Facultad
de
Ciencias

METASEC: APLICACIÓN PARA EL ANÁLISIS Y BORRADO DE METADATOS

METASEC: Application for the analysis and deletion of metadata

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Celso Gimeno Corbella

Director: Carlos Blanco Bueno

Febrero – 2024

Resumen

En la era digital actual, donde la información fluye a velocidades vertiginosas y la comunicación se ha vuelto omnipresente, la gestión adecuada de la seguridad cibernética se ha convertido en una prioridad ineludible. En este contexto, los metadatos, a menudo denominados “datos sobre datos”, que son esos pequeños fragmentos de información que acompañan a nuestros archivos digitales, juegan un papel fundamental en el ámbito de la ciberseguridad. Sin una gestión cuidadosa, los metadatos pueden convertirse en una potencial vulnerabilidad.

La sobreexposición de información sensible, como la ubicación precisa de un usuario al capturar una fotografía, puede ser explotada por amenazas cibernéticas. La privacidad se ve amenazada cuando se revelan detalles innecesarios sobre documentos, imágenes o videos, poniendo en riesgo la confidencialidad de la información.

En este escenario surge METASEC, como una herramienta diseñada para analizar y eliminar metadatos de manera efectiva. Se trata de un arma de doble filo, ya que está orientada no sólo a proteger nuestra información, sino que su objetivo primordial es analizar la información de terceros, mostrando los datos de una forma legible y ordenada, en los que se puedan aplicar filtros para no ver el ruido, es decir, toda esa información que no nos interesa.

Desarrollada como una aplicación de escritorio en Java para sistemas Windows, METASEC se apoya en Exiftool, una de las herramientas para la extracción y borrado de metadatos de ficheros más utilizadas y conocidas. Esto posibilita trabajar con grandes volúmenes de datos a una gran velocidad (dependiendo del procesador del usuario), ofreciendo así una solución más práctica, eficiente y legible en el análisis y gestión de metadatos en el complejo entorno de la seguridad digital.

Palabras clave:

Ciberseguridad, Metadatos, Archivos digitales, Privacidad, Aplicación de escritorio, Java, Windows.

Abstract

In the current digital era, where information flows at breathtaking speeds and communication has become omnipresent, the proper management of cybersecurity has become an unavoidable priority. In this context, metadata, often referred to as "data about data," which are those small fragments of information accompanying our digital files, plays a fundamental role in the realm of cybersecurity. Without careful management, metadata can become a potential vulnerability.

The overexposure of sensitive information, such as the precise location of a user when capturing a photograph, can be exploited by cyber threats. Privacy is jeopardized when unnecessary details about documents, images, or videos are revealed, putting the confidentiality of information at risk.

In this scenario, METASEC emerges as a tool designed to analyze and effectively remove metadata. It is a double-edged sword, not only focused on protecting our information but with the primary goal of analyzing third-party information. METASEC presents data in a readable and organized manner, allowing the application of filters to avoid noise, meaning all that information that is not of interest.

Developed as a desktop application in Java for Windows systems, METASEC uses Exiftool, one of the better known and used tool for the extraction and deletion of metadata.This enables working with large volumes of data at high speeds (depending on the user's processor), offering a more practical, efficient, and readable solution in the analysis and management of metadata in the complex environment of digital security.

Key words:

Cybersecurity, Metadata, Digital files, Privacy, Desktop application, Java, Windows.

Índice

1. Introducción y motivación

En los archivos digitales, los metadatos, a pesar de que no los vemos a simple vista, juegan un papel crucial al proporcionar información adicional sobre los datos que manejamos a diario. Sorprendentemente, la mayoría de las personas subestiman el valor de esta información, a menudo desconociendo su existencia. Esta falta de conciencia es comprensible, ya que raramente se enseña o enfatiza este aspecto en la educación convencional. Esto no significa que estemos completamente desprotegidos, empresas como Meta borran toda esta información de las publicaciones en sus redes sociales, añadiendo así una pequeña pero importante capa de seguridad.

Para comprender la transcendencia de los metadatos, imaginemos un escenario en el que Meta no realiza dicho borrado. Si configuramos nuestras imágenes para incluir la ubicación (una configuración que muchos olvidan o aceptan sin prestar atención), un atacante podría obtener detalles críticos, como las coordenadas GPS de nuestro hogar, lugar de trabajo y sitios frecuentados regularmente.

Afortunadamente, como se ha mencionado, algunas empresas han tomado conciencia de este riesgo y han implementado medidas de seguridad al respecto. Sin embargo, la cantidad abrumadora de información disponible en internet implica que gran cantidad de la información sensible aún no ha sido depurada.

Según el NISO (Oficina Nacional de Estándares de Información estadounidense), podemos categorizar los metadatos en cuatro tipos esenciales:

- **Metadatos Descriptivos**: Estos se enfocan en detallar el contenido, proporcionando información para identificar, localizar y comprender el recurso. Incluyen elementos como el título, autor y fecha de creación.

- **Metadatos Estructurales**: Su función es describir las relaciones entre las partes de un recurso, permitiendo comprender la organización interna de la información.

- **Metadatos Administrativos**: Esta categoría abarca la información necesaria para gestionar un recurso e incluye:

    - **Metadatos Técnicos:** Proporcionan información sobre archivos digitales, como el tipo de archivo y su tamaño.

    - **Metadatos de Preservación**: Ayudan en la gestión a largo plazo de archivos digitales, por ejemplo, mediante el uso de checksums.

    - **Metadatos de Derechos**: Ofrecen detalles sobre los derechos de propiedad intelectual, como los relacionados con el copyright.

- **Lenguajes de Marcado**: Estos lenguajes fusionan metadatos y contenido, siendo a veces empleados en recursos textuales para marcar elementos estructurales, señalar palabras con información semántica o definir detalles de

formato, como la cursiva. Ejemplos destacados de lenguajes de marcado son XML y HTML.

En el ámbito de la seguridad informática, el análisis de metadatos se erige como una herramienta esencial en el pentesting. Los profesionales de este sector se sumergen en la evaluación detallada de metadatos en busca de información valiosa que les oriente en una auditoría. Esto implica la extracción de datos como nombres de personas, direcciones de correo electrónico, versiones de software e incluso coordenadas GPS. No obstante, su utilidad se expande más allá del pentesting, ya que los analistas forenses pueden emplear este enfoque para discernir la autenticidad de información, como es el caso de las fake news. Ejemplo:

Ilustración 1: Imagen modificada.

Al analizar los metadatos de la Ilustración 1, se descubre que la fecha de creación coincide con la jornada de los eventos, aunque sorprendentemente, alrededor de las 11 de la noche. Esta anomalía, la creación de una imagen en un momento sin luz solar, despierta sospechas sobre posibles modificaciones. Este hallazgo impulsa una investigación más profunda, aplicando técnicas de detección de irregularidades que eventualmente conducen al descubrimiento de la imagen orignial, la Ilustración 2:

Ilustración 2: Imagen original.

Dada la transcendencia de los metadatos, emerge METASEC, una aplicación diseñada para visualizar y manipular de manera eficiente los metadatos sensibles. Esta herramienta proporciona una interfaz cómoda y legible, permitiendo realizar acciones tanto en el ámbito de la seguridad defensiva como en el ofensivo. Su denominación fusiona “META”, aludiendo a los metadatos, y “SEC”, haciendo referencia a la seguridad, encapsulando así la esencia de su propósito.

1.1. Objetivo

El objetivo principal de este proyecto es desarrollar una aplicación de escritorio para sistemas Windows, fácil de usar, donde los usuarios puedan analizar archivos en busca de información sensible y borrarla. Además de poder analizar los metadatos de páginas web estáticas, almacenar análisis y poder filtrar los metadatos. De esta forma el usuario tiene la comodidad de ver aquella información que le interesa cuando quiera y sin depender de una conexión a internet.

2. Material y métodos utilizados

En este capítulo se describen las tecnologías y herramientas utilizadas para garantizar la eficiencia y calidad del software.

2.1. Tecnologías y herramientas

Como entorno de programación, **Eclipse** se ha convertido en el candidato ideal. Es un IDE (entorno de desarrollo integrado) intuitivo y personalizable que facilita la escritura, compilación, depuración y gestión de proyectos. Su arquitectura modular permite la integración de una amplia variedad de complementos y extensiones como **WindowBuilder Editor**, que ha sido clave para realizar con rapidez y sencillez la interfaz de la aplicación, permitiendo interactuar visiblemente con los elementos que la componen.

El lenguaje de programación utilizado es **Java**. Es un lenguaje de programación orientado a objetos de propósito general que es ampliamente utilizado en el desarrollo software. Además de ser el lenguaje dominante en nuestra formación universitaria, Java es un lenguaje que soporta la programación multihilo, una característica muy importante y necesaria para satisfacer el deseo de una ejecución lo más rápida posible. Tiene librerías como **Java Swing**, aportando las estructuras necesarias para desarrollar la interfaz que, si bien no es la tecnología más moderna y que permita realizar los efectos más elaborados, esta librería contiene los recursos necesarios para adaptarse a la perfección al objetivo del proyecto.

Maven ha sido la elección para la gestión del proyecto. Esta herramienta simplifica y estandariza el proceso de construcción y gestión de proyectos automatizando tareas comunes en el ciclo de vida del desarrollo, como la gestión de dependencias y el empaquetado del proyecto.

Al tener una persistencia sencilla, el formato JSON era el indicado para almacenar los datos. Por ello se elige la librería **Jackson**. Esta biblioteca de procesamiento de JSON para Java proporciona las funcionalidades para convertir objetos Java en JSON y viceversa. Esta serialización y deserialización de datos la realiza de forma rápida y configurable. Jackson es perfecto para almacenar datos sencillos en una máquina local. Además, permite transportar e incluir en nuestra aplicación datos almacenados de una forma sencilla: solamente con mover un archivo (con el formato correcto) de un directorio a otro.

