Antes de la carga, el archivo se cifra utilizando *encryptFile(file, password)* en el componente *FileEncryptionUtil*. Este componente se encarga de cifrar el archivo

utilizando el algoritmo de cifrado definido. Una vez que el archivo se ha cifrado correctamente, *encryptFile* retorna el archivo cifrado.

Con el archivo cifrado se procede a cargarlo en el servicio en la nube a través de la API correspondiente. Si la carga del archivo se realiza con éxito, el servicio de nube retorna un mensaje de éxito a *CloudStorageController*. Este mensaje de éxito se propaga de vuelta al componente *SideBar.vue*, el cual entonces muestra un mensaje de éxito al usuario. En caso de error, se lanza una excepción, que se tratará de manera correspondiente para enviar al *frontend* un mensaje de error que se mostrará al usuario.

3.3.4.- Descargar archivo cifrado

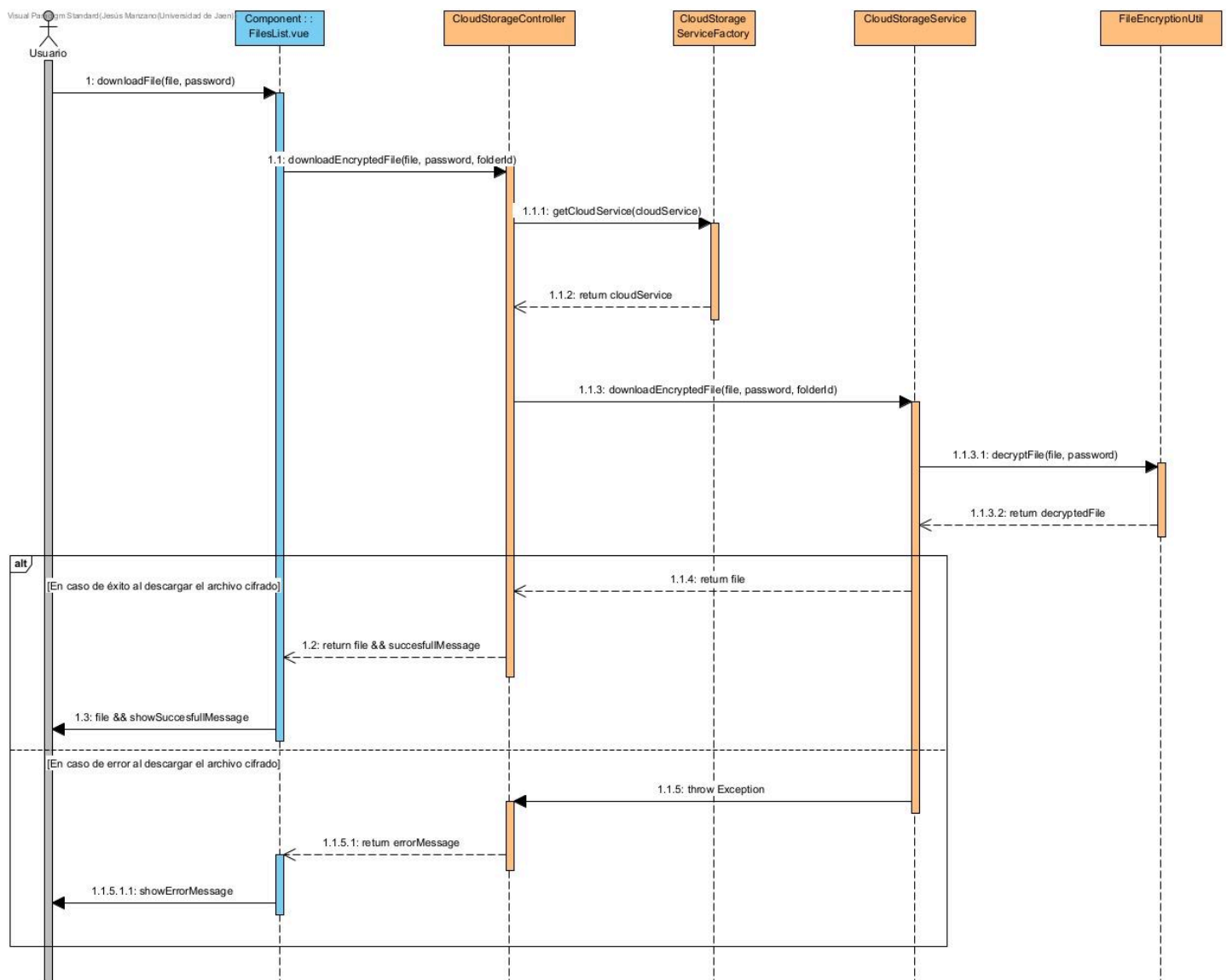


Ilustración 18. Diagrama de secuencia: Descargar archivo cifrado

El proceso de descarga de un archivo cifrado sigue una lógica similar al de la subida, con algunas diferencias clave. En este caso, el usuario inicia el proceso de descarga llamando al método *downloadFile(file, password)* desde el componente *FilesList.vue*. Las llamadas siguientes son similares, aunque evidentemente llamando a los métodos correspondientes del descifrado.

La principal diferencia reside en que una vez recibido el archivo del servicio en la nube se llama al método *decryptFile(file, password)* de la clase *FileEncryptionUtil* para descifrar el archivo obtenido. Este paso es esencialmente el inverso del cifrado que ocurre durante la subida de un archivo. Una vez descifrado, el archivo se retorna al *CloudStorageController*.

Al igual que en la subida si la descarga se hace correctamente o hay algún error, se mostrarán los mensajes correspondientes. Además de dichos mensajes, debe de aparecer el archivo descifrado en el apartado de descargas del navegador del cliente.

3.3.5.- Mover archivos entre nubes

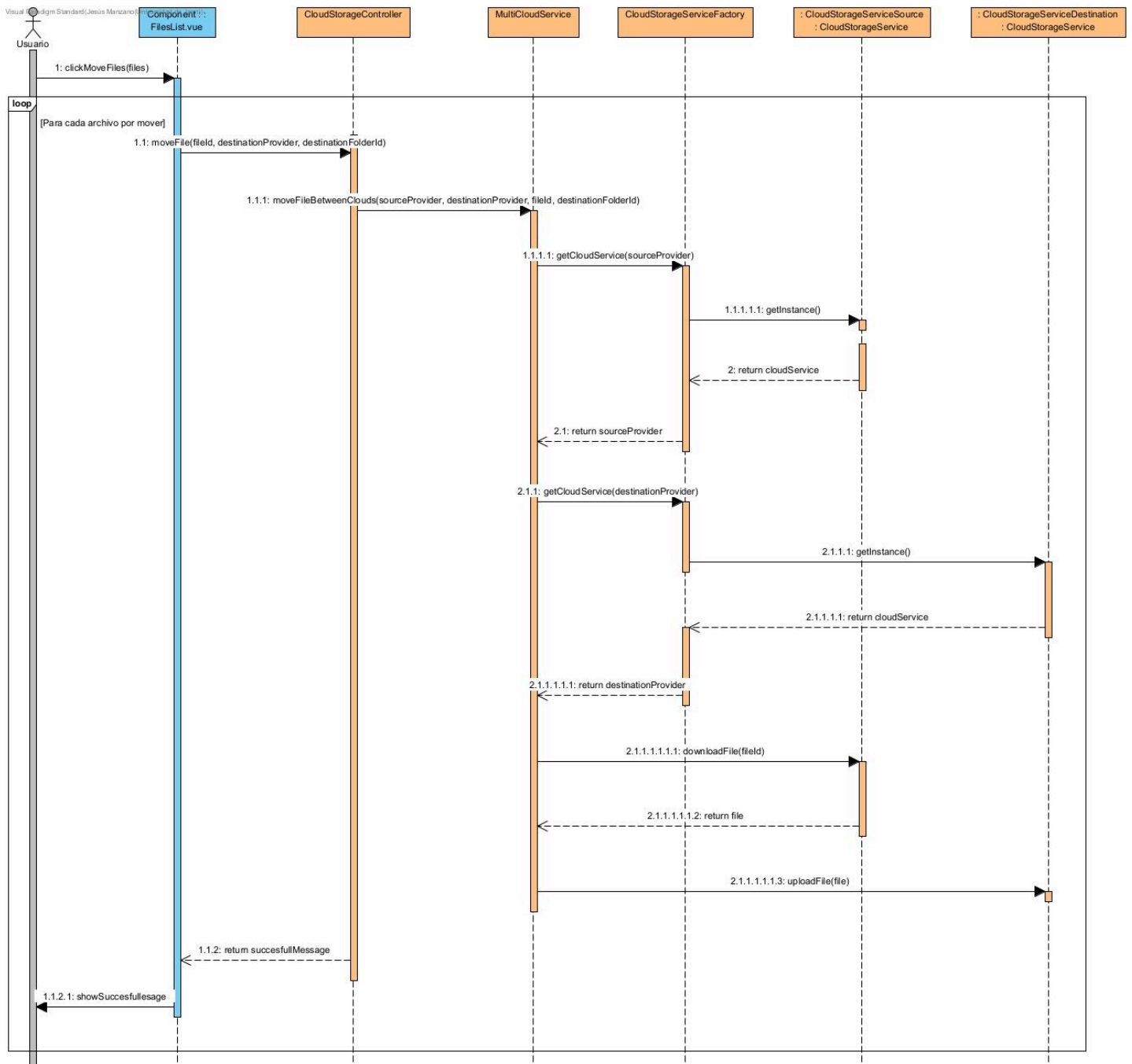


Ilustración 19. Diagrama de secuencia: Mover archivos entre nubes

El diagrama de secuencia muestra cómo se lleva a cabo la operación de mover archivos entre nubes, lo cual en realidad se traduce en copiar archivos de una nube origen a una nube destino. Esta funcionalidad es clave para demostrar la unificación entre diferentes servicios de almacenamiento en la nube de manera transparente. Para gestionar esta operación, se utilizan dos instancias diferentes de la clase *CloudStorageService*, denominadas *CloudStorageServiceSource* y *CloudStorageServiceDestination*.

CloudStorageServiceSource representa el servicio de almacenamiento en la nube desde el cual se copian los archivos. Implementa la interfaz *CloudStorageService* y se encarga de interactuar con la nube origen para la obtención de archivos. En el caso de *CloudStorageServiceDestination* representa el servicio de almacenamiento en la nube al cual se envían los archivos copiados. También implementa la interfaz *CloudStorageService* y maneja la carga de archivos en la nube destino.

Una vez aclaradas las características específicas, vamos a explicar cómo se estructura el flujo de eventos. Este comienza con la solicitud del usuario para mover archivos entre nubes. El sistema inicia el proceso en la vista correspondiente, enviando una solicitud por cada archivo (*fileId*) para copiarlo desde la nube origen (*sourceProvider*) hasta la nube final (*destinationProvider*) en la carpeta especificada (*destinationFolderId*).

La diferencia con respecto a los anteriores diagramas es que dicha funcionalidad al trabajar entre nubes y no en la misma, el controlador avisa a la clase *MultiCloudService*. Esta clase se encarga de obtener la instancia de la nube origen y destino gracias a la factoría, es decir a diferencia de los diagramas anteriores deberá hacerlo dos veces, una por cada nube. Una vez tenemos ambos servicios y el archivo que queremos copiar de la nube origen a la carpeta específica de la nube destino, debe descargar el archivo de la nube origen y subirlo a la nube destino en la carpeta especificada.

Tras la copia de archivos, se actualizan las vistas para reflejar los cambios. Aunque el diagrama no muestra explícitamente los mensajes de éxito o error, es importante señalar que, en la práctica, el sistema enviaría mensajes para confirmar la operación exitosa o reportar errores, basándose en la funcionalidad de los diagramas de secuencia anteriores, debido a que a fin de cuentas el proceso se basa en descargar archivos de la nube origen para subirlos a la nube destino, como ya se ha explicado en los diagramas anteriores.

3.4.- Diseño de la interfaz

La interfaz de la aplicación está formada por aquellos elementos con los que interaccionarán agentes externos a la aplicación. Se partirá del análisis de la interfaz teniendo en cuenta cómo se habían organizado los diferentes elementos. El diseño de esta interfaz ha sido un aspecto crucial en el desarrollo, ya que determina la usabilidad y la experiencia del usuario final.

3.4.1.- Wireframe de la interfaz gráfica del usuario

En primer lugar, se ha desarrollado un *wireframe* detallado y elaborado que proporciona una representación visual precisa de la interfaz de usuario. Este

wireframe es el resultado de numerosas modificaciones en el diseño. No es un simple esquema en blanco y negro; en su lugar, se ha optado por un diseño colorido y detallado que imita la apariencia final de la aplicación. El *wireframe* incluye elementos de diseño realistas como íconos, tipografía y paleta de colores que serán utilizados en la versión final de la aplicación.

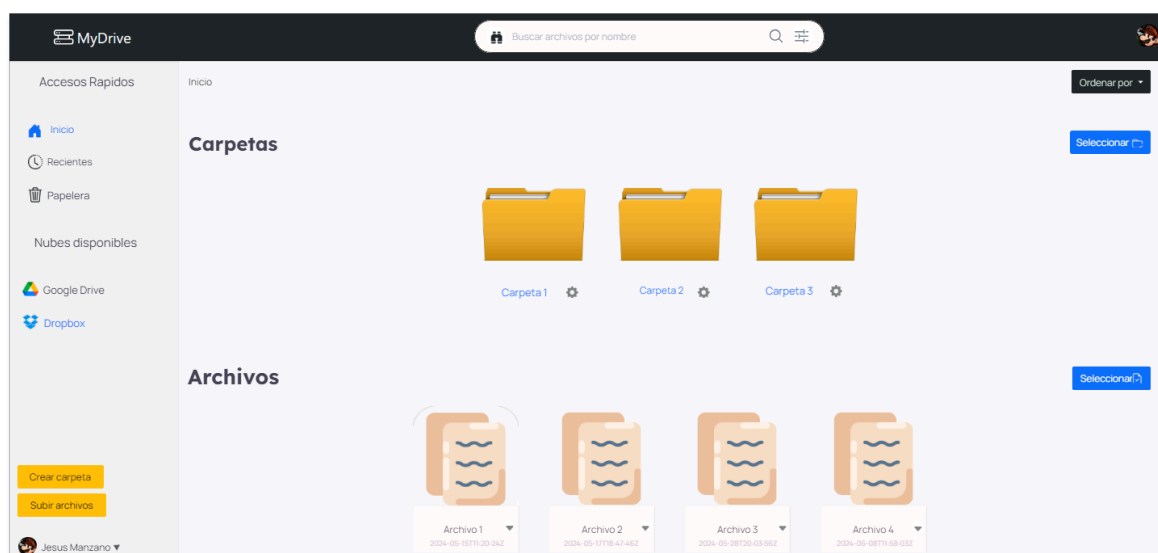


Ilustración 20. Diseño de la interfaz de usuario

En la parte izquierda o *sidebar*, se encuentra un menú de navegación vertical con accesos rápidos a diferentes secciones: Inicio, Recientes y Papelera. También se incluyen accesos a nubes disponibles como Google Drive y Dropbox, proporcionando a los usuarios una integración sencilla con sus servicios de almacenamiento en la nube preferidos. Además, en la parte inferior, se encuentran botones para crear una nueva carpeta y subir archivos. Estos botones están destacados en color amarillo, lo que los hace fácilmente identificables y accesibles para los usuarios. Además, en la esquina inferior izquierda, se muestra el nombre del usuario actual junto con un menú desplegable para opciones adicionales de cuenta.

En la parte superior central o *header*, hay una barra de búsqueda que permite a los usuarios buscar carpetas y archivos por nombre, así como un botón de opciones de búsqueda que permite filtrar la búsqueda a nivel de carpeta o global. A la derecha de cada título de archivos y carpetas, hay un botón de "Seleccionar" que permite a los usuarios realizar acciones masivas, como mover o eliminar múltiples *ítems* a la vez.

En la parte inferior, en el lado izquierdo encontramos la ruta completa desde la carpeta raíz hasta la carpeta actual. En este caso es por eso que solo se muestra "Inicio". En el lado derecho al mismo nivel, encontramos el botón de ordenamiento en que podemos ordenar los archivos por diferentes criterios como nombre (por defecto), fecha y tamaño.

En el centro de la pantalla o *main*, se presentan las carpetas y archivos disponibles. Las carpetas están representadas con iconos grandes y llamativos, con el nombre justo debajo como "Carpeta 1", "Carpeta 2", y "Carpeta 3". Cada carpeta incluye un pequeño icono de configuración que permite a los usuarios realizar acciones específicas como renombrar, mover o eliminar la carpeta. Debajo de las carpetas, se encuentran los archivos, que también están representados con iconos y su nombre debajo. Cada archivo muestra la fecha y hora de su última vista o modificación, facilitando la gestión de documentos a los usuarios.

3.4.2.- Mapa web jerárquico

El mapa web jerárquico de la aplicación refleja la estructura organizada y lógica de las distintas vistas y componentes de la interfaz, proporcionando una visión clara y ordenada de cómo los usuarios pueden navegar a través de esta. Se va a presentar un mapa web que describe la jerarquía de las páginas y sus relaciones.

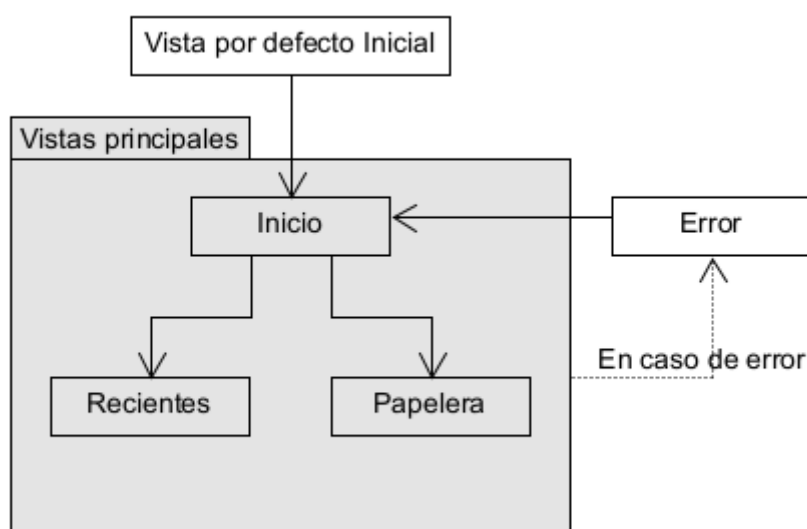


Ilustración 21. Mapa web jerárquico

La navegación comienza con la Vista por defecto Inicial, que actúa como la página de bienvenida o punto de entrada a la aplicación. Desde esta vista inicial, los usuarios son dirigidos a la Página de Inicio, que sirve como el centro de control principal de la aplicación.

En la Página de Inicio, los usuarios tienen dos opciones principales de navegación: Recientes y Papelera. La pestaña Recientes permite a los usuarios acceder rápidamente a los archivos y carpetas que han sido utilizados o modificados recientemente, proporcionando una manera eficiente de retomar el trabajo en documentos activos. Por otro lado, la pestaña Papelera almacena los archivos

eliminados, ofreciendo la posibilidad de restaurarlos o eliminarlos de manera definitiva según sea necesario.

Además de estas rutas principales, existe una Vista de Error que se activa en caso de que ocurra un error irremediable en la aplicación en cualquiera de las vistas principales. Esta vista proporciona a los usuarios información sobre el error ocurrido y ofrece la posibilidad de volver a la vista de Inicio a un estado estable, asegurando una experiencia de usuario sin interrupciones.

3.4.3.- Storyboards

El diseño de la interfaz de usuario no estaría completo sin una representación visual de los flujos de interacción del usuario. Los storyboards son una herramienta esencial en este proceso, ya que permiten visualizar cómo los usuarios navegarán y utilizarán las diferentes funciones de la aplicación. Estos diagramas narrativos ayudan a identificar posibles problemas de usabilidad y a asegurar que la experiencia del usuario sea lo más intuitiva y fluida posible.

En esta sección, se presentan algunos de los storyboards clave que ilustran los casos de uso más importantes y representativos de la aplicación. Estos storyboards muestran paso a paso las acciones que un usuario realiza para completar tareas específicas, como el inicio de sesión, mover archivos entre nubes y subir archivos cifrados. Cada storyboard destaca los puntos de interacción críticos y las transiciones entre diferentes vistas.

3.4.3.1.- Conectar un servicio de almacenamiento en la nube

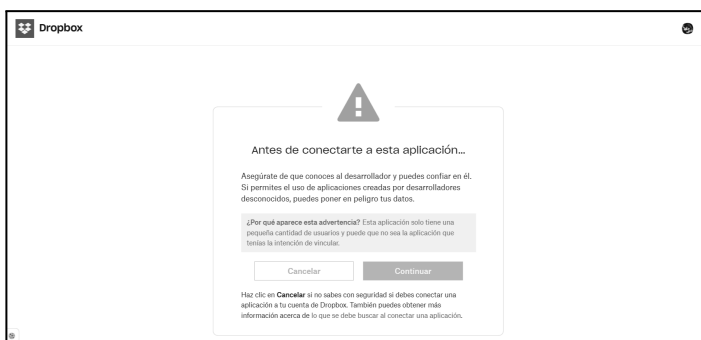
El storyboard inicial muestra el proceso de acceso a la aplicación, comenzando con la vista de Inicio antes de iniciar sesión en un servicio de almacenamiento en la nube y una vez pasada la página de inicio por defecto, que solo se usa para dar paso a la vista principal del gestor de archivos. Este flujo es fundamental para asegurar que los usuarios comprendan cómo comenzar a utilizar la aplicación de manera efectiva desde el primer momento.

