Repetición documental. Propuesta de almacenamiento único multi-aplicación. DBRMS, NoSQL y Simple Storage al rescate.

Autor: **Lic. Jorge Gerardo Fernández Lugo**

Director: **Mg. Ing. Gustavo Ramiro Rivadera**

Universidad Católica de Salta

Facultad de Ingeniería

Especialización en Administración de Bases de Datos

2023

Registro de Aprobación del Tribunal Evaluador

Fecha y Tipo de Defensa: 19/04/2023, Defensa Oral en modalidad virtual

Nota: 10 (DIEZ)

Presidente del Tribunal Evaluador: Mg. Ing. Fernando Rivera Bendorff

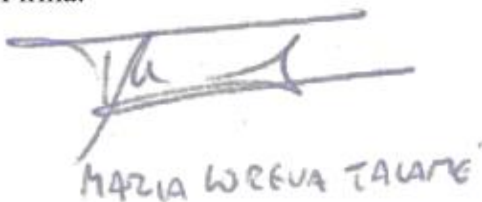
Firma:

Integrante del Tribunal Evaluador: Mg. Ing. Guillermina Nievas

Firma:

Integrante del Tribunal Evaluador: Mg. Lic. Lorena Talamé

Firma:



Resumen

La repetición de documentos iguales, dispersados en múltiples bases de datos, suele producirse cuando distintos sistemas utilizan los mismos documentos, y encuentran en la copia y en la duplicación, la forma más simple de compartir los mismos. Pero la repetición, como una solución simple para compartir, trae consigo el problema del desperdicio de almacenamiento producido por la duplicación junto con la pérdida de trazabilidad con respecto al versionado de un mismo documento.

El propósito del presente trabajo, es encontrar una forma de almacenar documentos, provenientes de múltiples sistemas, en un repositorio único. Permitir que los demás sistemas, mediante referencias y permisos, puedan acceder a los documentos, a través de una base de datos única multi-aplicación; la cual contemplara alojar los mencionados documentos en tres tipos distintos de almacenamiento como repositorio único: *relacional*, *NoSQL* y *almacenamiento simple de objetos*. Finalmente, como propósito añadido, incorporar a la solución un motor especializado en búsquedas, a fin de dotar al modelo de un mecanismo para realizar búsquedas complejas de texto.

El desarrollo del presente proyecto parte de la definición del problema, seguida de un estudio de los tres tipos de almacenamientos mencionados anteriormente. El resultado es el diseño de la base de datos única multi-aplicación, acompañada de un esquema de servicios para las operaciones CRUD. Desarrollado en PHP, un lote de pruebas alimenta la base de datos única, junto a los tres tipos de almacenamientos propuestos, probando la funcionalidad del modelo.

Palabras claves: repetición, documento, relacional, nosql, almacenamiento simple de objetos

Abstract

The repetition of the same documents, dispersed in multiple databases, usually occurs when different systems use the same documents, and find in copying and duplication, the simplest way to share them. But repetition, as a simple sharing solution, brings with it the problem of wasted storage caused by duplication along with loss of traceability regarding versioning of the same document.

The purpose of this work is to find a way to store documents from multiple systems in a single repository. Allow other systems, through references and permissions, to access the documents, through a single multi-application database; which will contemplate hosting the aforementioned documents in three different types of storage as a single repository: relational, NoSQL and simple object storage. Finally, as an added purpose, to incorporate a specialized search engine into the solution, in order to provide the model with a mechanism to perform complex text searches.

The development of this project starts from the definition of the problem, followed by a study of the three types of storage mentioned above. The result is the unique multi-application database design, accompanied by a service schema for CRUD operations. Developed in PHP, a batch of tests feeds the unique database, together with the three types of storage proposed, testing the functionality of the model.

Keywords: repetition, document, relational, nosql, simple object storage

Índice de contenidos

1. Introducción ... 1
2. El problema ... 2
2.1 Planteamiento del problema ... 2
2.2 Formulación del problema ... 3
2.3 Delimitación del problema ... 4
2.4 La cuestión documental en el ambiente académico ... 4
3. Objetivos ... 6
3.1 Objetivo General ... 6
3.2 Objetivos Específicos ... 6
4. Caso de estudio ... 7
4.1 Situación actual ... 7
5. Marco teórico ... 9
5.1 Motores de base de datos relacionales ... 9
5.2 Motores de base de datos NoSQL tipo documentales ... 10
5.3 Almacenamiento simple de objetos ... 11
6. Proyecto tecnológico: Diseño ... 13
6.1 Diseño de base de datos ... 13
6.2 Definiciones de tablas ... 16
6.3 Diseño de servicios para el acceso a documentos ... 21
7. Proyecto tecnológico: Implementación del almacenamiento único multi-aplicación ... 30
7.1 Implementación utilizando base de datos relacional ... 32
7.2 Implementación utilizando MongoDB ... 33
7.3 Implementación utilizando almacenamiento simple ... 34

8. Generación lote de pruebas 35
8.1 Modelo 35
8.2 Creación de las tablas 36
8.3 Factory 37
8.4 Poblado de las bases de datos 39
8.5 Test de recuperación 41
9. El motor de búsqueda 45
9.1 Elastic 45
9.2 Logstash 45
9.3 Elastic y Logstash en el almacenamiento único multi-aplicación 46
10. Conclusiones 52
Bibliografía 54

Índice de figuras

Figura 1: Costo de 1TB alojado en AWS S3 3

Figura 2: Almacenamiento único multi-aplicación 15

Figura 3: Definición de la tabla binary, para el caso de usarse un repositorio de documentos relacional 17

Figura 4: Definición del objeto GridFS, para el caso de usarse un repositorio NoSQL 17

Figura 5: Definición del un objeto SimpleStore, para el caso de usarse almacenamiento *simple* como repositorio de documentos 18

Figura 6: Definición de la tabla files 19

Figura 7: Definición de la tabla systems 20

Figura 8: Definición de la tabla users 20

Figura 9: Definición de la tabla file_systems 20

Figura 10: Definición de la tabla file_grants 21

Figura 11: Diagrama de flujo: creación de un documento 23

Figura 12: Diagrama de flujo: recuperación de un documento 25

Figura 13: Diagrama de flujo: actualización de un documento 27

Figura 14: Diagrama de flujo: eliminación de un documento 28

Figura 15: Diagrama de flujo: asignación de permisos de un documento 29

Figura 16: Esquema del proyecto tecnológico 30

Figura 17: Consulta a la tabla files 31

Figura 18: Contenido de la tabla files 31

Figura 19: Diseño tabla files 32

Figura 20: Ejemplo contenido de la tabla files 32

Figura 21: Colección que almacena los metadatos 33

Figura 22: Colección que almacena los chunks 33

Figura 23: Bucket y su contenido de documentos 34

Figura 24: Modelos del proyecto 35

Figura 25: Definición del modelo files, con sus atributos 36

Figura 26: Migraciones del proyecto 36

Figura 27: Ejemplo migración tabla files 37

Figura 28: FilesFactory.php: Factoría de la tabla files 38

Figura 29: Factorías del proyecto 39

Figura 30: Proceso de migración de tablas y poblado 39

Figura 31: Código generador de documentos de prueba 40

Figura 32: Tabla files. Registros generados mediante migración, factorización y poblado. ... 41

Figura 33: Contenido del bucket “multi-app-store.binaries” 42

Figura 34: Porciones (chunks) de documentos 42

Figura 35: Registros de la tabla relacional binaries 43

Figura 36: Recuperación del documento 4bae4215-5ef1-3785-a177-659c5db24232.pdf desde el almacenamiento simple 43

Figura 37: Recuperación del documento 4bae4215-5ef1-3785-a177-659c5db24232.pdf desde la especificación GridFS de MongoDB 44

Figura 38: Esquema del proyecto tecnológico, con Elasticsearch como motor de búsqueda . 47

Figura 39: Consulta full-text search, ordenada por relevancia de mayor a menor, limitada a los primeros 5 48

Figura 40: Ejemplo de contenido a buscar. Score=1,487 49

Figura 41: Ejemplo de contenido buscado, con un mejor score 1,750 50

Figura 42: Consulta full-text search, ordenada por relevancia de mayor a menor 51

Figura 43: Documentos que no cumplen con en el MATCH. Puede verse que el SCORE() en todos los casos es de 0 51

Figura 44: Proyecto almacenamiento único multi-aplicación, con elasticsearch como motor de búsquedas complejas 52

1. Introducción

El pasar de los tiempos, la evolución de los sistemas, la continua adaptación a nuevas funcionalidades, nos condujo a un punto tal, en el que la repetición de documentos, era la alternativa más rápida para responder a las necesidades de la organización, sin tener presente el alto costo en el que estábamos incurriendo a futuro.

Hoy es necesario establecer un punto en el tiempo, a partir del cual, el desarrollo de nuevos sistemas, la adaptación de antiguos sistemas y cualquier acción relacionada, gire en torno a un único repositorio de documentos (.doc, .rtf, .pdf); y que los análisis y programaciones se realicen teniendo como restricción una estructura de datos especializada para el almacenamiento único de documentos.

Con esto se pretende romper con el ciclo común, primero desarrollo, luego creo la estructura de datos. Sino que, por lo contrario, a partir de un diseño preestablecido de almacenamiento, ofrecer al analista y al programador, las herramientas para el guardado de documentos, en forma consensuada y uniforme para toda la organización. Pensar en la documentación de la organización en general, y no en los sistemas en particular.

Este proyecto tecnológico, pretende brindar una propuesta de solución global al almacenamiento de documentos para toda la organización, independiente del sistema desde el cual provenga mencionado documento. Para ello se investigarán tres tipos de tecnologías; DBRMS, NoSQL, y almacenamiento simple de objetos. Si bien existe también la posibilidad de almacenar documentos en un *filesystem*, este no será contemplado en el presente proyecto, ya que se pretende utilizar formas de almacenamiento, con características cercanas a motores de base de datos. No obstante, el almacenamiento simple de objetos, aunque no es un tipo de base de datos, se lo incluye en el presente proyecto por ser una tecnología con un costo muy bajo, escalable y resiliente con los datos frente a fallas de hardware.

Entonces, para las tecnologías contempladas, se evaluarán las fortalezas y debilidades de las mismas, y a partir de la combinación de lo mejor de ellas, llegar a un producto tecnológico que satisfaga los objetos del presente proyecto.

2. El problema

2.1 Planteamiento del problema

Nuestra organización objeto de estudio, consta de varias decenas de bases de datos, en las que se da el fenómeno de repetición de documentos. Es decir, un mismo documento, con exacto contenido, se encuentra almacenado en varias bases de datos.

Actualmente, y acentuado por el fenómeno mundial de la pandemia, el flujo de archivos de documentos digitales, cuyo contenido puede ser texto, imagen, o combinación de ellos, se ha incrementado notablemente. La necesidad de ofrecer al público una forma de enviar documentación que no sea en forma presencial, llevó a la generación de aplicaciones que son la puerta de entrada para estos documentos hacia las bases de datos.

La comunicación entre aplicaciones, por la que una base de datos “habla” con otra, es la que genera en nuestro caso particular, que el traspaso de información desde una base a otra se realice por medio de la copia de registros y por consiguiente la copia de documentos. El impacto de la copia de miles de registros en texto plano, tal vez es insignificante con los costos de almacenamiento que hoy hay, tal como lo muestra la Figura 1, el costo por almacenar 1TB de datos en AWS S3, en comparación, apenas supera el costo de tener un par proveedores de contenidos de streaming.

Imaginemos documentos de 16MB (usado por AWS S3 como base de cálculo), podemos almacenar 65.536 documentos a razón de 0,00036USD por cada uno por mes. No obstante, cuando hablamos de la copia, de miles de documentos de varios megabytes, y muchos de ellos producto de la duplicación, el impacto deja de ser insignificante para pasar a ser un problema en espacio y costo.

Para sumar a lo anterior, que solo habla del almacenamiento de documentos repetidos, hay que añadir que todo este almacenamiento es objeto de mantenimiento, de copias de seguridad, de políticas de retención. Si tomamos la expresión mínima de duplicidad de un documento, podemos decir que, para un determinado archivo, el mismo se halla en dos bases de datos, es objeto de dos tareas de backups, y esos backups (dos para el caso) son retenidos en la organización por X cantidad de tiempo.

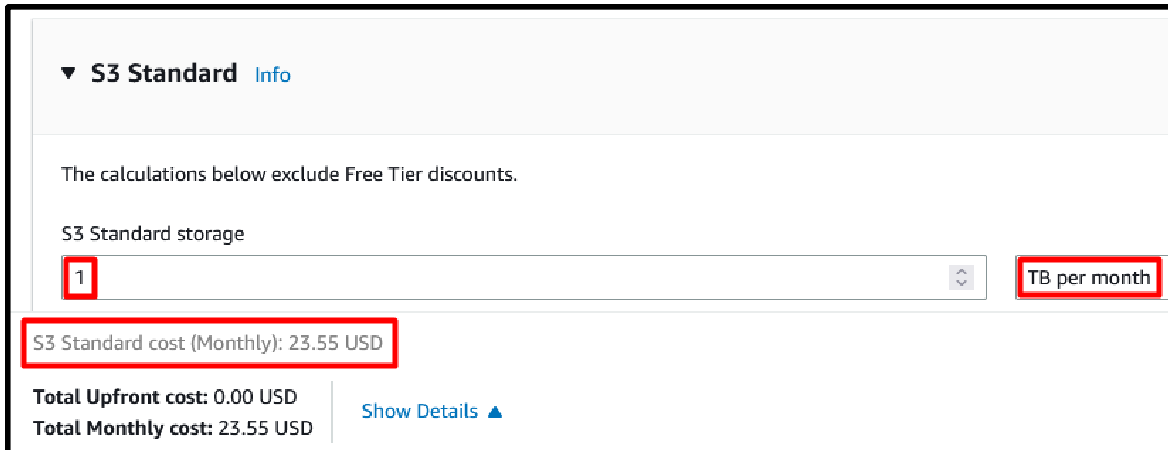


Figura 1: Costo de 1TB alojado en AWS S3[1]

En un ejemplo, 1 archivo de 5MB, ocupa en la organización 10MB de almacenamiento (por la duplicación), más 20MB de retención de backups (suponiendo que lo resguardamos 1 vez por semana, por 2 semanas de retención). Lo que sumando todo nos da, que por cada archivo ingresado a la organización de 5MB, requerimos en realidad 30MB de almacenamiento. Seis veces más de su tamaño es el espacio previsto para almacenar y retener un archivo duplicado.

La realidad es que nuestra organización objeto de estudio, hoy por su estructura de sistemas, triplica la documentación recibida sin modificarla en ningún paso previo. La misma información, con el exacto contenido de entrada, es almacenada tres veces, una en origen y dos veces más al menos.

2.2 Formulación del problema

Documentos (.doc, .rtf, .pdf) repetidos en múltiples bases de datos. Diseñar una única base de datos para almacenar documentos con las siguientes características:

- Independiente de las aplicaciones que la acceden.
- Que permita identificar al creador y propietario actual.
- Que permita versionado.

---

[1] https://calculator.aws/#/addService/S3
Valores al 17/03/2023

- Que permita búsquedas complejas de texto.

2.3 Delimitación del problema

Se establecen dos puntos en el tiempo, antes y después de la aplicación del presente proyecto.

Antes
Todos los documentos repetidos, almacenados en distintas bases de datos de la organización, quedan como están y serán objetos de estudio (en otro proyecto) acerca de:

- Ante la presencia de varios documentos, como determinar el valido.
- Migración de los mismos a la solución propuesta en este proyecto.
- Periodo de retención (conservación como históricos) de los documentos migrados.

Después
Los documentos que ingresen al sistema a partir de la aplicación del presente proyecto, ya cumplen con la característica principal de *unicidad*, y son almacenados teniendo en cuenta las características de este proyecto mencionadas en el punto 1.2.

2.4 La cuestión documental en el ambiente académico

Ya surgía la pregunta (Báez, 2007), sobre el lugar por excelencia para colocar documentos que se consideran terminados. Por terminados en nuestro caso, podemos considerar a los documentos que ya han sido firmados digitalmente y que poseen un carácter de inmutable.

Por otro lado, (Duque, 2020), también busca mitigar la problemática de archivos físicos y digitales. En nuestra organización objeto de estudio, los archivos físicos son los cientos de expedientes, que existen en dos formas: papel y digital. Y el formato digital es el que nos concierne, ya que es el que de manera sencilla se comparte por medio de la copia y duplicación, que termina siendo desmesurada y sin control.

Pero cuando surgen soluciones arbitrarias que tienden a borrar cualquiera y dejar uno, surgen pensamientos (Contreras Henao & Forero Guzmán, 2005) que afirman que, todos estos documentos son de un invaluable valor, lo cual justifica su archivo permanente y el cuidado de éstos para su futuro uso.

Todas estas cuestiones, también están presentes en este proyecto tecnológico, y serán abordadas algunas con más profundidad que otras, teniendo como objetivo el almacenamiento de documentos en un único lugar.

3. Objetivos

3.1 Objetivo General

Proporcionar a la organización objeto de estudio, en la forma de "*diseño de solución*", de una manera de gestionar los archivos de documentos, independientemente del sistema que lo genere, mediante un almacenamiento único, evitando la existencia de documentos repetidos.