Como marco para las pruebas software, **JUnit** proporciona anotaciones y clases para definir y organizar dichas pruebas, permitiendo la automatización y validación del comportamiento del código durante el desarrollo. Las pruebas presentaban una necesidad de abrir un servidor web, **Python** nos permite una manipulación sencilla, perfecta para abrir servidores de prueba en la maquina local.

Para la recolección y eliminación de metadatos, se emplea la herramienta **Exiftool**. La elección de esta herramienta se basa en su extensa y comprensible documentación, su capacidad de manejar diversos tipos de archivos y su sencilla instalación. Para obtener información detallada sobre su funcionamiento, se puede consultar su página web en https://exiftool.org/ .

En cuanto a la descarga de archivos a través de la red, se ha optado por la herramienta **Wget**. Esta elección se fundamenta en su eficacia comprobada, su amplio soporte para protocolos de transferencia de archivos (si bien en esta primera versión de METASEC no se usan todos, en futuras versiones se pretende extender el comportamiento a protocolos como FTP) y su versatilidad para manejar diversas situaciones. Para obtener detalles adicionales sobre esta herramienta, se recomienda visitar su sitio oficial https://www.gnu.org/software/wget/ .

La implementación de un instalador mediante **Smart Install Maker** se vuelve esencial para simplificar la experiencia del usuario al integrar herramientas fundamentales como Wget o Exiftool en el sistema. La instalación manual de estas herramientas puede ser un proceso tedioso y propenso a errores para usuarios menos experimentados.

2.2. Metodología

El éxito de un proyecto multifuncional requiere la implementación de una estructura metodológica sólida y sistemática para garantizar la cohesión y el éxito global del sistema. En este sentido, he adoptado un enfoque en cascada o secuencial, que organiza de manera lineal las distintas etapas del proyecto, iniciando con la recopilación de requisitos, seguido por el diseño, la implementación y culminando con las pruebas.

La fase inicial de este enfoque secuencial se centra en una recopilación minuciosa de los requisitos del proyecto. A través de un análisis detallado, se identificaron y documentaron de manera integral todas las funcionalidades necesarias. Este proceso estableció la base fundamental para el desarrollo subsiguiente, asegurando una comprensión completa de las necesidades y expectativas del proyecto.

Una vez recopilados todos los requisitos, se llevó a cabo el diseño de una arquitectura robusta, estableciendo las interrelaciones entre los diferentes componentes del sistema, y sentando las bases para el diseño y la implementación eficiente de cada elemento.

La fase de implementación se inició tras la finalización del diseño completo del proyecto. Se fueron implementando todas y cada una de las funcionalidades por completo, enfocándose en un componente a la vez. Este enfoque secuencial garantizó la coherencia del sistema a medida que cada elemento se integraba de manera progresiva, permitiendo ajustes y mejoras continuas.

Con la implementación finalizada, se procedió a realizar las pruebas de integración, sistema y aceptación. Esta fase se orientó hacia la identificación y corrección meticulosa de errores, además de verificar que todas las funcionalidades operaran de manera óptima y cumplieran con los requisitos preestablecidos.

2.3. Planificación del trabajo

El desarrollo del proyecto se estructuró en tres fases fundamentales: Planificación, desarrollo y documentación de la memoria.

La primera etapa inició con una reunión con el tutor, donde se exploraron diversas opciones para el Trabajo de Fin de Grado (TFG) y se evaluaron aquellas que podrían despertar mayor interés. Dado mi requisito personal de que el proyecto estuviera relacionado con la ciberseguridad, se llegó a la conclusión de que una aplicación centrada en el tratamiento de metadatos sería tanto relevante como viable en el tiempo que se requiere para un trabajo final de este tipo. Durante esta fase, se discutieron las funcionalidades de interés que podrían integrarse en la aplicación.

Con el tema definido, se dio inicio a una fase de investigación y pruebas de herramientas y tecnologías. El objetivo era evaluar la complejidad del proyecto y estimar la relación entre el tiempo estimado para este tipo de trabajo y las funcionalidades a implementar, asegurando así una ejecución exitosa conforme a las directrices establecidas para los Trabajos de Fin de Grado en el grado de Ingeniería Informática de la UC.

La investigación llevó a la creación de una pequeña aplicación piloto, utilizada como prueba de concepto. Esto permitió verificar la compatibilidad entre las diversas tecnologías y herramientas seleccionadas, anticipando posibles desafíos que podrían surgir y comprometer la viabilidad del proyecto. La fase de investigación concluyó con una tutoría adicional, donde se definieron los requisitos finales de la aplicación.

En la fase de desarrollo, se adoptó una metodología secuencial, como se detalla en la sección 4.2 del documento.

Con la aplicación completamente desarrollada y todos los objetivos alcanzados, se dio inicio a la redacción de la memoria. En la ilustración 3: Diagrama de Gantt, se muestra cómo se ha distribuido el tiempo en la realización de este trabajo.

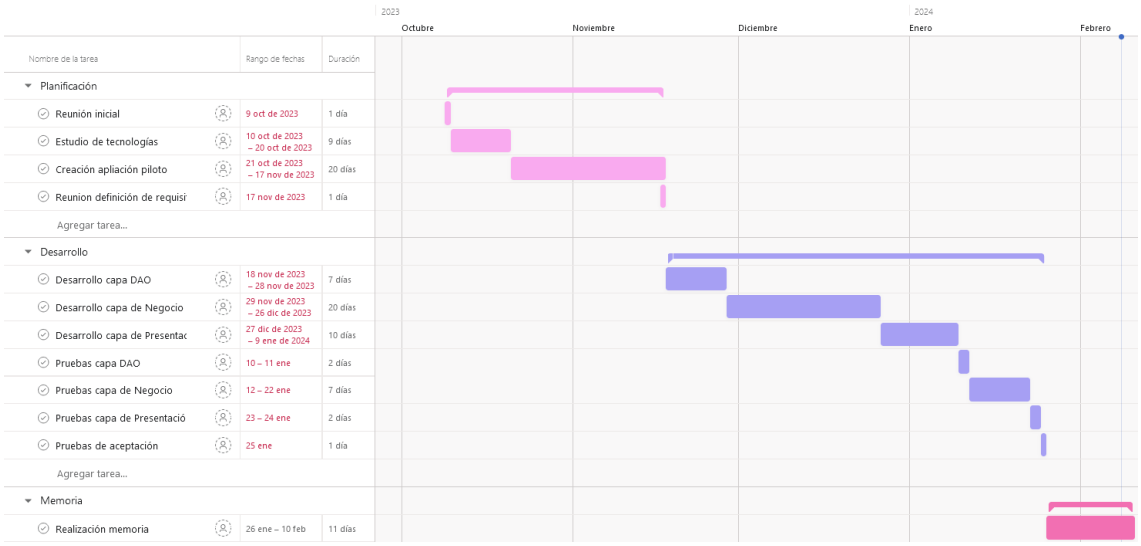

Ilustración 3: Diagrama de Gantt.

3. Análisis del problema

En este apartado, se explorará la visión de METASEC, con un breve resumen seguido de los requisitos tanto funcionales como no funcionales.

3.1. Resumen

METASEC, será una aplicación de escritorio diseñada específicamente para sistemas operativos Windows. Se presenta como una herramienta especializada en la recolección de información durante procesos de pentesting, aunque permita aplicarse en otros ámbitos por sus funcionalidades. Es importante destacar que este proceso de recopilación de datos se llevará a cabo en estricta conformidad con los marcos legales vigentes, asegurando la legalidad y ética de todas las operaciones realizadas.

El enfoque principal de METASEC estará en la extracción y análisis de metadatos asociados a archivos, con el propósito de proporcionar una visión detallada y significativa de la información contenida en ellos. Este análisis no solo se limitará a archivos individuales, sino que abarcará aquellos provenientes de diversas fuentes, como páginas web, directorios y archivos independientes.

La aplicación ofrecerá una interfaz intuitiva y práctica, diseñada para simplificar el proceso de filtrado de metadatos. Esta funcionalidad permitirá a los usuarios reducir la cantidad de información a analizar y, al mismo tiempo, focalizar la búsqueda en los elementos de mayor interés. METASEC será una herramienta eficiente y fácil de utilizar, permitiendo que incluso aquellos usuarios menos familiarizados con el pentesting puedan aprovechar sus capacidades.

Para aumentar el alcance de la aplicación, será capaz de borrar los metadatos de los archivos de los usuarios y almacenar los análisis para usarlos en un futuro. Este guardado se realizará en un formato que permita la manipulación de los datos, de forma sencilla mediante scripts.

En resumen, METASEC no solo se posicionará como una aplicación de recolección de información en el ámbito del pentesting, sino que se comprometerá a proporcionar una experiencia integral, legal y eficiente, enfocada en la manipulación y análisis de metadatos de archivos para potenciar las investigaciones y evaluaciones de seguridad.

3.2. Análisis de requisitos

En este apartado, se dictan los requisitos funcionales (Tabla 1) y no funcionales (Tabla 2) de METASEC, además de una representación de cómo gustaría que fuese la aplicación, en sus diferentes escenarios, sin perjuicio de que en la versión final se hayan realizado pequeños cambios que mejoren la estética o hagan más dinámica la interacción.