1



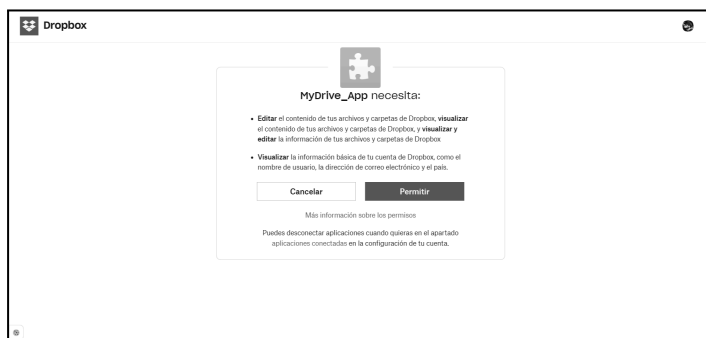
Antes de iniciar sesión en algún servicio

2



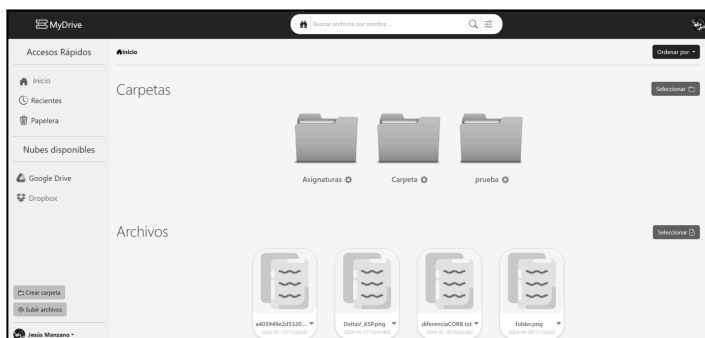
Redirección al usuario para iniciar sesión

3



Autorización del usuario a MyDrive

4



Redirección al usuario a la carpeta raíz de la nube

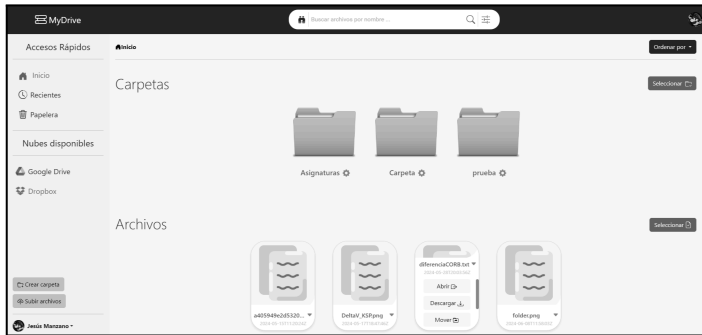
El storyboard mostrado, ilustra el proceso de inicio de sesión en un servicio de almacenamiento en la nube, específicamente Dropbox, y muestra las diferentes etapas que un usuario atraviesa para completar todo el proceso. Este es similar para otros servicios como Google Drive. Inicialmente, en la primera pantalla, se presenta la interfaz principal de la aplicación antes de que el usuario haya iniciado sesión en cualquier servicio de almacenamiento. En esta vista, se muestra un mensaje invitando al usuario a seleccionar uno de los servicios disponibles en la barra lateral izquierda, como Google Drive o Dropbox, con el fin de orientar al usuario hacia el siguiente paso sin abrumarlo con información. Al seleccionar Dropbox, el usuario es redirigido a una pantalla de autorización donde se le informa que debe conectarse a su cuenta de Dropbox para continuar. Esta pantalla incluye un mensaje de advertencia y detalles sobre los permisos necesarios, asegurando que el usuario entienda lo que implica la conexión de su cuenta y tiene la opción de cancelar o proceder con la autorización.

En la etapa siguiente, la aplicación *MyDrive* solicita permisos específicos de la cuenta de Dropbox del usuario. Esta pantalla es fundamental, ya que garantiza que la aplicación tenga acceso a los recursos necesarios de Dropbox para operar correctamente. El usuario debe aceptar estos permisos para proceder, lo que incluye la posibilidad de visualizar, modificar y eliminar archivos en su cuenta de Dropbox. Una vez que el usuario ha iniciado sesión y autorizado los permisos, se le redirige de nuevo a la aplicación *MyDrive*, a la pantalla de Inicio que se ha mostrado antes en el apartado anterior en el *wireframe*, donde ahora puede ver y gestionar sus archivos y carpetas en Dropbox.

3.4.3.2.- Mover un archivo entre diferentes servicios de almacenamiento

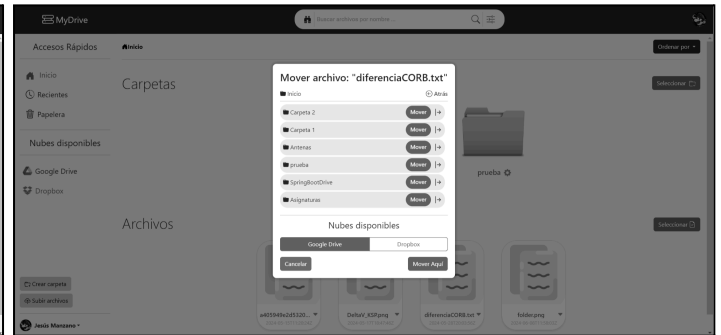
El storyboard que se va a mostrar, ilustra el proceso de mover archivos entre diferentes servicios de almacenamiento en la aplicación *MyDrive*, destacando la facilidad de gestión y transferencia de archivos entre servicios como Google Drive y Dropbox.

1



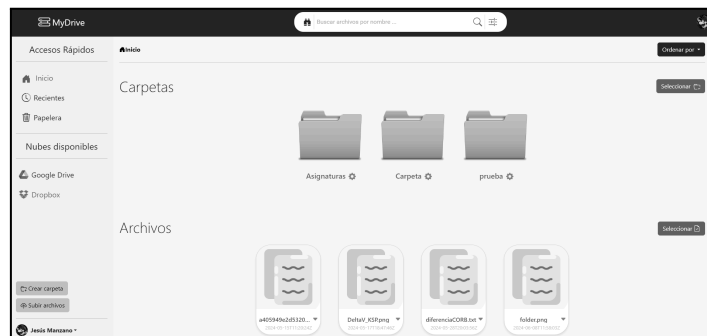
Seleccionar opción de mover

2



Escoger carpetas destino en la nube destino

3



Redirección al usuario a la vista de Inicio

En la primera pantalla, el usuario selecciona la opción de mover un archivo desde la interfaz principal de la aplicación. Esta acción se realiza desde la pantalla de Inicio, previamente descrita en el *wireframe*, donde se muestran las carpetas y archivos disponibles. Al seleccionar un archivo, se despliega un menú contextual que incluye la opción de "Mover". El usuario hace clic en esta opción para iniciar el proceso de transferencia.

En la segunda pantalla, se presenta una ventana emergente donde el usuario puede escoger la carpeta de destino dentro del servicio de almacenamiento al que desea mover el archivo. Esta ventana muestra una estructura jerárquica de las carpetas disponibles en los diferentes servicios de almacenamiento conectados a la aplicación. El usuario puede navegar por estas carpetas, seleccionando el destino adecuado para el archivo. En este ejemplo, se observa cómo el usuario puede seleccionar entre las carpetas disponibles en su cuenta de Dropbox, Google Drive u otros servicios conectados. De esta manera no solo podrá mover el archivo de dentro del propio servicio sino que también podrá copiarlo en otro servicio diferente. La interfaz es intuitiva, permitiendo al usuario ver claramente las opciones y hacer su selección con facilidad.

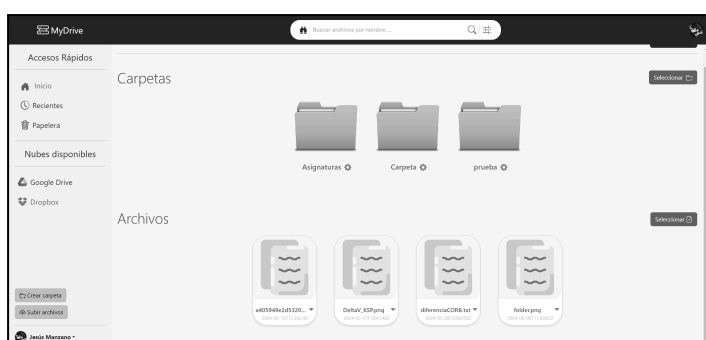
Finalmente, en la tercera pantalla, una vez que el usuario ha confirmado la carpeta de destino, la aplicación redirige al usuario de nuevo a la vista de Inicio del servicio en la nube origen. Aquí, se muestra un mensaje de éxito o error, en caso de que lo

haya. La interfaz actualiza la vista para reflejar cualquier cambio realizado, asegurando que el usuario tenga una visión actualizada de sus archivos y carpetas. Este storyboard demuestra la simplicidad y eficacia del proceso de mover archivos entre diferentes servicios de almacenamiento en la nube dentro de *MyDrive*, proporcionando una experiencia de usuario fluida y coherente.

3.4.3.2.- Subir un archivo cifrado

El storyboard proporcionado ilustra el proceso de subir un archivo cifrado a una carpeta actual en la aplicación *MyDrive*, destacando los pasos necesarios para asegurar la confidencialidad del archivo.

1



Pulsar opción para subir archivo en la carpeta actual

2



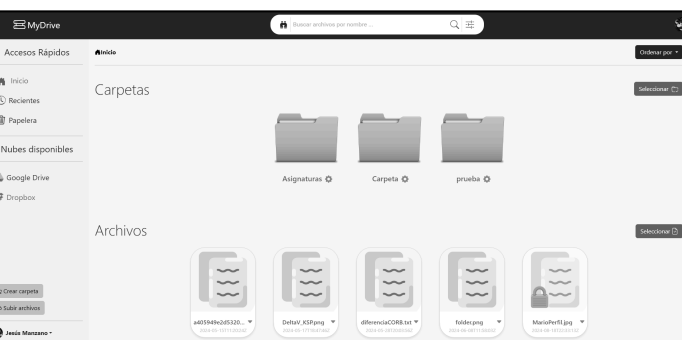
Seleccionar o arrastrar los archivos por subir

3



Modificar lista de archivos y establecer contraseña

4



Archivo subido con el candado indicativo

En la primera pantalla, el usuario pulsa la opción para subir un archivo dentro de la carpeta en la que se encuentra actualmente. Esta acción se realiza desde la pantalla de Inicio, previamente descrita en el *wireframe*, donde se muestran las carpetas y archivos disponibles. Al seleccionar la opción de "Subir archivo", se despliega una ventana emergente que inicia el proceso de carga.

En la segunda pantalla, el usuario tiene la opción de seleccionar o arrastrar los archivos que desea subir. La ventana emergente proporciona una interfaz clara y sencilla para añadir los archivos, permitiendo al usuario buscar en su dispositivo o arrastrar directamente los archivos a la zona designada. Este paso asegura que el

usuario pueda agregar fácilmente los archivos que necesita cargar en la carpeta seleccionada.

En la tercera pantalla, el usuario puede modificar la lista de archivos seleccionados y establecer una contraseña para cifrar el archivo antes de subirlo. Esta función es crucial para garantizar la seguridad y confidencialidad de los archivos sensibles. La interfaz permite al usuario ver la lista de archivos seleccionados, eliminar aquellos que no desea subir y establecer una contraseña de cifrado que será necesaria para acceder al archivo una vez subido.

Finalmente, en la cuarta pantalla, una vez que el archivo se ha subido correctamente, se muestra un mensaje de éxito o error en caso de que lo haya. Además, en la carpeta actual se señala que el archivo está cifrado con un icono de candado representativo. La interfaz actualiza la vista para reflejar cualquier cambio realizado, asegurando que el usuario tenga una visión actualizada de sus archivos y carpetas.

3.4.4.- Descripción de la API Rest

En este apartado, se describen las características del API REST de la aplicación, diseñado para gestionar operaciones de almacenamiento en la nube a través de múltiples proveedores como Google Drive y Dropbox. La interfaz se ha desarrollado para cumplir con los requisitos de las historias de usuario definidas en la fase de análisis. El controlador REST *CloudStorageController* se ha implementado para proporcionar una única interfaz a través de la cual los usuarios pueden interactuar con diferentes servicios de almacenamiento en la nube, facilitando la autorización OAuth, la gestión de archivos y carpetas, la búsqueda, la descarga y el manejo de archivos cifrados, entre otras funcionalidades.

Para documentar esta API de manera adecuada, se ha utilizado el estándar OpenAPI v3 [12], lo cual ha permitido generar una especificación detallada y coherente de los *endpoints*, los métodos HTTP utilizados (GET, POST, PUT, DELETE), los parámetros de entrada y salida, así como los códigos de estado y error que se devuelven en cada operación. Esta especificación ha sido integrada de manera dinámica en el *backend* de la aplicación, desarrollado con Spring Boot, gracias a la herramienta de Swagger UI. Permitiendo que la documentación se mantenga actualizada en tiempo real conforme se realizan cambios en los controladores.

Sin embargo, hubo un desafío durante la implementación de la documentación con OpenAPI, ya que el *swagger* no era capaz de detectar de manera global el parámetro "provider", que es un valor crucial para identificar el servicio de almacenamiento en la nube utilizado. Este parámetro debía poder ser establecido en

todos los endpoints de la API, pero debido a las limitaciones de Swagger, no se podía aplicar una configuración común a nivel global para todos los controladores que lo requerían.

Para solucionar este inconveniente y mantener un código más limpio y organizado, se optó por crear una anotación personalizada. Esta anotación se encarga de definir el parámetro "provider" en cada endpoint.

```
Java
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
@Operation(
    parameters = @Parameter(
        name = "provider",
        description = "Cloud provider",
        required = true,
        in = ParameterIn.PATH,
        schema = @Schema(
            allowableValues = {"google-drive", "dropbox"})
    )
)
public @interface ProviderParam {
}
```

Después de implementar la anotación personalizada y configurar correctamente los parámetros en cada uno de los endpoints, se puede visualizar el resultado final en Swagger UI. Para ello, es necesario arrancar el *backend* de la aplicación, que por defecto se levanta en *localhost:8080*, para posteriormente acceder a la ruta del Swagger desde el navegador:

```
Unset
http://localhost:8080/swagger-ui/index.html#/
```

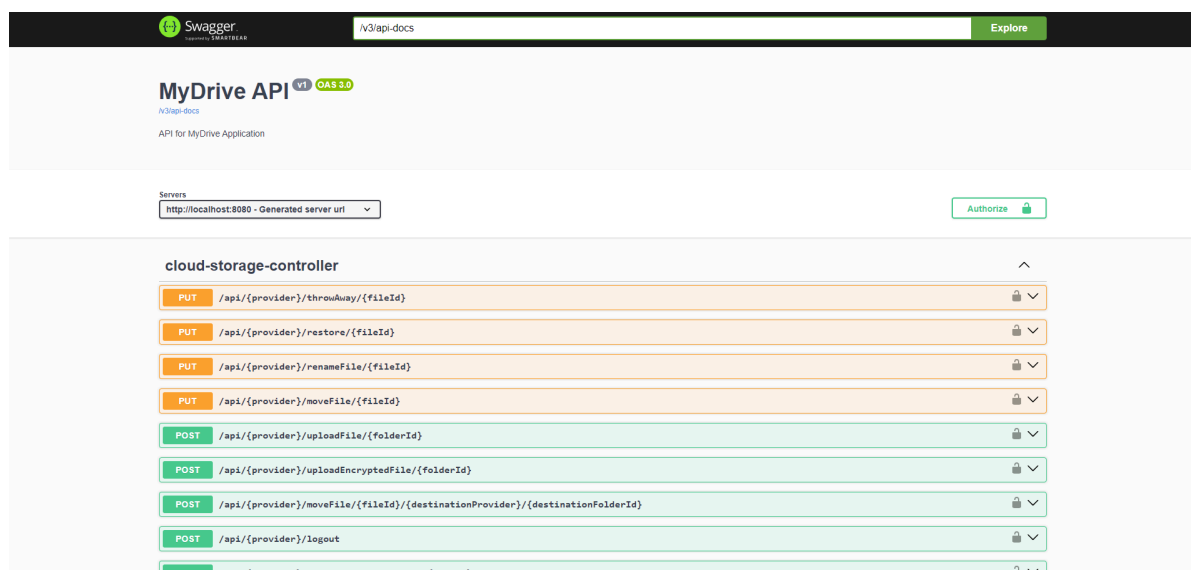


Ilustración 22. Documentación de la API con OpenAPI

La generación dinámica de la documentación mediante el uso de herramientas como Springdoc o Swagger UI dentro del ecosistema de Spring Boot permite a los desarrolladores y otros agentes externos interactuar con la API de manera intuitiva y en tiempo real a través de un navegador, facilitando el proceso de desarrollo, pruebas y futuras ampliaciones de la aplicación.

A continuación, se presenta una tabla general que muestra todas las características de la API, incluyendo los *endpoints* desarrollados para cubrir las funcionalidades especificadas en las historias de usuario. Esta tabla servirá como guía rápida para entender la estructura y el alcance del API, proporcionando una visión comprensiva de las operaciones que se pueden realizar y cómo se integran dentro del flujo general de la aplicación.

Endpoint	Método HTTP	Descripción	Parámetros de entrada	Códigos de respuesta
/api/{provider}/oauth/authorize	GET	Inicia el proceso de autorización OAuth con el proveedor de almacenamiento en la nube especificado.	provider: Nombre del proveedor	(Redirección a la URL de autorización)
/api/{provider}/oauth/callback	GET	Maneja la respuesta del proceso de autorización OAuth y autentica al usuario en la nube seleccionada.	provider: Nombre del proveedor code: Código autorización recibido	(Redirección a la URL principal de la aplicación)
/api/{provider}/oauth/check	GET	Verifica si el usuario está autenticado en el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor	Correct: 200 OK Error: 500 Internal Server

				Error
/api/{provider}/logout	POST	Cierra la sesión del usuario en el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/profilePhoto	GET	Obtiene la URL de la foto de perfil del usuario desde el servicio de almacenamiento en la nube.	provider: Nombre del proveedor	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/userName	GET	Obtiene el nombre de usuario desde el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/path/{folderId}	GET	Obtiene la ruta completa de una carpeta especificada por su ID en el servicio de almacenamiento.	provider: Nombre del proveedor folderId: ID de la carpeta	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/folders/{folderId}	GET	Lista las carpetas dentro de una carpeta especificada por su ID.	provider: Nombre del proveedor folderId: ID de la carpeta q (opcional): Término de búsqueda	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/files/{folderId}	GET	Lista los archivos dentro de una carpeta especificada por su ID.	provider: Nombre del proveedor folderId: ID de la carpeta q (opcional): Término de búsqueda.	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/recentFiles	GET	Obtiene una lista de archivos recientes en el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor maxDate: Fecha máxima para filtrar	Correct: 200 OK Error: 500 Internal Server Error

			q (opcional): Término de búsqueda.	
/api/{provider}/files/bin	GET	Lista los archivos en la papelera de reciclaje en el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor q (opcional): Término de búsqueda	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/searchFolder/{folderName}	GET	Busca carpetas por nombre en el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor folderName: Nombre de la carpeta a buscar	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/searchFile/{fileName}	GET	Busca archivos por nombre en el servicio de almacenamiento en la nube seleccionado.	provider: Nombre del proveedor fileName: Nombre del archivo a buscar	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/createFolder/{folderId}	POST	Crea una nueva carpeta en una carpeta especificada dentro del servicio de almacenamiento en la nube.	provider: Nombre del proveedor folderId: ID de la carpeta donde se creará la nueva carpeta name: Nombre de la nueva carpeta	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/uploadFile/{folderId}	POST	Sube un archivo a una carpeta especificada en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor folderId: ID de la carpeta donde se creará la nueva carpeta file: Archivo a subir	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/uploadEncryptedFile/{folderId}	POST	Sube un archivo cifrado a una carpeta especificada en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor folderId:	Correct: 200 OK Error: 500 Internal Server Error

			ID de la carpeta donde se creará la nueva carpeta file: Archivo a subir password: Contraseña para cifrar el archivo	Error
/api/{provider}/preview-link/{fileId}	GET	Obtiene la URL de vista previa para un archivo especificado por su ID en el servicio de almacenamiento.	provider: Nombre del proveedor fileId: ID del archivo a obtener la vista previa	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/download/{fileId}	GET	Descarga un archivo especificado por su ID desde el servicio de almacenamiento en la nube.	provider: Nombre del proveedor fileId: ID del archivo a descargar	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/downloadEncryptedFile/{fileId}	POST	Descarga un archivo cifrado especificado por su ID y lo descifra con la contraseña proporcionada.	provider: Nombre del proveedor fileId: ID del archivo a descargar password: Contraseña para descifrar el archivo	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/moveFile/{fileId}	PUT	Mueve un archivo a una carpeta especificada en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor fileId: ID del archivo a mover folderId: ID de la carpeta destino	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/renameFile/{fileId}	PUT	Renombra un archivo en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor fileId: ID del archivo a renombrar	Correct: 200 OK Error: 500 Internal Server Error

			name: Nuevo nombre del archivo	
/api/{provider}/throwAway/{fileId}	PUT	Mueve un archivo a la papelera de reciclaje en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor fileId: ID del archivo a mover a la papelera	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/restore/{fileId}	PUT	Restaura un archivo de la papelera de reciclaje en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor fileId: ID del archivo a mover a restaurar	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/delete/{fileId}	DELETE	Elimina permanentemente un archivo en el servicio de almacenamiento en la nube.	provider: Nombre del proveedor fileId: ID del archivo a eliminar	Correct: 200 OK Error: 500 Internal Server Error
/api/{provider}/moveFile/{fileId}/{destinationProvider}/{destinationFolderId}	POST	Mueve un archivo de un servicio de almacenamiento en la nube a otro.	provider: Nombre del proveedor fileId: ID del archivo a mover, destination Provider: Proveedor destino destination FolderId: ID de la carpeta destino	Correct: 200 OK Error: 500 Internal Server Error

Tabla 51. Guía de los endpoints de la API Rest de MyDrive

En la tabla presentada anteriormente, se han especificado los códigos de respuesta que se generan habitualmente para las operaciones exitosas y errores generales: 200 OK para indicar que la operación se realizó correctamente y 500 *Internal Server Error* para denotar un error en el servidor. Sin embargo, el sistema también puede generar otros códigos de respuesta dependiendo de las circunstancias específicas.

Uno de estos códigos adicionales es 400 *Bad Request*, que se devuelve cuando la solicitud enviada por el cliente está mal formulada o contiene errores. Esto puede

ocurrir si faltan parámetros necesarios, si los datos enviados no tienen el formato adecuado, o si se realiza una petición incompatible con las expectativas del servidor. El uso de este código es crucial, ya que informa al cliente que la solicitud debe corregirse antes de ser reenviada.

Además, el sistema incluye un manejo específico para ciertas limitaciones que pueden encontrarse al interactuar con los servicios de almacenamiento en la nube. Para estos casos, se ha definido una excepción particular denominada *CloudLimitationException*. Esta excepción se lanza cuando se detecta una restricción impuesta por el proveedor de almacenamiento. Cuando esta excepción se produce, el servidor responde con un código 409 *Conflict*. Este código es particularmente útil porque permite al *frontend* identificar la naturaleza específica del problema, proporcionando al usuario un mensaje claro sobre la limitación que ha ocurrido.

3.5.- Plan de pruebas

En este proyecto, se ha planificado un enfoque de pruebas basado en *test-later development* [8], un método en el que las pruebas se desarrollan y ejecutan después de que las funcionalidades han sido implementadas. Este enfoque permite priorizar la implementación de las funcionalidades, para luego validar su correcto funcionamiento mediante pruebas diseñadas específicamente para cada componente.

Las pruebas se centrarán en varios niveles del sistema, tanto en el *frontend* como en el *backend*. En el nivel de pruebas unitarias, se verificará que cada función o módulo individual opere de manera correcta y aislada. Para el *backend*, se utilizará JUnit⁹, una herramienta que permite realizar pruebas automatizadas sobre los métodos y clases del sistema, asegurando su correcto comportamiento. En el *frontend*, se empleará Jest¹⁰, un framework de pruebas en JavaScript que facilita la verificación de los componentes de la interfaz de usuario, garantizando que cada parte funcione sin errores en su contexto.

A nivel de integración, se realizarán pruebas para asegurar que los diferentes módulos del sistema interactúan entre sí de manera adecuada. Esto incluirá verificar que los servicios de almacenamiento en la nube y la lógica de negocio en el *backend* funcionen en conjunto como se espera. Además, se realizarán pruebas de integración para asegurar que el *frontend* se comunique correctamente con el *backend* y que los datos se procesen y muestren de manera adecuada. Las pruebas de integración ayudarán a detectar posibles problemas en la comunicación y en la combinación de diferentes partes del sistema.

⁹ Fuente: <https://junit.org/junit5/>

¹⁰ Fuente: <https://jestjs.io/>

Se utilizará Swagger UI de OpenAPI para llevar a cabo pruebas manuales sobre los *endpoints* y métodos del API REST. Esta herramienta permitirá validar que las interacciones externas con la API sean manejadas correctamente, incluyendo la validación de respuestas y el manejo de errores. Las pruebas automatizadas también se realizarán para asegurar que los *endpoints* del *backend* respondan adecuadamente a diversas solicitudes.