3.2 Objetivos Específicos

Basados en los requerimientos de la organización, los cuales brevemente son:

- Generar una solución que termine con el almacenamiento de documentos repetidos.
- Utilizar bases de datos relacionales para todo lo relacionado con transacciones.
- Utilizar tecnologías que tengan características de motores de bases de datos.
- Evaluar el almacenamiento simple de objetos como alternativa de bajo costo.

Se establecen los siguientes objetivos:

- Diseñar una base de datos única multi-aplicación
- Combinar distintas tecnologías de almacenamiento de datos.
- Utilizar MongoDB GridFS como almacenamiento NoSQL.
- Utilizar MinIO (https://min.io/) como almacenamiento simple de objetos.
- Utilizar MySQL como almacenamiento relacional.
- Implementación de Elasticsearch como motor de búsqueda.

4. Caso de estudio

4.1 Situación actual

Nuestra organización objeto de estudio, almacena la información documental no estructurada, en bases de datos relacionales PostgreSQL utilizando campos de tipo blob, que son almacenados en la tabla del catálogo de PostgreSQL llamada pg_largeobject.

El catálogo pg_largeobject[2] contiene los datos que componen los "objetos grandes". Un objeto grande se identifica mediante un OID asignado cuando se crea. Cada objeto grande se divide en segmentos o "páginas" lo suficientemente pequeñas como para almacenarlas convenientemente como filas en pg_largeobject. La cantidad de datos por página se define como LOBLKSIZE (que actualmente es BLCKSZ/4 {8192/4}, o normalmente 2 kB).

La implementación de large objects es propensa al almacenamiento disperso: pueden perderse páginas, y pueden ser más cortas que los bytes LOBLKSIZE, aunque no sean la última página del objeto. Las regiones que faltan dentro de un objeto grande se leen como ceros.

Un mantenimiento diario es necesario a fin de reclamar el espacio de los large objects eliminados, huérfanos, perdidos, producto de actualizaciones, eliminaciones realizadas por las aplicaciones sobre estos objetos.

Ahora, lo anterior describe los efectos del uso de large objects en el motor de base de datos. Pero la cuestión de este proyecto tecnológico va más allá de este aspecto técnico relacionado al funcionamiento del motor, sino que apunta al problema de la dispersión documental en forma de large objects a través de todas las bases de datos de esta organización.

Miles de documentos, almacenados en forma de blobs, en sus correspondientes pg_largeobject, en cada una de las bases de datos, son parte del problema de la dispersión de almacenamiento y al mismo tiempo son objetos de mantenimiento necesario para mantener la salud del motor y

[2] postgresql.org: 52.30. pg_largeobject
https://www.postgresql.org/docs/current/catalog-pg-largeobject.html

la economía de almacenamiento (que puede ser del tipo costoso, si son discos de alta performance).

El problema se agrava aún más, cuando hablamos que un mismo documento, con exactamente el mismo contenido, se encuentra ocasionando las situaciones mencionadas de dispersión, en distintas bases de datos. En vez de tener esta situación de dispersión multiplicada por uno, la tenemos multiplicada por tantas bases de datos como en las que se encuentre repetido el documento. Pasamos entonces de una situación ideal de X1 veces, a Xn veces, siendo n el número de repeticiones del documento en bases de datos.

5. Marco teórico

Partimos por definir el concepto de documento o archivo de documento para el presente proyecto. Se entenderá por cualquier forma de archivo digital con formato del tipo *pdf, rtf, doc, odt*, etc. Esta definición es a fin de desambiguar el concepto de documento desde el punto de vista de bases de datos documentales como MongoDB, por ejemplo.

Los archivos de documentos (Microsoft, 2022), son datos de la organización que no se pueden estructurar (un texto, una imagen, un informe, un expediente), y por ello se los almacena como archivos.

A continuación, se describen de manera breve, tres tipos de tecnologías de almacenamiento de información, con miras a tener presente alternativas, ventajas y desventajas, a fin de poder evaluar un camino para resolver el problema planteado en el presente proyecto tecnológico.

5.1 Motores de base de datos relacionales

La integración de los datos no estructurados en las bases de datos relacionales proporciona **ventajas** como ser:

- Capacidad de almacenamiento integrado y copias de seguridad.
- Búsqueda integrada.
- Seguridad de acceso a los documentos centralizada en el motor.
- Simplificación y homogeneización en la arquitectura de acceso a datos.
- Los blobs (archivos de documentos) se gestionan de manera transaccional.

Por otro lado, podemos encontrar las siguientes **desventajas**:

- Límites en la capacidad de almacenamiento:
    - 1GB PostgreSQL.
    - 2GB SQL Server.
    - 4GB Oracle.
- Incremento considerable en el tamaño de la base de datos.
- Incremento de costos operativos como ser, mantenimientos, backups.

- Ineficiente utilización de recursos valiosos, como ser la asignación de almacenamiento de alta performance a blobs.

5.2 Motores de base de datos NoSQL tipo documentales

Las bases de datos documentales, están diseñadas alrededor de la noción abstracta de documento, y difieren entre las distintas implementaciones, en la forma en la que codifican los documentos, como ser XML, YAML y JSON, y para formatos binarios el BSON.

Al ser un tipo de bases de datos no relacionales, una de sus principales características es que la información no está contenida en tablas. Por el contrario, están pensadas para el almacenamiento de datos semiestructurados, los cuáles se organizan en documentos.

Entonces, las bases de datos documentales son capaces de almacenar información en diferentes formatos sin una estructura definida. A pesar de que su estructura es completamente distinta, estas bases de datos permiten realizar las mismas operaciones básicas que las bases de datos relacionales, esto es, añadir, actualizar o eliminar información, además de realizar las pertinentes consultas por parte del usuario.

Otra característica importante de las bases de datos NoSQL (Project Voldemort. A distributed database., s.f.), es que no intenta satisfacer las propiedades *Atomicidad, Consistencia, Aislamiento y Durabilidad* (ACID en adelante). Por ejemplo, al hablar de consistencia de datos, cuando se toman múltiples escrituras en simultáneo, a lo largo de múltiples servidores, se vuelve un problema difícil, donde una solución[3] que optan las bases de datos NoSQL, es tolerar la posibilidad de inconsistencia, y resolverlas en el momento de la lectura.

Por lo tanto, se pueden mencionar las siguientes como características, aunque no definitorias, de una base de datos NoSQL:

- No usan el modelo relacional (tampoco el lenguaje SQL).
- Open source.
- Están diseñadas para correr en grandes clústeres.

[3] Project Voldemort. Consistency & Versioning
http://www.project-voldemort.com/voldemort/design.html

- Basadas en las necesidades de la web del siglo 21.
- No utilizan esquemas, permitiendo a los campos ser añadidos a cualquier registro sin controles.

Almacenar datos no estructurados en bases de datos documentales, parece ser la opción adecuada si se pretende si se pretende sacar **ventajas** para:

- Almacenar y consultar información semiestructurada sin una estructura definida.
- Albergar numerosos tipos de datos.
- Escritura rápida, dando prioridad a la disponibilidad de la escritura sobre la consistencia de los datos.

No obstante, su principal **desventaja** es:

- No garantizan ACID.

5.3 Almacenamiento simple de objetos

Un objeto (MinIO, 2020) son datos binarios, llamados a veces Binary Large Objects (BLOB). Blobs pueden ser imágenes, archivos de audio, hojas de cálculo, o hasta a veces código ejecutable. Las plataformas de almacenamiento de objetos, ofrecen herramientas y capacidades dedicadas para almacenamiento, recuperación y búsqueda de blobs.

Muchos servicios (como AWS S3, MinIO) usan buckets para organizar los objetos. Un bucket es similar a un directorio en un *filesystem*, donde cada bucket puede almacenar un número arbitrario de objetos. Estos buckets a su vez se pueden anidar en múltiples niveles.

Podemos mencionar que los servicios como AWS S3, MinIO, poseen las siguientes **ventajas**:

- Soluciones muy efectivas y especializadas en el mercado.
- Económico.
- Fácil de configurar.
- Resistente (según la oferta), con copia de seguridad incluida.

Y en cuanto a sus **desventajas**:

- Riesgos de accesibilidad si la aplicación de misión crítica (por ejemplo, en caso de denegación de servicio ataca al proveedor o al punto de acceso a Internet).

- Riesgo de confidencialidad.
- Se puede necesitar seguridad de acceso adicional (por ejemplo, gestión de identidad de AWS).
- Se debe ocupar de los aspectos de coherencia transaccional (por ejemplo, qué hacer si falla una actualización de blob).

6. Proyecto tecnológico: Diseño

6.1 Diseño de base de datos

Un sistema de almacenamiento único multi-aplicación deberá tener en cuenta entidades como los ***binarios***[4,5,6], que son los documentos archivos en sí (pdf, rtf, etc.); ***archivos***, como entidad que contenga todos los metadatos de los binarios (extensión, versión, dueño, acceso, etc.); ***sistemas*** y ***usuarios***, entidades que harán uso de los archivos. A esto una entidad que relacione a los usuarios con los sistemas en conjunto para acceder a los archivos (***file_systems***), y una entidad controladora de los permisos de los sobre los archivos (***file_grants***).

Este diseño tiene como principales entidades a *files* y *binary*. La entidad *files* representa el concepto de **archivo** desde el punto de vista de los sistemas que lo usan. Es la puerta de entrada hacia el **binario**. La entidad *binary* es el archivo físico en sí. El modelo, admite que un *file* tenga muchos *binaries*. Esto es a fin de que un *file* pueda tener muchas versiones del archivo físico.

La premisa en la que se basa el diseño es que, *cualquier modificación de un binario*, es en realidad, ***una nueva versión***. Esto es así, ya que se trata de un repositorio de documentos firmados, cuya modificación o eliminación, debe preservar el anterior por cuestiones de auditoría, y siendo siempre valido el documento de la última versión.

El usuario accede a los binarios a través del modelo files. La entidad ***files***, mediante su máximo valor del atributo ***version***, indica cual es el último binario (***binary_id***) activo tanto para operaciones de *read* como de *write*.

4 Microsoft | Learn. ReadAllBytes: ofrece como ejemplo de archivo binario una imagen https://learn.microsoft.com/es-es/dotnet/visual-basic/developing-apps/programming/drives-directories-files/how-to-read-from-binary-files

[5]IBM Documentación. Tipos de archivos. Aclara como tipo de archivo normal a un texto, binario y ejecutable https://www.ibm.com/docs/es/aix/7.1?topic=files-types

[6] Descripción del Sistema Actual de Análisis (Mancini, Galasso, & Banchieri, 2018). Donde expresa *Esta tarea genera un archivo binario, con un formato propio del sistema* http://sedici.unlp.edu.ar/bitstream/handle/10915/73244/Documento_completo.pdf?sequence=1

Un ***servicio***, devuelve a los usuarios, siempre el registro que posea el mayor valor de versión, asegurando de esta manera que se brinda el acceso al último binario activo y no a los históricos. Otro ***servicio***, se encargará de gestionar las modificaciones, que serán **transformadas**, en la generación de un nuevo registro en la tabla ***files*** con una nueva **versión** del mismo, y la generación de un nuevo ***binario***.

Aquí empezamos a hablar de servicios. Estos deberán ser programados para que las aplicaciones accedan al sistema de almacenamiento único multi-aplicación y realicen las operaciones de creación, lectura, actualización y borrado (CRUD en adelante), bajo la premisa indicada anteriormente. Pero la programación de esos servicios escapa al presente proyecto tecnológico.

A continuación, en la Figura 2, se expone el diseño de la base de datos central del almacenamiento único multi-aplicación. Por ser esta la encargada de coordinar todo el proceso, la misma se basa sobre un motor relacional que nos garantice las propiedades de ACID.

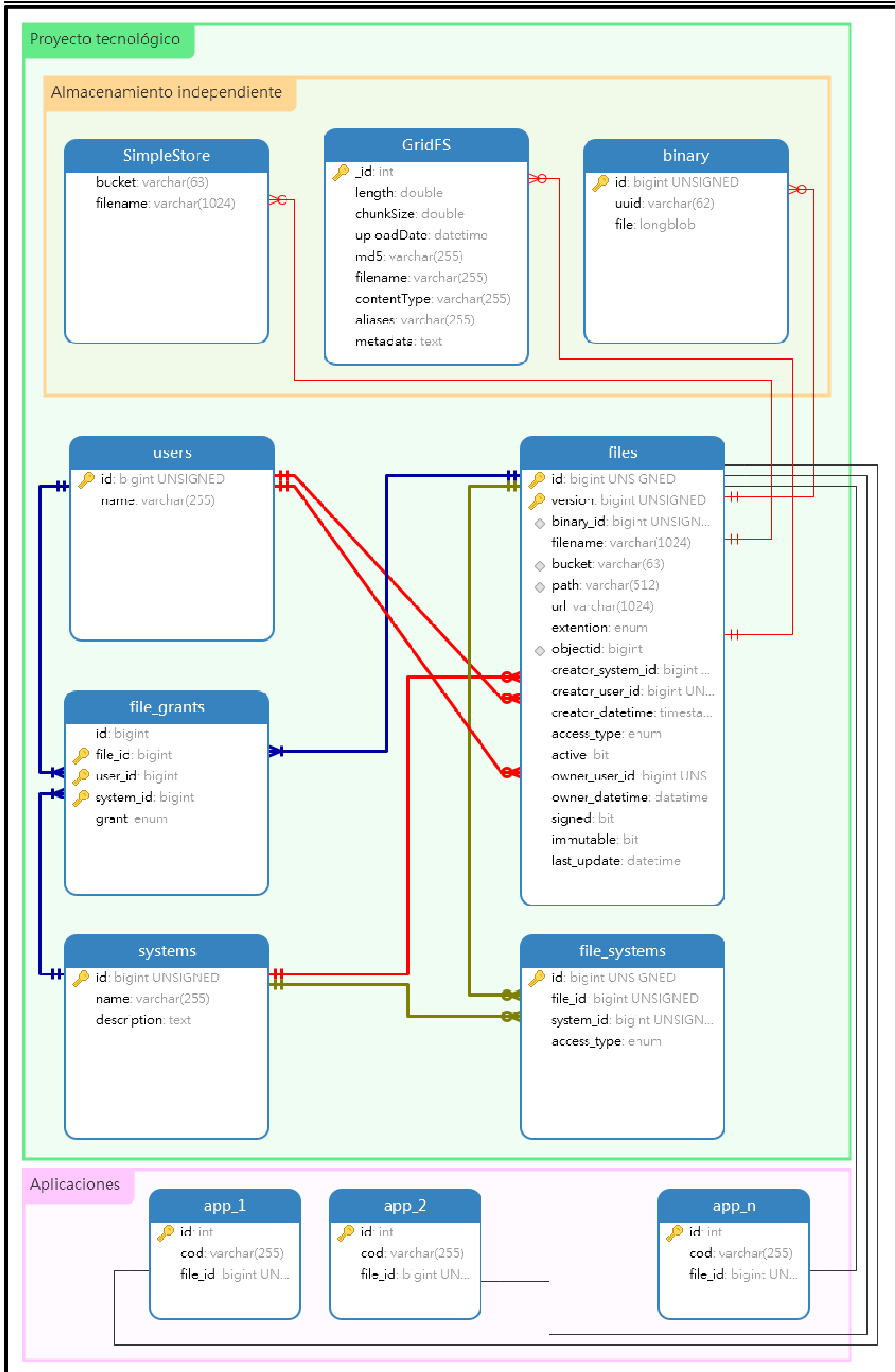


Figura 2: Almacenamiento único multi-aplicación

6.2 Definiciones de tablas

6.2.1 Almacenamiento independiente

El lugar donde residirán de forma definitiva los documentos, acorde a lo que venimos mencionando en este proyecto pueden ser de tres tipos, relacional, NoSQL y almacenamiento simple de objetos.

Ahora, es cuestión de que la organización, decida cuál es el apropiado según factores externos al presente proyecto de solución, como ser:

- Costo de almacenamiento on premises, en el caso de optar por una solución relacional o NoSQL, por ejemplo.
- Costo y grado de confianza, en el caso de optarse por un almacenamiento simple en la nube.

El proyecto contempla los tres tipos de almacenamiento, como posibles repositorios de los documentos de la organización. Gracias a la base de datos relacional del *almacenamiento único multi-aplicación*, se podrá optar por almacenar en una sola ubicación, o en todas al mismo tiempo. El lugar definitivo de los documentos, es relevante solo en cuestiones de costos y performance. La versatilidad del modelo presentado en la Figura 2, permite que la ubicación final de un documento sea irrelevante, pudiendo ampliarse a otros tipos. No obstante, nos circunscribimos a los tres tipos mencionados, por una cuestión de requerimientos (de la organización) mencionados anteriormente (1. Introducción).

Es el archivo binario, documento, *'doc', 'docx', 'rtf', 'pdf'*. Solo posee un *identificador* y el campo *objeto largo* que contiene el binario. En todos los casos habrá un identificador y el campo atributo que dependiendo la tecnología contendrá el documento.

Es por ello que se identifica como un almacenamiento independiente del modelo. Un atributo en la tabla files (***binary_id***, ***path***), será el encargado en mantener la relación hacia estos elementos.