Requisitos funcionales

ID	Descripción
RF1	El usuario debe de poder seleccionar el archivo o directorio a tratar de dos formas: Escribiendo la ruta hasta el recurso o navegando a través de las carpetas con un selector.
RF2	El usuario debe de poder analizar los metadatos de los tipos de archivos más comunes como documentos, imágenes y vídeos, directorios y páginas web estáticas (introduciendo una URL). En el caso de las webs estáticas, se analizarán aquellos recursos que se encuentren dentro del dominio y sean necesarios para poder mostrar la página web.
RF3	El usuario puede analizar recursos web individuales.
RF4	El análisis debe de mostrar el tipo, valor y ruta del metadato, además de los diferentes filtros que se indican más adelante.
RF5	Se debe de poder tener varios análisis abiertos a la vez.
RF6	Una vez analizados los recursos, se mostrarán el número de recursos procesados y se dará la opción de continuar con el análisis o guardarlo para ver más tarde.
RF7	El usuario debe de poder borrar los metadatos innecesarios de aquellos archivos que no necesiten de una configuración manual para ser borrados. También se deben de poder borrar metadatos de todos los archivos de un directorio. Se tiene que dar la opción de ver los metadatos restantes.
RF8	Tanto el análisis como el borrado, deben indicar la cantidad de archivos que han sido procesados.
RF9	Dada la tabla con los metadatos, se debe de poder filtrar por columna (filtros avanzados). Estos filtros aplicados deben de mostrarse y deben de poder ser eliminados individualmente.
RF10	Dada la tabla con los metadatos, se debe de disponer de filtros predefinidos que apliquen un conjunto de filtros sobre las columnas, de forma que el usuario no requiera de experiencia con el tratado de metadatos. Estos filtros predefinidos deben también de ser mostrados y eliminados individualmente.
RF11	Se debe de poder mezclar filtros predefinidos con filtros avanzados.
RF12	Un análisis debe de poder ser guardado y abierto en cualquier otro momento. Dicho análisis almacenará el conjunto total de los metadatos, sin filtros aplicados.
RF13	Todos los análisis y recursos descargados se deben de almacenar en un directorio creado automáticamente y al que pueda tener acceso el usuario.
RF14	Los análisis guardados deben de tener un formato que sea manipulable a la hora de usarse en un script.

Tabla 1: Requisitos funcionales

Requisitos no funcionales

ID	Descripción
RNF1	Tanto el análisis como el borrado de metadatos, debe de hacerse aprovechando las características del procesador de la máquina del usuario. Esto es haciendo uso de la programación multihilo.
RNF2	La interfaz debe de ser sencilla y fácil de entender.
RNF3	La aplicación debe de ser de escritorio para entornos Windows.
RNF4	Se debe de proporcionar un instalador que instale automáticamente cualquier herramienta externa que se utilice.
RNF5	La aplicación debe de estar en inglés.

Tabla 2: Requisitos no funcionales

3.3. Mockup de la aplicación

METASEC estará compuesta por una ventana de inicio (Ilustración 4), donde se presentarán las opciones para acceder a sus diversas funcionalidades: analizar, borrar y abrir proyectos almacenados. Si se elige analizar una URL, la aplicación descargará los archivos asociados y los procesará, eliminándolos una vez completado el análisis. En caso de tratarse de la ruta de un directorio o fichero, METASEC realizará el procesamiento de los metadatos sin eliminar los archivos originales. Por otro lado, al abrir un análisis previamente guardado (también conocido como proyecto), no será necesario repetir el procesamiento, ya que este se llevó a cabo anteriormente. Para la opción de borrar metadatos, simplemente se eliminarán sin necesidad de procesamiento adicional.

En cualquiera de las opciones, excepto borrar metadatos, se abrirá una ventana (Ilustración 5) que permitirá visualizar y filtrar los metadatos extraídos mediante una tabla. Se ofrecerán diversas opciones de filtrado: los filtros avanzados permitirán al usuario filtrar individualmente las columnas de la tabla, mientras que los predefinidos utilizarán filtros avanzados de manera transparente para el usuario. Por ejemplo, si se desea buscar un metadato que contenga un nombre específico, se utilizará un filtro avanzado para filtrar la columna de valor. En cambio, si se busca ver todos los nombres presentes, se puede emplear un filtro predefinido que filtre por nombres, lo que aplicará automáticamente un conjunto de filtros sobre la primera columna (tipo de metadato) para mostrar todos los nombres de personas disponibles, simplificando así la tarea del usuario.

Para el resto de las ventanas, no se han desarrollado mockups, ya que se espera que sean ventanas simples de transición.

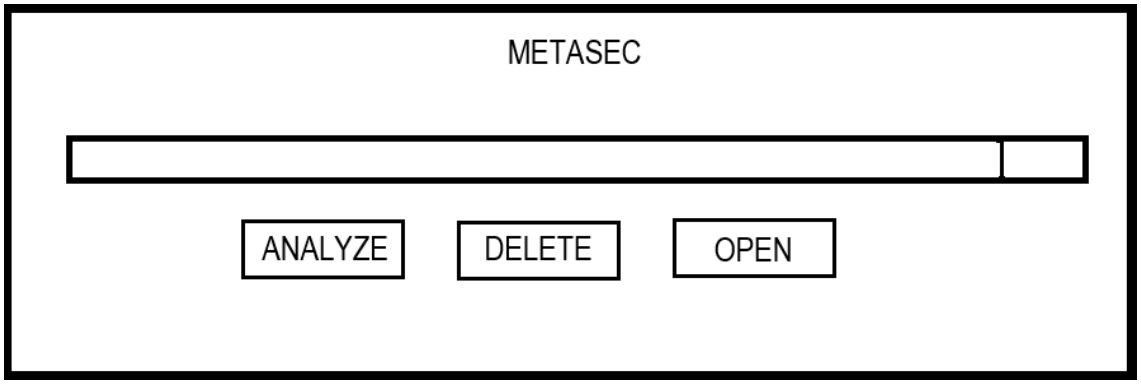

Ilustración 4: Mock ventana inicio METASEC.

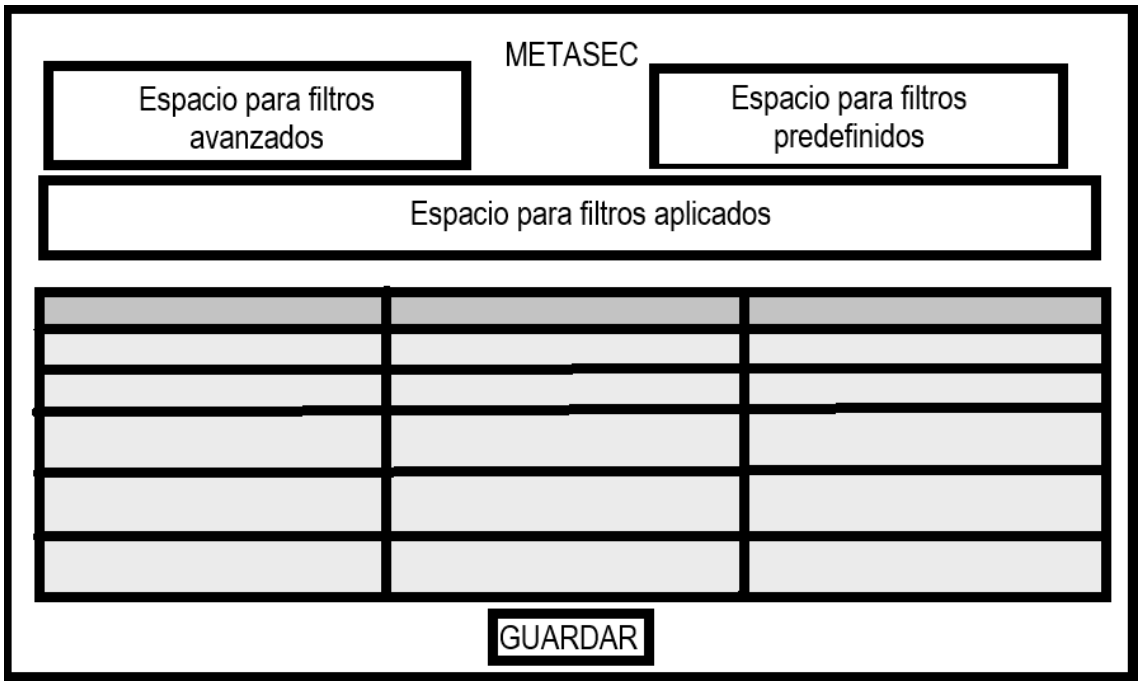

Ilustración 5: Mock ventana de análisis METASEC.

4. Diseño e implementación

En esta sección, se detalla el proceso de concepción y desarrollo de METASEC. Se explorarán los aspectos clave del diseño de la interfaz de usuario, la arquitectura software y las decisiones de implementación que han dado forma a esta herramienta.

4.1. Arquitectura

Para desarrollar la aplicación, se ha implementado una arquitectura en tres capas, como se puede ver en la ilustración 6, diseñada para promover la legibilidad, reutilización y modularidad del código. Estas tres capas, a saber, la capa DAO (Acceso a Datos), la capa Business (Negocio) y la capa de Presentación, han sido cuidadosamente estructuradas para separar las funcionalidades y responsabilidades de manera coherente.

La capa DAO, o capa de Acceso a Datos, se encarga de interactuar directamente con la fuente de datos, ya sea una base de datos, archivos o servicios externos. Aquí se gestionan las operaciones de lectura y escritura de datos, así como la manipulación de la información almacenada. Esta capa proporciona una abstracción de acceso a datos que permite al resto de la aplicación interactuar con la fuente de datos de manera independiente y cohesiva.