En el marco de la metodología ágil aplicada al proyecto, se planificarán pruebas de aceptación para cada sprint. Estas pruebas estarán alineadas con los objetivos y las historias de usuario de cada *sprint*, verificando que las funcionalidades implementadas cumplan con los requisitos especificados. Las pruebas de aceptación se realizarán al final de cada sprint para garantizar que los hitos alcanzados cumplen con los estándares de calidad previstos.

Finalmente, se prestará especial atención a la cobertura de todos los requisitos funcionales y no funcionales planteados inicialmente. Esto incluirá tanto la validación de las funcionalidades principales como de los itinerarios alternativos en cada historia de usuario.

3.5.1.- Plan de pruebas de aceptación del usuario (UAT)

En el contexto de proyectos ágiles gestionados bajo la metodología ágil, las Pruebas de Aceptación del Usuario (UAT) juegan un papel crucial para garantizar que el producto final cumpla con los requisitos y expectativas del usuario. A diferencia de las pruebas unitarias y de integración que se realizan durante el desarrollo, las pruebas de aceptación se centran en validar que las funcionalidades entregadas en cada *sprint* satisfacen las necesidades del cliente y cumplen con los criterios de aceptación predefinidos.

Estas pruebas no se automatizan como las pruebas unitarias o de integración; en su lugar, se basan en los criterios de aceptación establecidos para cada historia de usuario y se realizan manualmente. Las UAT se llevan a cabo por personas denominadas *testers* [9], quienes ejecutan pruebas en un entorno que simula el entorno de producción para verificar que las funcionalidades funcionan según lo esperado desde la perspectiva del usuario final.

En el caso de este proyecto, dado que el equipo está formado por una sola persona, yo seré el encargado de realizar todas las pruebas de aceptación. Esto implica una revisión exhaustiva de cada funcionalidad desarrollada en los *sprints*, asegurando que se ajusten a los criterios de aceptación y cumplan con las expectativas del cliente. Este enfoque manual garantiza que el producto se comporte correctamente en situaciones reales de uso, proporcionando una validación integral del trabajo

realizado en cada iteración. Cabe destacar que los criterios que se van a mostrar están alineados con las historias de usuario, pero representan los criterios específicos para completar cada sprint y sirven a modo de resumen. A continuación se presenta el plan de pruebas de aceptación por cada sprint:

Sprint 1 (semanas 1 y 2):

Conectar y desconectar servicios de almacenamiento en la nube: Verificar que la conexión a los servicios de almacenamiento en la nube se realiza correctamente, manejando adecuadamente credenciales y estados. Además, confirmar que la desconexión limpia todos los datos relacionados y no afecta otras funcionalidades del sistema.

Sprint 2 (semanas 3 y 4):

Obtener contenido de una carpeta y navegación entre carpetas: Confirmar que los usuarios pueden obtener todo el contenido de una carpeta y navegar entre carpetas de manera fluida. Verificar que la estructura de carpetas se actualiza correctamente en la interfaz.

Sprint 3 (semanas 5 y 6):

Creación de carpetas y subida de archivos: Validar que los usuarios pueden crear nuevas carpetas y subir archivos sin errores. Comprobar que los archivos se almacenan y visualizan correctamente, y que se manejan adecuadamente los posibles errores durante la subida.

Sprint 4 (semanas 7 y 8):

Subida de archivos (tarea final) y descarga de archivos: Realizar pruebas exhaustivas sobre el proceso de subida de archivos y confirmar que la descarga de archivos se realiza sin problemas, garantizando la integridad del archivo.

Sprint 5 (semanas 9 y 10):

Mover archivos y carpetas, y gestión de la papelera: Verificar que los usuarios pueden mover archivos y carpetas entre diferentes ubicaciones y nubes. Confirmar que los archivos en la papelera se recuperan correctamente y que las operaciones de mover y gestionar archivos cumplen con los requisitos.

Sprint 6 (semanas 11 y 12):

Gestión de la papelera y búsqueda de archivos: Asegurar que los archivos en la papelera se gestionan correctamente, y que la función de búsqueda permite a los usuarios encontrar carpetas y archivos de manera eficaz.

Sprint 7 (semanas 13 y 14):

Obtener archivos recientes y visualización de archivos: Confirmar que la funcionalidad de archivos recientes muestra correctamente los archivos más recientes y que los archivos se visualizan adecuadamente en la interfaz.

Sprint 8 (semanas 15 y 16):

Mover a la papelera y eliminar carpetas: Verificar que los archivos se mueven a la papelera correctamente y que las carpetas se eliminan sin problemas. Asegurar que estos cambios se reflejan correctamente en el sistema.

Sprint 9 (semanas 17 y 18):

Eliminar archivos y obtener información del usuario: Confirmar que los archivos se eliminan correctamente y que la información del usuario se muestra de manera precisa.

Sprint 10 (semanas 19 y 20):

Ordenar carpetas y archivos, vaciar papelera y renombrar archivo: Validar que el sistema permite ordenar carpetas y archivos, vaciar la papelera correctamente, y renombrar archivos sin errores.

Sprint 11 (semanas 21 y 22):

Restaurar archivos y crear carpeta espejo: Verificar que los archivos movidos a la papelera se restauran adecuadamente y que las carpetas espejo se crean y actualizan correctamente.

Sprint 12 (semanas 23 y 24):

Búsqueda unificada y compartir archivos: Confirmar que la búsqueda unificada funciona correctamente a través de diferentes servicios de almacenamiento y que los archivos se pueden compartir con otros usuarios según lo previsto.

4.- Implementación

4.1.- Arquitectura del sistema

La arquitectura del sistema desarrollado, nos permite mostrar de forma esquematizada los grandes elementos funcionales de la aplicación. Este diseño es el resultado de construir una aplicación web estilo cliente-servidor [10], que además debe conectar diferentes servicios de almacenamiento en la nube. En nuestro sistema, estos elementos se agrupan en *frontend*, *backend* y servicios de almacenamiento en la nube. Cada uno de estos componentes desempeña un rol esencial en la funcionalidad general del sistema, utilizando diferentes tecnologías para cumplir con sus responsabilidades.

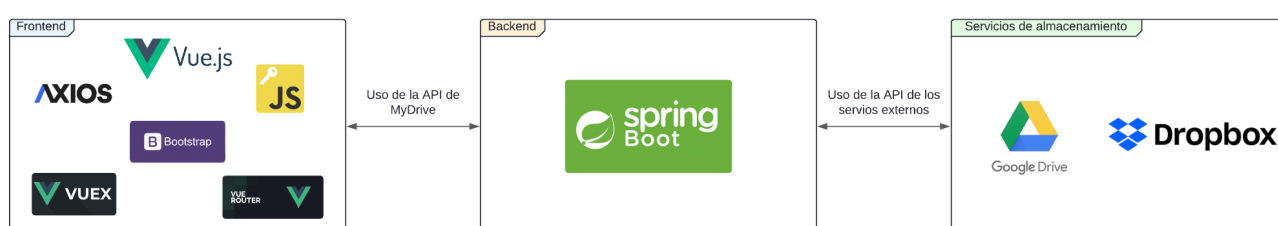


Ilustración 23. Diagrama de arquitectura del sistema

El *frontend* de la aplicación está implementado utilizando Vue.js, un *framework* JavaScript que facilita la creación de interfaces de usuario reactivas y dinámicas. Dentro del *frontend*, *Vuex* se encarga de la gestión del estado global de la aplicación, asegurando que los datos compartidos entre diferentes componentes sean consistentes. *Vue Router* maneja la navegación y las transiciones entre las diferentes vistas, proporcionando una experiencia de usuario fluida. Además, el *frontend* utiliza *Axios*¹¹ para realizar peticiones HTTP al *backend*, permitiendo la comunicación y el intercambio de datos entre el *frontend* y el *backend*. También se integra *Bootstrap* para proporcionar estilos y componentes predefinidos que mejoran la apariencia y la funcionalidad de la interfaz de usuario. Finalmente, se emplea la biblioteca *Crypto*¹² para realizar operaciones de cifrado en el *frontend*, lo que aumenta la seguridad de los datos transmitidos.

El *backend*, por su parte, está implementado con *Spring Boot*, un *framework* de Java que facilita la creación de aplicaciones empresariales. El *backend* es responsable de manejar la lógica de negocio, gestionar las solicitudes HTTP entrantes y comunicarse con los servicios de almacenamiento en la nube. El *frontend* utiliza la API de *MyDrive* para interactuar con estos servicios, proporcionando una capa de abstracción que permite a los usuarios acceder a múltiples servicios de almacenamiento en la nube a través de una interfaz unificada.

¹¹ Fuente: <https://github.com/axios/axios>

¹² Fuente: <https://www.npmjs.com/package/crypto-js>

El diagrama del sistema también muestra los servicios de almacenamiento en la nube, que incluyen Google Drive y Dropbox. Estos servicios son integrados mediante sus respectivas APIs, lo que permite realizar operaciones como la subida, descarga y gestión de archivos en la nube. El *backend* utiliza estas APIs para comunicarse con los servicios externos, garantizando la transferencia de datos entre la aplicación y los servicios de almacenamiento en la nube.

En resumen, la arquitectura del sistema se divide en tres grandes componentes: *frontend*, *backend* y servicios de almacenamiento en la nube, mostrando cómo gracias a las tecnologías y herramientas usadas el usuario puede interactuar con la aplicación produciendo el paso de información de izquierda a derecha y viceversa. En los apartados siguientes, se comentará de forma más detallada la organización de estos grandes elementos y su comunicación, explicando cómo cada uno contribuye a la funcionalidad general del sistema y cómo se han implementado las interacciones entre ellos. Este enfoque modular y basado en capas permite asignar responsabilidades a una de una manera más manejable y organizada.

4.2.- Arquitectura del software

La arquitectura del software desarrollado sigue un enfoque de arquitectura software basada en capas. Este enfoque divide el sistema en diferentes capas, cada una con responsabilidades específicas, lo que facilita el mantenimiento, la escalabilidad y la robustez del sistema. En nuestra aplicación por un lado, se comentarán cuales son a nivel general estas capas y por otro, como se ha agrupado y organizado todo el software en paquetes, partiendo ambos del diagrama del sistema, dividido en *frontend*, *backend* y *servicios externos*, cada una desempeñando roles esenciales en la funcionalidad general del sistema.

4.1.1.- Arquitectura basada en capas

La arquitectura basada en capas es un enfoque ampliamente utilizado en el desarrollo de software que organiza el sistema en diferentes niveles o capas [11], cada una con responsabilidades específicas y bien definidas. Este enfoque promueve la separación de preocupaciones. A continuación, se va a describir la teoría detrás de las capas comunes en una arquitectura software, que presenta un esquema general de una arquitectura basada en capas, para posteriormente tener una referencia teórica con la que comparar el diagrama de capas de nuestra aplicación.

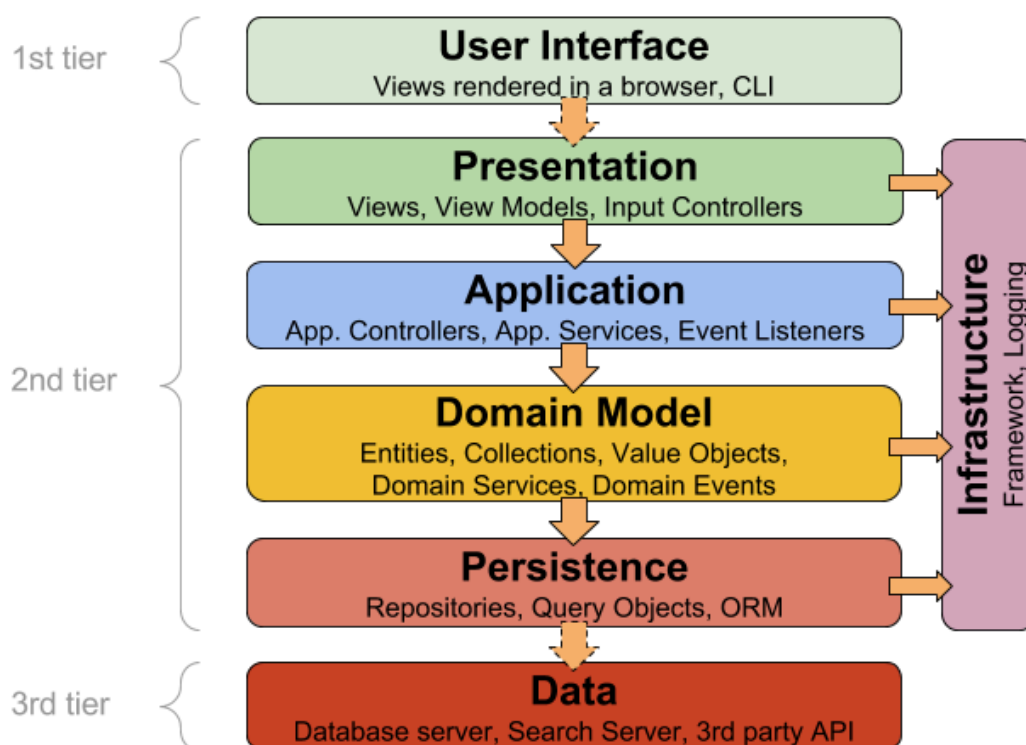


Ilustración 24. Arquitectura software basada en capas¹³

La arquitectura en capas divide el sistema en niveles jerárquicos donde cada capa se encarga de una responsabilidad específica. Las capas más comunes en una arquitectura software incluyen:

- **User Interface (Interfaz de Usuario):** Esta capa es responsable de la interacción con el usuario. Contiene vistas que se renderizan en el navegador o en una interfaz de línea de comandos (CLI). Su principal objetivo es proporcionar una experiencia de usuario amigable y accesible.
- **Presentation (Presentación):** Maneja la lógica de presentación de datos y la comunicación con la capa de aplicación. Incluye vistas, modelos de vista y controladores de entrada. Actúa como intermediario entre la interfaz de usuario y la lógica de negocio de la aplicación.
- **Application (Aplicación):** Esta capa contiene los controladores de la aplicación, servicios de la aplicación y gestores de eventos. Gestiona las peticiones del usuario y coordina la lógica de negocio y el flujo de datos.
- **Domain Model (Modelo de Dominio):** Representa los conceptos clave y la lógica de negocio de la aplicación. Incluye entidades, colecciones, objetos de

¹³ Fuente: <https://www.baoss.es/arquitecturas-de-software-en-capas-pasado-y-presente/>

valor, servicios de dominio y eventos de dominio. Es el núcleo donde se define cómo opera el sistema.

- **Persistence (Persistencia):** Maneja la interacción con las bases de datos u otros mecanismos de almacenamiento. Contiene repositorios, objetos de consulta y ORM (Object-Relational Mapping). Su responsabilidad es garantizar que los datos se almacenen y recuperen de manera eficiente y segura.
- **Data (Datos):** Esta capa incluye los servidores de bases de datos, servidores de búsqueda y APIs de terceros. Se encarga del almacenamiento y recuperación física de los datos.
- **Infrastructure (Infraestructura):** Proporciona los marcos y herramientas necesarios para soportar el funcionamiento del sistema, como los frameworks y la gestión de logs.

Después de comprender la teoría detrás de la arquitectura en capas y los roles específicos de cada capa, es momento de aplicar estos conceptos a nuestra aplicación. A continuación, se presentará el diagrama de capas específico de nuestra aplicación, mostrando cómo se conforma cada una de ellas y cómo se distribuyen las responsabilidades de acuerdo con los principios teóricos previamente discutidos. Este diagrama nos permitirá visualizar claramente la organización y el flujo de datos dentro del sistema en base a las capas del mismo, facilitando la comprensión de su funcionamiento y la identificación de las áreas clave de interacción entre las capas.

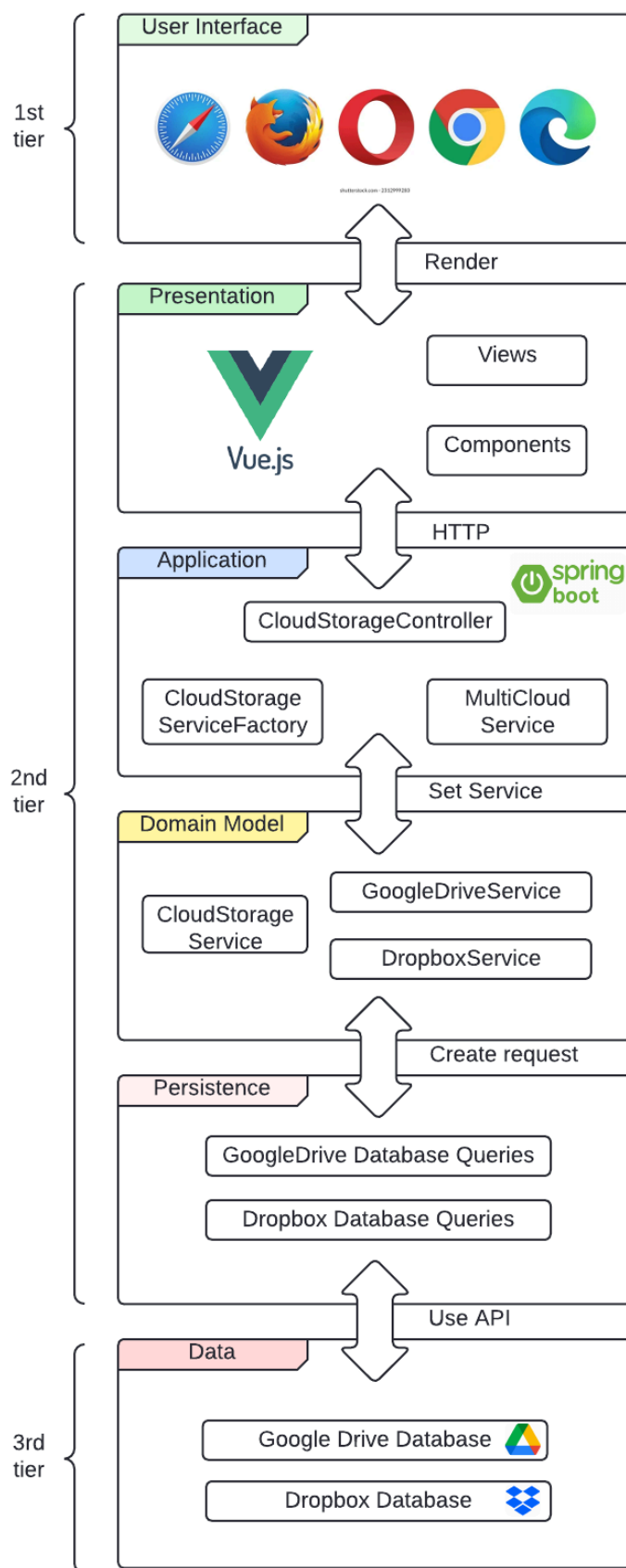


Ilustración 25. Diagrama de capas

En el diagrama de capas de nuestra aplicación se ha mostrado aquellos elementos más importantes dentro de cada capa. Existen otras clases que ya vimos en el apartado de diseño que son necesarias para hacer funcionar la aplicación al completo pero sólo se mostrarán aquellas que son clave en cada capa. En primer lugar, tenemos la capa de *User Interface* (Interfaz de Usuario) que se corresponde con la parte del *frontend* de nuestra aplicación, donde las vistas son renderizadas en el navegador utilizando *Vue.js*. La capa *Presentation* (Presentación) incluye la parte del *frontend* con las vistas junto con sus componentes para manejar los datos recibidos y en consecuencia la lógica de presentación. La capa de *Application* (Aplicación) corresponde al *backend*, los controladores de la aplicación y servicios gestionan las peticiones y hacen de intermediarios antes de interactuar con la verdadera lógica de negocio. La capa de *Domain Model* (Modelo de Dominio) en el caso de nuestra aplicación, incluiría aquellos servicios más específicos encargados de comunicarse con los servicios de almacenamiento en la nube, representando la clave de nuestra aplicación y nuestra lógica de negocio. La capa de *Persistence* (Persistencia) se encarga de interactuar con los servicios de almacenamiento en la nube. Debido a que nuestra aplicación no tiene una base de datos, esta capa se encarga de formular consultas especializadas para cada servicio. En nuestro caso esta capa estaría dentro de la anterior debido a que las clases o servicios encargados de la comunicación, hacen sus correspondientes consultas. La capa de *Data* (Datos) representa la base de datos de los servicios de almacenamiento en la nube como Google Drive y Dropbox. La capa de *Infrastructure* (Infraestructura) proporciona el marco y herramientas necesarias para soportar el funcionamiento del sistema. Todas estas herramientas y tecnologías utilizadas serán explicadas de manera detallada en el apartado de detalles sobre la implementación.

4.1.2.- Arquitectura de paquetes

La arquitectura de paquetes en nuestra aplicación se centra en la organización lógica y estructural del código fuente, facilitando el mantenimiento, la escalabilidad y el entendimiento de futuros posibles desarrolladores. En este enfoque, el código se agrupa en paquetes coherentes según sus responsabilidades y funcionalidades específicas, asegurando una separación clara de preocupaciones. A diferencia de la arquitectura basada en capas, que agrupa funcionalidades en niveles jerárquicos, la arquitectura de paquetes organiza el código en agrupaciones lógicas basadas en responsabilidades específicas. Cada paquete encapsula una parte del sistema, promoviendo la reutilización de código y la modularidad. En nuestra aplicación, hemos considerado cada componente del diagrama del sistema como un paquete general, que contendrá sus subpaquetes correspondientes, teniendo también de esta manera: *frontend*, *backend* y servicios de almacenamiento.