6.2.1.a Base de datos relacional (MySQL)

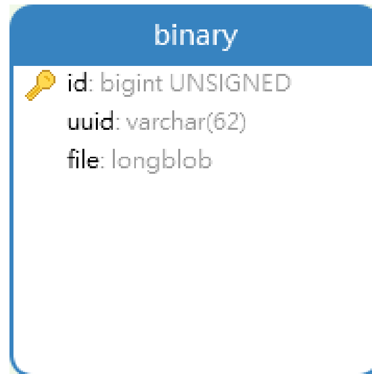


Figura 3: Definición de la tabla binary,
para el caso de usarse un repositorio de documentos relacional

El almacenamiento de documentos en una base de datos relacional, como se muestra en la Figura 3, solo requiere de un campo del tipo ***longblob*** (MySql). En la definición se incluye un identificador único (*id*) autonumérico, y también un identificador (*uuid*) que se utiliza como nombre del documento.

6.2.1.b Base de datos NoSQL (MongoDB)

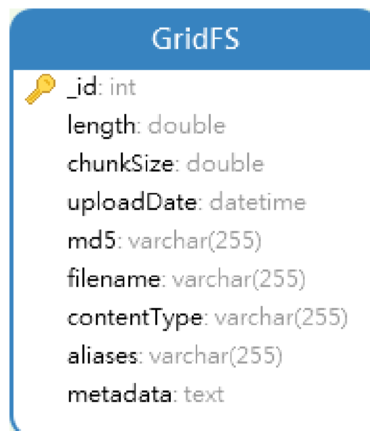


Figura 4: Definición del objeto GridFS,
para el caso de usarse un repositorio NoSQL

En el caso del almacenamiento basado en GridFS (MongoDB), los documentos son almacenados dentro de una colección, como se muestra en la Figura 4, la cual contiene un identificador único (*_id*), más otros atributos como ser la longitud del documento en bytes (*length*), el tamaño de cada porción en la que se subdivide el archivo (*chunkSize*), la fecha en la que fue creado (*uploadDate*), la forma humana de identificar el archivo (*filename*), etc.

6.2.1.c SimpleStore

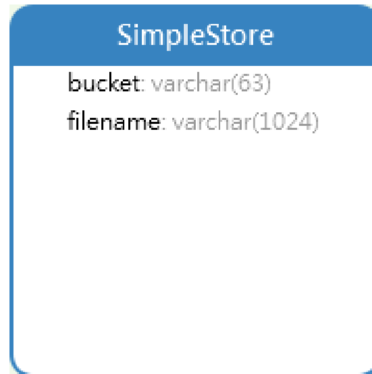


Figura 5: Definición del un objeto SimpleStore,
para el caso de usarse almacenamiento *simple* como repositorio de documentos

Finalmente, el guardado de documentos en un *almacenamiento simple* (AWS S3, Minio), requerirá de dos atributos (Figura 5). El nombre de su contenedor (*bucket*) y el nombre del archivo (*filename*). Con estos dos atributos, podemos realizar todas las operaciones necesarias sobre los documentos almacenados en este medio.

6.2.2 Files

Posee información sobre los metadatos del binario almacenado. Su diseño, esquematizado en la Figura 6, nos presenta los siguientes atributos:

id: autonumérico. Junto con *version* forman la clave primaria de la tabla.

version: id de la versión del documento. Número correlativo que se incrementa en uno cada vez que se genera una nueva versión del documento. Son clave primaria junto con *id*.

binary_id: el identificador de la tabla relacional contenedora del archivo binario. Posee la propiedad *unique*, por lo que solo puede haber, un solo *binary_id*, por cada par *id+versión*.

filename: es el nombre[7] del objeto.

bucket: nombre del bucket[8] contenedor.

---

[7] Creating object key names
https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-keys.html
[8] Bucket naming rules
https://docs.aws.amazon.com/AmazonS3/latest/userguide/bucketnamingrules.html

path: ruta del binario almacenado. Posee una longitud de 512, que permite almacenar rutas del tipo AWS S3[9].

bucket + **path**: poseen la propiedad *unique*, por lo que solo puede haber, un solo par bucket+path, por cada par *id+versión*.

Figura 6: Definición de la tabla files

url: almacena la ruta compartida para un objeto.

extention: tipo de extensión del binario almacenado. Enumeración de valores: 'doc','docx','rtf','pdf'.

objectid: almacena el identificador al documento MongoDB. Posee la propiedad *unique*, por lo que solo puede haber, un solo *objectid*, por cada par *id+versión*.

creator_system_id: indica el sistema que creó el archivo.

creator_user_id: indica el usuario creador del archivo.

creator_datetime: fecha hora de creación del archivo.

access_type: acceso global al archivo. Enumeración de valores: 'private','shared'.

active: true, si está activo; o false, en caso de ser histórico.

owner_user_id: indica el dueño actual del archivo.

owner_datetime: indica a partir de qué fecha hora es dueño.

signed: indica si el documento ha sido firmado digitalmente.

immutable: si está en true, ya no se pueden generar nuevas versiones del archivo.

last_update: última fecha de modificación del archivo.

[9] Querying data using select: Note https://docs.aws.amazon.com/AmazonS3/latest/userguide/querying-glacier-archives.html

El modelo **files** se basa en el concepto que, cualquier modificación al archivo, es una nueva versión, es un nuevo binario, estableciendo como activo este último y estableciendo como histórico todos los anteriores.

6.2.3 Systems - Users

Estas entidades identifican a los actores. Los sistemas (Figura 7) y los usuarios (Figura 8), que a través de los primeros accederán a los archivos. Solo los sistemas que estén agregados como registros de la tabla systems, tendrán la capacidad de poder generar nuevos documentos.

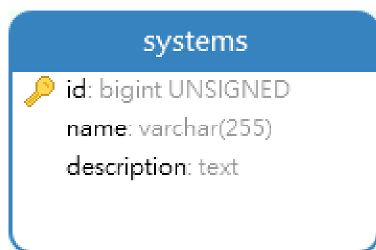


Figura 7: Definición de la tabla systems

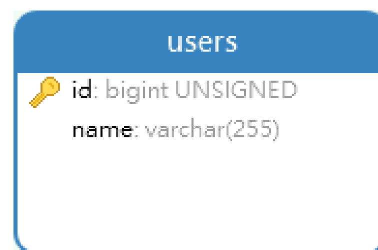


Figura 8: Definición de la tabla users

6.2.4 File_systems

Esta entidad (Figura 9) controla el acceso a los binarios, relacionado *sistema* y *archivos*. Permite identificar a todos los sistemas que acceden a un determinado archivo. También proporciona un nivel superior de control, por encima del acceso individual de cada usuario, estableciendo de manera general si un sistema puede acceder al archivo y en qué forma.

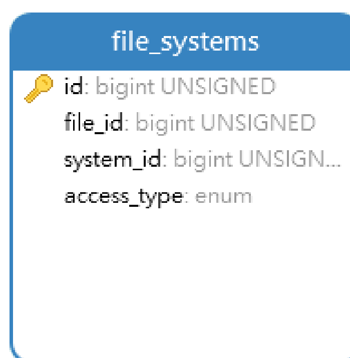


Figura 9: Definición de la tabla file_systems

El atributo *access_type*, puede optar por los siguientes valores:

- **'none'**: bloque el acceso a al archivo, para *system_id*. Este bloqueo es superior a los permisos individuales de los usuarios.

- **'readonly'**: permite el acceso al archivo, para *system_id*, en modo de lectura. Este permiso es superior a los permisos individuales de los usuarios. Habilita a los usuarios a acceder en modo *lectura* como máximo.
- **'readwrite'**: permite el acceso a los archivos, para *system_id*, en modo *lectura* y *escritura*. Este permiso habita a los usuarios, dependiendo sus atributos particulares, acceder para leer y/o escribir.

6.2.5 File_grants

La entidad *file_grants* (Figura 10), administra el acceso a los archivos por medio del atributo *grant*, relacionando las entidades *archivo*, *usuario* y *sistema*.

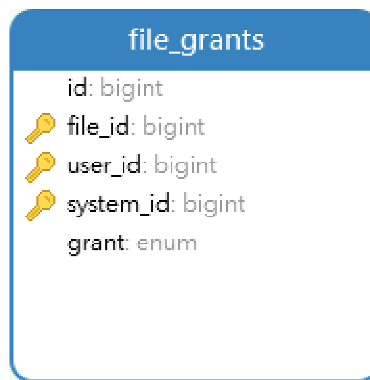


Figura 10: Definición de la tabla file_grants

El atributo grant, puede optar por los siguientes valores: **R**(*read/lectura*), **W**(*write/escritura*), **RW**(*read-write/lectura-escritura*).

Aquí se puede dar una serie de situaciones, dependiendo la combinación *usuario/sistema*. Por ejemplo:

> Un usuario, logueado al sistema *SYS1*, puede tener acceso R sobre el archivo con *file_id=1*. Pero el mismo usuario, logueado al sistema *SYS2*, puede tener acceso RW sobre el mismo *file_id=1*.

6.3 Diseño de servicios para el acceso a documentos

A fin de que exista un elemento controlador del acceso a los documentos, regulador de los permisos, y monitor de las acciones que se ejerzan sobre los mismos; se propone la

implementación de servicios encargados de mediar entre los usuarios/sistemas y el almacenamiento definitivo de documentos.

Estos servicios se basan en la información contenida en las tablas *users*, *systems*, *file_grants*, *file_systems* y finalmente *files*. Por ejemplo, solo los sistemas y usuarios contenidos en estas tablas, serán capaces de generar y almacenar documentos. De manera similar, las capacidades que los usuarios y sistemas dispondrán sobre los documentos serán las almacenadas en *file_grants*. La información del alcance compartido de los documentos se aloja en *file_systems*.

La propuesta de servicios es independiente del tipo de almacenamiento optado. Más aún es compatible con los tres tipos de almacenamientos propuestos (RDBMS, NoSQL document database - MongoDB, S3 simple store). En definitiva, los servicios son el medio de comunicación desde y hacia los documentos.

Se deberán desarrollar servicios específicos, a fin de que las distintas aplicaciones, puedan lograr una correcta y coordinada manipulación de los documentos almacenados en el almacenamiento único multi-aplicación. Correcta en el sentido que siempre se recupere la última versión del documento almacenado. Coordinada en el sentido que puedan accederlo basados en los permisos que poseen.

Teniendo presente el acrónimo CRUD, se definirán los siguientes servicios:

- C: (create) Creación del documento.
- R: (read) Recuperación del documento almacenado.
- U: (update) Generación de una nueva versión del documento.
- D: (delete) Marcado como no activo de manera lógica.

Tanto el update como el delete, son de manera lógica, a fin de conservar el elemento físico por cuestiones de auditoría hacia atrás.

Así mismo, se añaden el servicio de:

- Asignación de permisos.

6.3.1 (create) Creación del documento

Un documento puede ser creado (como se muestra en la Figura 11) por un usuario *logueado* (user_id) correctamente a un sistema (system_id), teniendo en cuenta que, en la base de datos *multi-aplicación*:

- El sistema (*system_id*) exista en la tabla *systems*.
- El sistema (*system_id*) tenga permiso de ***readwrite***, en el atributo *access_type*, de la tabla *file_systems*.
- El usuario (*user_id*) tenga permiso de ***RW***, en el atributo *grant*, de la tabla *file_grants*.

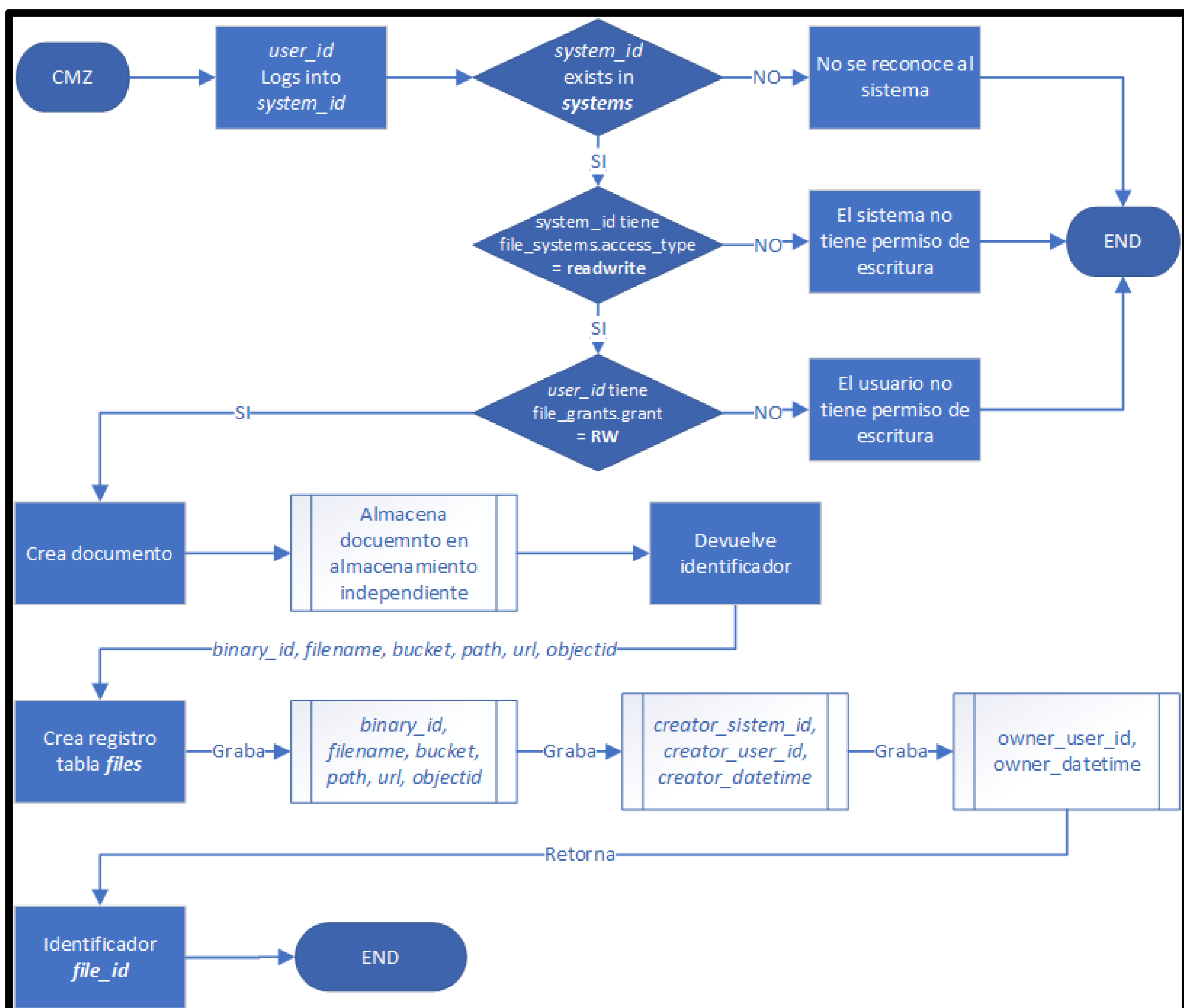


Figura 11: Diagrama de flujo: creación de un documento

El documento creado tiene las siguientes propiedades:

- Relativas a la ubicación del documento: ***identificadores*** hacia los distintos almacenamientos.
  - binary_id
  - filename
  - bucket
  - path
  - url
  - objectid
- Relativas a la creación: indican ***quién***, ***desde dónde***, y ***cuándo***, fue creado el documento. Estos atributos permanecen inalterables a lo largo de la vida del documento.
  - creator_system_id
  - creator_user_id
  - creator_datetime
- Relativos a la durabilidad del documento: indican el tipo de acceso al documento, si es la versión activa o no, si posee firma digital, si ya no se lo puede tomar para crear nueva versión, su número de versión y su última fecha de actualización.
  - access_type
  - active
  - signed
  - inmutable
  - last_update
  - version
- Relativos al propietario actual: indica quien es el actual dueño del documento. Quien es el usuario que puede compartir el documento y crear una nueva versión del mismo
  - owner_user_id
  - owner_datetime

6.3.2 (read) Recuperación del documento almacenado

El servicio que recupera los documentos (Figura 12) desde el almacenamiento (RDS, S3, NoSQL), parte del login de un usuario a un determinado sistema, con la posterior solicitud de

un documento por medio de su *file_id*. A fin de poder recuperar el mismo, se requerirá que, en la base de datos *multi-aplicación*:

- El sistema (*system_id*) exista en la tabla *systems*.
- El sistema (*system_id*) tenga permiso mayor o igual que ***readonly***, en el atributo *access_type*, de la tabla *file_systems*.
- El usuario (*user_id*) tenga permiso mayor o igual que ***R***, en el atributo *grant*, de la tabla *file_grants*.

Si el conjunto de atributos *file_id*, *user_id*, *system_id* son validados, y cumplen con los requisitos mínimos para acceder al documento, establecidos en las tablas *file_grants* y *file_systems* de la base de datos *multi-aplicación*, entonces se procederá a acceder a su ubicación física y se devolverá al usuario el documento.