La capa Business, también conocida como capa de Negocio, encapsula la lógica empresarial y las reglas de negocio de la aplicación. Aquí se implementan los procesos y operaciones que manipulan los datos de acuerdo con las reglas y requisitos del negocio. Esta capa actúa como intermediaria entre la capa de Acceso a Datos y la capa de Presentación, asegurando que las operaciones realizadas en los datos sean coherentes y estén alineadas con la lógica del negocio.

Finalmente, la capa de Presentación se encarga de la interfaz de usuario y la interacción con el usuario final. Aquí se definen y gestionan los elementos visuales, como formularios, botones y menús, así como las acciones y eventos asociados a ellos. Esta capa se comunica con la capa de Negocio para obtener y presentar la información relevante al usuario, proporcionando una experiencia de usuario intuitiva y coherente.

En conjunto, estas tres capas forman un marco de desarrollo sólido y escalable que facilita el mantenimiento y la evolución de la aplicación a lo largo del tiempo. La separación clara de responsabilidades entre las capas permite una mayor flexibilidad y adaptabilidad, promoviendo una arquitectura de software robusta y eficiente.

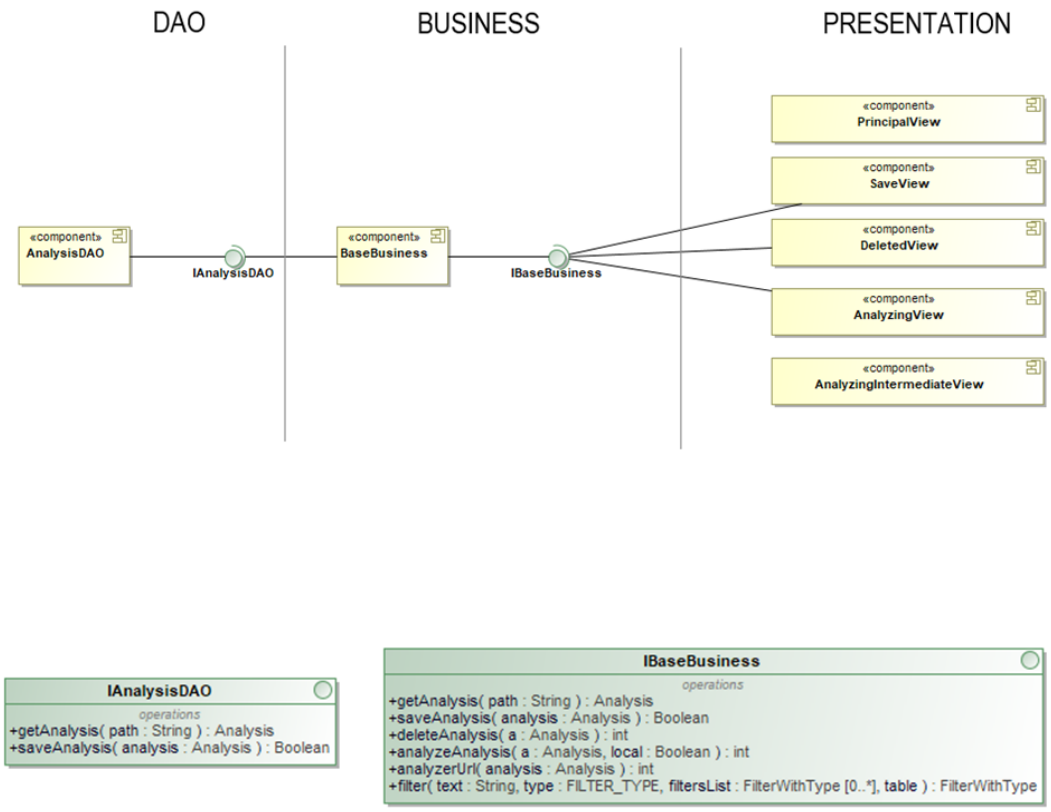

Ilustración 6: Arquitectura en 3 capas.

En la arquitectura de METASEC, vemos claramente las 3 capas: DAO, Negocio y Presentación. Estas capas se comunican entre sí a través de interfaces, asegurando así un adecuado aislamiento de los componentes y una mayor flexibilidad en el desarrollo y mantenimiento del software.

Para facilitar esta comunicación, se han definido dos interfaces:

- IAnalysisDAO: Esta interfaz facilita a la capa de negocio los métodos necesarios para gestionar la persistencia de los análisis. Implementada por AnalysisDAO, proporciona las funciones para obtener y guardar análisis.

- IBaseBusiness: Ofrece a la capa de presentación la lógica necesaria para su funcionamiento, evitando la necesidad de implementarla directamente. Implementada por BaseBusiness, incluye métodos que actúan como puente entre la capa de presentación y la capa de acceso a datos (guardar y almacenar análisis) además de los métodos necesarios para manipular y analizar los metadatos de un análisis.

A continuación, se habla en detalle de cada una de las capas.

4.2. Capa DAO

Esta capa está compuesta por la interfaz IAnalysisDAO y la clase que la implementa AnalysisDAO. Las operaciones que proporciona se describen en la siguiente ilustración:

```
7  public interface IAnalysisDAO {
8
9      /**
10      * Gets an analysis that is in a json format.
11      *
12      * @param path The path of the json.
13      * @throws IOException If there is any problem in the file mapping.
14      * @return The analysis. Null if the analysis doesn't exist.
15      *
16      */
17     public Analysis getAnalysis(String path) throws IOException;
18
19     /**
20      * Saves an analysis in the projects folder created by the application (userPath\METASEC\projects).
21      *
22      * @param analysis The analysis to be saved.
23      * @throws IOException If there was any problem with the file mapping process.
24      * @return True if everything was ok, False if there is a project with that name.
25      */
26     public boolean saveAnalysis(Analysis analysis) throws IOException;
27 }
```

Ilustración 7: Operaciones capa DAO.

En nuestro modelo de dominio (Ilustración 8), la clase principal es Analysis, que representa el conjunto de metadatos de un determinado path o URL. Esta clase está compuesta por una lista de objetos Metadata, que representan los metadatos asociados a cada fichero. Esta estructura de datos permite organizar y almacenar de

manera coherente la información recopilada durante el proceso de análisis de metadatos.

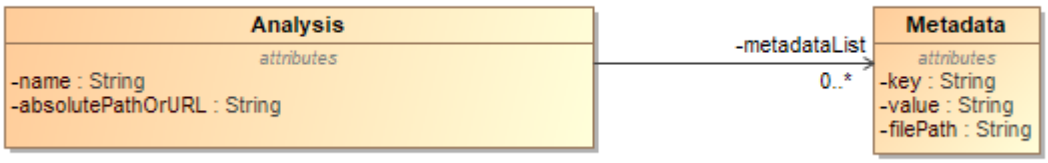

Ilustración 8: Modelo de datos capa DAO.

Para gestionar la persistencia de estos datos, se ha optado por almacenarlos localmente en un archivo JSON, usando la librería Jackson para el correcto mapeado de los datos. Este enfoque ofrece varias ventajas, como la portabilidad y la facilidad de acceso a los datos. Además, el formato JSON es ampliamente compatible y fácilmente interpretable, lo que simplifica el proceso de lectura y escritura de los datos en el sistema de archivos además de su actualización. Las anotaciones Jackson utilizadas, se representan en esta ilustración:

```
11 @JsonTypeName("analysis")
12 public class Analysis {
13
14     private String name;
15     private String absolutePath;
16     @JsonProperty("metadataList")
17     private List<Metadata> metadataList;
```

```
5  @JsonTypeName("metadata")
6  public class Metadata {
7
8      private String key;
9      private String value;
10     private String filePath;
```

Ilustración 9: Anotaciones Jackson.

Aunque la actualización de información no se implemente en la aplicación, la elección de JSON como formato de almacenamiento se fundamenta en su facilidad para ser manipulado mediante scripts, una característica esencial para los profesionales del sector. La estructura que adquiere el JSON se muestra en la ilustración 10.

```
{
    "name": "tfgPrueba",
    "absolutePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png",
    "metadataList": [
        {
            "key": "File Name",
            "value": "inicial.png",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        },
        {
            "key": "Directory",
            "value": "C:/Users/gimen/Desktop/UC/TFG/MOCKUPS",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        },
        {
            "key": "File Size",
            "value": "7.6 kB",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        },
        {
            "key": "File Modification Date/Time",
            "value": "2024:01:27 21:23:24+01:00",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        },
        {
            "key": "File Access Date/Time",
            "value": "2024:01:27 21:36:05+01:00",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        },
        {
            "key": "File Creation Date/Time",
            "value": "2024:01:27 21:23:24+01:00",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        },
        {
            "key": "File Permissions",
            "value": "-rw-rw-rw-",
            "filePath": "C:\\Users\\gimen\\Desktop\\UC\\TFG\\MOCKUPS\\inicial.png"
        }
    ]
}
```


Ilustración 10: Ejemplo con estructura JSON.

4.3. Capa de Negocio

La capa de negocio, ubicada entre la capa DAO y la capa de Presentación, está compuesta por la interfaz IBaseBusiness (Ilustración 11), que proporciona las operaciones, y el componente BaseBusiness, que contiene, entre otras, la clase que implementa dichas operaciones (BaseBusiness). Estos dos elementos se apoyan en el modelo de dominio que se describe en el siguiente apartado.