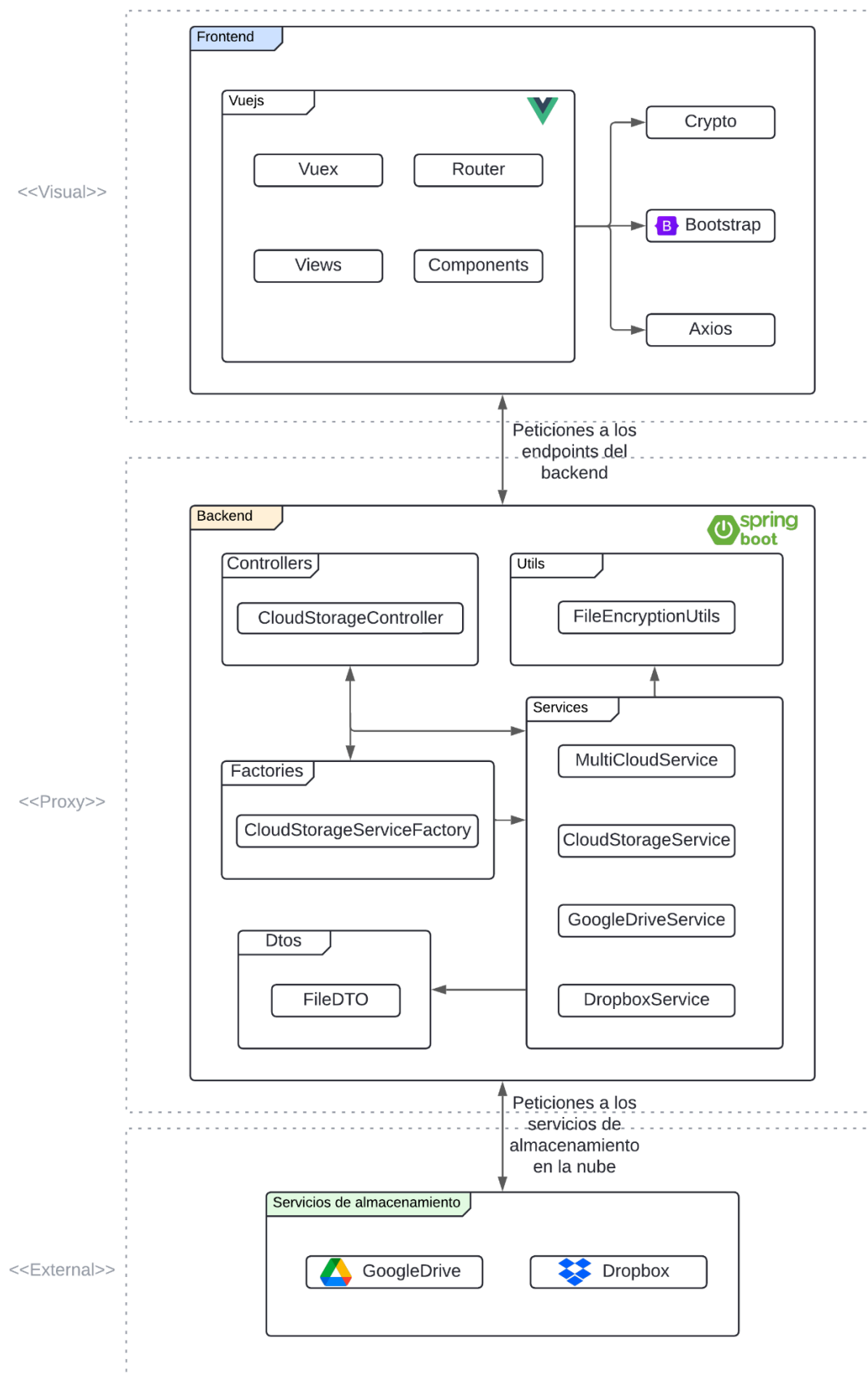


Ilustración 26. Diagrama de paquetes

En la parte superior del diagrama, podemos encontrar el *frontend*. En dicho paquete vemos los mismos *frameworks* y tecnologías ya mostradas en el diagrama del sistema, pero organizados de manera más específica. El paquete Vuejs contiene subpaquetes que agrupan todo lo relacionado con las vistas, componentes, Vuex y Vue Router. Este paquete global utiliza tecnologías adicionales como Bootstrap, Crypto y Axios para proporcionar funciones específicas y estilos a la aplicación. Bootstrap se encarga de los estilos y componentes predefinidos, Crypto maneja operaciones de cifrado para mayor seguridad y Axios facilita la comunicación HTTP con el *backend*, como ya habías comentado en el apartado de arquitectura del sistema.

En un nivel inferior, encontramos el *backend*. El diagrama del sistema solo mostraba Spring Boot como el marco principal, reflejando que todo el *backend* está desarrollado utilizando esta tecnología. Sin embargo, en el diagrama de paquetes, vemos una organización más detallada. Los controladores, como el *CloudStorageController*, están agrupados en un paquete específico que maneja las solicitudes HTTP entrantes. Los servicios, incluyendo *MultiCloudService*, *CloudStorageService*, *GoogleDriveService* y *DropboxService*, están organizados en otro paquete y son responsables de la lógica de negocio y la interacción con los servicios de almacenamiento en la nube. Además, tenemos paquetes dedicados a fábricas (*Factories*), como *CloudStorageServiceFactory*, que gestionan la creación de instancias de servicios, y utilidades (*Utils*), como *FileEncryptionUtils*, que proporcionan funciones comunes de cifrado. Los objetos de transferencia de datos (DTOs), como *FileDTO*, también están encapsulados en su propio paquete, facilitando el intercambio de datos entre el *frontend* y el *backend*.

Como último nivel, podemos encontrar el de Servicios Externos. La aplicación se integra con servicios de almacenamiento en la nube como Google Drive y Dropbox. Para Google Drive, se utilizan las bibliotecas como *google-api-client* y *google-api-services-drive*, que permiten realizar operaciones como subida, descarga y gestión de archivos. Para Dropbox, se emplea la biblioteca *dropbox-core-sdk*, proporcionando funcionalidades similares para la gestión de archivos. Estos servicios externos se encargan de almacenar y gestionar los archivos en la nube, garantizando la transferencia segura y eficiente de datos.

4.2.- Detalles sobre implementación

Para la implementación del proyecto, se han utilizado una amplia variedad de *frameworks* y tecnologías existentes. En esta sección se detalla el desarrollo tanto del *frontend* como del *backend* del proyecto, abarcando las herramientas y bibliotecas específicas empleadas en cada caso, así como ejemplos de código relevantes para ilustrar su integración y funcionalidad.

4.2.1.- Implementación del frontend

En el caso del *frontend*, ha sido desarrollado utilizando Vue.js como *framework* principal, acompañado de Vuex para la gestión del estado global de la aplicación. Vue.js en su versión 3 ha sido empleado para aprovechar sus características más recientes, lo cual facilita la reutilización de código y la modularidad. Vuex, en su versión 4, asegura una gestión centralizada y predecible del estado, crucial para la coherencia del estado a lo largo de la aplicación.

Aunque Vue 3 también soporta Pinia como una alternativa moderna a Vuex para la gestión del estado, se ha optado por utilizar Vuex por varias razones. En primer lugar, Vuex ha estado en el ecosistema de Vue durante mucho más tiempo, lo cual significa que es una tecnología probada y ampliamente adoptada con una gran cantidad de recursos y documentación disponible. Esta estabilidad y madurez resultan beneficiosas para el proyecto. Además, Vuex 4 está completamente compatible con Vue 3, lo cual facilita la transición y el uso de las nuevas características de Vue 3 sin necesidad de cambiar el enfoque de gestión del estado.

Para las peticiones HTTP, se ha utilizado Axios, una biblioteca que simplifica las operaciones asincrónicas con el *backend*, asegurando una comunicación eficiente y manejable. La versión 1.6.8 de Axios ha sido seleccionada por su estabilidad y amplia adopción en la comunidad. Además, para la presentación de notificaciones y mensajes al usuario, se ha integrado Vue Toast en su versión más reciente 3.0.6, permitiendo alertas visuales atractivas y no intrusivas.

En cuanto a los estilos y diseño, Bootstrap 5 ha sido la elección principal, proporcionando una base sólida para un diseño receptivo y moderno. Bootstrap facilita la creación de interfaces de usuario atractivas y consistentes con una mínima personalización, lo cual acelera el desarrollo. Además, se ha añadido Popper.js para poder mostrar *tooltips* y *popovers*, mejorando la experiencia de usuario y reduciendo el tiempo de desarrollo.

Para cifrar las contraseñas del lado del cliente, se ha utilizado la biblioteca crypto.js en su versión 4.2, la cuál se encarga de cifrar la contraseña introducida en texto claro por el usuario a un hash, tanto para la subida de archivos como para la descarga. El algoritmo aplicado está basado en SHA-256.

La gestión de dependencias se maneja mediante el archivo *package.json*, utilizando *npm* (Node Package Manager). *package.json* contiene todas las dependencias requeridas para el *frontend*, incluyendo bibliotecas de JavaScript y *frameworks* como Vue.js, Axios, Bootstrap y otras herramientas esenciales para el desarrollo y el funcionamiento del *frontend*. *npm* facilita la instalación y actualización de estas dependencias, asegurando que el entorno de desarrollo sea reproducible y

consistente. El listado completo del archivo ubicado dentro de la jerarquía del proyecto *frontend* es el siguiente :

JavaScript

```
"dependencies": {  
  "@popperjs/core": "^2.11.8",  
  "@vuesimple/vs-toast": "^3.0.6",  
  "axios": "^1.6.8",  
  "bootstrap": "^5.3.3",  
  "core-js": "^3.8.3",  
  "crypto-js": "^4.2.0",  
  "popper.js": "^1.16.1",  
  "vue": "^3.2.13",  
  "vue-router": "^4.3.0",  
  "vuex": "^4.1.0"  
},
```

4.2.2.- Implementación del backend

El backend se ha desarrollado con Spring Boot en su última versión 3.2.2, permitiendo aprovechar sus capacidades para construir aplicaciones robustas y de alto rendimiento con una configuración mínima. Spring Boot facilita la integración de varios componentes y servicios, manteniendo una arquitectura limpia y modular. Además, se ha utilizado el kit de desarrollo de java (*JDK*) en su versión 19. En este caso no se ha utilizado la última versión debido a que en las últimas versiones había ciertas incompatibilidades con las APIs de los servicios de almacenamiento en la nube.

La autenticación y la autorización se gestionan mediante el protocolo *OAuth 2.0*, el cuál se ha implementado manualmente, para tener un mayor control sobre el proceso de autenticación en cada nube. Además, gracias al ámbito o *Scope* que ofrece Spring Boot, se puede hacer que cada servicio tenga un *SCOPE_SESSION*, lo cuál permite tener una instancia única de ese servicio por cada sesión activa del usuario. Por esta razón resultó más sencillo realizar el cierre de sesión del usuario que varió la planificación inicial de tareas que ya comentamos, debido a que simplemente había que eliminar la instancia de ese servicio.

4.2.2.1.- Acceso a Google Drive

Para la integración con servicios de almacenamiento en la nube, se han utilizado las APIs de Google Drive y Dropbox. La interacción con Google Drive, implementada en el servicio de *GoogleDriveService*, se maneja a través de dos bibliotecas principales.

Primero, la biblioteca oficial de Google API Client¹⁴ (*com.google.api-client:google-api-client*), versión 2.4.0, proporciona las herramientas básicas y las clases necesarias para la autenticación, el transporte HTTP, y la serialización/deserialización de JSON. Esta biblioteca incluye componentes como *GoogleCredential* para la autenticación OAuth 2.0, *HttpTransport* para el manejo de las solicitudes HTTP y *JsonFactory* para la manipulación de datos JSON.

Además, se utiliza la biblioteca específica de Google Drive API¹⁵ (*com.google.apis:google-api-services-drive*), versión v3-rev20240327-2.0.0. Esta biblioteca contiene clases y métodos específicos para interactuar con la Google Drive API v3, como la clase *Drive*, que permite la gestión de archivos y carpetas, y métodos específicos para operaciones como *files().list()*, *files().get()*, *files().create()*, entre otros. Estas dos bibliotecas combinadas permiten realizar operaciones como subir, descargar, mover y gestionar archivos en Google Drive de manera segura y eficiente.

Java

```
public void authenticateUser(String code) throws Exception {
    GoogleTokenResponse response = flow.newTokenRequest(code)
        .setRedirectUri(redirectUri).execute();
    Credential cred = flow.createAndStoreCredential(response, null);

    drive = new Drive.Builder(HTTP_TRANSPORT, JSON_FACTORY, cred)
        .setApplicationName("MyDrive").build();
}
```

En el código anterior podemos ver cómo es el proceso para, una vez obtenido el código del protocolo OAuth 2.0, obtener el token de acceso para autenticar al usuario y crear una instancia del servicio de Google Drive para ese usuario.

Java

```
public List<FileDTO> getFoldersInFolder(String folderId, String folderName)
    throws Exception {

    String query = "'" + folderId
        + "' in parents and trashed=false and " +
        "mimeType='application/vnd.google-apps.folder'";
```

¹⁴ Fuente: <https://developers.google.com/api-client-library/java?hl=es-419>

¹⁵

Fuente:

<https://developers.google.com/resources/api-libraries/documentation/drive/v3/java/latest/com/google/api/services/drive/Drive.html>

```

// Agregar filtro por nombre de carpeta si se proporciona
if (!folderName.isEmpty()) {
    query += " and name contains '" + folderName + "'";
}

// Consulta para obtener todas las subcarpetas en esta carpeta
FileList allFolders = drive.files().list().setQ(query)
    .setFields("files(id,name,thumbnailLink,mimeType,viewedByMeTime," +
        "modifiedByMeTime,createdTime,size,description)")
    .execute();

List<FileDTO> foldersDTOList = new ArrayList<>();

// Modificamos los parámetros antes de convertirlo en DTO

for (File folder : allFolders.getFiles()) {
    folder.setSize(0L);
    foldersDTOList.add(convertToFileDTO(folder));
}

return foldersDTOList;
}

```

El método *getFoldersInFolder* se utiliza para obtener una lista de carpetas dentro de una carpeta específica en Google Drive. Primero, se construye una consulta que filtra por el *folderId* proporcionado, asegurando que solo se incluyan carpetas no eliminadas (*trashed=false*) y de tipo *application/vnd.google-apps.folder*. Si se proporciona un nombre de carpeta (*folderName*), se añade un filtro adicional para buscar carpetas cuyo nombre contenga dicho valor. Luego, se ejecuta la consulta utilizando el método *list()* de *drive.files()*, especificando los campos deseados en los resultados, como *id*, *name*, *thumbnailLink*, *mimeType*, *viewedByMeTime*, *modifiedByMeTime*, *createdTime*, *size*, y *description*. Posteriormente, se inicializa una lista para almacenar los objetos *FileDTO* que representarán las carpetas encontradas. Para cada carpeta obtenida, se ajusta su tamaño a cero, ya que se considera que las carpetas tienen un peso de cero para no complicar la implementación y se convierte en un objeto *FileDTO* antes de agregarlo a la lista final, que es devuelta por el método.

4.2.2.2.- Acceso a Dropbox

La integración con Dropbox, implementada en el servicio de *DropboxService*, se realiza mediante la biblioteca *Dropbox Core SDK*¹⁶ (*com.dropbox.core:dropbox-core-sdk*), versión 7.0.0. Esta biblioteca proporciona las

¹⁶ Fuente: <https://github.com/dropbox/dropbox-sdk-java>

herramientas necesarias para interactuar con la API de Dropbox, permitiendo realizar operaciones de gestión de archivos como subir, descargar, mover y eliminar archivos en Dropbox. La biblioteca Dropbox Core SDK incluye componentes para la autenticación OAuth, manejo de solicitudes HTTP y manejo de respuestas JSON, similar a las bibliotecas de Google, asegurando una integración robusta y eficiente con el servicio de almacenamiento en la nube de Dropbox.

Java

```
public void authenticateUser(String code) throws Exception {

    // Cambiar el code para obtener un token de acceso
    String tokenUrl = "https://www.dropbox.com/oauth2/token";
    RestTemplate restTemplate = new RestTemplate();
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);

    MultiValueMap<String, String> map = new LinkedMultiValueMap<>();
    map.add("code", code);
    map.add("grant_type", "authorization_code");
    map.add("redirect_uri", redirectUri);
    map.add("client_id", clientId);
    map.add("client_secret", clientSecret);

    HttpEntity<MultiValueMap<String, String>> request =
        new HttpEntity<>(map, headers);
    ResponseEntity<String> response =
        restTemplate.postForEntity(tokenUrl, request, String.class);

    if (response.getStatusCode().is2xxSuccessful()) {
        // Obtener del JSON el token de acceso
        ObjectMapper mapper = new ObjectMapper();
        JsonNode rootNode = mapper.readTree(response.getBody());
        String accessToken = rootNode.get("access_token").asText();

        // Usar el access token para autenticarse en Dropbox
        DbxRequestConfig config = new DbxRequestConfig("MyDrive");
        client = new DbxClientV2(config, accessToken);

        initializeTemplateId();
    }
}
```

Al igual que en Google Drive, este es el método que, a partir del código obtenido del protocolo OAuth 2.0, obtiene el *token* de acceso para posteriormente autenticar al usuario en Dropbox. En este caso, el proceso es mucho más elaborado. Primero, se construye una URL de *token* específica para Dropbox y se utiliza *RestTemplate* para enviar una solicitud HTTP POST a esta URL. La solicitud incluye varios parámetros

necesarios, como el código de autorización, el tipo de concesión, el URI de redirección, el ID del cliente y el secreto del cliente. Si la respuesta de Dropbox es exitosa, se analiza el JSON de la respuesta para extraer el *token* de acceso utilizando *ObjectMapper*. Luego, este *token* de acceso se utiliza para configurar el cliente de Dropbox con *DbxRequestConfig* y *DbxClientV2*, permitiendo así la autenticación del usuario en Dropbox. Finalmente, se inicializa el ID de la plantilla con *initializeTemplateId*, ya que es necesario para realizar ciertas operaciones.

Comparado con el método *authenticateUser* de Google Drive, que simplemente crea y almacena credenciales utilizando la biblioteca proporcionada por Google, el método de Dropbox requiere una gestión manual de las solicitudes HTTP, análisis de respuestas JSON y configuración del cliente. Esto añade complejidad adicional al proceso de autenticación para Dropbox en comparación con Google Drive.

```
Java
public List<FileDTO> getFoldersInFolder(String folderId,
                                       String folderName) throws Exception {
    if (folderId.equals("root"))
        folderId = ""; // root para dropbox es cadena vacía

    // Realizar la consulta para obtener todas las carpetas en esta carpeta
    ListFolderResult result = client.files().listFolder(folderId);

    List<FileDTO> foldersDTOList = new ArrayList<>();

    // Recorrer todas las páginas ignorando la paginación
    while (true) {
        for (Metadata metadata : result.getEntries()) {
            if (metadata instanceof FolderMetadata &&
                metadata.getName().contains(folderName)) {
                FolderMetadata folderMetadata = (FolderMetadata) metadata;
                FileDTO folderDTO = convertToFileDTO(folderMetadata);
                foldersDTOList.add(folderDTO);
            }
        }

        if (!result.hasMore()) break;
        result = client.files().listFolderContinue(result.getCursor());
    }

    return foldersDTOList;
}
```

El método para obtener las carpetas dentro de una carpeta en Dropbox también tiene sus particularidades y una estructura más elaborada en comparación con su equivalente en Google Drive.

En Google Drive, el método *getFoldersInFolder* construye una consulta específica utilizando el formato de consulta de Google Drive API, estableciendo criterios como el *folderId*, el estado de *trashed* (papelera) y el *mimeType* para identificar carpetas. Luego, se ejecuta esta consulta y se procesa el resultado para convertir cada carpeta en un objeto *FileDTO*.

En Dropbox, el método también obtiene una lista de carpetas dentro de una carpeta, pero el proceso es diferente debido a la estructura de la API de Dropbox. Aquí, se utiliza el método *listFolder* del cliente de Dropbox para obtener los elementos dentro de una carpeta. Dropbox devuelve los resultados en páginas, por lo que el método debe manejar la paginación utilizando un bucle *while* para iterar sobre todas las páginas de resultados. Cada entrada en el resultado se verifica para asegurarse de que es una carpeta (*FolderMetadata*), y luego se convierte en un objeto *FileDTO*.

La diferencia principal radica en cómo cada API maneja la estructura de carpetas y la paginación. Google Drive permite una consulta más directa y específica utilizando su lenguaje de consulta, mientras que Dropbox requiere un manejo manual de la paginación y verificación del tipo de entrada. Esto provoca que en Dropbox tengamos que obtener muchos más archivos y filtrarlos específicamente en el *backend* de nuestra aplicación, pudiendo ralentizar ciertas peticiones. Además, en Dropbox, el *folderId* raíz se representa como una cadena vacía, mientras que en Google Drive se representa como *root*, lo cual es otra diferencia significativa a tener en cuenta. Estas diferencias reflejan las particularidades de cada servicio de almacenamiento en la nube y la necesidad de adaptar el código para trabajar con sus respectivas API.

Un aspecto técnico crucial en el desarrollo de la aplicación es la configuración de dependencias y versiones necesarias para garantizar un entorno coherente y estable. La gestión adecuada de estas dependencias es esencial para evitar conflictos y asegurar que todas las bibliotecas funcionen correctamente entre sí. En el *backend*, hemos utilizado Maven como herramienta de gestión de dependencias. Maven no solo permite manejar las bibliotecas y sus versiones de manera eficiente, sino que también proporciona una estructura estándar para el proyecto y facilita la automatización de tareas comunes como la compilación y el empaquetado.

El archivo *pom.xml* en el *backend* es el núcleo de la configuración de Maven, donde se definen todas las dependencias necesarias para el proyecto, incluyendo las versiones específicas de las bibliotecas. Esto asegura la compatibilidad entre las diferentes bibliotecas y facilita las actualizaciones, ya que Maven puede resolver automáticamente las dependencias transitivas y garantizar que todas las versiones sean compatibles entre sí. Además, el uso de *Maven Central Repository* proporciona acceso a un amplio conjunto de bibliotecas bien mantenidas y probadas, lo que contribuye a la estabilidad y seguridad del proyecto.

4.2.2.3.- Cifrado de archivos

En términos de cifrado y seguridad de datos, se ha implementado la clase *FileEncryptionUtil*, esta clase utiliza las utilidades de cifrado personalizadas de las bibliotecas de *Java Cryptography Architecture*¹⁷ (JCA). Este enfoque asegura que los archivos sensibles sean cifrados antes de ser almacenados en la nube, protegiendo la información contra accesos no autorizados. Esta utilidad realiza dos operaciones principales: cifrado y descifrado de archivos. Durante el cifrado, se genera una sal y un vector de inicialización (IV) aleatorios, que se utilizan junto con una contraseña proporcionada por el usuario para derivar una clave secreta mediante el algoritmo PBKDF2 con HMAC-SHA256. Esta clave se utiliza para configurar el cifrador en modo de cifrado AES con padding PKCS5 y modo de operación CBC. El archivo cifrado resultante contiene la sal y el IV, seguidos por los datos cifrados del archivo original. Durante el descifrado, se leen la sal y el IV del archivo cifrado, se deriva la clave secreta con la misma contraseña y se configura el descifrador para revertir el proceso de cifrado, recuperando así los datos originales. De esta manera se puede prescindir de la necesidad de una base de datos para almacenar la contraseña de cada archivo y reducir el esfuerzo necesario para completar dicha funcionalidad.

Java

```
public static void encryptFile(File inputFile, File outputFile,
                               String password) throws Exception {

    // Generate salt and IV
    byte[] salt = new byte[SALT_SIZE];
    byte[] iv = new byte[IV_SIZE];
    SecureRandom random = new SecureRandom();
    random.nextBytes(salt);
    random.nextBytes(iv);

    // Derive key
    SecretKey secretKey = deriveKey(password, salt);

    // Configure the cipher
    Cipher cipher = Cipher.getInstance(TRANSFORMATION);
    cipher.init(Cipher.ENCRYPT_MODE, secretKey, new IvParameterSpec(iv));

    // Write salt and IV to the output file
    try (FileOutputStream fos = new FileOutputStream(outputFile);
         CipherOutputStream cos = new CipherOutputStream(fos, cipher);
         FileInputStream fis = new FileInputStream(inputFile)) {

        fos.write(salt);
```

¹⁷ Fuente: <https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>

```
        fos.write(iv);

        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = fis.read(buffer)) != -1) {
            cos.write(buffer, 0, bytesRead);
        }
    }
}
```

Para ilustrar esta funcionalidad de cifrado, se ha presentado el método *encryptFile*, el cual toma un archivo de entrada (*inputFile*), un archivo de salida (*outputFile*) y una contraseña (*password*) como parámetros. Este método se encarga de cifrar el archivo de entrada y guardar el archivo cifrado resultante en el archivo de salida. Primero, se generan valores aleatorios de sal y IV, que son necesarios para el proceso de cifrado. Luego, utilizando la contraseña proporcionada y la sal generada, se deriva una clave secreta mediante el método *deriveKey*. Esta clave es utilizada para configurar el cifrador con el algoritmo AES en modo CBC con padding PKCS5.