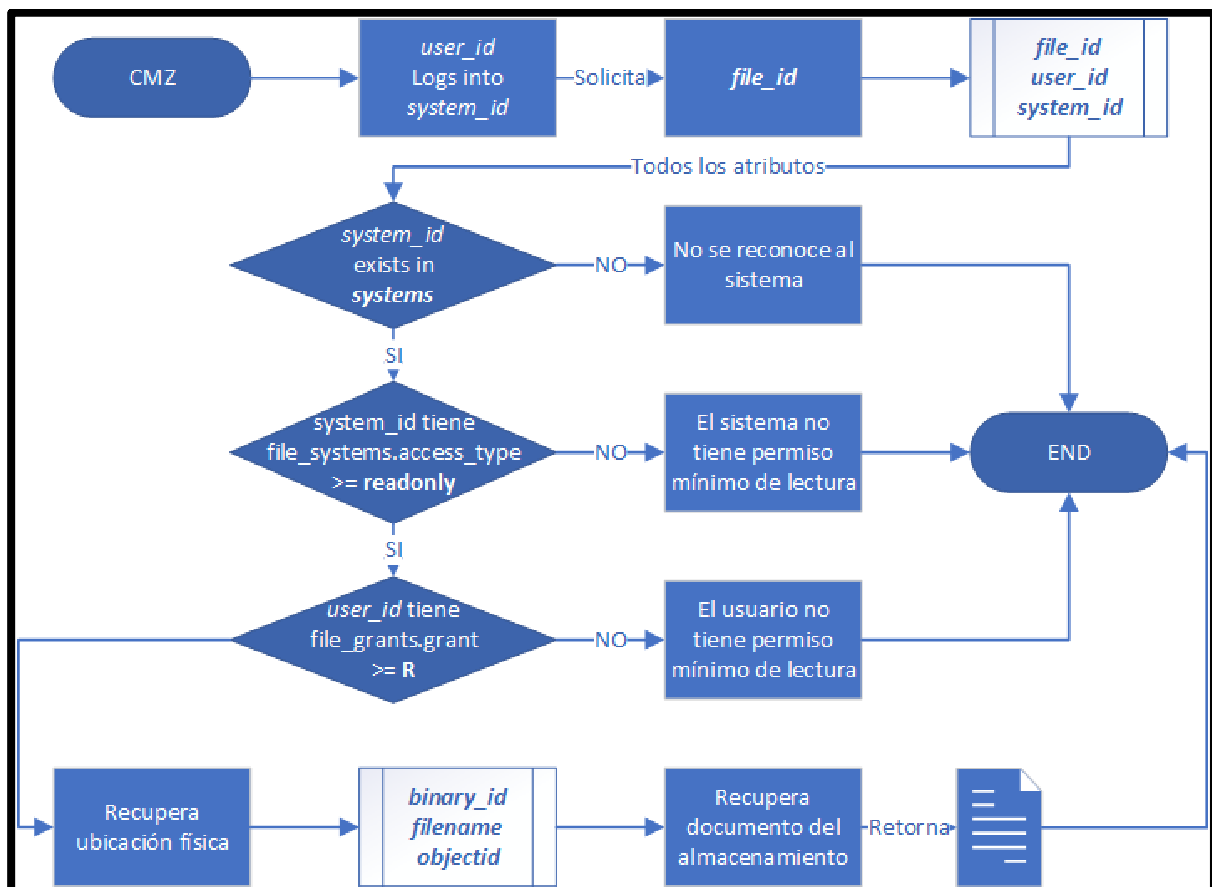


Figura 12: Diagrama de flujo: recuperación de un documento

6.3.3 (update) Generación de una nueva versión del documento

La actualización de un documento, es en realidad, la generación de un nuevo documento, el cual conserva el mismo ***file_id***, pero un identificador de ***versión***[10] incrementado en uno con respecto al anterior. Esto es a fines de auditoría interna y requisitos de la organización. Las versiones anteriores no están disponibles para el usuario o los sistemas, pero si están disponibles para los servicios de este sistema de almacenamiento único multi-aplicación.

Luego se pensará una manera de mantener al mínimo la cantidad de versiones de documentos que se almacenan con propósitos históricos o de auditoría, y se implementará un procedimiento de eliminación, conservando siempre el de mayor ***versión*** disponible. Este tema no es parte de este proyecto tecnológico, ya que el nivel de versionado a conservar, depende de políticas de la organización y cuestiones legales; por lo cual excede a los propósitos del presente.

Se entiende que, en la tabla *files*, un documento con un determinado ***file_id***, para su máximo valor en el atributo ***versión***, es el último documento que se considera como válido para ser devuelto a los usuarios/sistemas que lo solicitan.

A fin de poder actualizar un documento, un usuario debe estar *logueado* (user_id) correctamente a un sistema (system_id), solicitar de la tabla *files*, el [id+max(versión)], del documento que pretende actualizar.

Luego, tal como se esquematiza en la Figura 13, se deberá tener en cuenta que, en la base de datos *multi-aplicación*:

- El sistema (*system_id*) exista en la tabla *systems*.
- El sistema (*system_id*) tenga permiso de ***readwrite***, en el atributo *access_type*, de la tabla *file_systems*.
- El usuario (*user_id*) tenga permiso de ***RW***, en el atributo *grant*, de la tabla *file_grants*.

Cumpliéndose los mencionados requerimientos, se puede crear un nuevo documento, el cual como ya se mencionó, mantendrá el mismo *files.id*, pero incrementara en uno el *files.version*.

[10] What's new in PostgreSQL 9.2: 1.1 Index-only scans
El ejemplo, se muestra una búsqueda sobre una tabla de 100 millones de registros, y se plantean dos situaciones: sin indexación y *con indexación*. Los resultados son de 364.390,127 ms y *4.297,405 ms*, respectivamente.

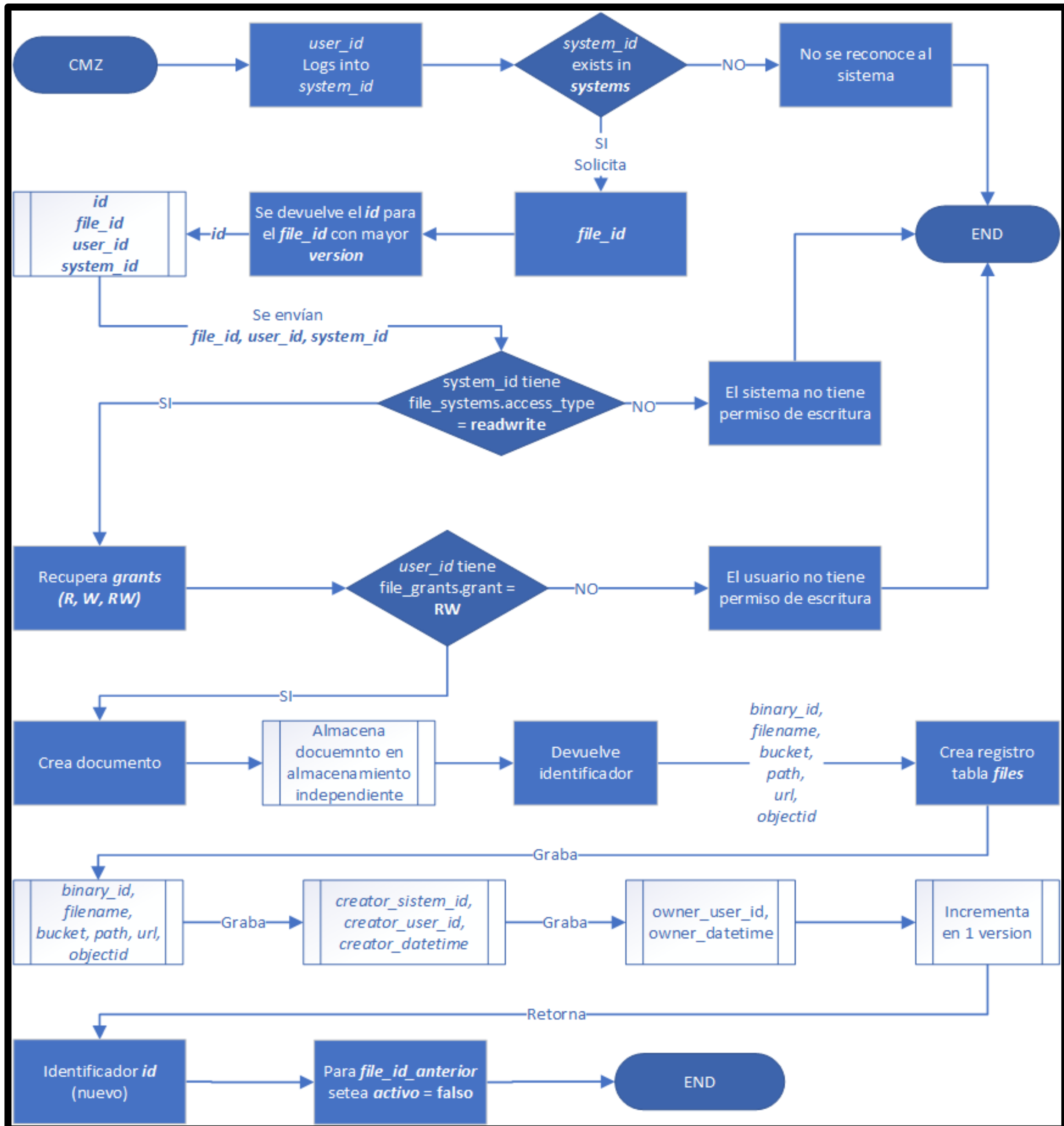


Figura 13: Diagrama de flujo: actualización de un documento

6.3.4 (delete) Marcado como no activo de manera lógica

La eliminación de un documento (Figura 14), similar a la modificación, es una operación lógica. Se realiza estableciendo el atributo ***active*** en ***false***. Se busca mantener el documento anterior en el almacenamiento para fines de auditoría, mientras que lógicamente el mismo ya no es visible.

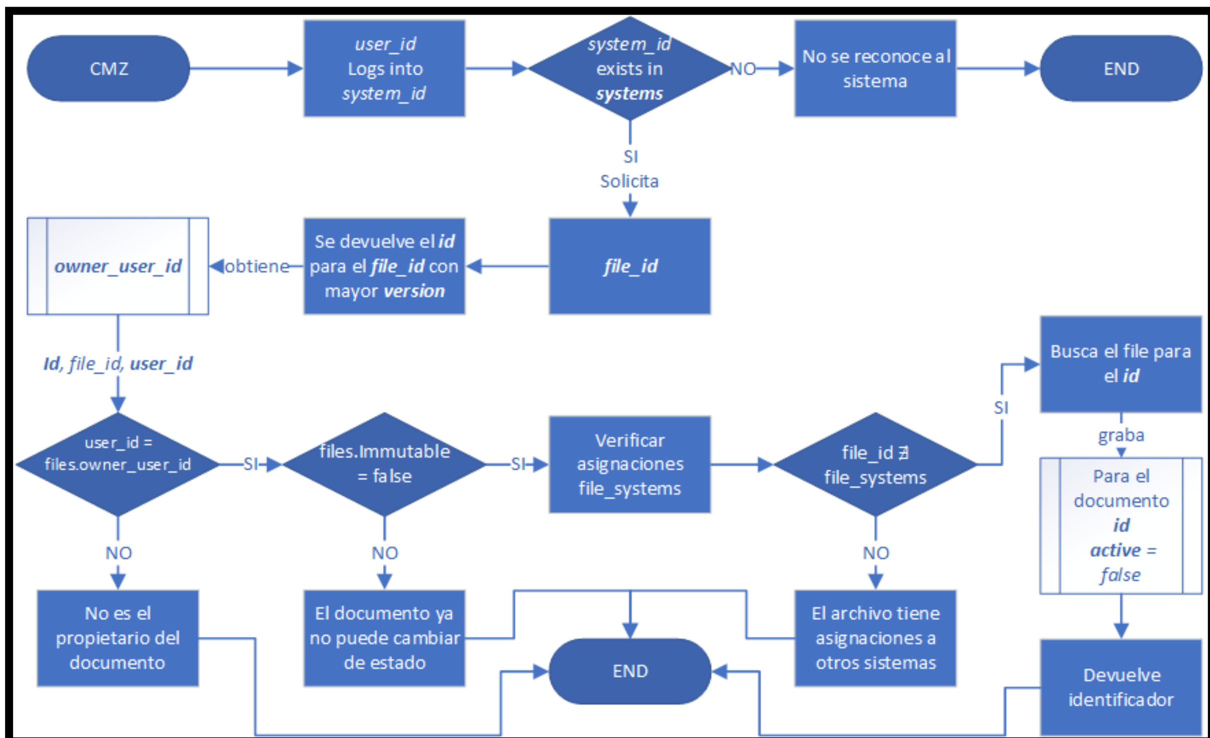


Figura 14: Diagrama de flujo: eliminación de un documento

No obstante, para los servicios del sistema de *almacenamiento único multi-aplicación*, si es posible acceder al mismo, si fuese necesario.

El proceso de borrado parte con un usuario logueado al sistema. Luego el mismo solicita un documento por medio de su ***file_id***. Se retorna el mismo y se recupera el ***owner_user_id*** que identifica al dueño actual del documento. El siguiente parámetro que se recupera es el ***immutable***, el mismo debe estar en false (*immutable=true* indica que ya no se puede modificar el documento). Se verificará también que no existan asignaciones del documento a otros sistemas en la tabla *file_systems*. Solo puede existir la referencia al sistema desde el que se accede para la eliminación.

Si el ***user_id*** del usuario que lo recuperó, y el ***owner_user_id*** coinciden, y el parámetro ***immutable*** se encuentra en ***false***, y finalmente no existen referencias al documento desde otros sistemas, se procede al establecimiento del ***active = false***. En cualquier otro caso, no se puede proceder con la eliminación del documento.

6.3.5 Asignación de permisos

La asignación de permisos (Figura 15) es el proceso en el cual, el usuario propietario del documento (*owner_user_id*) inserta en la tabla *file_grants* un nuevo registro permitiendo el acceso hacia el *file_id*, para el usuario *user_id*, desde el sistema *system_id*, con los permisos descritos en *grant*.

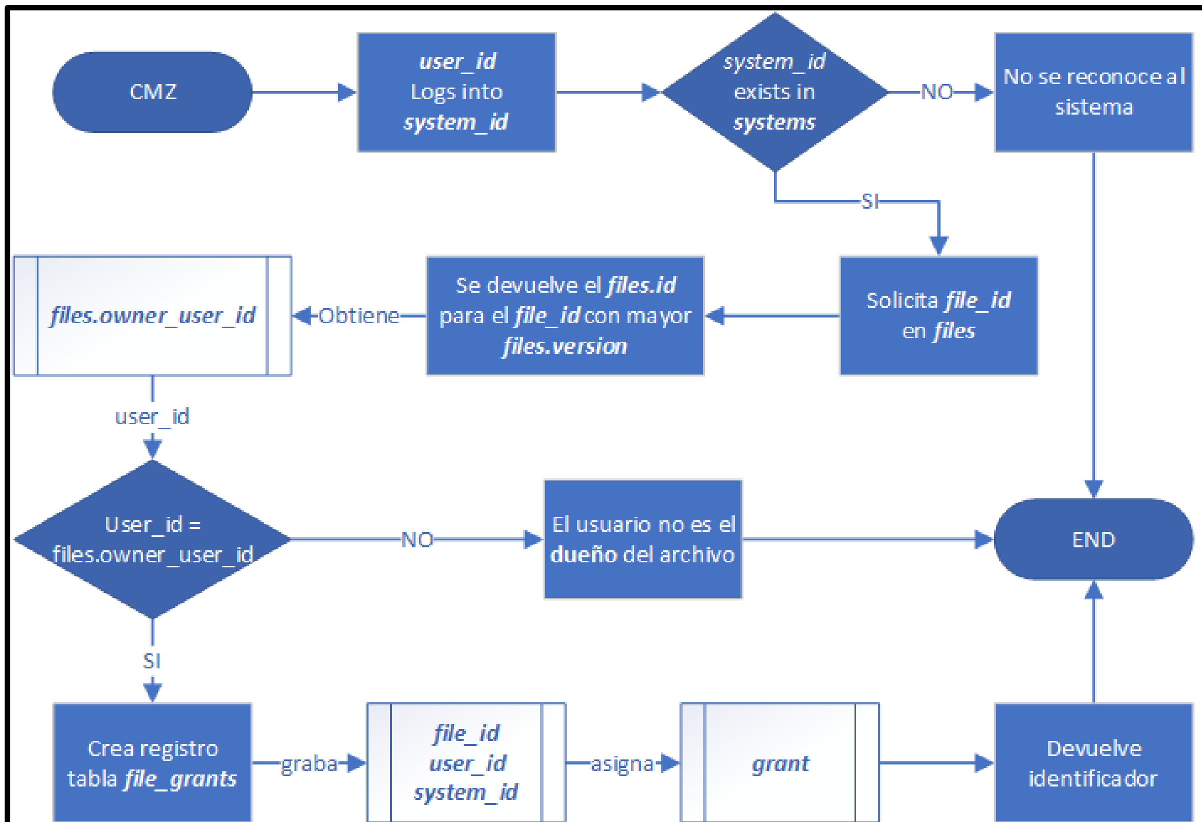


Figura 15: Diagrama de flujo: asignación de permisos de un documento

Esta tabla, será consultada cada vez que se solicite un documento (*file_id*). Se tiene en cuenta en todo momento el par usuario (*user_id*), sistema (*system_id*). Un usuario desde un sistema, podría *tener acceso* a un documento, y no tenerlo desde otro sistema. Es por ello, que la validación se realiza en conjunto usuario/sistema.