Dichas operaciones son:

```java
public interface IBaseBusiness {

    /**
     * Gets an analysis that is in a json format.
     *
     * @param path The path of the json.
     * @throws IOException If there is any problem in the file mapping.
     * @return The analysis. Null if the analysis doesn't exist.
     *
     */
    public Analysis getAnalysis(String path) throws IOException;

    /**
     * Saves an analysis.
     *
     * @param analysis The analysis to be saved.
     * @throws IOException If there was any problem with the file mapping process.
     * @return True if everything was ok, False if there is a project with that name.
     */
    public boolean saveAnalysis(Analysis analysis) throws IOException;

    /**
     * Deletes the exif metadata of the file or files from the directory which has
     * the same path as the value of the variable absolutePath of the argument a.
     *
     * @param a The analysis that contains the metadata that we want to delete.
     *
     * @return The number of files that have been analyzed.
     */
    public int deleteAnalysis(Analysis a);

    /**
     * Creates the structure of the analysis by creating and adding all the Metadata
     * objects to its list of metadata.
     *
     * @param a The analysis that has the path that we want to analyze.
     * @param local Indicates weather it is a local analysis or an online analysis.
     *
     * @return The number of files that have been analyzed.
     */
    public int analyzeAnalysis(Analysis a, boolean local);


    /**
     * Downloads an analyzes a URL then, deletes what has been downloaded.
     * @param analysis The analysis with the url.
     * @return the amount of files that have been processed.
     * @throws URISyntaxException
     * @throws IOException
     * @throws InterruptedException
     * @throws ImpossibleToConnectException
     */
    public int analyzeUrl(Analysis analysis) throws URISyntaxException, IOException, InterruptedException, ImpossibleToConnectException;

    /**
     * Adds a filter. If text is null or empty, it filters with the filters written
     * in the filtersList.
     *
     * @param text        Text that is going to act as the filter.
     * @param type        Type of filter.
     * @param filtersList List with the filters and it's type.
     * @param table       Table to be filtered.
     * @return The FilterWithType object.
     */
    public FilterWithType filter(String text, FILTER_TYPE type, List<FilterWithType> filtersList, JTable table);

}
```

Ilustración 11: Operaciones capa de Negocio.

Para poder llevar a cabo las operaciones de análisis y borrado de metadatos, se utilizan dos herramientas clave en esta aplicación: Exiftool y Wget. Juntas permiten obtener los archivos, extraer sus metadatos o bien eliminarlos. Estas dos herramientas, junto con la programación multihilo, ofrecen un rendimiento elevado a la hora de tratar con metadatos (Ilustración 12).

```java
private void analyzeAnalysis(String path, Analysis a, boolean local) {

    // Get the file from the path
    File file = new File(path);

    // Analyze directories
    if (file.isDirectory()) {
        // Parallel processing
        File[] files = file.listFiles();
        List<Thread> threads = new ArrayList<Thread>();

        for (File f : files) {
            Thread thread = new Thread(() -> {
                analyzeAnalysis(f.getAbsolutePath(), a, local);
            });
            thread.start();
            threads.add(thread);

        }

        threads.forEach(thread -> {
            try {
                thread.join();
            } catch (InterruptedException e) {
                return;
            }
        });

    }

    // Analyze files and add the metadata to the list
    if (file.isFile()) {
        try {

            ProcessBuilder processBuilder = new ProcessBuilder("exiftool", path);
            Process process = processBuilder.start();
            incrementFileCounter();
            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;
            boolean primero = true;
            while ((line = reader.readLine()) != null) {
                if (primero) {
                    primero = false;
                    continue;
                }
                String[] partes = line.split(":", 2);
                if (partes.length == 2) {
                    a.getMetadataList().add(new Metadata(partes[0].trim(), partes[1].trim(), path));
                }
            }
            if(!local) {
                file.delete();
            }

        } catch (IOException e) {
            return;
        }
    }

}
```

sección multihilo

uso exiftool

Ilustración 12: Demostración programación multihilo y uso de Exiftool.

Modelo de dominio

En esta capa, se encuentran definidas las clases que conforman el modelo de dominio (Ilustración 13), como se detalló en la capa anterior, que incluye las clases Analysis y Metadata. A estas se suman las clases FilterWithType, Constants, y dos enumerados: FILTER_TYPE y ANALYSIS_TYPE.

La clase Analysis representa una unidad de análisis, conteniendo información como el nombre y la ruta o URL del directorio o recurso web que representa, así como una lista de metadatos asociados a dicho análisis.

La clase Metadata, por otro lado, representa un metadato individual, y está compuesta por una clave (key), que define el tipo de metadato, un valor (value), que contiene la

información asociada al metadato, y la ruta del archivo (FilePath) al que corresponde dicho metadato durante el proceso de análisis.

La clase FilterWithType desempeña un papel fundamental en la gestión de filtros sobre los metadatos. Actúa como una clase auxiliar que facilita la manipulación y gestión de los filtros, albergando el propio filtro, el texto del filtro y el tipo de filtro aplicado.

Constants, como su nombre indica, es una clase auxiliar diseñada para almacenar constantes utilizadas en la aplicación como, por ejemplo, los tipos de filtros predefinidos.

Los enumerados FILTER_TYPE y ANALYSIS_TYPE representan los tipos de filtros y análisis respectivamente. FILTER_TYPE define los distintos tipos de filtro disponibles, como KEY, VALUE y PATH para los filtros avanzados, y NAME, EMAIL y GPS para los filtros predefinidos. Por otro lado, ANALYSIS_TYPE define los tipos de análisis, como LOCAL para análisis realizados sobre archivos locales, INTERNET para análisis a través de una URL, y PROJECT para análisis guardados previamente.

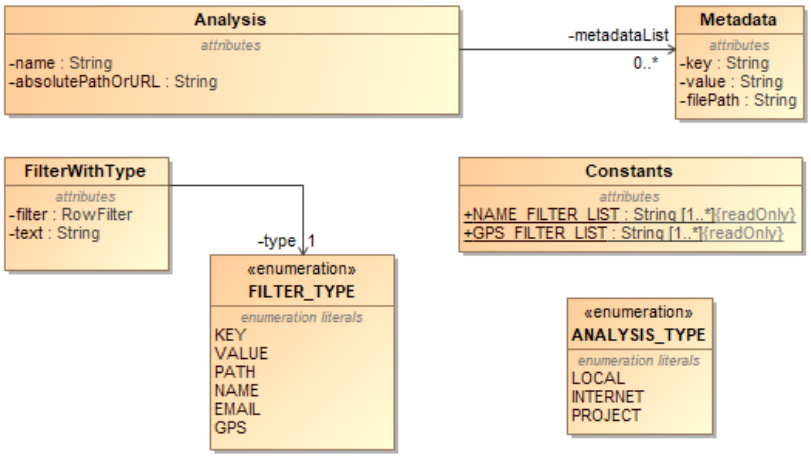

Ilustración 13: Modelo de dominio

4.4. Capa de presentación

La capa de presentación constituye el entorno visual de la aplicación, donde el usuario interactúa con el sistema. Está conformada por una serie de vistas diseñadas para adaptarse a distintos escenarios de uso. Cada vista, correspondiente a un escenario específico, se implementa como una clase individual denominada "View". Esta clase encapsula y gestiona el comportamiento de un JFrame personalizado, lo que facilita la legibilidad del código al separar claramente los elementos estructurales del JFrame de los funcionales o de comportamiento de la vista. De esta manera, la estructura del código se simplifica, mejorando su mantenibilidad y comprensión. En el caso de los errores, para ese JFrame (ErrorFrame) no se crea un view, ya que por su sencillez puede obviarse.

```java
public class SaveView {

    private SaveFrame frame;
    private IBaseBusiness business;

    public SaveView(AnalyzingView analyzingView) {
        business = new BaseBusiness();
        frame = new SaveFrame();
        frame.setVisible(true);
        frame.setLocationRelativeTo(analyzingView.getFrame());
        frame.getBtnNewButton().addActionListener(new ActionListener() {

            // Save the project.
            @Override
            public void actionPerformed(ActionEvent e) {

                analyzingView.getAnalysis().setName(frame.getTextField().getText());
                boolean result;
                try {
                    result = business.saveAnalysis(analyzingView.getAnalysis());
                    if (!result) {
                        ErrorFrame ef = new ErrorFrame("ERROR!! There is one existing project with that name");
                        ef.setLocationRelativeTo(frame);
                    }
                    frame.dispose();
                } catch (IOException e1) {
                    ErrorFrame ef = new ErrorFrame("ERROR!! Error processing the file");
                    ef.setLocationRelativeTo(frame);
                }

            }
        });
    }

    public SaveFrame getFrame() {
        return frame;
    }
}
```

Ilustración 14: Ejemplo relación view-JFrame

Como ya se ha comentado, la capa de presentación se compone de varias vistas que guían la navegación del usuario a través de la aplicación. Estas vistas y su navegación asociada son las siguientes:

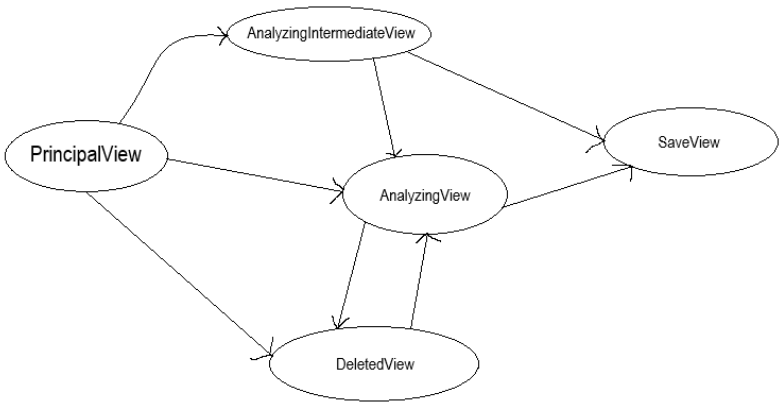

Ilustración 15: Diagrama de navegación.

PrincipalView: (Ilustración 16) Esta es la interfaz principal desde donde se selecciona el tipo de acción a realizar después de ingresar la ruta o URL(en caso necesario): Analizar en internet, analizar en la máquina local, borrar metadatos o abrir proyecto.