Posteriormente, se escribe la sal y el IV en el archivo de salida antes de comenzar a escribir los datos cifrados. Esto es necesario para que el proceso de descifrado pueda reconstruir la clave secreta más adelante. Finalmente, se lee el archivo de entrada en bloques y se escriben los datos cifrados en el archivo de salida mediante un flujo de cifrado *CipherOutputStream*.

5.- Pruebas

En la fase de pruebas de este proyecto, se ha empleado una combinación de herramientas y enfoques tanto para el *backend* como para el *frontend*, con el objetivo de asegurar que cada componente del sistema funcione correctamente y que las interacciones entre ellos se desarrollen sin problemas.

5.1.- Pruebas realizadas en el backend

En el *backend*, se ha optado por utilizar *JUnit* como marco de pruebas principal. *JUnit* es ampliamente reconocido en el ecosistema Java por su robustez y flexibilidad, lo que lo convierte en una herramienta ideal para realizar pruebas unitarias y de integración. Durante las pruebas unitarias, se centró en validar la funcionalidad de los servicios que interactúan con Google Drive y Dropbox, así como la fábrica que permite obtener el servicio de almacenamiento correspondiente en función de los parámetros proporcionados. Cada uno de estos componentes es crítico para la correcta operación del sistema, ya que permiten gestionar la conexión y operación con distintos servicios de almacenamiento en la nube.

Un aspecto importante a destacar es el uso de objetos *Mock* durante estas pruebas. Los objetos *Mock* son esenciales en las pruebas unitarias cuando es necesario aislar el componente que se está probando, evitando dependencias externas que puedan influir en los resultados de la prueba. Por ejemplo, en este caso, el servicio de Google Drive y el de Dropbox tienen dependencias en la API externa de cada plataforma. Utilizar objetos *Mock* permite simular las respuestas de estas APIs sin necesidad de realizar llamadas reales, lo que no sólo acelera las pruebas, sino que también permite controlar y prever todos los escenarios posibles, incluyendo aquellos en los que la API pueda responder con errores o datos inesperados.

Otra razón clave para el uso de objetos *Mock*¹⁸, es la necesidad de evitar depender de datos específicos almacenados en cuentas personales durante las pruebas. Si los tests se basaran en la obtención de datos de una cuenta personal, es probable que estos datos no estén disponibles si otro desarrollador intenta ejecutar las pruebas en su entorno. Para garantizar la consistencia y fiabilidad de las pruebas, sería necesario contar con cuentas especiales destinadas exclusivamente a la realización de pruebas, lo cual podría complicar el proceso y agregar una capa adicional de complejidad al desarrollo.

Además de las pruebas unitarias, se han llevado a cabo pruebas de integración utilizando *MockMvc*¹⁹. *MockMvc* es una herramienta poderosa para probar

¹⁸ Fuente: <https://github.com/mockito/mockito>

¹⁹ Fuente: <https://spring.io/guides/gs/testing-web>

controladores en aplicaciones Spring, ya que permite simular peticiones HTTP y verificar cómo se comportan los *endpoints* sin necesidad de desplegar la aplicación en un servidor real. Esto resulta especialmente útil para validar que las interacciones entre los controladores y el resto de la lógica de la aplicación se realizan de manera adecuada. En este proyecto, *MockMvc* ha sido crucial para verificar que los *endpoints* que gestionan las operaciones de almacenamiento en la nube responden correctamente, manejan los errores de forma apropiada y devuelven las respuestas esperadas en diferentes escenarios.

Este enfoque ha permitido realizar pruebas exhaustivas sin depender de un entorno de producción real, asegurando que el *backend* funcione de manera robusta y fiable en la mayoría de las condiciones. En total, se han implementado 17 pruebas automatizadas para el *backend*, que abarcan diversas funcionalidades clave y validan tanto las pruebas unitarias como las de integración. Los detalles de estas pruebas se presentarán en los apartados siguientes.

5.1.1.- Pruebas unitarias realizadas al servicio de google drive

En esta sección se presentan las pruebas unitarias realizadas al servicio de Google Drive. Estas pruebas tienen como objetivo asegurar que los métodos implementados funcionen correctamente bajo diferentes condiciones, incluidas tanto situaciones normales como casos de error. A continuación, se muestra un resumen de las pruebas realizadas, especificando el método probado, los datos de entrada utilizados y los resultados esperados.

Servicio	Método probado	Datos de entrada	Resultados Esperados
Google Drive	getFoldersInFolder(String, String)	ID de la carpeta raíz ("root"), filtro vacío ("")	Lista con dos carpetas (folder1, folder2) con detalles correctos (nombre, ID, etc.)
Google Drive	getFoldersInFolder(String, String)	ID de la carpeta raíz ("root"), filtro vacío ("")	Lanzamiento de excepción RuntimeException con el mensaje "API Error"
Google Drive	getRecentFiles(String, String)	Fecha máxima, filtro vacío ("")	Lista con un archivo reciente (testFile)
Google Drive	searchFiles(String)	Término de búsqueda ("searchFile")	Lista con dos archivos encontrados (searchFile1, searchFile2)
Google Drive	downloadEncryptedFile(String, String)	ID del archivo ("testFileId"), contraseña ("testPassword")	Archivo descargado y descifrado correctamente ("testContent")

Tabla 52. Tests unitarios para el servicio de Google Drive

Las pruebas unitarias realizadas se han centrado en asegurar que los métodos clave del servicio de Google Drive funcionen correctamente, con especial atención a aquellos que manejan la obtención y manipulación de datos más allá de una simple confirmación de éxito. Estas pruebas son cruciales porque evalúan cómo el servicio interactúa con datos reales, como la obtención de carpetas, la búsqueda de archivos, y la descarga de archivos cifrados, lo que es esencial para el correcto funcionamiento de la aplicación.

Aunque la mayoría de los métodos del servicio comparten similitudes en su lógica interna, se ha optado por probar solo algunos de ellos para garantizar una cobertura eficiente sin redundancia. Esto ha permitido validar la funcionalidad general del servicio sin la necesidad de probar cada método de manera exhaustiva.

5.1.2.- Pruebas unitarias realizadas al servicio de dropbox

En esta sección se presentan las pruebas unitarias realizadas para el servicio de Dropbox. Al igual que con el servicio de Google Drive, el foco de las pruebas ha sido verificar que las funcionalidades críticas, especialmente aquellas que devuelven datos importantes como archivos y carpetas, funcionen correctamente. Debido a la similitud entre los métodos implementados en el servicio, se ha optado por probar solo un conjunto representativo de ellos, centrándose en los que manipulan datos significativos.

Servicio	Método probado	Datos de entrada	Resultados Esperados
Dropbox	getFoldersInFolder(String, String)	ID de la carpeta raíz ("root"), filtro vacío ("")	Lista con dos carpetas (folder1, folder2) con detalles correctos (nombre, ID, etc.)
Dropbox	getFoldersInFolder(String, String)	ID de la carpeta raíz ("root"), filtro vacío ("")	Lanzamiento de excepción RuntimeException con el mensaje "API Error"
Dropbox	getRecentFiles(String, String)	Fecha máxima, filtro vacío ("")	Lista con un archivo reciente (recentFile)
Dropbox	searchFiles(String)	Término de búsqueda ("searchFile")	Lista con dos archivos encontrados (searchFile1, searchFile2)
Dropbox	downloadEncryptedFile(String, String)	ID del archivo ("testFileId"), contraseña ("testPassword")	Archivo descargado y descifrado correctamente ("testContent")
Dropbox	deleteFile(String)	ID del archivo ("some-file-id")	Lanzamiento de excepción <i>CloudLimitationException</i> con el mensaje "No se puede eliminar el archivo debido a limitaciones en la nube"

Tabla 53. Tests unitarios para el servicio de Dropbox

Las pruebas unitarias realizadas sobre el servicio de Dropbox han seguido un enfoque similar al utilizado para el servicio de Google Drive. Se han probado los mismos métodos clave que en Google Drive, como la recuperación de carpetas, la búsqueda de archivos y la descarga de archivos cifrados, para asegurar que el servicio de Dropbox maneja correctamente estas operaciones y excepciones.

Adicionalmente, en Dropbox se probó el método *deleteFile*, el cual no está disponible en el servicio debido a limitaciones inherentes. Por ello, se ha verificado que este método lance una excepción personalizada (*CloudLimitationException*). Esta prueba asegura que el sistema maneja adecuadamente los casos en los que una funcionalidad no es compatible, proporcionando una respuesta clara y manejable frente a estas limitaciones.

5.1.3.- Pruebas unitarias realizadas a la factoría

Las pruebas unitarias realizadas a la *CloudStorageServiceFactory* verifican que la factoría se comporta correctamente al instanciar y proporcionar los servicios de almacenamiento en la nube correspondientes. Los siguientes tests aseguran que la factoría devuelve los servicios adecuados para los identificadores especificados, utilizando *mocks* para los servicios de Google Drive y Dropbox.

Servicio	Método probado	Datos de entrada	Resultados Esperados
CloudStorageServiceFactory	getCloudService(String) "google-drive"	Identificador "google-drive"	Instancia de googleDriveService devuelta
CloudStorageServiceFactory	getCloudService(String) "dropbox"	Identificador "dropbox"	Instancia de dropboxService devuelta

Tabla 54. Tests unitarios para la factoría de la aplicación

5.1.4.- Pruebas de integración realizadas

Las pruebas de integración verifican que los componentes del sistema interactúan correctamente entre sí y que las respuestas de la API cumplen con las expectativas en escenarios de uso real. En este apartado, se han realizado pruebas para los servicios de Google Drive y Dropbox, así como para el manejo de excepciones. A continuación, se detallan las pruebas realizadas:

Servicio	Escenario de prueba	Peticiones	Resultados Esperados
Google Drive	Obtener carpetas	Solicitud GET a <code>/api/google-drive/folders/root</code>	Respuesta con estado HTTP 200 OK y lista de carpetas en formato JSON
Google Drive	Manejar excepción al obtener archivos	Solicitud GET a <code>/api/google-drive/files/root</code>	Respuesta con estado HTTP 500 <i>INTERNAL SERVER ERROR</i> y mensaje de error
Dropbox	Obtener carpetas	Solicitud GET a <code>/api/dropbox/folders/root</code>	Respuesta con estado HTTP 200 OK y lista de carpetas en formato JSON
Dropbox	Manejar excepción al intentar eliminar un archivo	Solicitud DELETE a <code>/api/dropbox/delete/{fileId}</code>	Respuesta con estado HTTP 409 <i>CONFLICT</i> y mensaje de limitación en la nube

Tabla 55. Tests de integración para el backend de la aplicación

En las pruebas de integración realizadas, se simulan los datos que devolverán los métodos del servicio, los cuales ya fueron probados previamente mediante pruebas unitarias. Esto incluye la simulación de listas de objetos *FileDTO*, que representan carpetas y archivos. Al hacerlo, se asegura que las respuestas de los métodos del servicio se comporten como se espera cuando son integradas con el resto del sistema.

Al simular las respuestas, se pueden verificar aspectos clave del flujo de datos, desde la recuperación de información en los servicios hasta su transformación en formato JSON y su envío al cliente. En este caso, se utiliza *mockMvc* para realizar solicitudes HTTP a las APIs y comprobar que el contenido de las respuestas se ajusta a las expectativas definidas. Esto permite validar que los datos simulados se procesan correctamente y que el sistema responde adecuadamente tanto en situaciones normales como en escenarios de error.

Así, estas pruebas de integración no solo confirman que el sistema funciona correctamente con los datos simulados, sino que también aseguran que la integración completa, se comporta según lo esperado.

5.2.- Pruebas realizadas en el frontend

En el *frontend*, se ha utilizado *Jest*, un *framework* de pruebas muy popular en el entorno de desarrollo con Vue.js, para verificar el comportamiento de las funcionalidades principales de la interfaz de usuario. Al igual que en el *backend*, fue necesario implementar mocks, en este caso para simular las respuestas que *Axios*, la biblioteca utilizada para realizar peticiones HTTP al *backend*, recibiría. Este enfoque fue fundamental para probar la lógica de la aplicación en el *frontend* sin

depender del *backend* real, lo que permitió desarrollar y verificar las funcionalidades de forma aislada.

Debido a que en el *frontend* existen muchos métodos dedicados exclusivamente a la visualización de elementos, como *pop ups*, menús y otros aspectos puramente visuales, estos no fueron objeto de pruebas exhaustivas. La mayor parte del código del *frontend* se enfoca en la gestión de estos elementos visuales, los cuales no contienen lógica de negocio significativa que afecte el comportamiento de la aplicación. En cambio, los tests unitarios se dirigieron a los componentes que sí gestionan y modifican elementos cruciales en el *frontend*, como *FolderList.vue* y *FileList.vue*, que contienen lógica de negocio importante para la funcionalidad de la aplicación. Es por esta razón que, sólo se implementaron 6 pruebas en total para el *frontend*.

A continuación, se presentan las pruebas realizadas para estos componentes específicos, que abarcan la lógica que maneja la interacción con el usuario y la modificación de los datos clave en la interfaz.

5.2.1.- Pruebas unitarias realizadas a *FolderList.vue*

Para probar el componente *FolderList.vue*, se empleó *Jest* junto con *Vue Test Utils*²⁰ y *axios-mock-adapter*²¹ para simular las respuestas de la API. Estas pruebas se enfocan en verificar la correcta funcionalidad de la lógica del componente, asegurando que la integración con el *backend* se maneje adecuadamente y que el estado del componente se actualice en consecuencia.

1. **Test de obtención de carpetas:** En esta prueba se verifica que el componente *FolderList* sea capaz de recuperar y actualizar la lista de carpetas desde el *backend* correctamente. Se simula una respuesta exitosa de la API utilizando *MockAdapter*, devolviendo una lista de carpetas mockeadas. El componente llama al método *getFolders*, que hace una solicitud HTTP para obtener las carpetas. Luego, se asegura que el *array folders* del componente se actualice correctamente con la respuesta de la API. La prueba verifica que el componente maneja la respuesta del *backend* y actualiza su estado adecuadamente.
2. **Test de manejo de errores al obtener carpetas:** Esta prueba asegura que el componente maneje correctamente los errores en la solicitud al *backend*. Se configura *MockAdapter* para simular una respuesta de error (código 500) cuando se intenta recuperar las carpetas. Se llama al método *getFolders* del componente y se maneja el error, asegurándose de que la lista de carpetas se

²⁰ Fuente: <https://test-utils.vuejs.org/>

²¹ Fuente: <https://www.npmjs.com/package/axios-mock-adapter>

mantenga vacía en caso de una falla en la solicitud. Además, se simula la supresión de errores en la consola para evitar mensajes de error no deseados durante la prueba.

5.2.2.- Pruebas unitarias realizadas a *FileList.vue*

Para el componente *FileList.vue*, se han implementado pruebas unitarias utilizando *Jest* y *Vue Test Utils* para verificar el comportamiento del componente y su interacción con la API. Las pruebas se centran en asegurar que el componente maneje correctamente la obtención, visualización y ordenación de archivos, así como la descarga de archivos.

1. **Test de obtención de archivos:** Esta prueba verifica que el componente *FileList* sea capaz de recuperar y actualizar la lista de archivos desde el *backend*. Se utiliza *MockAdapter* para simular una respuesta exitosa de la API, proporcionando una lista de archivos *mockeados*. Tras invocar el método *getFiles*, la prueba asegura que el array *files* del componente se actualice con los archivos recibidos de la API.
2. **Test de manejo de errores al obtener archivos:** Aquí se prueba la capacidad del componente para manejar errores en la solicitud a la API. Se configura *MockAdapter* para simular una respuesta de error (código 500) cuando se intenta obtener los archivos. Tras llamar al método *getFiles*, la prueba verifica que el array *files* permanezca vacío en caso de error, asegurando que el componente maneje adecuadamente la situación y no muestre archivos no deseados.
3. **Test de ordenación de archivos:** Este test valida que el componente ordene correctamente los archivos por nombre. Se configuran datos *mockeados* para representar archivos con nombres desordenados. Después de asignar estos archivos al estado del componente, se llama al método *sortFiles* para ordenar los archivos. La prueba asegura que la lista de archivos se ordene correctamente por nombre, verificando que el componente maneje la lógica de ordenación como se espera.
4. **Test de descarga de archivos:** En esta prueba se verifica la funcionalidad de descarga de archivos. Se simula una solicitud de descarga que devuelve un *blob* de datos. Se *mockea* la creación de un objeto URL para el *blob* y se asegura que el método *downloadFile* del componente haga la solicitud HTTP correcta y maneje la respuesta adecuadamente. También se comprueba que se muestre una notificación de éxito utilizando *VsToast* después de completar la descarga.

Estas pruebas cubren los aspectos críticos del componente *FileList*, validando que maneje adecuadamente la obtención, visualización, ordenación y descarga de archivos. A través del uso de *mocks* y *stubs*, se asegura que el componente interactúe correctamente con las dependencias externas y que su lógica interna funcione como se espera en diferentes escenarios.

5.3.- Pruebas con herramientas externas

En esta sección, se describen las pruebas realizadas utilizando herramientas externas para validar el correcto funcionamiento de los *endpoints* del *backend* en un entorno autenticado. A diferencia de las pruebas unitarias y de integración, que se realizaron utilizando *mocks* para simular el comportamiento de los componentes del sistema, estas pruebas requieren una autenticación real en el servicio y se enfocan en validar la interacción completa entre el cliente y el servidor.

Para llevar a cabo estas pruebas, aunque es común el uso de herramientas como Postman, en este proyecto se ha optado por utilizar el Swagger UI proporcionado por OpenAPI. Swagger UI, que ya se había mencionado en el apartado de diseño de la interfaz, facilita la documentación y prueba de los endpoints del controlador de la API REST. Esta herramienta permite ejecutar solicitudes HTTP autenticadas directamente desde el navegador y comprobar que los datos que devuelve el *backend* son correctos y cumplen con las especificaciones definidas.

Uno de los casos de uso probados fue la obtención de todas las carpetas del directorio raíz de Dropbox. Obviamente para saber si los datos devueltos son correctos debemos conocer qué carpetas tenemos en dicho directorio, junto con sus metadatos. Todos los endpoints han sido probados, pero solo voy a enseñar el resultado en JSON de la respuesta de esta operación para no alargar mucho la memoria, ya que el resto de métodos, devuelven datos similares o solo un mensaje de éxito. Para ello, se utilizó el endpoint correspondiente y se validó que la respuesta JSON contenía la información correcta.

JavaScript

```
[
  {
    "id": "id:ebTx7PZsaqwAAAAAAAABg",
    "name": "a405949e2d53201e.png",
    "thumbnailLink": null,
    "lastTimeViewed": "2024-05-15T11:20:24Z",
    "size": 1076,
    "encrypted": false
  },
  {
```

```
"id": "id:ebTx7PZsaqwAAAAAAAAABw",  
"name": "DeltaV_KSP.png",  
"thumbnailLink": null,  
"lastTimeViewed": "2024-05-17T18:47:46Z",  
"size": 1830368,  
"encrypted": false  
}  
]
```

6.- Conclusiones

Al mirar atrás en el desarrollo de *MyDrive*, puedo afirmar con confianza que los objetivos iniciales han sido alcanzados con éxito, aunque no sin desafíos en el camino. Desde el principio, me propuse realizar un estudio exhaustivo de las necesidades y soluciones en el ámbito de la gestión unificada de almacenamiento en la nube. Este estudio fue fundamental para entender las carencias y expectativas de los usuarios en un entorno donde múltiples servicios de almacenamiento coexisten, pero rara vez se integran de manera eficiente. A partir de este análisis, desarrollé un sistema que no solo aborda estas necesidades, sino que también adopta principios de diseño y metodologías de desarrollo web robustas y escalables.

El diseño del sistema fue uno de los hitos más importantes del proyecto. Enfocado en arquitecturas modernas y principios de diseño como la modularidad y la reutilización de componentes, el sistema está orientado a facilitar su mantenimiento y futuras ampliaciones. Esta orientación hacia el diseño también se refleja en la elección de las tecnologías, donde opté por tecnologías modernas que incluye Vue.js para el *frontend* y Spring Boot para el *backend*. La combinación de estas tecnologías no solo permitió un desarrollo ágil y eficiente, sino que también asegura que el sistema esté preparado para adaptarse a cambios futuros, algo que considero esencial en un entorno tecnológico en constante evolución.

El proceso de selección del entorno de desarrollo fue crítico para el éxito del proyecto. La elección de tecnologías como Vue.js y Spring Boot no fue al azar. Estas herramientas fueron seleccionadas por su capacidad para manejar las demandas de una aplicación web compleja y por su amplia comunidad de soporte, lo que facilita la resolución de problemas y la integración de nuevas funcionalidades. Sin embargo, no todo fue perfecto. Uno de los desafíos más grandes fue manejar la integración entre diferentes APIs de servicios de almacenamiento en la nube, cada una con sus particularidades y limitaciones. Este obstáculo me llevó a aprender y aplicar estrategias para adaptar el código a todos los servicios integrados de manera transparente y profundizar en el manejo de errores, lo que al final resultó en una aplicación más robusta y fiable.

Desde un punto de vista personal, los aspectos más relevantes y fascinantes del proyecto fueron, sin duda, la integración de múltiples servicios en la nube y la implementación de una interfaz de usuario intuitiva y funcional. Integrar servicios como Google Drive y Dropbox en una única plataforma supuso un reto considerable, pero también me permitió profundizar en el manejo de APIs externas y en la implementación de protocolos más complejos y avanzados para la autorización por parte del usuario como OAuth 2.0. La interfaz de usuario, por otro lado, me brindó la oportunidad de explorar en profundidad Vue.js y de experimentar con el diseño de una UX que realmente facilite la vida del usuario.

Reflexionando sobre lo que este proyecto ha aportado a mis conocimientos adquiridos durante el Grado, puedo decir que ha sido una experiencia sumamente enriquecedora. No solo me ha permitido aplicar de manera práctica muchos de los conceptos teóricos que aprendí, sino que también me ha proporcionado nuevas habilidades y conocimientos, especialmente en el ámbito del desarrollo web y la integración de servicios en la nube. Además, me ha enseñado a enfrentar y superar desafíos complejos, a tomar decisiones en cuanto a la selección de tecnologías y a trabajar de manera metódica y estructurada para alcanzar los objetivos propuestos.

En conclusión, considero que *MyDrive* no solo ha cumplido con los objetivos iniciales, sino que ha superado mis expectativas en muchos aspectos. El enfoque propuesto, basado en una arquitectura moderna y el uso de tecnologías actuales, ha demostrado ser eficaz y adecuado para el desarrollo de una solución web que responde a las necesidades actuales del mercado. Si bien hubo desafíos, la experiencia adquirida y el resultado final del proyecto me han dejado una profunda satisfacción y una sólida base para futuros proyectos en el ámbito del desarrollo de software.

6.1.- Mejoras y trabajos futuros

El proyecto *MyDrive*, aunque ha alcanzado muchos de sus objetivos clave, aún tiene un largo camino por recorrer en términos de funcionalidades adicionales que podrían aportar un valor significativo a los usuarios. A lo largo del desarrollo, algunas historias de usuario con una prioridad más baja no se implementaron debido al tiempo de desarrollo que requerirían y el alcance del proyecto. Estas funcionalidades, aunque sean útiles, no se consideraron esenciales y son un punto de partida para futuras aplicaciones.