7. Proyecto tecnológico: Implementación del almacenamiento único multi-aplicación

La esencia del presente proyecto tecnológico, radica en la posibilidad de almacenar documentos indistintamente de la aplicación de la cual provengan, en cualquier tipo de almacenamiento. De esta forma cualquier documento puede ser compartido por distintas aplicaciones, accediendo por su identificador único en el sistema (***file_id***), sin la necesidad de duplicar el mismo y ser almacenado en otra base de datos dependiente de la aplicación con la cual se comparte.

A continuación, en la Figura 16, se esquematiza el proyecto tecnológico.

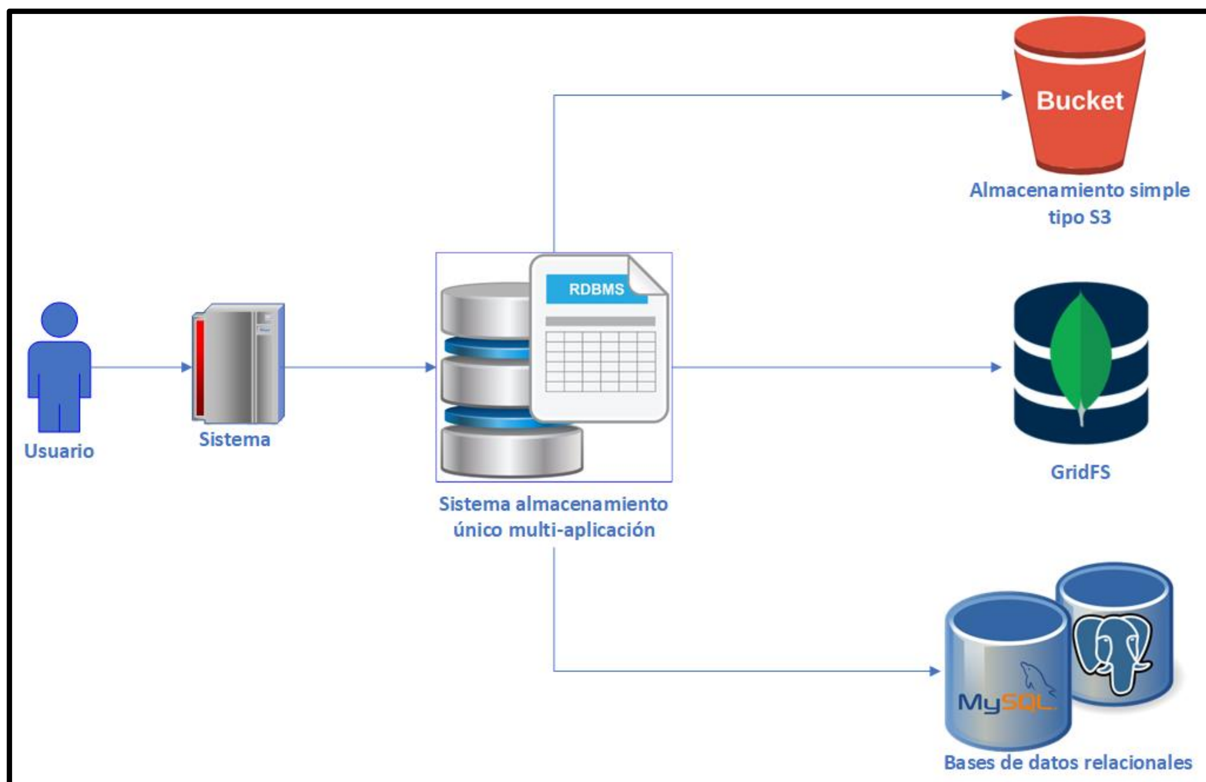


Figura 16: Esquema del proyecto tecnológico

Pero esto solo se puede lograr si se implementa un sistema que coordine todos en conjunto. Un sistema que va a estar compuesto de una base de datos relacional central que contenga la información que relaciona los usuarios, los sistemas, los accesos y los documentos con sus distintas versiones y ubicaciones.

Esta base de datos central, necesariamente será relacional, ya que debemos asegurar ACID sobre las relaciones entre usuarios, sistemas, permisos y documentos. No contiene los documentos como tales, sino que contiene la información necesaria para llegar a ellos. Operará en conjunto con los servicios explicados en el punto 6, y estos servicios dependerán de la información contenida en ella.

La tabla que coordina los documentos (Figura 17), con sus distintas posibilidades de ubicaciones es ***files***[11], en ella podemos encontrar su identificador único ***id***, que se enlaza al identificador único del binario ***binary_id*** para el caso de un almacenamiento de documentos en DBRMS; también el ***filename*** y ***bucket*** para el caso del almacenamiento en un Simple Storage, y finalmente el ***objectid*** para el caso NoSQL (MongoDB).

```
SELECT id, binary_id, filename, bucket, path, objectid FROM `files`;
```

Figura 17: Consulta a la tabla files

El resultado del *query* anterior, se muestra en la Figura 18, a continuación.

id	binary_id	filename	bucket	path	objectid
1	1	fac7b3b4-2f5f-3f4c-a0b3-4766bbb6acea.pdf	multi-app-store.binaries	fac7b3b4-2f5f-3f4c-a0b3-4766bbb6acea.pdf	62f66f7347b5c476820cd051
2	2	7a539292-13f6-3092-bcfb-88b5216ed214.pdf	multi-app-store.binaries	7a539292-13f6-3092-bcfb-88b5216ed214.pdf	62f66f7447b5c476820cd054
3	3	ba4326b4-59d6-3078-8cd7-413ab5b9ed56.pdf	multi-app-store.binaries	ba4326b4-59d6-3078-8cd7-413ab5b9ed56.pdf	62f66f7547b5c476820cd056
4	4	3af36321-1b97-39f8-8c57-1e2c04d9eeb9.pdf	multi-app-store.binaries	3af36321-1b97-39f8-8c57-1e2c04d9eeb9.pdf	62f66f7647b5c476820cd058
5	5	9129c515-309a-3849-8cc9-1486fbb0946d.pdf	multi-app-store.binaries	9129c515-309a-3849-8cc9-1486fbb0946d.pdf	62f66f7747b5c476820cd05a
6	6	03dcd56c-ca16-3415-84c0-b7be4120b57e.pdf	multi-app-store.binaries	03dcd56c-ca16-3415-84c0-b7be4120b57e.pdf	62f66f7847b5c476820cd05c
7	7	49c06d11-7cea-3e36-b0f6-edb1a54022d9.pdf	multi-app-store.binaries	49c06d11-7cea-3e36-b0f6-edb1a54022d9.pdf	62f66f7947b5c476820cd05e
8	8	ca4d5df3-3cf2-3342-8662-1e1bfbbcee0e.pdf	multi-app-store.binaries	ca4d5df3-3cf2-3342-8662-1e1bfbbcee0e.pdf	62f66f7a47b5c476820cd060
9	9	b3bb9545-904a-3094-b9b6-51d43960184d.pdf	multi-app-store.binaries	b3bb9545-904a-3094-b9b6-51d43960184d.pdf	62f66f7c47b5c476820cd062
10	10	76200d3e-3978-3939-867f-9f2a3aea4ac5.pdf	multi-app-store.binaries	76200d3e-3978-3939-867f-9f2a3aea4ac5.pdf	62f66f7d47b5c476820cd064

Figura 18: Contenido de la tabla files

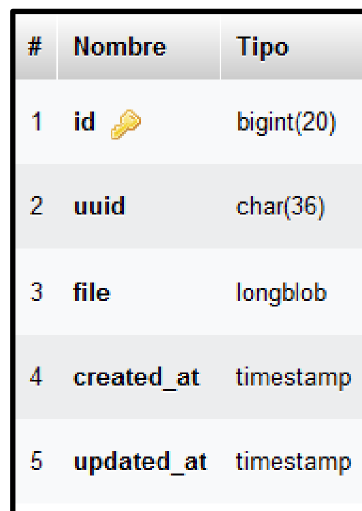
[11] La decisión del lugar donde residirán los archivos de documentos, si es en un almacenamiento relacional, NoSQL, o almacenamiento simple, se explicó en *6.2.1 Almacenamiento independiente*

7.1 Implementación utilizando base de datos relacional

La implementación basándonos en un DBRMS para almacenar los documentos definitivos sería, una tabla binaries con la siguiente estructura.

El campo ***id*** de esta tabla ***binaries*** (Figura 19), es el ***binary_id*** de la tabla ***files***. Ellos son los que relacionan a los binarios con el usuario/sistema que lo accede a través de la tabla files.

Esta tabla contiene como campo destacable el denominado ***file*** que es de tipo ***longblob***. Este permite almacenar al binario en sí.



#	Nombre	Tipo
1	id	bigint(20)
2	uuid	char(36)
3	file	longblob
4	created_at	timestamp
5	updated_at	timestamp

Figura 19: Diseño tabla files

Otro campo importante es el ***uuid***, el cual es el nombre único del binario dentro del sistema de almacenamiento multi-aplicación. A continuación (Figura 20), ejemplo de contenido de la tabla *files*.

id	uuid	file	created_at	updated_at
1	fac7b3b4-2f5f-3f4c-a0b3-4766bbb6acea	[BLOB - 2.0 KB]	2022-08-12 15:19:13	2022-08-12 15:19:13
2	7a539292-13f6-3092-bcfb-88b5216ed214	[BLOB - 2.0 KB]	2022-08-12 15:19:15	2022-08-12 15:19:15
3	ba4326b4-59d6-3078-8cd7-413ab5b9ed56	[BLOB - 2.1 KB]	2022-08-12 15:19:16	2022-08-12 15:19:16
4	3af36321-1b97-39f8-8c57-1e2c04d9eeb9	[BLOB - 2.3 KB]	2022-08-12 15:19:17	2022-08-12 15:19:17
5	9129c515-309a-3849-8cc9-1486fbb0946d	[BLOB - 2.4 KB]	2022-08-12 15:19:18	2022-08-12 15:19:18

Figura 20: Ejemplo contenido de la tabla files

7.2 Implementación utilizando MongoDB

La implementación basada en NoSQL, con el caso particular de MongoDB, se realiza por medio de la especificación GridFS. La misma se utiliza para almacenar y recuperar archivos que excedan el límite de 16MB del tamaño de los BSON-documents.

En lugar de almacenar un archivo en un solo documento, GridFS divide el archivo en partes o chunks (fragmentos), y almacena cada chunk como un documento separado. De forma predeterminada, GridFS utiliza un tamaño de chunk predeterminado de 255 kB; es decir, GridFS[12] divide un archivo en fragmentos de 255 kB con la excepción del último fragmento. El último trozo es tan grande como sea necesario.

GridFS usa dos colecciones para almacenar archivos. Una colección almacena los fragmentos de archivos (Figura 22) y la otra almacena metadatos de archivos (Figura 21).

```
_id: ObjectId('62f66f7347b5c476820cd051')
chunkSize: 261120
filename: "fac7b3b4-2f5f-3f4c-a0b3-4766bbb6acea.pdf"
contentType: "application/pdf"
metadata: Object
  created_at: 2022-08-12T15:19:15.141+00:00
  updated_at: 2022-08-12T15:19:15.141+00:00
  uuid: "2079e688-1a52-11ed-b27e-000c29cd5ee3"
  downloads: 0
length: 2056
uploadDate: 2022-08-12T15:19:15.374+00:00
md5: "ba98dd0a2d7d2c904a91bb3a66b248ec"
```

Figura 21: Colección que almacena los metadatos

```
_id: ObjectId('62f66f7347b5c476820cd053')
files_id: ObjectId('62f66f7347b5c476820cd051')
n: 0
data: BinData(0, 'JVBERi0xLjcKMSAwIG9iago8PCAvVHlwZS
```

Figura 22: Colección que almacena los chunks

[12] MongoDB. GridFS.
https://www.mongodb.com/docs/v6.0/core/gridfs/

El ***objectid*** de la tabla ***files***, es el atributo ***_id*** de la ***colección***. Por medio de su valor, los usuarios/sistemas pueden recuperar documentos almacenados en una base de datos NoSQL (MongoDB) mediante la especificación GridFS.

7.3 Implementación utilizando almacenamiento simple

Esta implementación es la más alejada del concepto tradicional de una base de datos. Es por demás parecida a la estructura de archivos y carpetas de una computadora común. No obstante, es una opción más que brinda este proyecto tecnológico de almacenamiento multi-aplicación.

Se define un espacio de almacenamiento, como se muestra en la Figura 23, en este ejemplo ***multi-app-store.binaries***, el cual servirá de contenedor (bucket) para alojar todos los documentos.

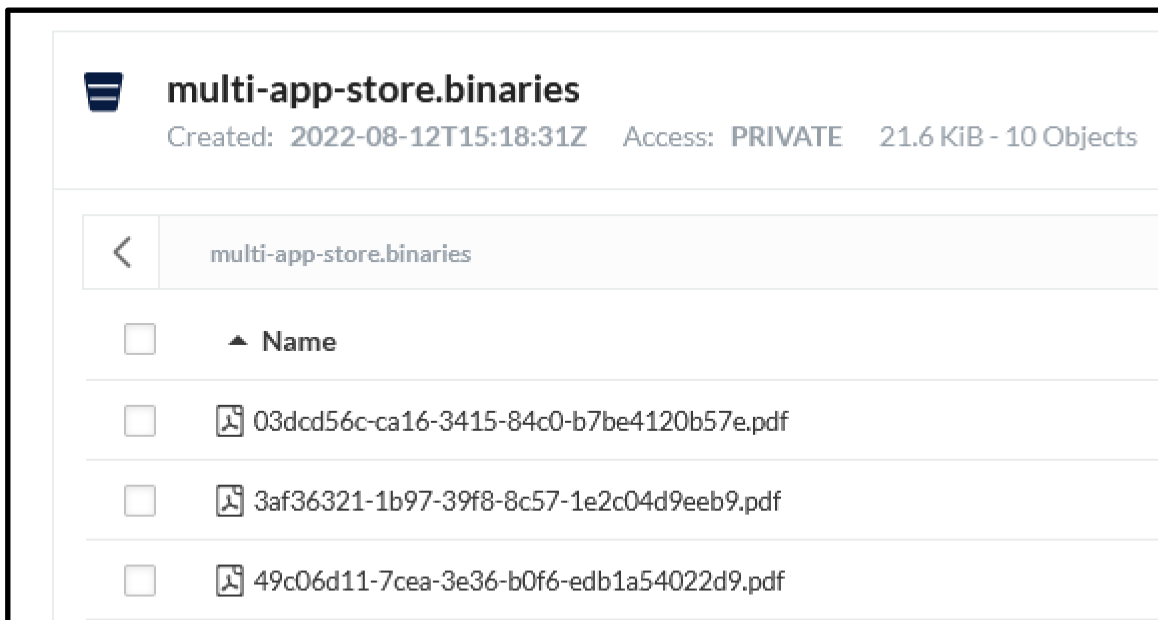


Figura 23: Bucket y su contenido de documentos

Los valores de los campos ***filename*** y ***bucket*** de la tabla ***files***, son mapeados hacia el almacenamiento simple, como el ***contenedor*** (bucket) y dentro de él, el ***nombre del archivo*** a recuperar. Una vez más, estos atributos de la tabla ***files***, son los que relacionan a los binarios con los usuarios/sistemas.

8. Generación lote de pruebas

A fin de realizar pruebas de inserción de documentos, relaciones entre las tablas, y recuperación de los mismos, se desarrolla un programa en PHP utilizando como framework Laravel; el cual genera un lote de pruebas mediante sus factorías y seeders, que produce como resultado múltiples documentos del tipo PDF, los cuales son guardados en los tres tipos de almacenamiento propuestos.

8.1 Modelo

Se necesitaron crear modelos para todos los objetos definidos en el diseño (punto 6.1). Laravel incluye Eloquent[13], un mapeador relacional de objetos (ORM) que hace posible interactuar con la base de datos. Al usar Eloquent, cada tabla de la base de datos tiene un "Modelo" (Figura 24) correspondiente que se usa para interactuar con esa tabla. Además de recuperar registros de la tabla de la base de datos, los modelos Eloquent también le permiten insertar, actualizar y eliminar registros de la tabla.

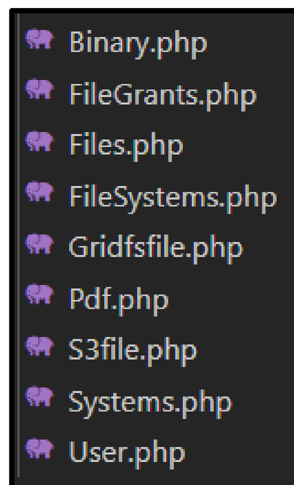


Figura 24: Modelos del proyecto

[13] Laravel. Eloquent: Getting Started. Introduction. https://laravel.com/docs/9.x/eloquent#introduction

A continuación (Figura 25), se presenta una el código utilizado para definir el modelo *files* en este caso.

```
class Files extends Model
{
    use Notifiable;

    protected $fillable = [
        'binary_id', 'path', 'extention', 'creator_system_id', 'creator_user_id', 'creator_datetime',
        'access_type', 'active',
        'owner_user_id', 'owner_datetime',
        'signed', 'immutable',
        'last_update', 'version'
    ];
}
```

Figura 25: Definición del modelo files, con sus atributos

8.2 Creación de las tablas

Laravel define a las migraciones[14] (Figura 26), como el control de versiones de la base de datos.