- Analizar en internet: Tras ingresar la URL y hacer clic en el botón correspondiente, se abre un AnalyzingIntermediateView.
- Analizar en local: Después de ingresar la ruta y hacer clic en el botón correspondiente, se abre un AnalyzingIntermediateView.
- Borrar metadatos: Tras ingresas la ruta y hacer clic en el botón correspondiente, se abre un DeletdView.
- Abrir proyecto: Muestra un sistema de navegación de archivos para seleccionar el proyecto deseado. Después de la selección, se abre un AnalyzingView.
- En caso de error: Si ocurre algún error, se abre un ErrorFrame para indicar el problema.

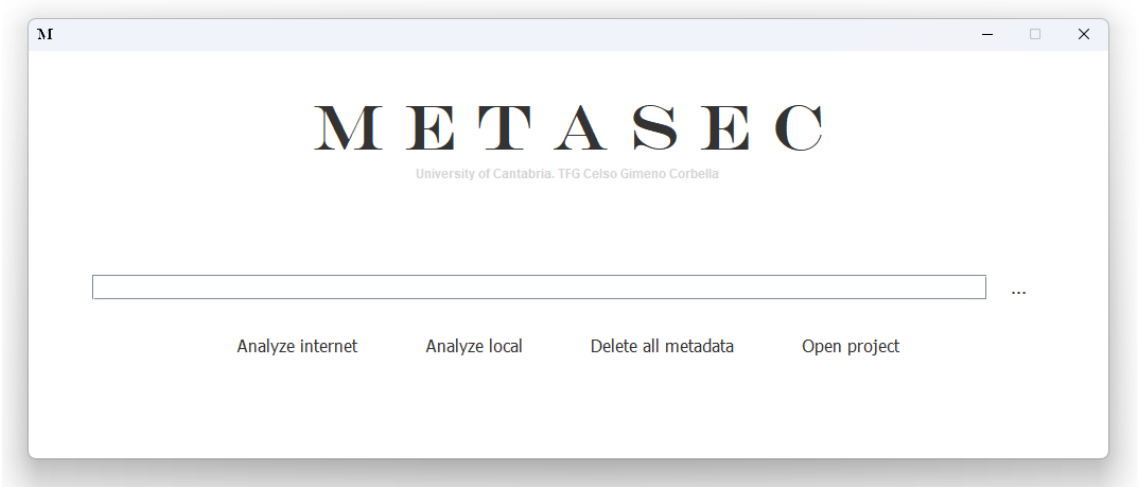

Ilustración 16: PrincipalView.

AnalyzingIntermediateView: (Ilustración 17) Aquí se visualiza la cantidad de archivos procesados y se ofrece la opción de ver el análisis de inmediato o guardarlo para su revisión posterior.

- Ver ahora: Abre un AnalyzingView con los metadatos del análisis.
- Ver más tarde: Abre un SaveView para guardar el análisis.

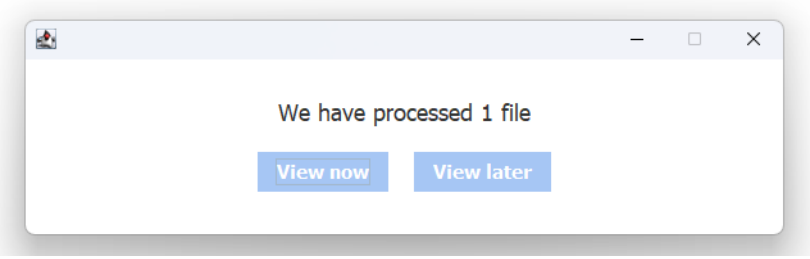

Ilustración 17: AnalyzingIntermediateView.

DeletedView: (Ilustración 18) Muestra la cantidad de archivos procesados y ofrece la opción de ver los metadatos de esos archivos abirendo un AnalyzingView.

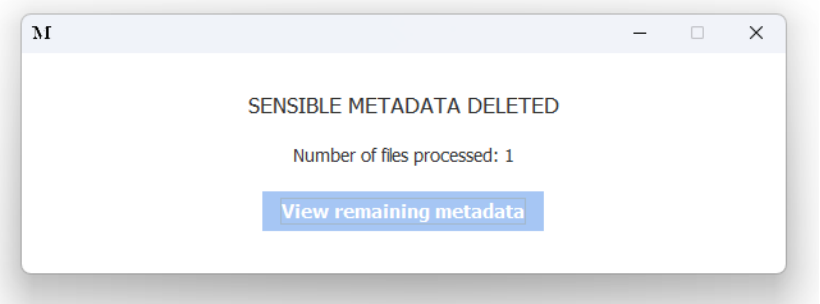

Ilustración 18: DeletedView.

AnalyzingView: (Ilustraciones 19 y 20) Muestra los metadatos del análisis en una tabla y permite filtrar por las columnas de la tabla, así como borrar dichos filtros (mediante un botón). También proporciona la opción de borrar metadatos y guardar el análisis (se abren un DeletedView y un SaveView respectivamente). Si se accede desde un DeletedView, la opción de borrar no estará disponible.

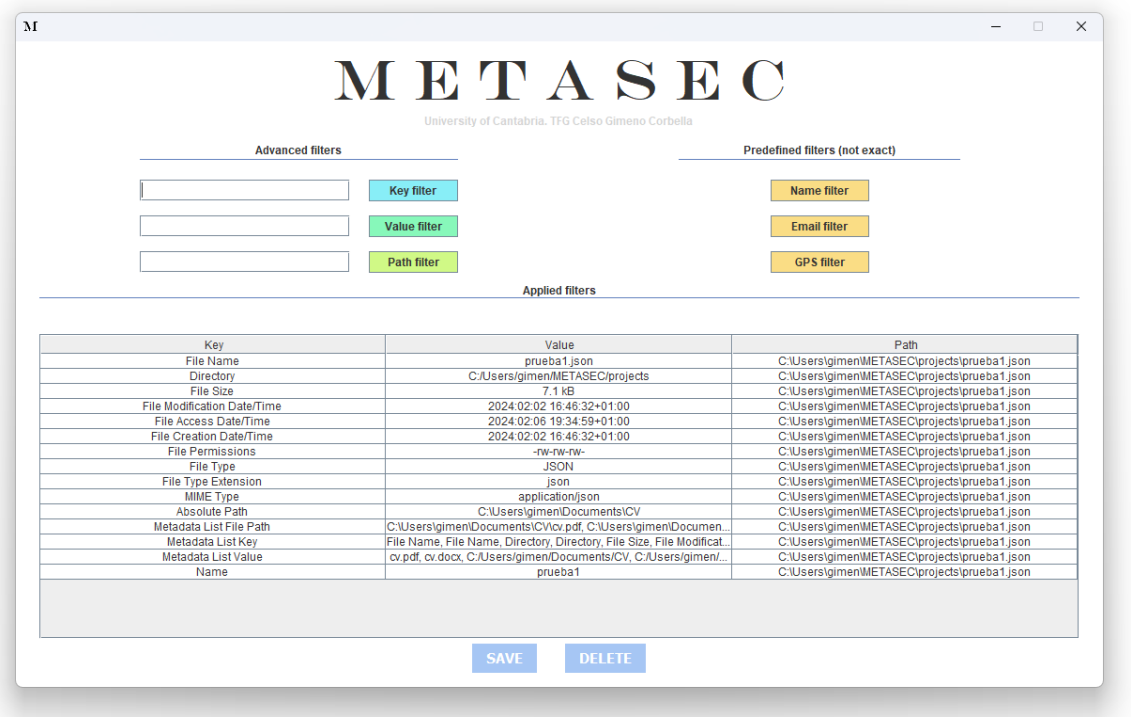

Ilustración 19: AnalyzingView.

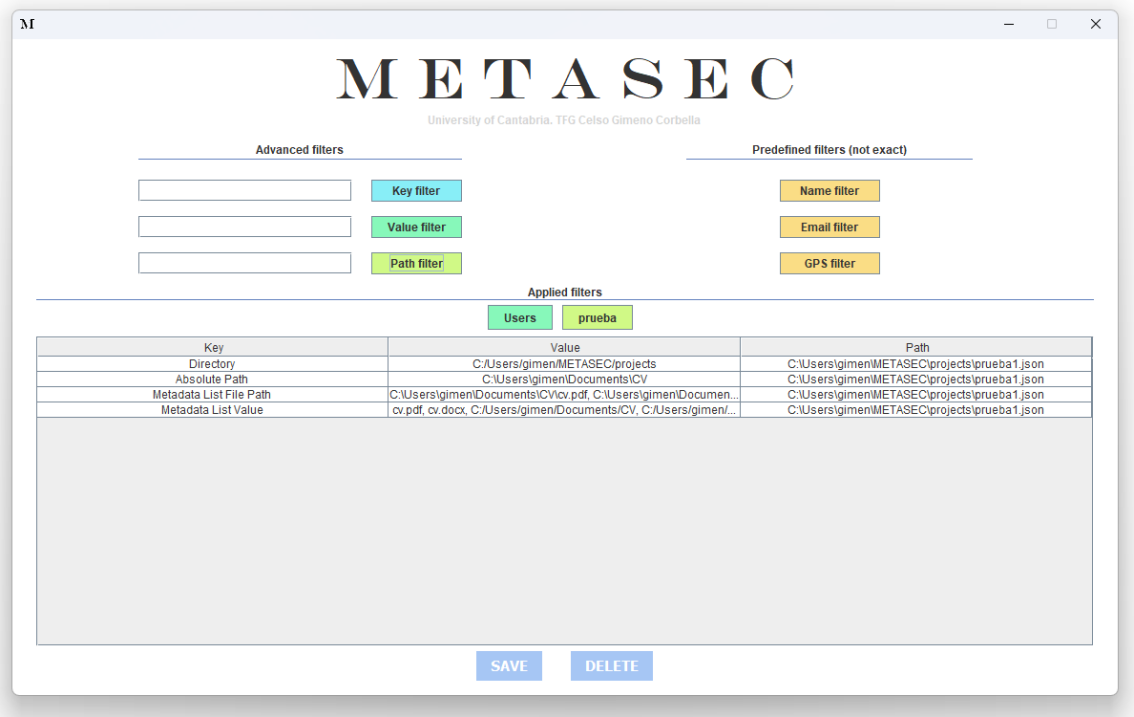

Ilustración 20: AnalyzingView con filtros aplicados.