Entre estas funcionalidades se encuentran operaciones más avanzadas, como la búsqueda unificada entre todos los servicios de almacenamiento en la nube (HU24). Esta característica permitiría a los usuarios buscar archivos y carpetas en varias plataformas desde un único lugar, lo que aumentaría significativamente la eficiencia en la gestión de archivos dispersos en diferentes servicios.

Otra funcionalidad importante que quedó fuera del alcance inicial es la sincronización entre nubes, como la creación de carpetas espejo (HU23) que se replicarían automáticamente en diferentes servicios de almacenamiento. Esta característica sería extremadamente útil para los usuarios que necesitan mantener sus archivos sincronizados en varias plataformas, pero su implementación presenta desafíos significativos. Además de las consideraciones técnicas para garantizar que los archivos y carpetas se repliquen correctamente y en tiempo real, también habría que manejar los posibles conflictos de sincronización, errores en las API de terceros y asegurar una experiencia de usuario fluida.

Estas funcionalidades son solo algunas de las muchas que podrían implementarse para ampliar la experiencia del usuario al tener sus servicios de almacenamiento en la nube unificados en una sola aplicación.

También se descartaron funcionalidades relacionadas con la interacción entre usuarios, como la posibilidad de compartir archivos (HU22) o carpetas con otros usuarios. Estas características, que permitirían a los usuarios generar enlaces para compartir archivos o carpetas directamente desde *MyDrive*, implicaría no solo la implementación de una interfaz de usuario intuitiva, sino también una integración segura y eficiente con las APIs de los servicios de almacenamiento, para gestionar los permisos de acceso y la seguridad de los datos compartidos. Aunque es una funcionalidad valiosa, especialmente para usuarios que colaboran frecuentemente con otros, su implementación se dejó fuera para concentrar esfuerzos en desarrollar funcionalidades más centrales.

Finalmente, la ampliación más interesante sería añadir otros servicios de almacenamiento en la nube además de Google Drive y Dropbox. Sin embargo, también quedó como un objetivo para futuras expansiones del proyecto, debido a que todas las funcionalidades debían de replicarse de manera concreta para cada servicio, lo cuál requeriría una gran cantidad de tiempo. Aunque la arquitectura de *MyDrive* está diseñada para ser modular y soportar la integración con múltiples servicios, cada nuevo servicio añade una capa adicional de complejidad en términos de desarrollo y mantenimiento. Considerando el tiempo y los recursos disponibles, se optó por enfocarse en integrar bien los servicios más populares y conocidos, dejando la puerta abierta para la inclusión de otros en el futuro.

7.- Bibliografía

- [1] E. Gamma, Design patterns : elements of reusable object-oriented software, 37a ed. Read., Mass., 1995. [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén.
- [2] J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time. New York : Crown Bus., 2014.
- [3] “Radware Bot Manager Captcha”. IOPscience. [Online]. Disponible: <https://iopscience.iop.org/article/10.1088/1757-899X/801/1/012136/meta>
- [4] Contributors to Wikimedia projects. “Cloud storage - Wikipedia”. Wikipedia, the free encyclopedia. [Online]. Disponible: <https://tinyurl.com/5e58n8za>
- [5] J. Bloch, Effective Java, 3a ed. Addison-Wesley Prof., 2017. [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén.
- [6] C. Walls, Spring in action, 5a ed. Shelter Isl., New York, 2019. [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén.
- [7] H. Djirdeh, Fullstack Vue: The Complete Guide to Vue.js. CreateSp. Indep. Publ. Platf., 2018.
- [8] B. Beizer, Software Testing Techniques. Dreamtech Press, 2003.
- [9] L. Crispin, Agile testing: A practical guide for testers and agile teams. Addison-Wesley, 2008. [Online]. Disponible en Recursos Electrónicos de la Universidad de Jaén.
- [10] Contributors to Wikimedia projects. “Client–server model - Wikipedia”. Wikipedia, the free encyclopedia. [Online]. Disponible: https://en.wikipedia.org/wiki/Client–server_model
- [11] “Software Architecture Patterns”. O'Reilly Online Learning. [En línea]. Disponible: <https://tinyurl.com/2n8kycdu>
- [12] “OpenAPI Specification - Version 3.1.0 | Swagger”. API Documentation & Design Tools for Teams | Swagger. [Online]. Disponible: <https://swagger.io/specification/>

Apéndice I. Descripción de contenidos suministrados

A continuación, se describirá la estructura del contenido suministrado en el archivo .ZIP, el cual proporciona toda la información necesaria para entender, analizar y evaluar el proyecto *MyDrive*, facilitando el acceso tanto a la documentación como al código fuente y a una demostración visual de la aplicación en funcionamiento. Está organizado de la siguiente manera:

1. **Memoria:** Esta carpeta contiene el documento de la memoria del proyecto en formato PDF. Aquí se recoge toda la documentación escrita que describe el proceso de desarrollo del proyecto *MyDrive*, incluyendo los objetivos, metodología, resultados obtenidos y conclusiones.
2. **Código Fuente:** Esta carpeta incluye todo el proyecto de *MyDrive*, organizado en su estructura correspondiente. Dentro de esta carpeta, se encuentran las siguientes subcarpetas:

MyDrive: Contiene el código fuente completo del proyecto, dividido en:

- a. **Frontend:** Incluye el código relacionado con la interfaz de usuario desarrollado en Vue.js, organizado según la estructura de carpetas necesaria para su ejecución.
 - b. **Backend:** Contiene el código del servidor, junto con todas las configuraciones necesarias. Dentro de esta carpeta, se encuentra una subcarpeta adicional:
 - i. **Javadoc:** Esta subcarpeta contiene un archivo *index.html* que permite visualizar en el navegador la documentación generada automáticamente del *backend* de la aplicación, facilitando la comprensión de la estructura y las funciones implementadas.
3. **Vídeo de funcionamiento:** Esta carpeta contiene un vídeo que muestra el funcionamiento de la aplicación *MyDrive*. El vídeo sirve como una demostración visual de las principales funcionalidades y características implementadas, ofreciendo una visión clara de cómo se utiliza el sistema.
 4. **Diagramas de secuencia:** Dentro de esta carpeta se encuentra un archivo .vpp de Visual Paradigm que contiene todos los diagramas de secuencia del sistema. Estos diagramas representan el flujo de interacción entre los diferentes componentes del sistema *MyDrive*, detallando cómo se gestionan las solicitudes y las respuestas entre el *frontend* y el *backend*, así como la comunicación con los distintos servicios de almacenamiento en la nube integrados en la aplicación.

Apéndice II. Manual de Instalación del sistema

En este Apéndice II. Manual de Instalación del Sistema, se describen los pasos necesarios para la instalación y configuración inicial de los elementos requeridos para la ejecución del sistema *MyDrive*, tanto en el *backend* como en el *frontend*.

Antes de comenzar con la instalación y configuración del *backend* y *frontend* de *MyDrive*, es crucial configurar adecuadamente las credenciales necesarias en los servicios externos. Para ello, debe realizar la configuración de su aplicación tanto en la consola de Google Developers como en la plataforma de desarrolladores de Dropbox. Esto incluye la creación de proyectos, la obtención de claves API y la configuración del flujo de autorización OAuth. En estos portales, obtendrá los secretos correspondientes como *client_id* y *client_secret*, que son esenciales para que su aplicación pueda interactuar con las APIs de Google Drive y Dropbox. Además, deberá configurar las URIs de redirección para ejecutar la aplicación en local, como `http://localhost:8080/api/dropbox/oauth/callback` para Dropbox y la URI equivalente para Google Drive. Asegúrese de que su aplicación esté registrada correctamente y que tenga todos los permisos necesarios para interactuar con las APIs.

II.1.- Configuración del backend

Instalación del JDK:

Asegúrese de tener instalado el JDK 19. Es posible que el proyecto funcione con versiones posteriores, pero no se garantiza la compatibilidad con versiones anteriores.

Descarga del proyecto:

Descargue el proyecto *MyDrive* en su máquina local.

Configuración de Propiedades Secretas para Dropbox:

Navegue al directorio:

```
Unset
/backend/src/main/resources
```

Verifique si existe el archivo *application-secrets.properties*. Si no existe, créelo con el siguiente contenido:

Java

```
dropbox.clientId="your-client-id"  
dropbox.clientSecret="your-client-secret"
```

Configuración de Propiedades Secretas para Google Drive:

En el mismo directorio (*/backend/src/main/resources*), verifique si existe la carpeta *keys*. Si no existe, créela.

Dentro de la carpeta *keys*, añada el archivo *credentials.json* que contiene las credenciales necesarias para acceder a la API de Google Drive.

Arranque del backend

Con los pasos anteriores completados, el *backend* debería estar listo para arrancar sin problemas. Si queremos lanzarlo mediante maven sin necesidad de un entorno de desarrollo, debemos asegurarnos de tener instalado *maven*, ir al directorio *backend* y ejecutar el siguiente comando:

Unset

```
mvn spring-boot:run
```

II.2.- Configuración del Frontend

Instalación de Node.js:

Asegúrese de tener instalada la versión v20.12.2 de Node.js. Es posible que el proyecto funcione con versiones posteriores, pero no se garantiza la compatibilidad con versiones anteriores.

Instalación de Vue CLI:

Instale Vue CLI, asegurándose de tener la versión *@vue/cli 5.0.8*. Puede funcionar con versiones superiores, pero no se recomienda el uso de versiones anteriores.

Instalación de Dependencias:

Navegue a la carpeta del *frontend* del proyecto. Si no existe la carpeta *node_modules*, instale todas las dependencias necesarias ejecutando en la terminal:

Unset

```
npm i
```

Levantamiento del Servidor Frontend:

Una vez instaladas las dependencias, puede levantar el servidor visual del *frontend* ejecutando:

```
Unset  
npm run serve
```

Esto iniciará el servidor de desarrollo y permitirá ver la aplicación en funcionamiento en su navegador.

Con estos pasos completados, tanto el *backend* como el *frontend* del sistema *MyDrive* estarán configurados y listos para su uso. Asegúrese de haber seguido todas las instrucciones de configuración de credenciales para garantizar que la aplicación pueda comunicarse correctamente con los servicios externos.

Apéndice III. Manual de Usuario

En este manual, se describen las funcionalidades principales de la aplicación *MyDrive*, proporcionando una visión general de cómo utilizar el sistema para gestionar el almacenamiento en la nube de manera eficiente. A continuación se detallan las funcionalidades implementadas, acompañadas de capturas de pantalla que ilustran el uso de cada una.

III.1.- Inicio de sesión y autenticación

Una vez en la vista de inicio del gestor de archivos, los usuarios deben seleccionar el servicio de almacenamiento en la nube al que desean autenticarse, ya sea Google Drive o Dropbox. Una vez seleccionado, se les redirige a la página de autenticación del servicio elegido. Los usuarios deben proporcionar las credenciales correspondientes para autorizar a *MyDrive* a acceder a su cuenta. Esta autenticación permite que la aplicación obtenga los permisos necesarios para gestionar los archivos y carpetas del servicio seleccionado.

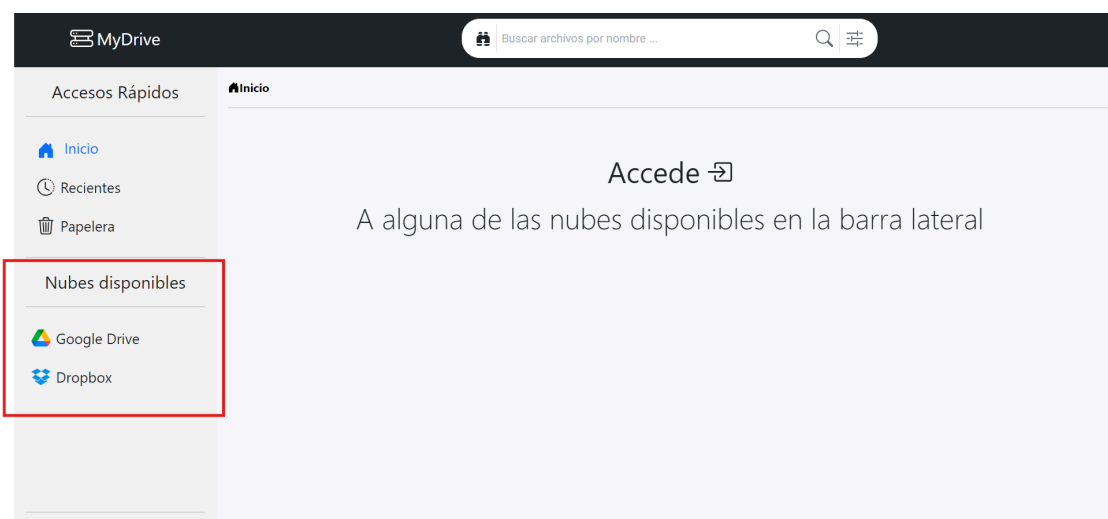


Ilustración 27. Nubes disponibles antes de iniciar sesión

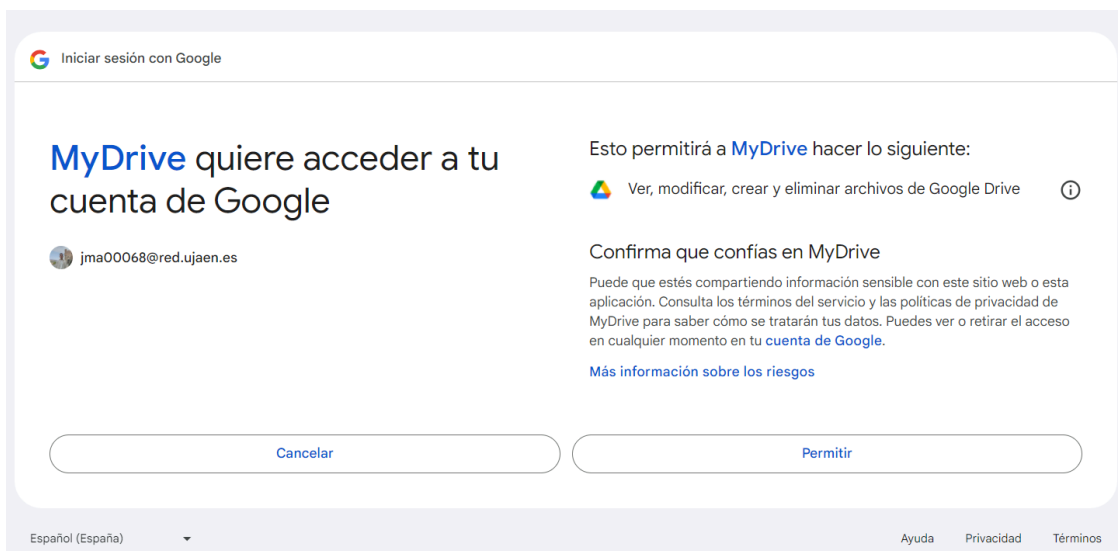


Ilustración 28. Proceso de autenticación y autorización

III.2.- Subida de Archivos

La aplicación permite a los usuarios subir nuevos archivos a la carpeta en la que se encuentre actualmente. Para hacerlo, los usuarios deben seleccionar el archivo desde su dispositivo y confirmar la subida. La barra de progreso muestra el estado de la carga.

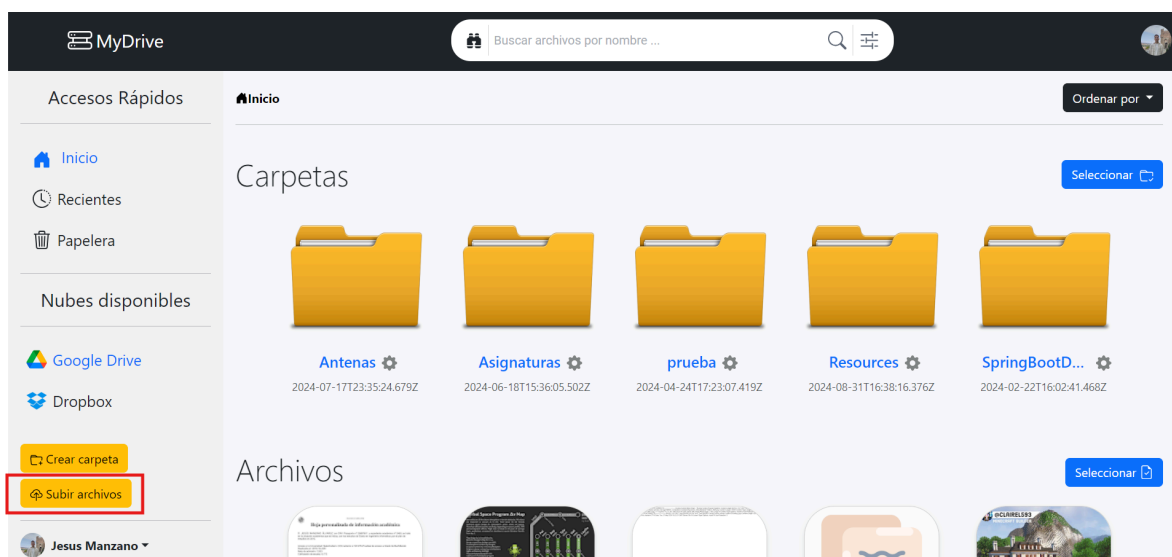


Ilustración 29. Opción para subir archivos

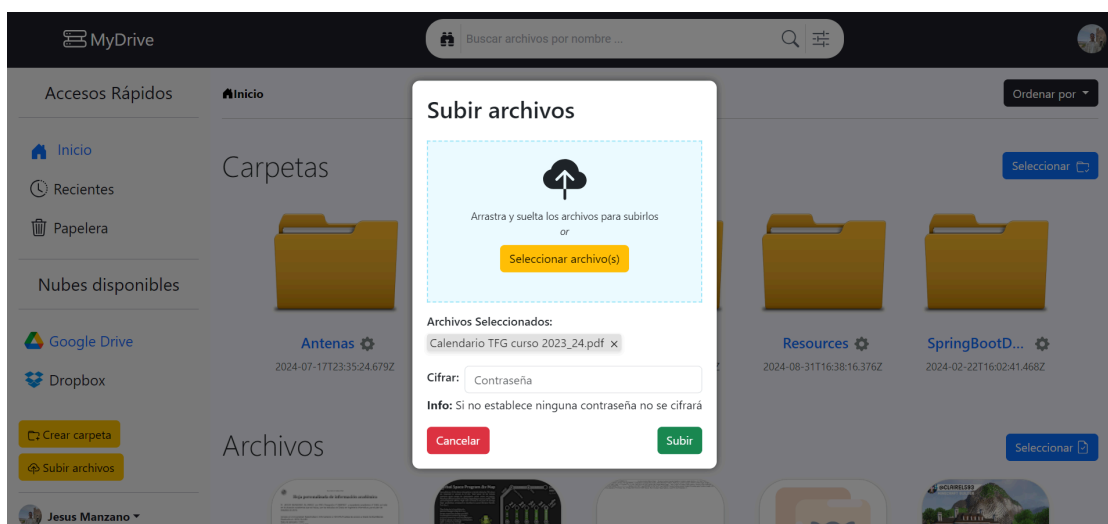


Ilustración 30. Selección de archivos a subir

Como podemos ver en la captura anterior, podemos modificar la lista de archivos seleccionados e incluso establecer una contraseña con la cuál se cifrarán. Si no se establece ninguna contraseña no se cifra. Para confirmar la subida se debe pulsar el botón subir y esperar a que se suban.

III.3.- Descarga de archivos

Los usuarios pueden descargar archivos desde la aplicación a su dispositivo local. Basta con seleccionar el archivo deseado y hacer clic en el botón de descarga. Otra opción es pulsar el botón azul a la derecha del título de archivos que permite seleccionar múltiples archivos. La descarga se inicia automáticamente y el archivo se guarda en la ubicación predeterminada del navegador.

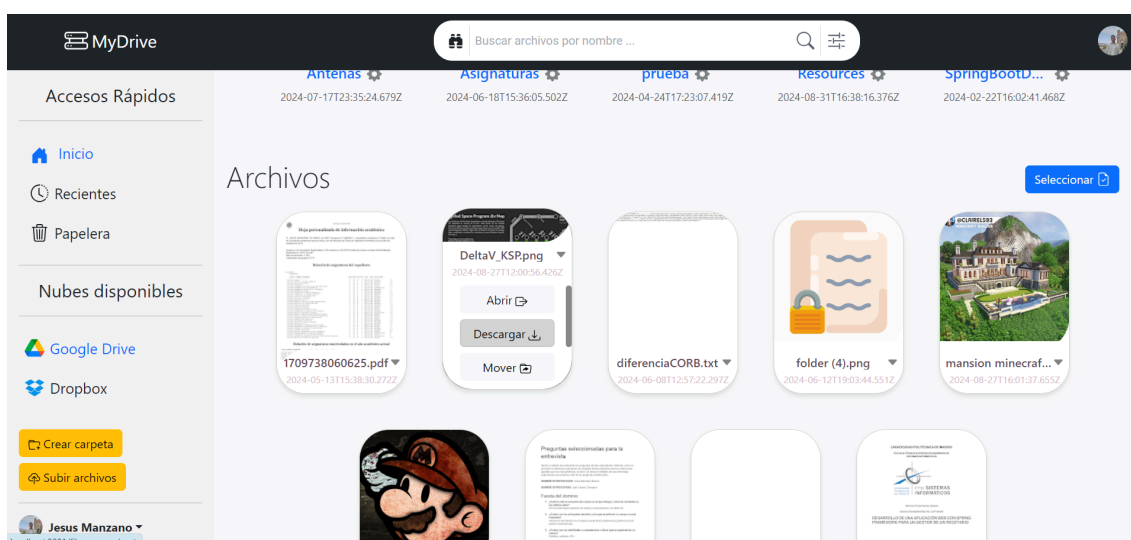


Ilustración 31. Descargar un único archivo

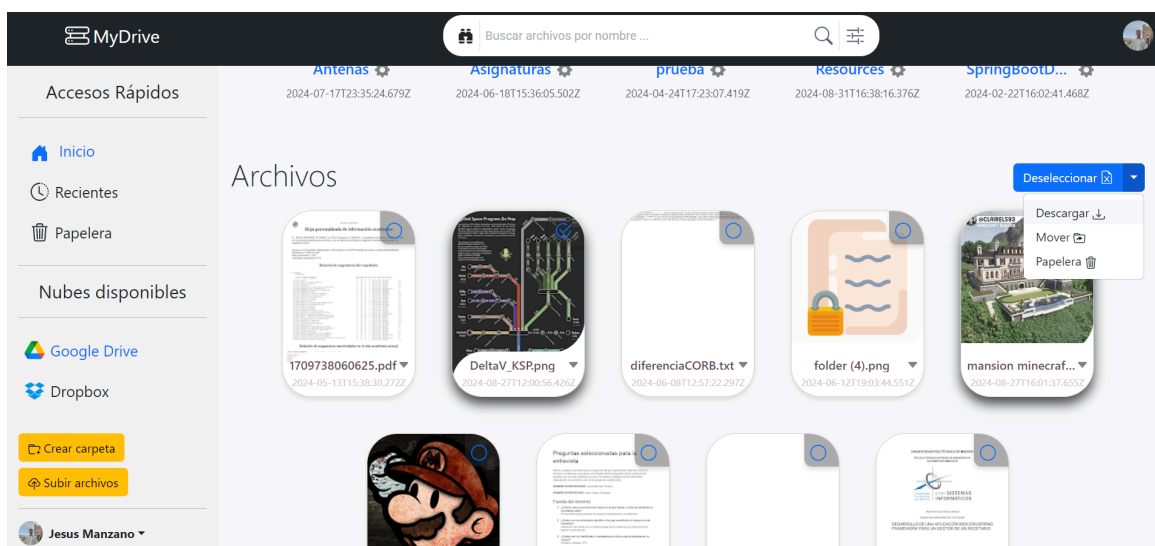


Ilustración 32. Descargar múltiples archivos

Si tratas de descargar un archivo cifrado aparecerá un ventana flotante que te pedirá la contraseña que pusiste originalmente a dicho archivo.