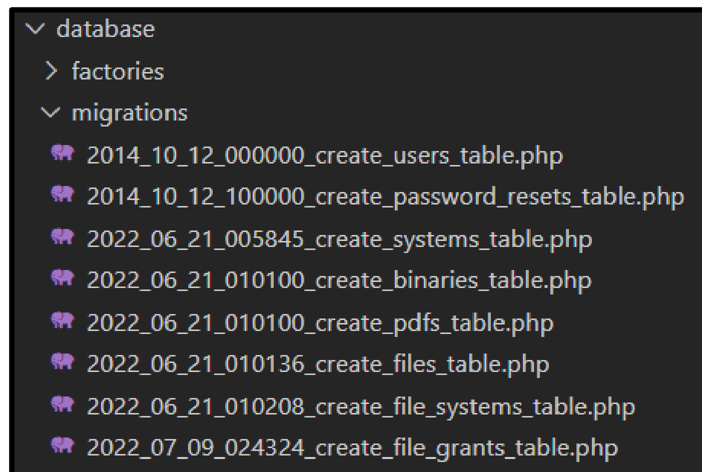


Figura 26: Migraciones del proyecto

El *Schema Facade* de Laravel proporciona soporte agnóstico de base de datos para crear y manipular tablas en todos los sistemas de base de datos compatibles con Laravel. Por lo general, las migraciones usarán este *facade* para crear y modificar tablas y columnas de bases de datos.

[14] Laravel. Database: Migrations. Introduction
https://laravel.com/docs/9.x/migrations#introduction

Para fines de ejemplo, exponemos la creación de la tabla ***files*** (Figura 27). Se define en primer lugar el atributo id, el cual lleva las propiedades sin signo y valor único. Seguido la definición de los distintos campos de la tabla con sus nombres, tipos y longitudes.

```
Schema::create('files', function (Blueprint $table) {
    $table->bigIncrements('id');
    // FK
    $table->bigInteger('binary_id')->unsigned()->unique();
    $table->foreign('binary_id')->references('id')->on('binaries');
    //
    $table->text('filename', 1024);
    $table->text('bucket', 63);
    $table->text('path', 512);
    $table->text('url', 1024);
    $table->enum('extention', ['doc', 'docx', 'rtf', 'pdf']);
    $table->text('objectid', 256);

    // FK
    $table->bigInteger('creator_system_id')->unsigned()->nullable();
    $table->foreign('creator_system_id')->references('id')->on('systems');
    // FK
    $table->bigInteger('creator_user_id')->unsigned()->nullable();
    $table->foreign('creator_user_id')->references('id')->on('users');
    //
    $table->timestamp('creator_datetime');
    $table->enum('access_type', ['private', 'shared']);
    $table->boolean('active');
    // FK
    $table->bigInteger('owner_user_id')->unsigned()->nullable();
    $table->foreign('owner_user_id')->references('id')->on('users');
    //
    $table->timestamp('owner_datetime')->useCurrent()->useCurrentOnUpdate();
    $table->boolean('signed');
    $table->boolean('immutable');
    $table->timestamp('last_update')->useCurrent()->useCurrentOnUpdate();
    $table->bigInteger('version');
    //
    $table->timestamps();
});
```

Figura 27: Ejemplo migración tabla files

También se definen las claves foráneas. Estas son las que permiten la relación hacia las demás tablas definidas en el sistema de almacenamiento multi-aplicación.

8.3 Factory

Al probar una aplicación o inicializar una base de datos, es posible que deba insertar algunos registros en la base de datos. En lugar de especificar manualmente el valor de cada columna,

Laravel permite definir un conjunto de atributos predeterminados para cada uno de sus modelos de Eloquent utilizando fábricas[15] de modelos.

Entonces, llegamos a la instancia en la cual, por medio de las fábricas (*factories*), generamos los datos ficticios que poblarán la base de datos. Para ello se definirá una a una cada factoría para cada uno de los modelos del almacenamiento único multi-aplicación. Como ejemplo, se muestra la definición de la factoría *FilesFactory.php* (Figura 28).

```php
return [
    'binary_id' => $binaryMysqlToArray['id'],
    'filename' => $bynaryMongoDBToArray['filename'],
    'bucket' => env('MINIO_BUCKET'),
    'path' => $bynaryS3fileToArray['path'],
    'url' => $bynaryS3fileToArray['url'],
    'extention' => $faker->randomElement(array('pdf')),
    'objectid' => $bynaryMongoDBToArray['oid'],

    'creator_system_id' => $faker->numberBetween(1, $seeds_sys),
    'creator_user_id' => $faker->numberBetween(1, $seeds_usr),
    'creator_datetime' => $faker->dateTimeBetween($startDate, $endDate),
    'access_type' => $faker->randomElement(array('private', 'shared')),
    'active' => $faker->numberBetween(1, 2),

    'owner_user_id' => $faker->numberBetween(1, $seeds_usr),
    'owner_datetime' => $faker->dateTimeBetween($startDate, $endDate),

    'signed' => $faker->numberBetween(1, 2),
    'immutable' => $faker->numberBetween(1, 2),

    'last_update' => $faker->dateTimeBetween($startDate, $endDate),
    'version' => $faker->numberBetween(1, 5),
];
```

Figura 28: FilesFactory.php: Factoría de la tabla files

[15] Laravel. Eloquent: Factories. Introduction
https://laravel.com/docs/9.x/eloquent-factories#introduction

A través del ***fake helper***, las fábricas tienen acceso a la biblioteca *Faker* PHP, que le permite generar convenientemente varios tipos de datos aleatorios para probar y sembrar, estos se muestran en la Figura 29 a continuación.

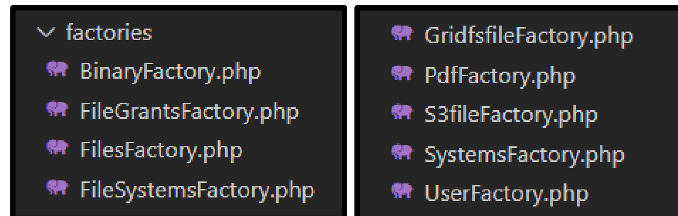


Figura 29: Factorías del proyecto

También se programaron procesos especiales diseñados para generar documentos PDF, más no es tema a desarrollar para este proyecto. Solo lo mencionamos como una necesidad de este proceso de testeo.

8.4 Poblado de las bases de datos

El proceso de migración puede incluir como en el presente ejemplo, inicialmente un *rollback* de todas las migraciones anteriores (Figura 30). Seguido las migraciones que *generarán* cada una de las tablas del diseño del almacenamiento único multi-aplicación, y acto final el *poblado (seeding)* de la base de datos.

```
$ php artisan migrate:refresh --seed
Rolling back: 2022_07_09_024324_create_file_grants_table
Rolled back:  2022_07_09_024324_create_file_grants_table (0.02 seconds)
Rolling back: 2022_06_21_010208_create_file_systems_table
Rolled back:  2022_06_21_010208_create_file_systems_table (0.02 seconds)
Rolling back: 2022_06_21_010136_create_files_table
Rolled back:  2022_06_21_010136_create_files_table (0.02 seconds)
Rolling back: 2022_06_21_010100_create_pdfs_table
Rolled back:  2022_06_21_010100_create_pdfs_table (0.02 seconds)
Rolling back: 2022_06_21_010100_create_binaries_table
Rolled back:  2022_06_21_010100_create_binaries_table (0.02 seconds)
Rolling back: 2022_06_21_005845_create_systems_table
Rolled back:  2022_06_21_005845_create_systems_table (0.02 seconds)
Rolling back: 2014_10_12_100000_create_password_resets_table
Rolled back:  2014_10_12_100000_create_password_resets_table (0.02 seconds)
Rolling back: 2014_10_12_000000_create_users_table
Rolled back:  2014_10_12_000000_create_users_table (0.02 seconds)
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (0.05 seconds)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (0.04 seconds)
Migrating: 2022_06_21_005845_create_systems_table
Migrated:  2022_06_21_005845_create_systems_table (0.02 seconds)
Migrating: 2022_06_21_010100_create_binaries_table
Migrated:  2022_06_21_010100_create_binaries_table (0.07 seconds)
Migrating: 2022_06_21_010100_create_pdfs_table
Migrated:  2022_06_21_010100_create_pdfs_table (0.07 seconds)
Migrating: 2022_06_21_010136_create_files_table
Migrated:  2022_06_21_010136_create_files_table (0.39 seconds)
Migrating: 2022_06_21_010208_create_file_systems_table
Migrated:  2022_06_21_010208_create_file_systems_table (0.18 seconds)
Migrating: 2022_07_09_024324_create_file_grants_table
Migrated:  2022_07_09_024324_create_file_grants_table (0.8 seconds)
Seeding: UserTableSeeder
Seeding: SystemsTableSeeder
Seeding: FilesTableSeeder
Database seeding completed successfully.
```

Figura 30: Proceso de migración de tablas y poblado

Esta migración, debe poblar los tres tipos de almacenamientos que contempla el proyecto tecnológico. A modo de ejemplo (Figura 31), exponemos el código PHP compuesto de cuatro sectores bien definidos.

```
// Genera PDF
$pdf = factory(Pdf::class, 1)->create();
$pdfToArray = $pdf->toArray();

$uuid = $pdfToArray[0]['uuid'];
$file = $pdfToArray[0]['file'];
$updated_at = $pdfToArray[0]['updated_at'];
$created_at = $pdfToArray[0]['created_at'];
$id = $pdfToArray[0]["id"];

// Genera Binario MySQL
$binaryMysql = factory(Binary::class, 1)->create(['uuid' => $uuid, 'file' => $file]);
$binaryMysqlToArray = $binaryMysql->toArray()[0];

// Genera Binario Minio
$bynaryS3file = factory(S3file::class, 1)->make(['uuid' => $uuid]);
$bynaryS3fileToArray = $bynaryS3file->toArray()[0];

// Genera Binario MongoDB
$bynaryMongoDB = factory(Gridfsfile::class, 1)->make(['uuid' => $uuid]);
$bynaryMongoDBToArray = $bynaryMongoDB->toArray()[0];
```

Figura 31: Código generador de documentos de prueba

Genera PDF: Se encarga de generar por medio de una factoría dedicada a este fin, documentos PDF de pruebas. Estos serán almacenados en una tabla intermedia. De este paso solo rescatamos para el proyecto, la variable ***$pdfToArray***, que contiene la información del documento recientemente generado y que luego será almacenada adecuadamente en la tabla ***files***.

Genera Binario MySQL: almacena el PDF de prueba generado en el paso anterior en la tabla binaries para el caso en el que se opte por un almacenamiento relacional para el proyecto de almacenamiento único multi-aplicación. Este es el caso, en el cual, la organización que implementa este proyecto, opta por un almacenamiento único basado en un modelo de base de datos relacional.

Genera Binario Minio: en el caso que el almacenamiento elegido para la unificación de los documentos, sea un almacenamiento simple, esta parte de código, almacena el PDF del primer

paso en un almacén tipo S3, y en este caso particular, en el almacenamiento que nos provee Minio[16].

Genera Binario MongoDB: finalmente, si se optase por almacén único ***MongoDB***, esta factoría nos permitirá almacenar el PDF ya mencionado, dentro de la especificación GridFS.

Luego, el resultado será un documento PDF, almacenado en tres tipos distintos de almacenamiento distintos, pero compartiendo las mismas características identificatorias (id, filename, creator, owner, etc). La idea de fondo de esto, es que las aplicaciones puedan acceder a los mismos, independientemente de su ubicación física real, basándose en la información contenida en la tabla ***files***.

8.5 Test de recuperación

El proceso de generación de datos de prueba genera los siguientes registros para la tabla files, que tendrán su correspondencia con los demás almacenamientos planteados. Esto se ve en la figura a continuación Figura 32:

id	binary_id	filename	bucket	objectid
1	1	4bae4215-5ef1-3785-a177-659c5db24232.pdf	multi-app-store.binaries	6306d23cbebdb9e631019cc1
2	2	3f10a0d4-7367-3550-8626-4802b5568498.pdf	multi-app-store.binaries	6306d23dbebdb9e631019cc4
3	3	bd2c6909-17d1-3c46-a25b-29358df2ab3d.pdf	multi-app-store.binaries	6306d23fbebdb9e631019cc6

Figura 32: Tabla files. Registros generados mediante migración, factorización y poblado.

Recordemos que la tabla files, es la controladora de la ubicación de un documento, ubicado en alguno de los tres tipos de almacenamientos definidos en el proyecto. El atributo *binary_id*, identifica la ubicación en el almacenamiento relacional, *bucket* y *filename*, identifican la ubicación en el almacenamiento simple, y *objetid* hace lo mismo en el almacenamiento NoSQL.

[16] Multi-Cloud Object Storage
https://min.io/

En el almacenamiento simple, el proceso de factorización y poblado, anexo los siguientes documentos del tipo pdf al ***bucket*** "multi-app-store.binaries" (Figura 33).

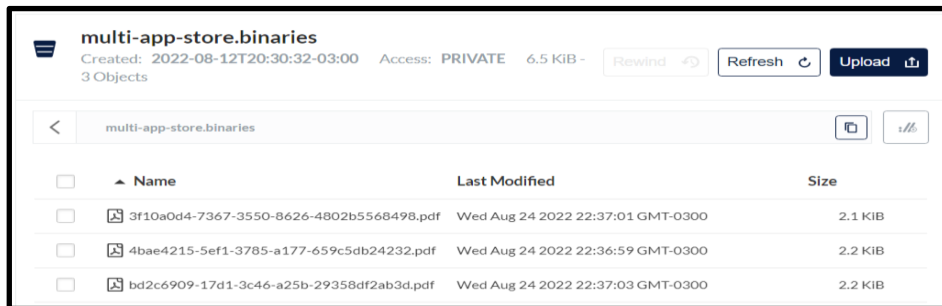


Figura 33: Contenido del bucket "multi-app-store.binaries"

Para el almacenamiento NoSQL, MongoDB, se crearon documentos en la colección ***multi-app-store.binaries.files***, que a su vez están relacionados con la colección ***multi-app-store.binaries.chunks***, que bajo la especificación GridFS almacena los documentos, en este caso del tipo pdf, en porciones (chunks), tales como se muestran a continuación (Figura 34).

Figura 34: Porciones (chunks) de documentos

En el caso del almacenamiento relacional (Figura 35), se insertaron los siguientes registros, con los documentos del tipo pdf inmersos en el campo ***file*** del tipo ***large object***.

id	uuid	file	created_at	updated_at
1	4bae4215-5ef1-3785-a177-659c5db24232	[BLOB - 2.2 KB]	2022-08-25 01:36:58	2022-08-25 01:36:58
2	3f10a0d4-7367-3550-8626-4802b5568498	[BLOB - 2.1 KB]	2022-08-25 01:37:00	2022-08-25 01:37:00
3	bd2c6909-17d1-3c46-a25b-29358df2ab3d	[BLOB - 2.2 KB]	2022-08-25 01:37:01	2022-08-25 01:37:01

Figura 35: Registros de la tabla relacional binaries

La Figura 36, es un ejemplo de la recuperación del documento ***4bae4215-5ef1-3785-a177-659c5db24232.pdf*** desde el almacenamiento simple (MinIO):