SaveView: (Ilustración 21) Permite introducir un nombre para el análisis y guardarlo pulsando el botón correspondiente. En caso de que haya un error, se abre un ErrorFrame indicando dicho error.

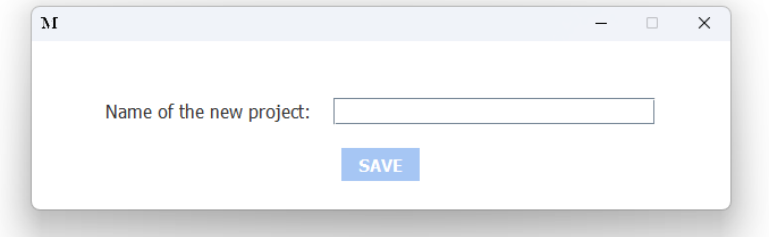

Ilustración 21: SaveView.

ErrorFrame: (Ilustración 22) Muestra el error que se ha producido.

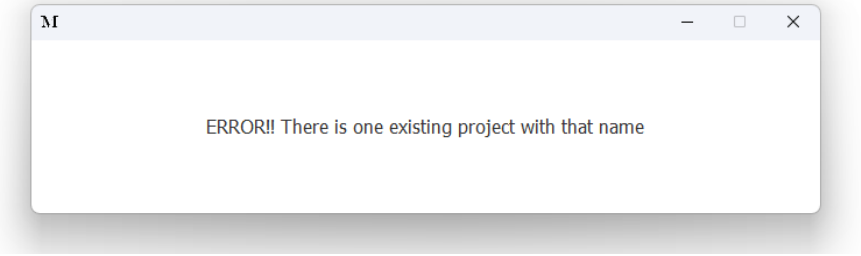

Ilustración 22: ErrorFrame.

5. Pruebas

En el desarrollo de cualquier aplicación, la garantía de calidad es un aspecto fundamental para asegurar su correcto funcionamiento y la satisfacción del usuario final. Para lograr esto, es crucial llevar a cabo una serie de pruebas exhaustivas en diferentes etapas del ciclo de desarrollo. En el caso de METASEC, se ha planificado un proceso de pruebas que abarca desde la integración de componentes individuales con el sistema hasta la validación final de la funcionalidad del sistema. A continuación, exploraremos en detalle cada tipo de prueba que se implementará en METASEC:

1. Pruebas unitarias: Verifican el correcto funcionamiento de los métodos y clases de forma aislada. Para ello se pueden usar librerías que faciliten dicho aislamiento. En este proyecto no se han realizado pruebas unitarias, la explicación de por qué se encuentra en el apartado 5.1 .

2. Pruebas de Integración: Las pruebas de integración se centran en verificar que los diferentes módulos y componentes de la aplicación funcionen de manera conjunta y coordinada. Este tipo de pruebas garantiza que las distintas partes de la aplicación interactúen correctamente entre sí, evitando posibles conflictos o incompatibilidades. En el caso de METASEC, se realizarán pruebas de integración de forma automatizada para asegurar la correcta comunicación y funcionalidad entre:

    - Capa DAO y máquina local.
    - Capa de Negocio y capa DAO.
    - Capa de Negocio y máquina local.
    - Capa de Negocio y herramientas necesarias (Exiftool y wget).

3. Pruebas de Sistema: Las pruebas de sistema tienen como objetivo evaluar el comportamiento y el rendimiento del sistema en su conjunto, una vez que todos los componentes individuales han sido integrados. Estas pruebas se enfocan en validar que la aplicación cumple con todos los requisitos funcionales y no funcionales especificados en los documentos de requisitos.

4. Pruebas de Aceptación: Las pruebas de aceptación tienen como objetivo final validar que la aplicación cumple con las expectativas y necesidades del usuario final. Estas pruebas se realizan desde la perspectiva del usuario y se centran en evaluar la usabilidad, la experiencia del usuario y la conformidad con los requisitos del usuario. En el caso de METASEC, las pruebas de aceptación se llevarán a cabo por el tutor de este TFG, Carlos Blanco, al que se le proporcionará la aplicación y mostrará su conformidad al respecto.

5.1. Pruebas unitarias

Como se comentaba anteriormente, no se han realizado este tipo de pruebas. Esta decisión se basa en un análisis detallado de cada componente y sus funcionalidades.

Empecemos por la capa DAO, donde se encuentran dos métodos cuya función principal es la escritura o recuperación de archivos del sistema de ficheros local. En este caso, realizar pruebas unitarias no cubriría adecuadamente la lógica principal de estos métodos, ya que no se abordaría la interacción con el sistema de ficheros en sí. En cambio, las pruebas de integración nos permiten evaluar tanto la interacción con el sistema de ficheros como los diferentes comportamientos de los métodos. Por lo tanto, se ha optado por no realizar pruebas unitarias en la capa DAO.

En la capa de Negocio, dos métodos actúan como puente entre la capa DAO y la de Presentación, lo que significa que solo necesitamos verificar que este puente funcione correctamente. Respecto a los demás métodos, estos se basan en otros métodos privados y en herramientas instaladas en la máquina local. Realizar pruebas unitarias en este caso podría no capturar todos los posibles escenarios, mientras que las pruebas de integración nos permiten abordar una gama más amplia de situaciones.

Finalmente, en la capa de Presentación, no se han identificado métodos relevantes aparte de aquellos relacionados con la adición de un botón de filtro. Dada la simplicidad de esta función, la prueba se realiza visualmente para garantizar su correcto funcionamiento.

Esta estrategia nos permite enfocar nuestros esfuerzos de prueba en aquellas áreas donde realmente se necesita una cobertura exhaustiva.

5.2. Pruebas de integración

En el apartado 5.1 Pruebas unitarias, ya se ha explicado por qué se obviaban y por qué pasar directamente a las pruebas de integración tiene más sentido.

Para las pruebas de integración de ambas capas (Negocio y DAO), se ha empleado JUnit5 como el marco de pruebas de integración. En cuanto a las técnicas de prueba, se han utilizado principalmente técnicas de caja negra para evaluar el comportamiento funcional de la aplicación desde una perspectiva externa. Estas técnicas se centran en probar la funcionalidad de la aplicación sin tener en cuenta su estructura interna, lo que nos permite validar que la aplicación cumple con los requisitos especificados sin necesidad de conocer su implementación subyacente.

Pruebas de integración capa DAO

Se pueden encontrar en:

/tfgmetasec/src/test/java/tfgmetasec/AnalysisDAOIntegrationTest.java

A continuación, se muestra una guía en la que se identifican las pruebas y los diferentes comportamientos que se analizan:

+getAnalysis(path: String): Analysis

Identificador	Contexto / Entrada	Comportamiento / Retorno
IAD.1a	path de un JSON correcto.	Retorna un objeto de la clase Analysis a corde con lo descrito en el JSON.
IAD.1b	path de un JSON incorrecto	IOException
IAD.1c	path de un JSON vacio.	IOException
IAD.1d	path de un fichero que no es un JSON.	IOException
IAD.1e	path de un fichero que no existe.	Retorna null

+saveAnalysis(analysis: Analysis): boolean

Identificador	Contexto / Entrada	Comportamiento / Retorno
IAD.2a	No existe el directorio en el que se guardan los análisis. Tampoco existe un análisis guardado con el nombre del objeto anlaysis.	Se crean los directorios “METASEC” y “projects”. Se guarda en ese directorio el análisis. Retorna True.
IAD.2b	Ya existe el directorio en el que se guardan los análisis. Además, ya hay un análisis guardado con ese nombre.	Retorna False.
IAD.2c	Ya existe el directorio con el que se guardan los análisis. No existe un análisis con el nombre del objeto analysis.	Se almacena el análisis en el directorio “projects”. Retorna True.

En las pruebas se puede ver, que el comportamiento de los métodos utilizados de la clase ObjectMapper no están comprobados. Esto se debe a dos razones: La primera de ellas es que se tratan de comprobaciones de muy bajo nivel y la segunda, es que dichas comprobaciones ya se espera de la librería que estén realizadas.