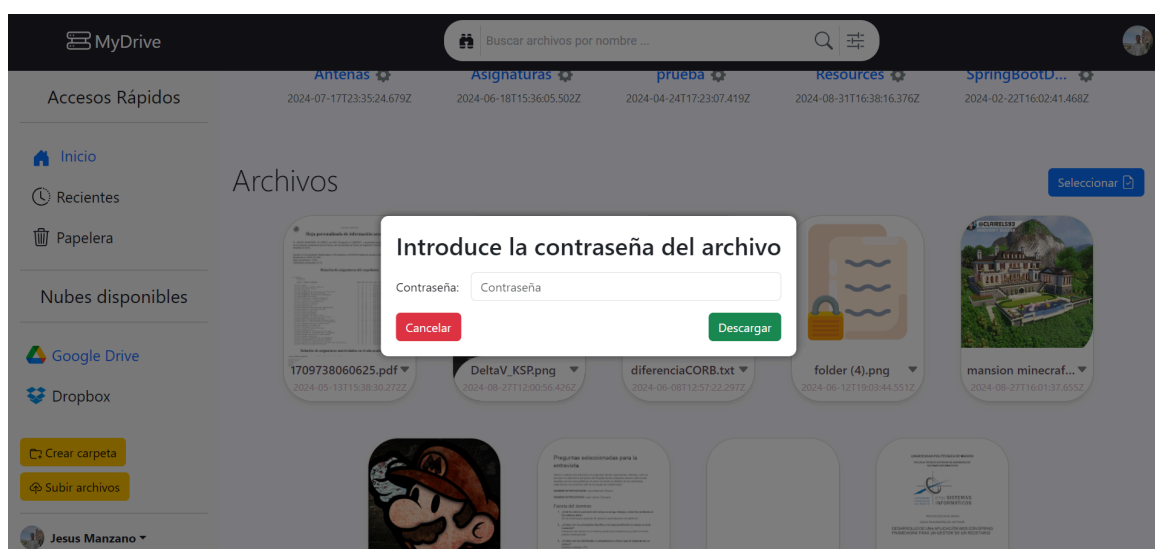


Ilustración 33. Descarga de un archivo cifrado

Para confirmar la descarga simplemente hay que pulsar el botón de “Descargar” y en caso contrario el de “Cancelar”.

III.4.- Creación de una nueva carpeta

Para organizar mejor los archivos, los usuarios pueden crear nuevas carpetas dentro del servicio de almacenamiento. La carpeta se creará dentro de la carpeta en la que se encuentra actualmente. La aplicación proporciona una interfaz simple para nombrar y crear carpetas.

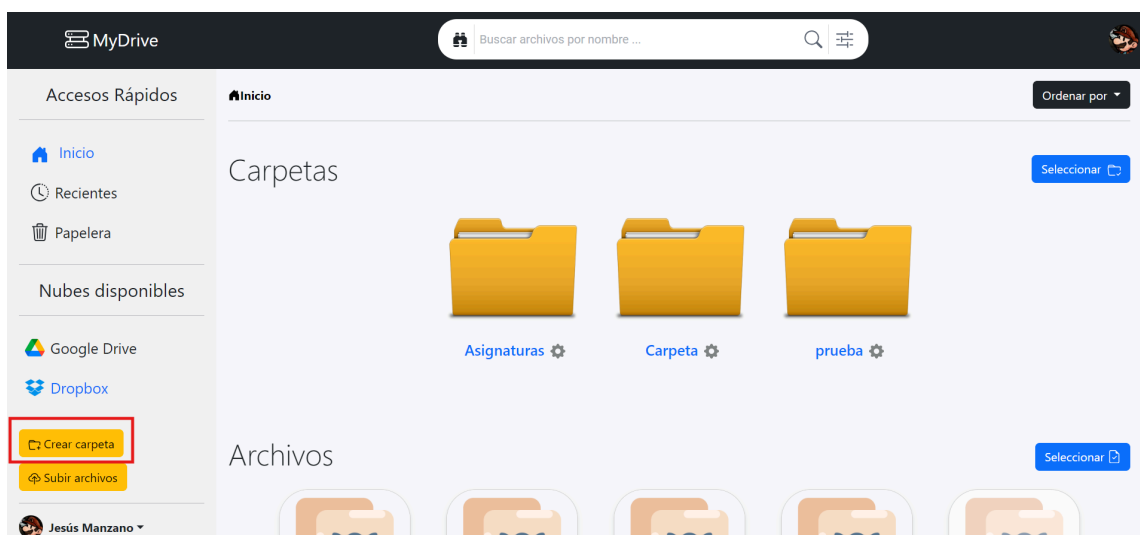


Ilustración 34. Selección para crear una carpeta

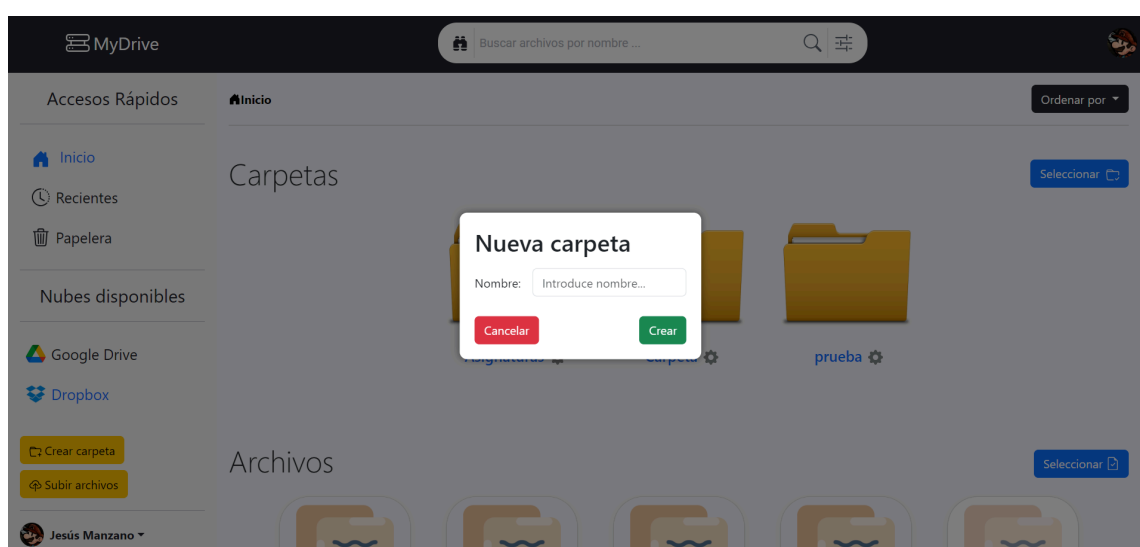


Ilustración 35. Nombre de la nueva carpeta

Para confirmar la creación de dicha carpeta, deberá pulsar sobre el botón de “Crear” y en caso contrario sobre “Cancelar”.

III.5.- Búsqueda de archivos y carpetas

La funcionalidad de búsqueda permite a los usuarios encontrar archivos y carpetas dentro del servicio de almacenamiento. Los usuarios deben ingresar palabras clave en la barra de búsqueda. Adicionalmente, el usuario puede establecer un filtro de búsqueda para buscar dentro de la carpeta o a nivel global en dicho servicio de almacenamiento.

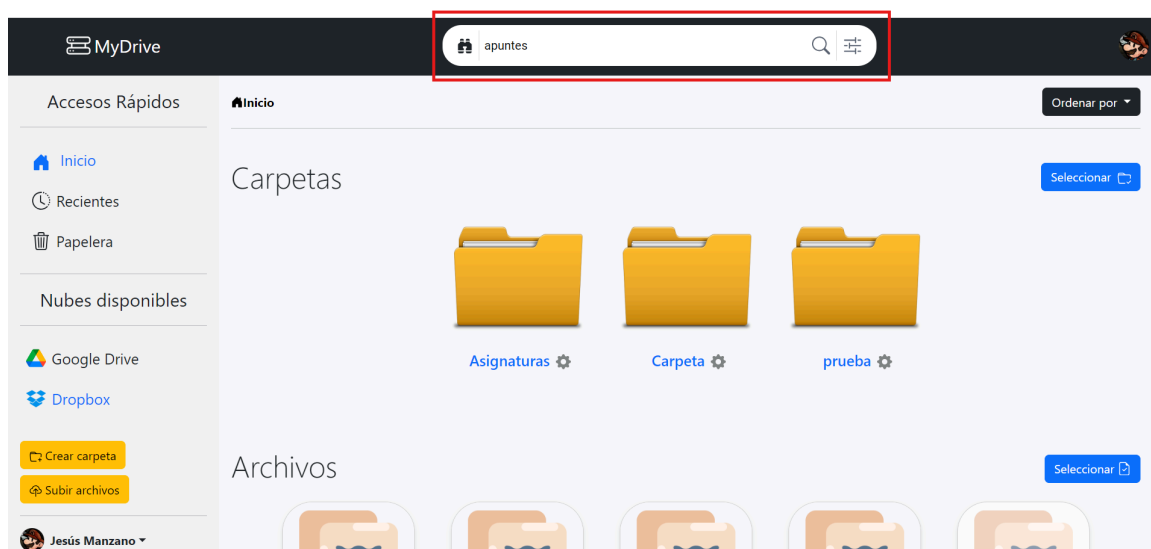


Ilustración 36. Introducción palabra para buscar

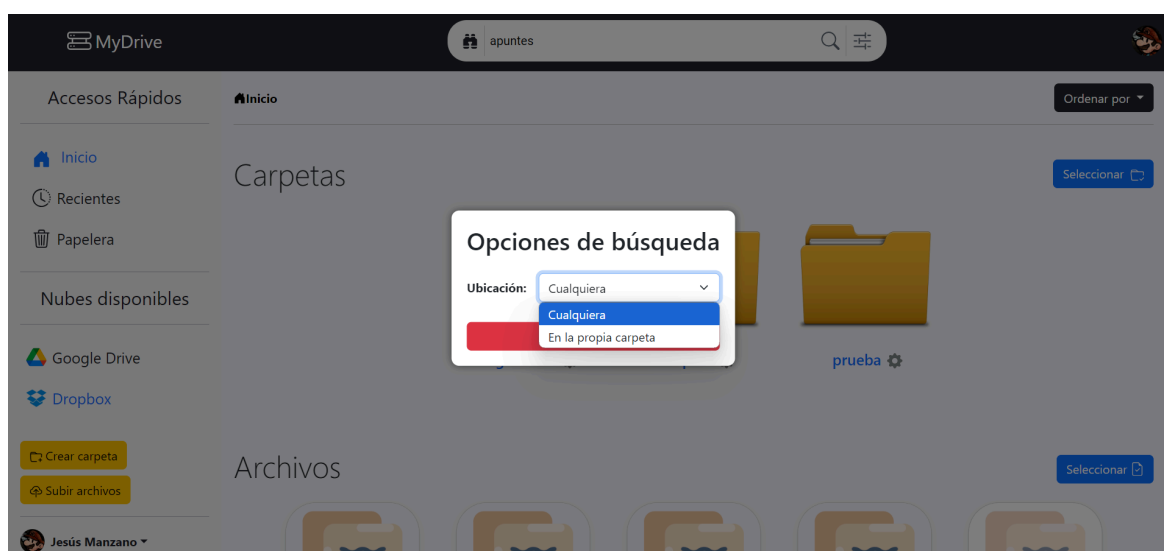


Ilustración 37. Aplicar filtros de búsqueda

Si se pulsa sobre el botón de opciones de búsqueda justo a la derecha de la lupa, podremos buscar dentro de la propia carpeta o a nivel global. Por defecto, se encuentra seleccionado a nivel global ya que de esta manera también aparecerán las coincidencias dentro de la propia carpeta actual.

Finalmente, para buscar se puede pulsar sobre la lupa una vez introducida la palabra o pulsar sobre la tecla INTRO del teclado.

III.6.- Apertura de Archivos

MyDrive permite a los usuarios abrir archivos. Al no permitir esta opción de forma nativa, la aplicación redirige al usuario al archivo en cuestión para visualizarlo en el

servicio en el que está almacenado. Esta funcionalidad es útil para revisar documentos rápidamente sin tener que descargar el archivo en local.

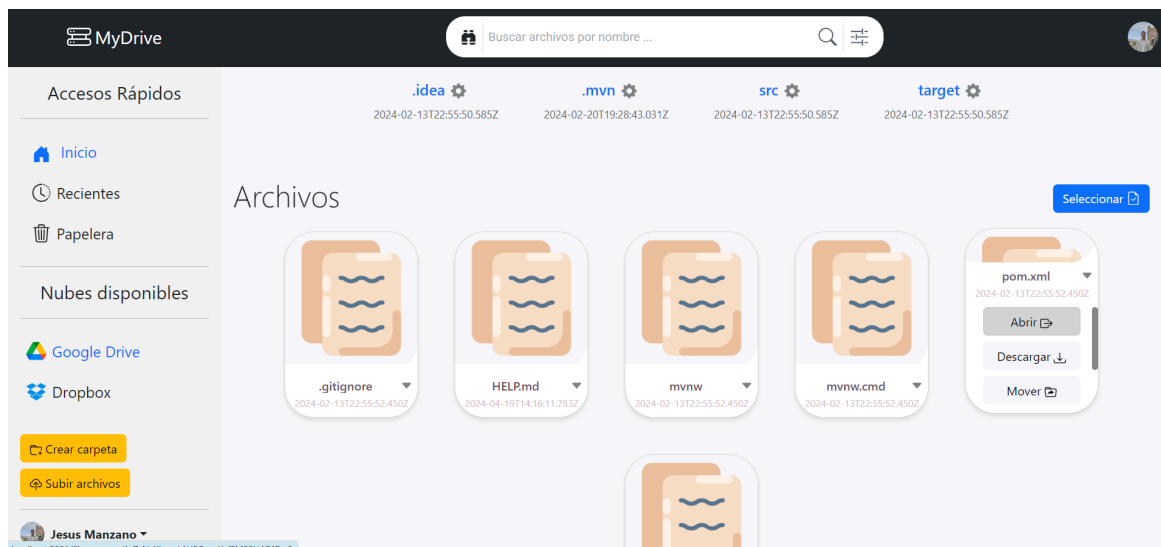


Ilustración 38. Previsualizar un archivo

III.7.- Renombrar carpetas o archivos

Los usuarios pueden cambiar el nombre de cualquier archivo o carpeta dentro de la aplicación. Para hacerlo, deben seleccionar el archivo o carpeta, hacer clic en la opción de renombrar, y luego ingresar el nuevo nombre deseado. Esta función es útil para mantener un sistema de archivos bien organizado y fácil de navegar.

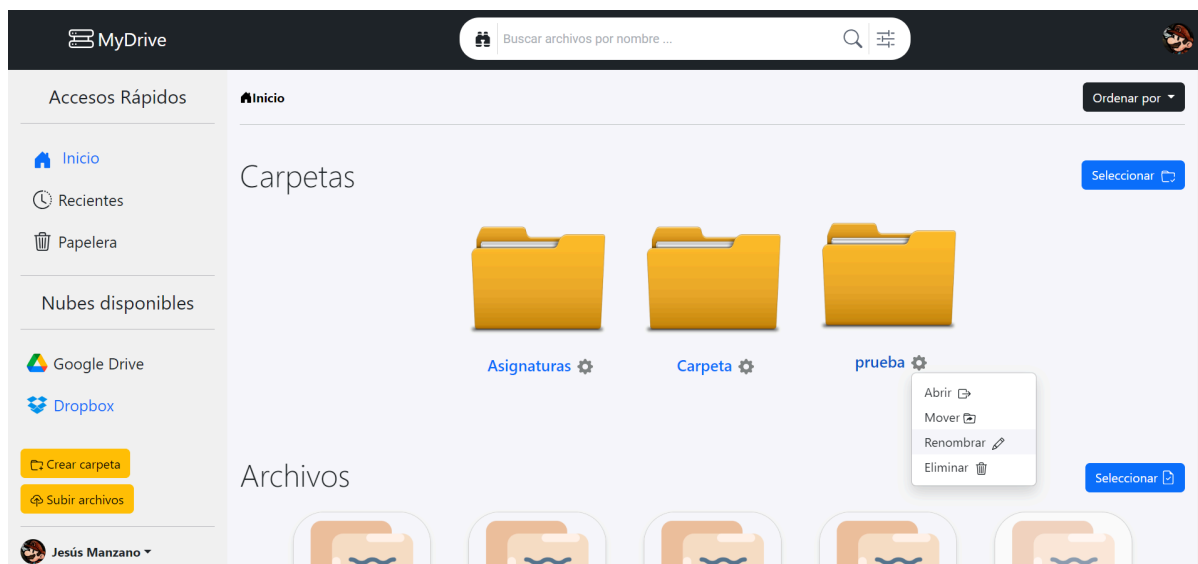


Ilustración 39. Opción para renombrar archivo

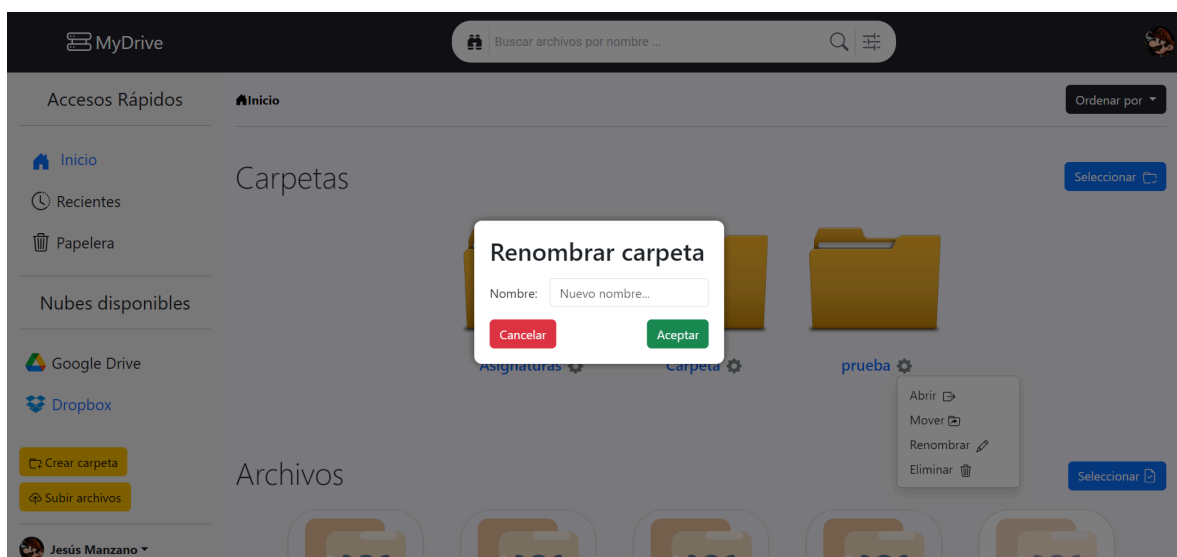


Ilustración 40. Renombrar carpeta

Al igual que se puede renombrar un carpeta, de la misma manera se aplicaría para un archivo. Para completar el proceso y renombrarla, se debe pulsar sobre “Aceptar” y en caso contrario sobre “Cancelar”.

III.8.- Mover archivos o carpetas

MyDrive permite a los usuarios mover archivos o carpetas de un lugar a otro. Al seleccionar un archivo o carpeta y la opción de mover, el usuario puede elegir la carpeta de destino dentro del servicio actual. Si el usuario quiere mover archivos y el destino es un servicio de almacenamiento diferente al original, en lugar de mover, se realizará una copia de los archivos en la nueva ubicación, permitiendo así una gestión flexible entre diferentes servicios de almacenamiento.

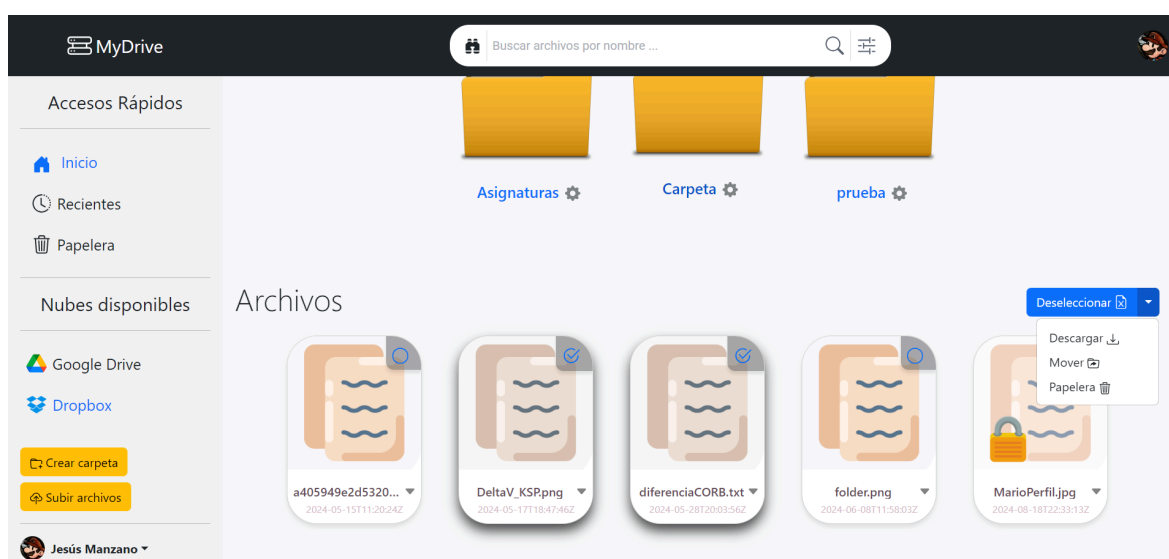


Ilustración 41. Mover archivos

Al igual que en otras funcionalidades comentadas anteriormente en este manual, se puede hacer con múltiples archivos o carpetas o únicamente con uno. Para operaciones múltiples hay que usar siempre los botones azules de la derecha, tanto para carpetas como archivos.