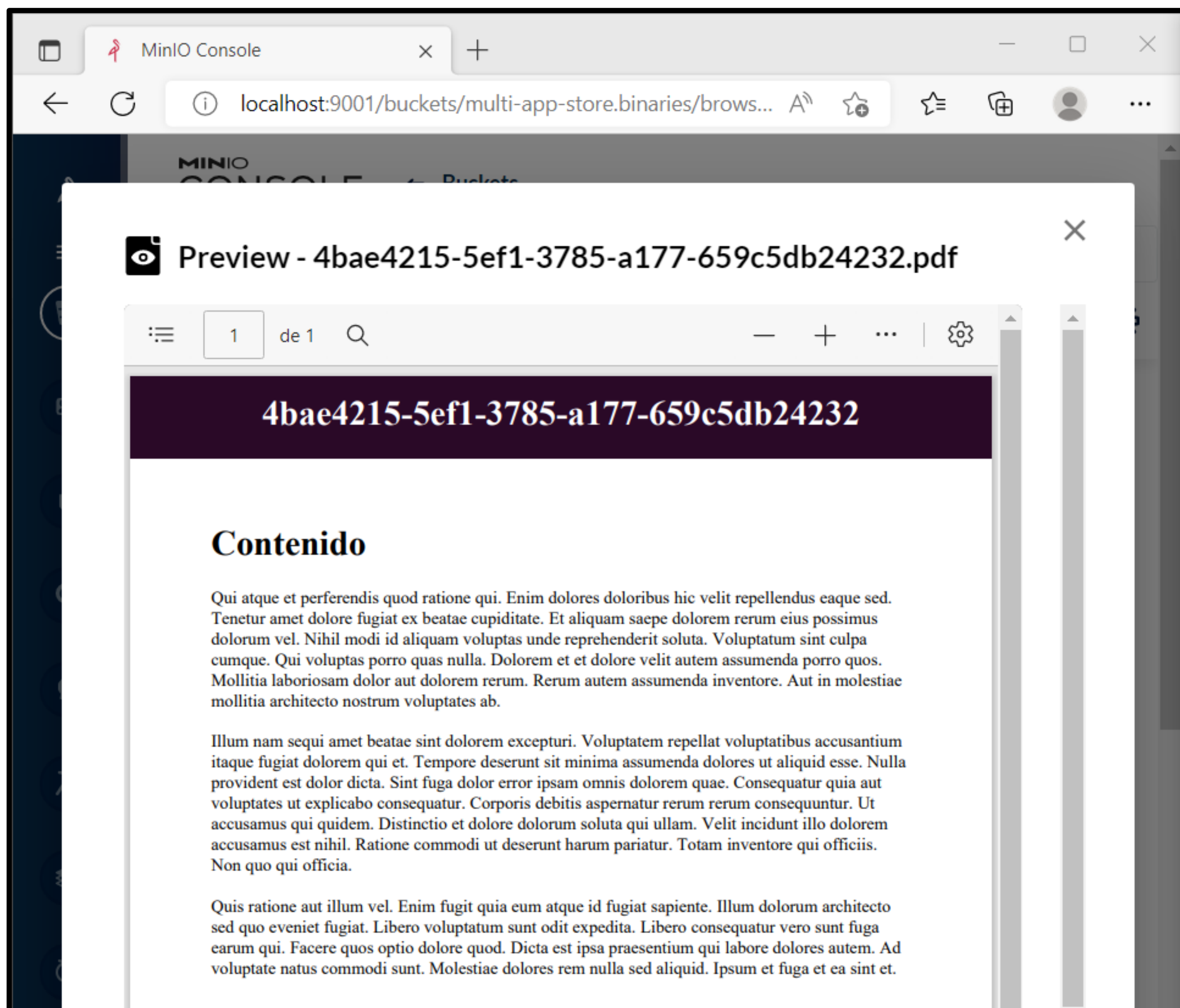


Figura 36: Recuperación del documento 4bae4215-5ef1-3785-a177-659c5db24232.pdf desde el almacenamiento simple

La Figura 37, es un ejemplo de la recuperación del mismo documento, ***4bae4215-5ef1-3785-a177-659c5db24232.pdf*** desde el *GridFS* de MongoDB:

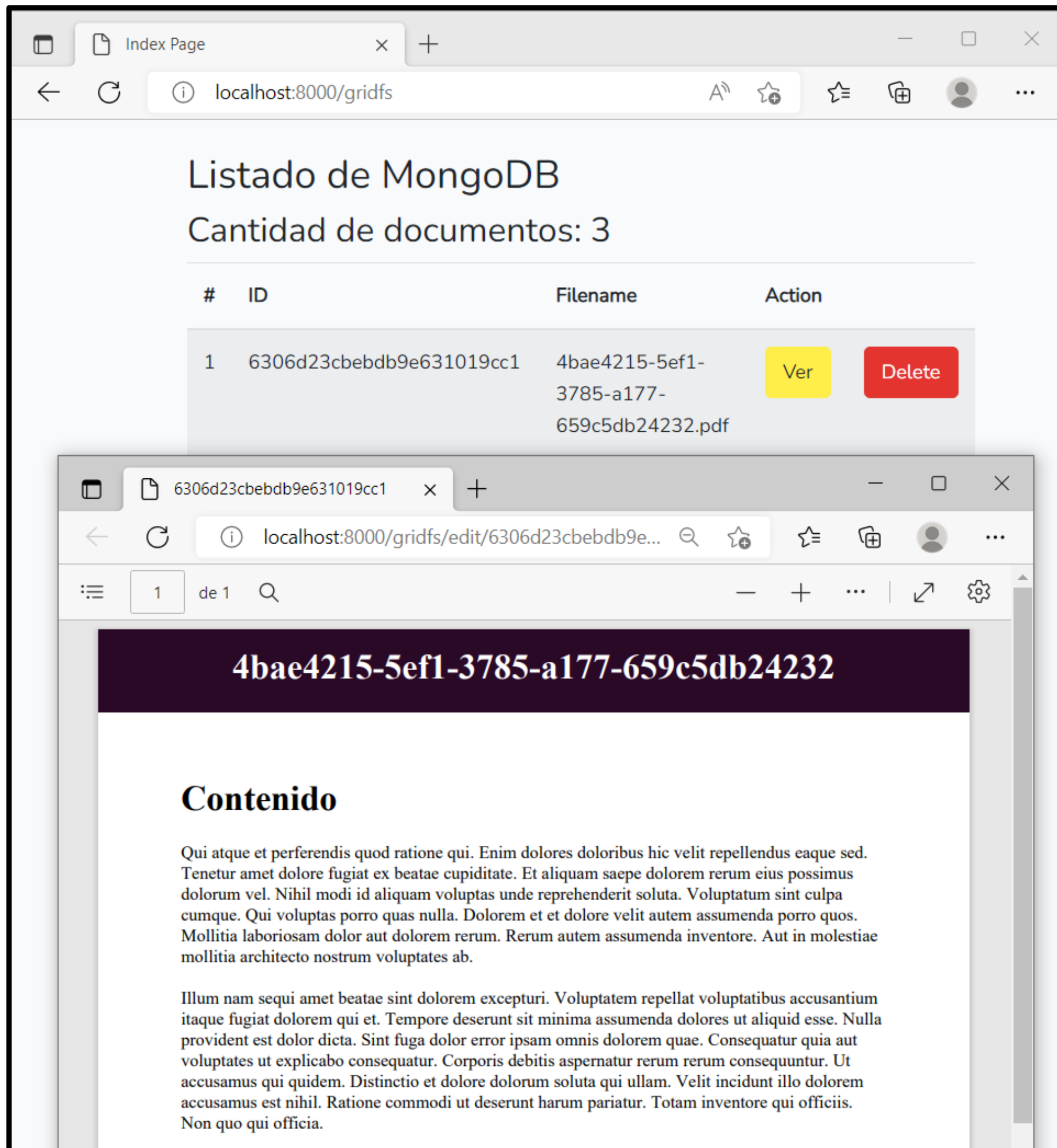


Figura 37: Recuperación del documento 4bae4215-5ef1-3785-a177-659c5db24232.pdf desde la especificación GridFS de MongoDB

9. El motor de búsqueda

9.1 Elastic

Elasticsearch (elastic.co, 2022) es un motor de búsqueda y analítica de RESTful distribuido, que permite almacenar de forma central datos para búsquedas a gran velocidad. Elasticsearch permite realizar y combinar muchos tipos de búsquedas: estructuradas, no estructuradas, geográficas, métricas.

Elasticsearch[17] proporciona búsqueda y análisis casi en tiempo real para todo tipo de datos. Ya sea texto estructurado o no estructurado, datos numéricos o datos geoespaciales, Elasticsearch puede almacenarlos e indexarlos de manera eficiente de una manera que admite búsquedas rápidas.

9.2 Logstash

Logstash (elastic.co, Logstash, 2022) es un pipeline de procesamiento de datos gratuito y abierto del lado del servidor que ingiere datos de una multitud de fuentes, los transforma y luego los envía a tu "escondite" favorito.

Logstash ingesta, transforma y envía a Elastic de forma dinámica datos independientemente de su formato o complejidad.

Logstash admite una variedad de entradas de múltiples fuentes. Ingiere fácilmente desde logs, métricas, aplicaciones web, almacenes de datos y varios servicios de AWS, todo de una manera de transmisión continua.

A medida que los datos viajan de la fuente al almacén, los filtros Logstash parsean cada evento, identifican los campos con nombre para crear la estructura y los transforman para que converjan en un formato común para un análisis y un valor comercial más poderosos.

[17] elastic.co. What is Elasticsearch?
https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html

9.3 Elastic y Logstash en el almacenamiento único multi-aplicación

9.3.1 Marco teórico

Elastic permite realizar consultas de texto completo[18] en campos de texto analizados. La cadena de consulta se procesa con el mismo analizador que se aplicó al campo durante la indexación.

Algunos tipos de consultas que se pueden realizar son:

match query: es la consulta estándar para realizar consultas de texto completo, incluidas las consultas de coincidencia aproximada y de frase o proximidad.

match_phrase query: al igual que la consulta de coincidencia, pero se utiliza para hacer coincidir frases exactas o coincidencias de proximidad de palabras.

query_string query: admite la sintaxis de cadena de consulta compacta de Lucene, lo que le permite especificar las condiciones AND|OR|NOT y la búsqueda de varios campos dentro de una única cadena de consulta. Solo para usuarios expertos.

simple_query_string query: una versión más simple y robusta de la sintaxis query_string adecuada para exponer directamente a los usuarios.

9.3.2 Implementación

Estos tipos de consulta nos permiten realizar búsquedas complejas de texto sobre nuestra base de datos única multi-aplicación. Este es el caso en el que necesitamos recuperar un documento, pero no por su identificador, sino basado en su contenido. En este caso se deberá recorrer los documentos buscando el texto, frase o palabra deseado.

En los casos de los almacenamientos relacional y NoSQL MongoDB, se podrán realizar consultas, pero no directamente al BLOB o al GridFS. Dado que estos solo son los contenedores de los archivos. Entonces se necesitaría agregar un campo con el texto plano, de manera de poder realizar sobre él la búsqueda. Pero este tipo de búsqueda no son muy performantes, sobre todo en el ambiente relacional.

---

[18] elastic.co. Full text queries
https://www.elastic.co/guide/en/elasticsearch/reference/current/full-text-queries.html

Para el caso del almacenamiento simple de objetos, aquí se complica más, ya que se deberán abrir uno a uno los archivos para poder realizar la mencionada búsqueda. Otra vez el tema de la performance se ve afectada.

Aquí es donde ingresa Elastic en conjunto con Logstash. Este último permite recuperar la información almacenada en los documentos PDFs y pasarlos a formato plano para luego almacenarlos en Elastic. Logstash puede conectarse a un relacional, a MongoDB o a un almacenamiento simple, para recuperar datos desde los documentos, manipularlos y enviarlos finalmente al repositorio de Elastic.

Finalmente, Elastic provee los mecanismos para realizar búsquedas complejas sobre su repositorio.

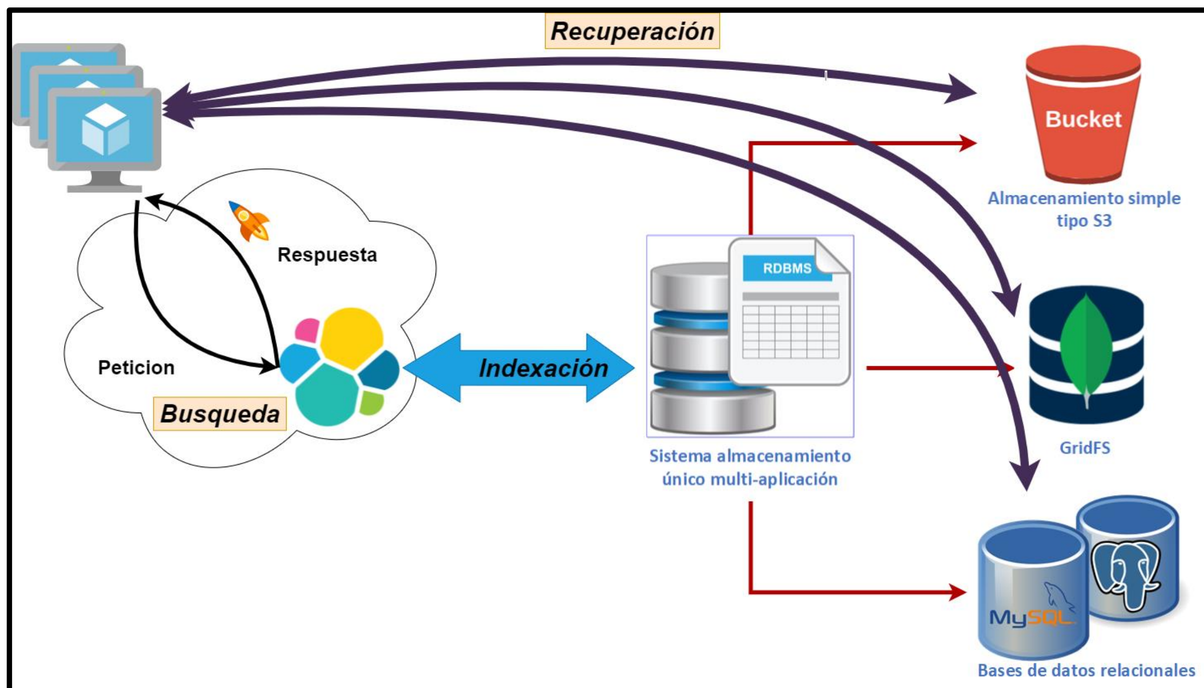


Figura 38: Esquema del proyecto tecnológico, con Elasticsearch como motor de búsqueda

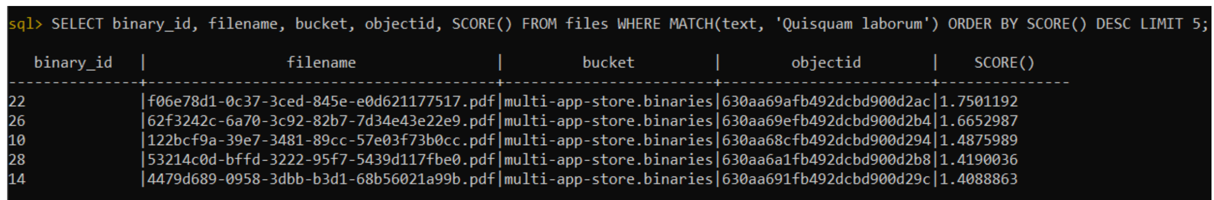
El proyecto tecnológico (Figura 38), tiene previsto una indexación continua de la tabla ***files***, mediante el recolector ***Logstash***, para almacenar su contenido en ***Elastic***. De esta forma, las aplicaciones pueden realizar las búsquedas de documentos directamente sobre Elastic, para obtener resultados con sus respectivos identificadores de documentos, y acceder directamente al almacenamiento apropiado según su identificador.

9.3.3 Ejemplo

Supongamos un lote de 30 documentos, todos PDFs, con distintos contenidos de texto. Y se desea realizar una búsqueda por todos los documentos que contengan las palabras *quisquam laborum*, indistintamente si están juntas o no.

Elasticsearch nos permite realizar búsquedas de texto completo de manera muy fácil:

```
SELECT binary_id, filename, bucket, objectid, SCORE()
FROM files WHERE MATCH(text, 'Quisquam laborum')
ORDER BY SCORE() DESC;
```



sql> SELECT binary_id, filename, bucket, objectid, SCORE() FROM files WHERE MATCH(text, 'Quisquam laborum') ORDER BY SCORE() DESC LIMIT 5;

binary_id	filename	bucket	objectid	SCORE()
22	f06e78d1-0c37-3ced-845e-e0d621177517.pdf	multi-app-store.binaries	630aa69afb492dcbd900d2ac	1.7501192
26	62f3242c-6a70-3c92-82b7-7d34e43e22e9.pdf	multi-app-store.binaries	630aa69efb492dcbd900d2b4	1.6652987
10	122bcf9a-39e7-3481-89cc-57e03f73b0cc.pdf	multi-app-store.binaries	630aa68cfb492dcbd900d294	1.4875989
28	53214c0d-bffd-3222-95f7-5439d117fbe0.pdf	multi-app-store.binaries	630aa6a1fb492dcbd900d2b8	1.4190036
14	4479d689-0958-3dbb-b3d1-68b56021a99b.pdf	multi-app-store.binaries	630aa691fb492dcbd900d29c	1.4088863

Figura 39: Consulta full-text search,
ordenada por relevancia de mayor a menor, limitada a los primeros 5

Esta consulta (Figura 39), emplea el operador MATCH para realizar la búsqueda de las palabras deseadas. La función SCORE()[19], retorna la relevancia de la entrada dada (**Quisquam laborum**). Cuanto más alto el valor, más relevante es la entrada en la consulta.

El documento a continuación (Figura 40), presentado como ejemplo, está en tercer lugar de relevancia, ya que solo contiene tres coincidencias en la búsqueda, las dos marcadas y un *Quisquam* más en el segundo párrafo.

---

[19] Full-text search functions. SCORE()
https://www.elastic.co/guide/en/elasticsearch/reference/7.17/sql-functions-search.html

122bcf9a-39e7-3481-89cc-57e03f73b0cc

Contenido

Vel soluta distinctio ab et quia. Velit ut ut harum et laboriosam nisi amet. Et unde illo itaque optio quidem. Quae non commodi voluptatem natus aliquid nemo. Facilis eos rem nisi voluptas corporis. Eligendi est quis amet et nostrum omnis. Vitae aut eum optio in. Facere dolorem ut quis et voluptate. Voluptas necessitatibus possimus accusantium impedit deserunt. Ut nesciunt et rerum tempore libero. At est autem delectus reiciendis. Sequi veritatis nesciunt enim quaerat facilis. Qui praesentium quis distinctio itaque ut quas explicabo ut. Eligendi sint consectetur ipsum facere dolores quam dolore.

Quisquam laborum unde voluptates repellendus nostrum. Deleniti architecto neque adipisci quibusdam et doloremque quibusdam corrupti. Doloribus dolor dolorem odio quo iusto non. Quaerat et iste quidem. Neque eos neque unde et unde alias. Quisquam commodi in dicta aspernatur debitis delectus et atque. Magnam odit sed non enim molestias debitis. Sit commodi vel incidunt officia optio reprehenderit quas. Voluptatum eligendi expedita quam architecto esse voluptatem itaque. Culpa et quia non reprehenderit minima quam impedit. Rerum facere corporis occaecati amet. Ratione perferendis libero numquam dolores quia sit ad. Et corrupti et esse ab libero. Odit qui rem alias modi quia occaecati.

Atque nihil odio consequatur quia aut. Odit qui fugit inventore sed. Ratione earum tenetur placeat laboriosam ducimus velit dignissimos. Quia necessitatibus aliquam eaque. Aspernatur quidem dolor soluta laboriosam delectus. Est quibusdam eaque nostrum vel maxime nisi laborum. Est sint harum repellat dolor doloremque delectus. Quia quasi ut fugiat quaerat nulla. Voluptatem quod ipsam odit omnis fugiat ab. Officiis suscipit omnis reprehenderit. Illum quo voluptas expedita omnis. Ab error voluptas dignissimos.

Figura 40: Ejemplo de contenido a buscar.
Score=1,487

El siguiente documento (Figura 41), es el que mayor relevancia tiene. En este caso, el mismo posee seis coincidencias en su contenido, por lo que obtiene el mejor score.

f06e78d1-0c37-3ced-845e-e0d621177517

Contenido

Quibusdam aut dicta velit rem et eligendi. Quos velit quia sed est. Consequuntur blanditiis quia aperiam eveniet possimus eaque. Enim ullam omnis sint labore doloribus harum nulla perspiciatis. Eaque id atque voluptatem in enim aut molestiae. In consequatur iste neque vel amet fugit voluptates omnis. Quod soluta architecto quibusdam ullam. Error omnis adipisci amet dolor rerum et. Vero quasi maiores deleniti et officia aut. Velit temporibus illo voluptas. Vel exercitationem error quod laborum et est. Qui velit nihil illo amet. Ducimus accusamus eaque recusandae ex culpa a qui.

Modi aspernatur iure iusto id iure quia corporis sequi. Vel est corporis quaerat maiores expedita rerum quibusdam. Ullam cupiditate sint aspernatur quae amet ut. Eos laboriosam facere tempora nihil autem accusamus at. Maiores tempora adipisci nulla quod. Nesciunt est reprehenderit eos quaerat aliquid sed. Qui quia non eum. Repellendus accusamus asperiores itaque hic. Sit voluptates itaque et illum deleniti. Eos dolorum mollitia ipsa exercitationem qui. Ut quisquam in laborum dolores. Ratione minima doloremque odit odio. Magni quae natus quia accusamus. Quidem quam voluptatem accusamus id qui et.

Deleniti odit quisquam aliquam et voluptatem ducimus sunt aut. Velit illum dolor reprehenderit et voluptatum sit consequatur. Odio suscipit nihil unde vero id. Natus et perferendis repudiandae aspernatur. Labore voluptas vitae totam quis. Laborum mollitia excepturi ipsa aut est id ex ducimus. Quisquam qui corporis eum tempore quos quod et. A consequuntur vitae est nihil fugit molestiae occaecati.

Figura 41: Ejemplo de contenido buscado,
con un mejor score 1,750

Esta consulta a continuación (Figura 42), nos da todos los documentos, ordenados por orden de relevancia de mayor a menor, permitiéndonos identificar, cuales documentos satisfacen en mayor medida, el criterio de búsqueda, con respecto a los demás:

```
SELECT binary_id, filename, bucket, objectid, SCORE()
FROM files WHERE MATCH(text, 'Quisquam laborum')
ORDER BY SCORE() DESC;
```

```
sql> SELECT binary_id, filename, bucket, objectid, SCORE() FROM files WHERE MATCH(text, 'Quisquam laborum') ORDER BY SCORE() DESC;
   binary_id   |               filename               |         bucket         |         objectid         |    SCORE()
---------------+--------------------------------------+------------------------+--------------------------+--------------
22             |f06e78d1-0c37-3ced-845e-e0d621177517.pdf|multi-app-store.binaries|630aa69afb492dcbd900d2ac|1.7501192
26             |62f3242c-6a70-3c92-82b7-7d34e43e22e9.pdf|multi-app-store.binaries|630aa69efb492dcbd900d2b4|1.6652987
10             |122bcf9a-39e7-3481-89cc-57e03f73b0cc.pdf|multi-app-store.binaries|630aa68cfb492dcbd900d294|1.4875989
28             |53214c0d-bffd-3222-95f7-5439d117fbe0.pdf|multi-app-store.binaries|630aa6a1fb492dcbd900d2b8|1.4190036
14             |4479d689-0958-3dbb-b3d1-68b56021a99b.pdf|multi-app-store.binaries|630aa691fb492dcbd900d29c|1.4088863
15             |f2b62253-5c39-34de-a468-76306d3d2582.pdf|multi-app-store.binaries|630aa692fb492dcbd900d29e|1.3718189
11             |fe2fa139-7c37-3185-bc17-6adba71ac08c.pdf|multi-app-store.binaries|630aa68dfb492dcbd900d296|1.1962606
16             |8fe7dbec-4eb4-3fa2-86b7-4cec3d2cedcf.pdf|multi-app-store.binaries|630aa694fb492dcbd900d2a0|1.1962606
27             |2b623e96-56c0-3280-b128-0ed0467bf7a5.pdf|multi-app-store.binaries|630aa6a0fb492dcbd900d2b6|1.1962606
20             |e6eeb62a-e2fd-3c62-9143-e026a4f24d84.pdf|multi-app-store.binaries|630aa698fb492dcbd900d2a8|1.1217626
21             |cb76cf09-8461-3b92-94ae-8866838fdf03.pdf|multi-app-store.binaries|630aa699fb492dcbd900d2aa|0.8294221
19             |c88d65f6-485e-30c5-8c30-b1f8d1148c2b.pdf|multi-app-store.binaries|630aa697fb492dcbd900d2a6|0.810756
29             |d4b4a5a4-0f48-358f-8a9e-a1bc042ac6cb.pdf|multi-app-store.binaries|630aa6a2fb492dcbd900d2ba|0.79291165
3              |274db899-c4cf-33c6-90e1-44ba4e6c09ac.pdf|multi-app-store.binaries|630aa682fb492dcbd900d286|0.79291165
17             |923f14ba-00e7-3e8c-b5e2-6ca0fcde3d46.pdf|multi-app-store.binaries|630aa695fb492dcbd900d2a2|0.77583575
8              |b35aee5c-2af8-3ce7-81ac-31ecb4b9a257.pdf|multi-app-store.binaries|630aa689fb492dcbd900d290|0.6406787
2              |9166bad1-6128-3804-94f7-2856ccd73694.pdf|multi-app-store.binaries|630aa681fb492dcbd900d284|0.6186738
23             |d0a23f60-73ae-371b-aae6-7f47944b577f.pdf|multi-app-store.binaries|630aa69bfb492dcbd900d2ae|0.6186738
30             |cd5a66e8-6cef-3f61-9b08-7c09309bdccb.pdf|multi-app-store.binaries|630aa6a3fb492dcbd900d2bc|0.5981303
9              |ff7cbf17-4c12-3d4f-8891-24cd5f4047d0.pdf|multi-app-store.binaries|630aa68bfb492dcbd900d292|0.57890725
18             |2a7b9d04-402f-349e-8e0a-4eb89546a50f.pdf|multi-app-store.binaries|630aa696fb492dcbd900d2a4|0.5608813
24             |58ff3f15-dbc3-3f08-87c9-87412f185b9b.pdf|multi-app-store.binaries|630aa69cfb492dcbd900d2b0|0.543944
12             |7ad10719-cdf3-3178-858a-128a9edb45ab.pdf|multi-app-store.binaries|630aa68ffb492dcbd900d298|0.543944
6              |b9866b4b-01fa-3f40-bca9-168f299fcb3a.pdf|multi-app-store.binaries|630aa686fb492dcbd900d28c|0.5129636
```

Figura 42: Consulta full-text search,
ordenada por relevancia de mayor a menor.

Finalmente, la Figura 43, nos permite ver los documentos que no se encuentran en la consulta anterior.

```
sql> SELECT binary_id, filename, bucket, objectid, SCORE() FROM files WHERE NOT MATCH(text, 'Quisquam laborum');
   binary_id   |               filename               |         bucket         |         objectid         |    SCORE()
---------------+--------------------------------------+------------------------+--------------------------+--------------
25             |6db7f57a-9c66-3530-bcbf-9c8590f430d4.pdf|multi-app-store.binaries|630aa69dfb492dcbd900d2b2|0.0
7              |be740367-7f59-3cc6-a109-e4e939784ad5.pdf|multi-app-store.binaries|630aa687fb492dcbd900d28e|0.0
1              |aeb4a920-26fd-324e-b703-274cd79bee6a.pdf|multi-app-store.binaries|630aa680fb492dcbd900d281|0.0
13             |15c75889-319f-3076-b627-611b83428a76.pdf|multi-app-store.binaries|630aa690fb492dcbd900d29a|0.0
4              |e18a6b51-6661-368b-a0ef-04a58e2c63aa.pdf|multi-app-store.binaries|630aa683fb492dcbd900d288|0.0
5              |bc00e851-392d-3e20-8cb7-a8e9c6d4bc31.pdf|multi-app-store.binaries|630aa684fb492dcbd900d28a|0.0
```

Figura 43: Documentos que no cumplen con en el MATCH.
Puede verse que el SCORE() en todos los casos es de 0

10. Conclusiones

La imagen a continuación (Figura 44) representa el proyecto tecnológico completo, con los límites entre lo que es el ***almacenamiento único multi-aplicación***, y la ***búsqueda*** a través del ***motor Elasticsearch***.

El proyecto tecnológico consta de un ***sistema de almacenamiento único multi-aplicación***, compuesto por una ***base de datos central relacional*** (punto 6.1) y ***servicios*** para controlar las operaciones CRUD (punto 6.3) por parte de los usuarios/sistemas.

El ***almacenamiento final*** de los documentos, puede ser de cualquier tipo (punto 7). Relacional, MongoDB (NoSQL), o almacenamiento simple de objetos, dependiendo solamente de la organización.

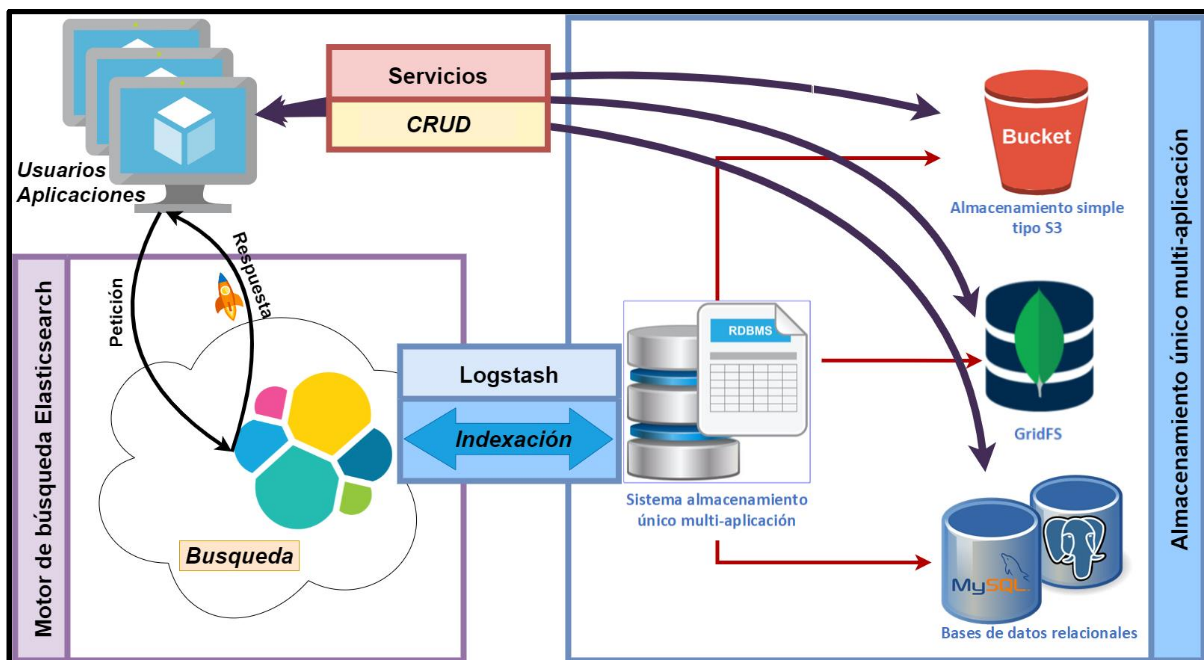


Figura 44: Proyecto almacenamiento único multi-aplicación,
con elasticsearch como motor de búsquedas complejas

A fin de poder realizar ***búsqueda de texto***, se implementa el motor de búsqueda ***Elasticsearch*** (punto 9), que permite búsquedas complejas, a alta velocidad. El nexo entre Elasticsearch y la base de datos central única multi-aplicación, es ***Logstash***, el que de manera continua indexa el contenido de la tabla files (tabla coordinadora de los documentos).

Pruebas de almacenamiento se realizaron (punto 8), a fin de evaluar la viabilidad de las distintas propuestas de almacenamiento que se plantearon en el presente proyecto tecnológico. Se desarrolló una sencilla aplicación en PHP que almacena un mismo documento en los tres tipos de almacenamientos propuestos. Almacena sus ubicaciones, y los recupera por medio de sus identificadores.

Como conclusión final, el enfoque fundamental de este proyecto es la ***creación*** de la ***base de datos única multi-aplicación***, que será la encargada de coordinar el almacenamiento de los documentos independientemente de la aplicación donde provengan. Y la ***programación*** de los ***servicios que coordinan las operaciones CRUD*** sobre los documentos. Si bien la programación de los mismos no corresponde al alcance de este proyecto, se acompañan los diagramas de flujo de estos procesos.

Secundario al proyecto es la aplicación de Elasticsearch para la implementación de full-text search para la búsqueda compleja a alta velocidad. Pero se deja sentado la factibilidad del mismo y la manera de implementarlo.

Como pensamiento final, se espera que este proyecto tecnológico, solucione el problema del almacenamiento repetitivo de documentos, con su implicancia negativa sobre la utilización de espacio para tal fin. Y al mismo tiempo, ser una mejor forma de aprovechar el mencionado recurso, brindando un mecanismo apropiado para el almacenamiento y recuperación de documentos de manera global.

Bibliografía

Amazon Web Services, I. (2022). *Amazon Simple Storage Service*. Obtenido de https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Báez, I. C. (2007). *Wiki, intranet y repositorios documentales. SciELO*. Obtenido de http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1024-94352008001200014

Contreras Henao, F., & Forero Guzmán, F. (2005). *Diseño de un modelo para la implantación de un sistema de gestión documental en áreas u organizaciones jurídicas.* Obtenido de https://repository.javeriana.edu.co/bitstream/handle/10554/7476/Tesis185.pdf?sequence=1&isAllowed=y

Duque, J. F. (2020). *Optimización del repositorio documental de la empresa Multidimensionales S.A.S a través del sharepoint de Microsoft.* Obtenido de https://expeditiorepositorio.utadeo.edu.co/bitstream/handle/20.500.12010/14234/Trabajo%20de%20grado.pdf?sequence=1&isAllowed=y

elastic.co. (2022). *Elastic*. Obtenido de Elasticsearch: https://www.elastic.co/es/elasticsearch/

elastic.co. (2022). *Logstash*. Obtenido de Centraliza, transforma y almacena tus datos: https://www.elastic.co/es/logstash/

Elasticsearch B.V. (2022). *Elastic Docs*. Obtenido de https://www.elastic.co/guide/index.html

Laravel LLC. (2022). *Documentation*. Obtenido de Documentation

Mancini, M. J., Galasso, C. L., & Banchieri, M. A. (2018). Software para Decodificación y Análisis de Archivos Binarios en Computadoras Navales. *XXIV Congreso Argentino de Ciencias de la Computación*, 1. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/73244/Documento_completo.pdf?sequence=1

Microsoft. (2022). *Comparar opciones para almacenar objetos Blob (SQL Server)*. Obtenido de https://learn.microsoft.com/es-es/sql/relational-databases/blob/compare-options-for-storing-blobs-sql-server?view=sql-server-ver15

MinIO, I. (2020). *MinIO High Performance Object Storage*. Obtenido de https://docs.min.io/minio/baremetal/

MongoDB, I. (2022). *MongoDB Documentation*. Obtenido de https://www.mongodb.com/docs/

Project Voldemort. A distributed database. (s.f.). Obtenido de https://www.project-voldemort.com/voldemort/

The PostgreSQL Global Development Group. (2022). Obtenido de Documentation: https://www.postgresql.org/docs/