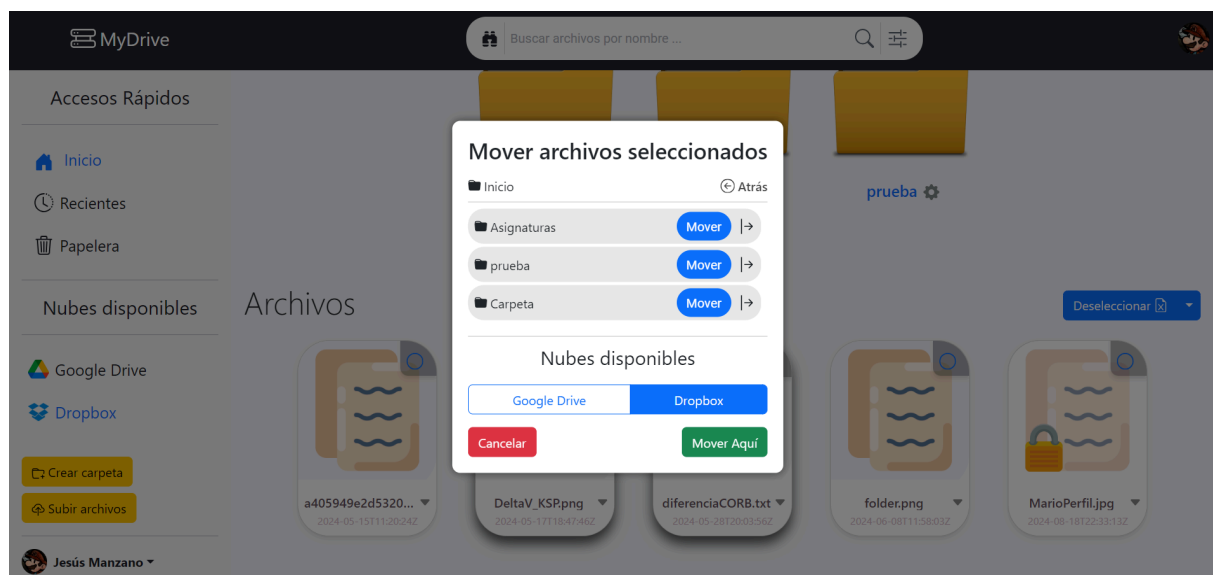


Ilustración 42. Mover archivos

Para confirmar el movimiento de los archivos seleccionados, se puede pulsar sobre los botones de “Mover” que tiene cada carpeta y se moverán a ella, o entrar en cada carpeta con las flechas que permiten navegar por los directorios hasta llegar al directorio destino y pulsar el botón de “Mover Aquí”. En caso contrario, se puede pulsar sobre “Cancelar”.

III.9.- Ordenación de Archivos

Los usuarios pueden ordenar los archivos y carpetas por nombre, fecha de modificación o tamaño. Esta funcionalidad facilita la organización de los contenidos según las preferencias del usuario. Por defecto, la ordenación está marcada por nombre, lo que quiere decir que se ordena todo por orden alfabético.

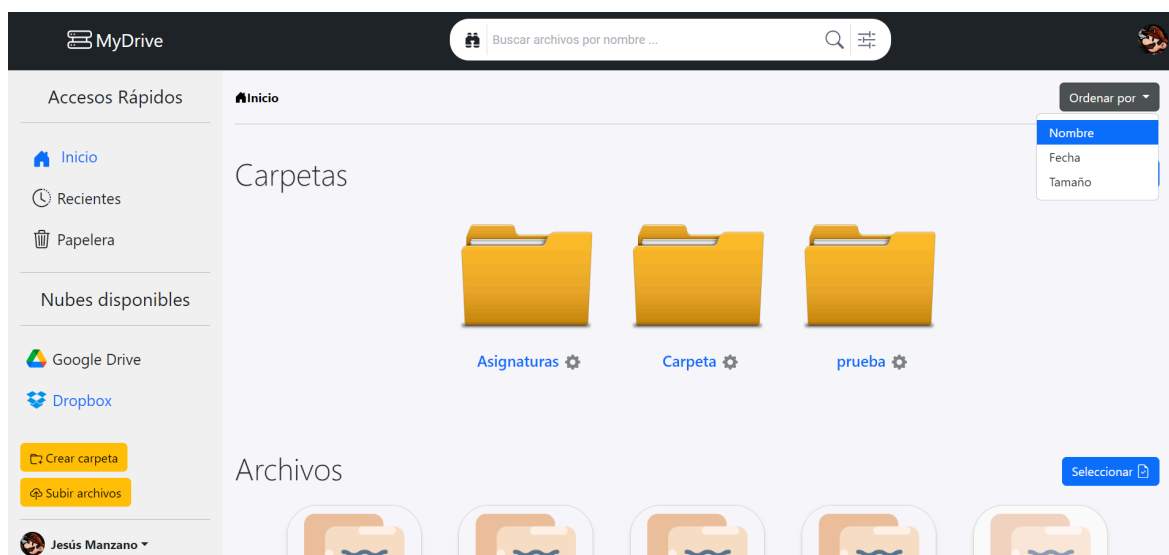


Ilustración 43. Ordenación de todo el contenido

III.10.- Periodo máximos de archivos recientes

MyDrive incluye una funcionalidad que permite a los usuarios establecer el período máximo durante el cual se consideran archivos como "recientes". Los usuarios pueden elegir entre opciones como 1 día, 1 semana, 1 mes o 1 año, siendo 1 semana la opción predeterminada. Los archivos que coincidan con el período seleccionado serán mostrados en la pestaña de "Recientes", brindando un acceso rápido a los documentos últimamente usados sin necesidad de buscarlos manualmente.

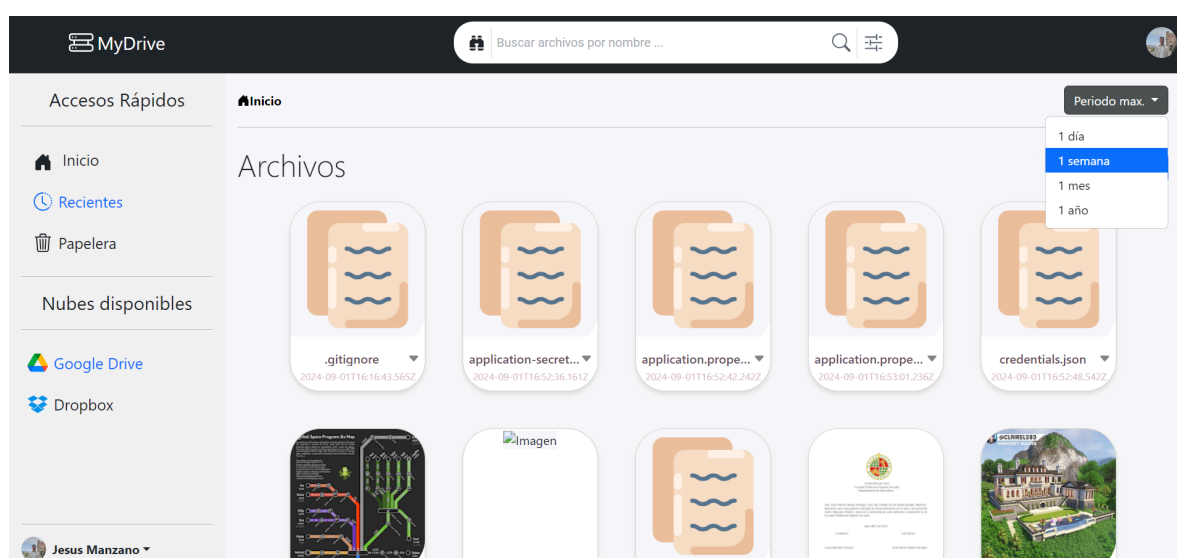


Ilustración 44. Establecimiento de periodo máximo de archivos recientes

III.11.- Mover archivos a la papelera de reciclaje

Cuando un usuario decide eliminar un archivo, este se mueve a la papelera de reciclaje del servicio de almacenamiento en la nube correspondiente. Esto significa que el archivo no se elimina inmediatamente, sino que se guarda temporalmente en la papelera, permitiendo su recuperación en caso de un borrado accidental.

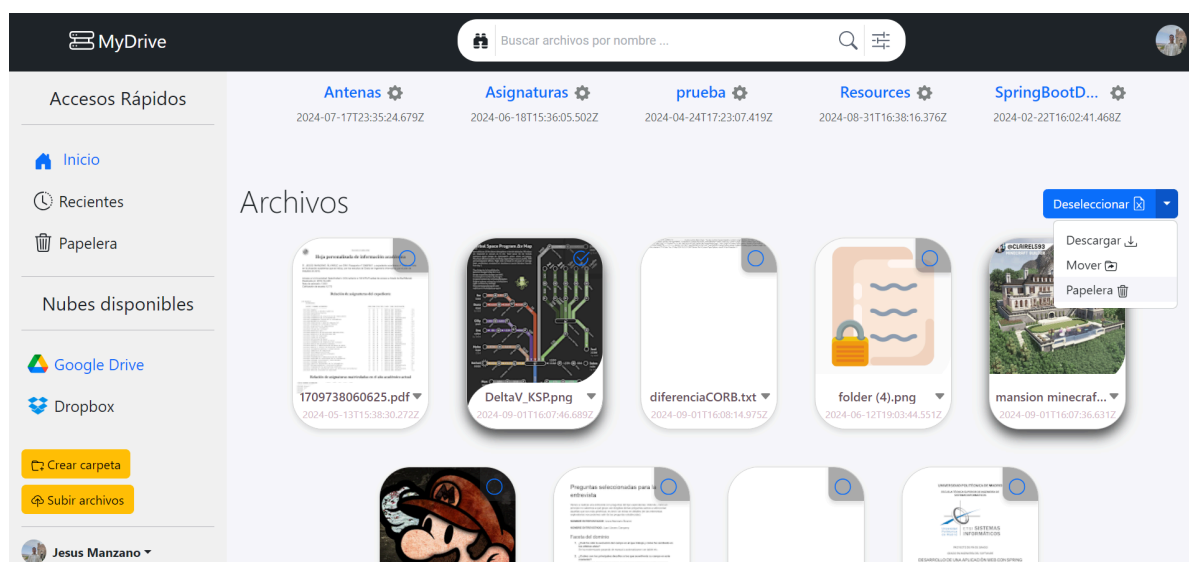


Ilustración 45. Opción para enviar a la papelera los archivos seleccionados

Para mover un archivo a la papelera, seleccione el archivo deseado, elija la opción "Eliminar" y confirme la acción o puede hacerlo de manera múltiple usando el botón de seleccionar azul y pulsar en "papelera" una vez seleccionados los archivos que se quieren eliminar.

III.12.- Eliminar carpetas

A diferencia de los archivos, cuando se elimina una carpeta en *MyDrive*, esta no se envía a la papelera. En su lugar, la aplicación solicita una confirmación del usuario antes de eliminarla definitivamente. Esta acción es irreversible, por lo que es importante asegurarse antes de proceder.

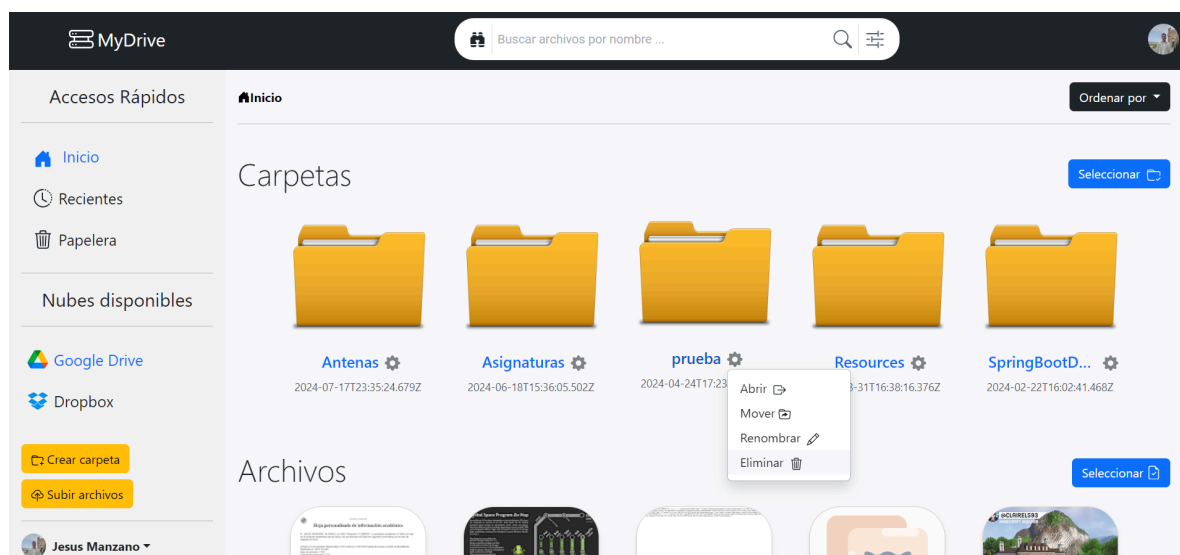


Ilustración 46. Opción para eliminar una carpeta

De la misma forma que en otras opciones, se puede eliminar de manera individual o múltiple gracias al botón de “Seleccionar” correspondiente.

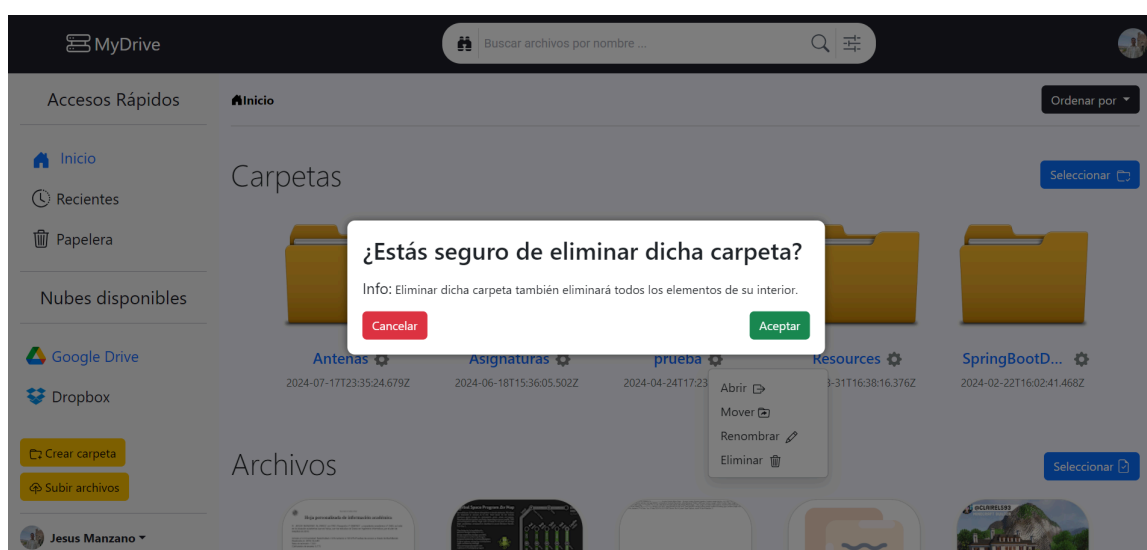


Ilustración 47. Eliminación de una carpeta

Tras pulsar “Eliminar”, aparecerá un mensaje de confirmación indicando que si eliminamos esta carpeta, se eliminará todo su contenido junto a ella. Si estamos extremadamente seguros de esta operación, se debe pulsar sobre “Aceptar” y en caso contrario sobre “Cancelar”.

III.13.- Eliminación Permanente de Archivos desde la Papelera

Una vez que los archivos se encuentran en la papelera de reciclaje, el usuario tiene la opción de eliminarlos de forma permanente. Al acceder a la papelera, se puede seleccionar cualquier archivo para proceder a su eliminación definitiva, lo que

liberará el espacio ocupado por dicho archivo en la nube. Este proceso es irreversible.

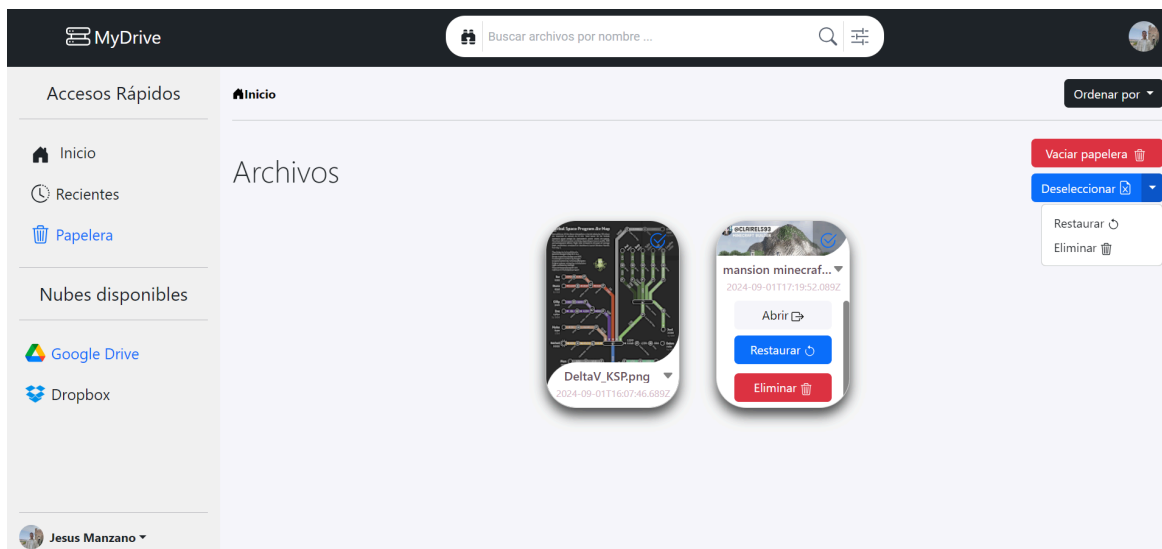


Ilustración 48. Eliminación de archivos

Como podemos observar, se pueden eliminar los archivos mediante el botón de seleccionar, e indicando aquellos que queremos eliminar, podemos hacerlo únicamente con un archivo. O si queremos limpiar la papelera, se puede pulsar sobre “Vaciar papelera” que eliminará todos los archivos.

III.14.- Restaurar archivos

MyDrive permite a los usuarios restaurar archivos que han sido movidos a la papelera de reciclaje. Esta funcionalidad es útil si se elimina un archivo por error y se desea recuperarlo sin perder su ubicación original en la estructura de carpetas. Para restaurar un archivo, el usuario debe acceder a la papelera de reciclaje, seleccionar el archivo que desea recuperar y elegir la opción "Restaurar". El archivo volverá a su ubicación original dentro del servicio de almacenamiento, como si nunca hubiera sido eliminado.

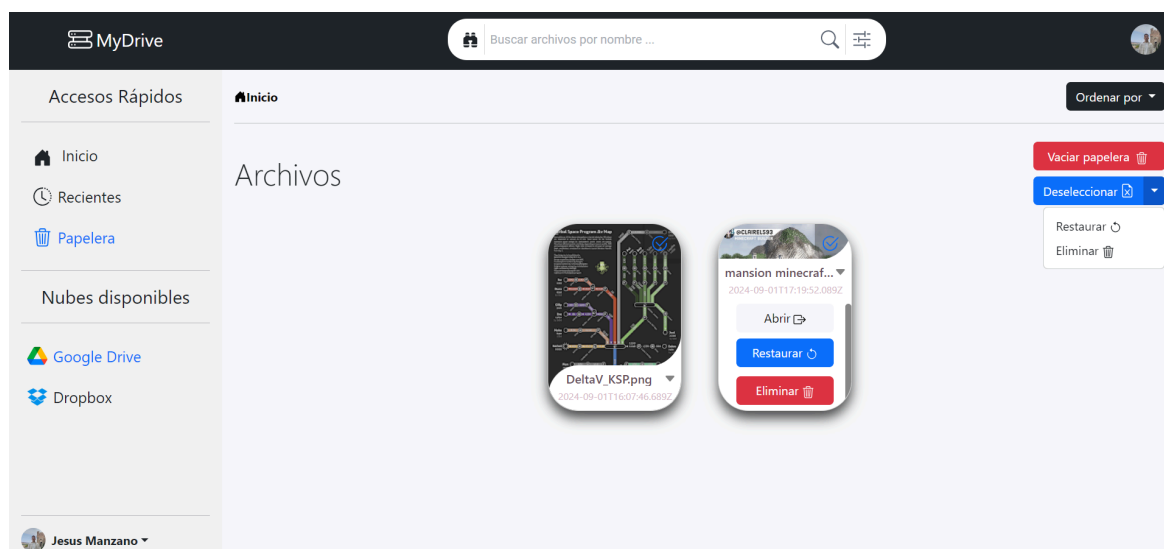


Ilustración 49. Restaurar archivos

Para restaurar archivos se puede hacer únicamente con uno o se puede hacer una selección múltiple tal y como hemos visto ya en otras funcionalidades.

III.15.- Cerrar sesión en un servicio de almacenamiento en la nube

MyDrive ofrece a los usuarios la opción de cerrar sesión en sus cuentas de Google Drive o Dropbox en cualquier momento. Esta funcionalidad es importante para asegurar que la sesión de un usuario no quede abierta en dispositivos compartidos o públicos. Para cerrar sesión, el usuario debe dirigirse a su foto o datos de perfil en la parte inferior izquierda o superior derecha y seleccionar la opción "Cerrar Sesión". Esto desvincula la cuenta del servicio de almacenamiento y finaliza la sesión actual, protegiendo así la seguridad y privacidad de los datos del usuario.

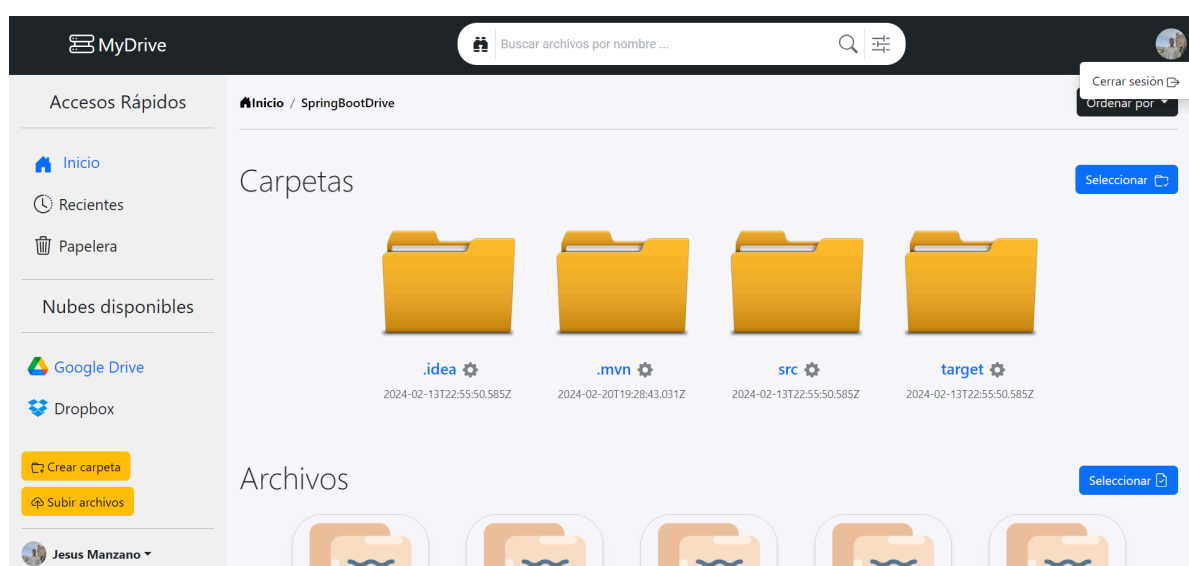


Ilustración 50. Cerrar sesión en un servicio de almacenamiento en la nube