A continuación, se presenta como ejemplo de implementación el test para el método saveAnalysis (Ilustración 23). Es crucial que, al finalizar el test, el estado del sistema permanezca inalterado. Por ello, se incluyen dos secciones claramente identificadas: Pre-configuration y Post-configuration. Estas secciones tienen como único propósito preservar el estado del sistema y realizar cualquier preparación necesaria antes de ejecutar cada parte del test.

```java
@Test
public void testSaveAnalysis() {

    // Pre-configuration.
    String home = System.getProperty("user.home");
    String appPath = home + File.separator + "METASEC";
    String projectsPath = appPath + File.separator + "projects";
    File projectsDirectory = new File(projectsPath);
    File appDirectory = new File(appPath);
    File appDirectoryAux = null;
    if(appDirectory.exists()) {
        appDirectoryAux = new File(appDirectory.getParent(), "METASEC_TEMP");
        appDirectory.renameTo(appDirectoryAux);

    }
    Analysis a1 = new Analysis("a1");
    a1.setName("a1");
    a1.setMetadataList(null);
    Analysis a2 = new Analysis("a2");
    a2.setName("a2");
    a2.setMetadataList(null);


    // IAD.2a METASEC directory and file don't exist.
    assertDoesNotThrow(() -> {
        assertTrue(dao.saveAnalysis(a1));
        Analysis a = dao.getAnalysis(projectsPath + File.separator + a1.getName() + ".json");
        assertEquals(a1.getName(), a.getName());
        assertEquals(a1.getAbsolutePath(), a.getAbsolutePath());
        assertNull(a.getMetadataList());
    });

    // IAD.2b METASEC directory and file exist.
    assertDoesNotThrow(() -> {
    assertFalse(dao.saveAnalysis(a1));
    });

    // IAD.2c METASEC directory exists and file doesn't.
    assertDoesNotThrow(() -> {
        assertTrue(dao.saveAnalysis(a2));
        Analysis a = dao.getAnalysis(projectsPath + File.separator + a2.getName() + ".json");
        assertEquals(a2.getName(), a.getName());
        assertEquals(a2.getAbsolutePath(), a.getAbsolutePath());
        assertNull(a.getMetadataList());
    });


    // Post-configuration.
    File f1 = new File(projectsPath + File.separator + a1.getName() + ".json");
    File f2 = new File(projectsPath + File.separator + a2.getName() + ".json");
    f1.delete();
    f2.delete();
    projectsDirectory.delete();
    appDirectory.delete();

    if (appDirectoryAux != null) {
        System.out.println("p control");
        appDirectoryAux.renameTo(new File(appDirectory.getParent(), "METASEC"));
        appDirectoryAux.delete();
    }


}
```

Ilustración 23: Ejemplo test integración capa DAO

Pruebas de integración capa de Negocio

Se pueden encontrar en:

/tfgmetasec/src/test/java/tfgmetasec/BaseBusinessIntegrationTest.java

A continuación, se muestra una guía en la que se identifican las pruebas y los diferentes comportamientos que se analizan:

+getAnalysis(path: String): Analysis

Identificador	Contexto / Entrada	Comportamiento / Retorno
IBB.1a	path de un JSON correcto.	Retorna un objeto de la clase Analysis a corde con lo descrito en el JSON.
IBB.1b	path de un JSON incorrecto	IOException
IBB.1c	path de un JSON vacio.	IOException
IBB.1d	path de un fichero que no es un JSON.	IOException
IBB.1e	path de un fichero que no existe.	Retorna null

+saveAnalysis(analysis: Analysis): boolean

Identificador	Contexto / Entrada	Comportamiento / Retorno
IBB.2a	No existe el directorio en el que se guardan los análisis. Tampoco existe un análisis guardado con el nombre del objeto anlaysis.	Se crean los directorios “METASEC” y “projects”. Se guarda en ese directorio el análisis. Retorna True.
IBB.2b	Ya existe el directorio en el que se guardan los análisis. Además, ya hay un análisis guardado con ese nombre.	Retorna False.
IBB.2c	Ya existe el directorio con el que se guardan los análisis. No existe un análisis con el nombre del objeto analysis.	Se almacena el análisis en el directorio “projects”. Retorna True.

+analyzeUrl(analysis: Analysis): int

Identificador	Contexto / Entrada	Comportamiento / Retorno
IBB.3a	Análisis vacío que contiene la URL de un recurso de un directorio en la máquina local (levantado en un servidor) con dos ficheros y un directorio con otro fichero.	Análisis con el listado completo de los metadatos de los archivos. Se retorna el número de ficheros procesados. No queda rastro de los ficheros descargados para su análisis.
IBB.3b	Análisis vacío que contiene la URL de un recurso de un fichero en la máquina local (levantado en un servidor).	Análisis con el listado completo de los metadatos del fichero. Se retorna el número de ficheros procesados. No queda rastro de los ficheros descargados para su análisis.
IBB.3c	Análisis vacío con la URL de un recurso que no existe.	Exception
IBB.3d	Análisis vacío con una URL mal formada.	Exception

+analyzeAnalysis(analysis: Analysis, local: boolean): int

Identificador	Contexto / Entrada	Comportamiento / Retorno
IBB.4a	Análisis vacío con el path de un directorio que contiene dos ficheros y un directorio con otro fichero. La variable local a true.	Análisis con el listado completo de los metadatos de los ficheros. Retorna el número de ficheros procesados.
IBB.4b	Análisis vacío con el path de un fichero. La variable local a true.	Análisis con el listado completo de los metadatos del fichero. Retorna el número de ficheros procesados.
IBB.4c	Análisis vacío con el path de un directorio vacío. La variable local a true.	Análisis sin metadatos. Retorna el número de ficheros procesados.

+deleteAnalysis(analysis: Analysis): int

Identificador	Contexto / Entrada	Comportamiento / Retorno
IBB.5a	Análisis con el path de un fichero con metadatos intactos.	Se han borrado los metadatos esperados del fichero. Se retorna el número de ficheros procesados.
IBB.5b	Análisis con el path de un fichero con los metadatos borrados.	Se mantienen los metadatos del fichero. Se retorna el número de ficheros procesados.
IBB.5c	Análisis con el path de un directorio vacío.	Se retorna el número de ficheros precesados.
IBB.5d	Análisis con el path de un directorio con dos ficheros y un directorio con un fichero.	Se borran los metadatos de todos los ficheros. Se retorna el número de ficheros procesados.

+filter(text: String, type: FilterWithType, filtersList: List, table: JTable): FilterWithType

Identificador	Contexto / Entrada	Comportamiento / Retorno
IBB.6a	Text es null. Tabla con tres filas. No hay filtros aplicados.	La tabla no se modifica. Se retorna null.
IBB.6b	Text está vacío (“”). Tabla con tres filas. No hay filtros aplicados.	La tabla no se modifica. Se retorna null.
IBB.6c	Tabla con tres filas. No hay filtros aplicados. Se quiere añadir un KEY_FILTER que lo cumple solo una fila.	La tabla ahora solo tiene una fila. Se retorna y añade a la lista de filtros un nuevo

		objeto de la clase FilterWithType, con el mismo filtro, tipo y texto que la entrada.
IBB.6d	Tabla con cuatro filas. Hay un filtro aplicado. Se quiere añadir un VALUE_FILTER que solo cumple una fila.	La tabla ahora solo tiene una fila. Se retorna y añade a la lista de filtros un nuevo objeto de la clase FilterWithType, con el mismo filtro, tipo y texto que la entrada.
IBB.6e	Tabla con cuatro filas. Hay un filtro aplicado. Se quiere añadir un PATH_FILTER que solo cumple una fila.	La tabla ahora solo tiene una fila. Se retorna y añade a la lista de filtros un nuevo objeto de la clase FilterWithType, con el mismo filtro, tipo y texto que la entrada.
IBB.6f	Tabla con cuatro filas. No hay filtros aplicados. Se quiere añadir un filtro del tipo NAME que cumplen dos filas.	La tabla ahora tiene dos filas. Se retorna y añade a la lista de filtros un nuevo objeto de la clase FilterWithType, con el mismo filtro, texto y tipo que la entrada.
IBB.6g	Tabla con cuatro filas. No hay filtros aplicados. Se quiere añadir un filtro del tipo EMAIL que cumplen dos filas.	La tabla ahora tiene dos filas. Se retorna y añade a la lista de filtros un nuevo objeto de la clase FilterWithType, con el mismo filtro, texto y tipo que la entrada.
IBB.6h	Tabla con cuatro filas. Hay dos filtros aplicados. Se quiere añadir un filtro del tipo GPS que cumplen dos filas.	La tabla ahora tiene dos filas. Se retorna y añade a la lista de filtros un nuevo objeto de la clase FilterWithType, con el mismo filtro, texto y tipo que la entrada.
IBB.6i	Tabla con cuatro filas. No hay filtros aplicados. Se quiere añadir un filtro que no cumple ninguna fila.	La tabla ahora tiene 0 filas. Se retorna y añade a la lista de filtros un nuevo objeto FilterWithType, con el mismo filtro, tipo y texto que la entrada.

5.3. Pruebas de sistema

Una vez finalizado el código y, repasado y corregido los problemas encontrados en las pruebas de integración, hay que comprobar que las funcionalidades descritas en los requisitos funcionales se satisfacen. Por lo que se ejecuta la aplicación y manualmente se va a comprobar que la aplicación satisface dichos requisitos.

Ahora que sabemos que todos los requisitos funcionales están satisfechos, procedemos con los no funcionales. Éstos también se van a comprobar manualmente, pero aquí vamos a poner como ejemplo el que es más complicado de comprobar:

RNF1: *Tanto el análisis como el borrado de metadatos, debe hacerse aprovechando las características del procesador de la máquina del usuario. Esto es haciendo uso de la programación multihilo.*

La forma en la que se ha probado el RNF1 es utilizando las propias herramientas que Windows nos ofrece. En este caso vamos a realizar una ejecución de METASEC, y usando la aplicación 'Administrador de tareas', vamos a poder comprobar la cantidad de procesos en ejecución que tenemos en un momento determinado. Para que de tiempo a poder tomar capturas de pantalla, la muestra tiene que ser grande, por lo que pruebo a analizar los metadatos de todos los archivos de mi escritorio. En la ilustración 24, se puede ver cómo existen numerosos procesos 'exiftool.exe' y cómo el porcentaje de uso de la CPU sube hasta el 100%. En el caso de borrar los metadatos, ocurre exactamente lo mismo.

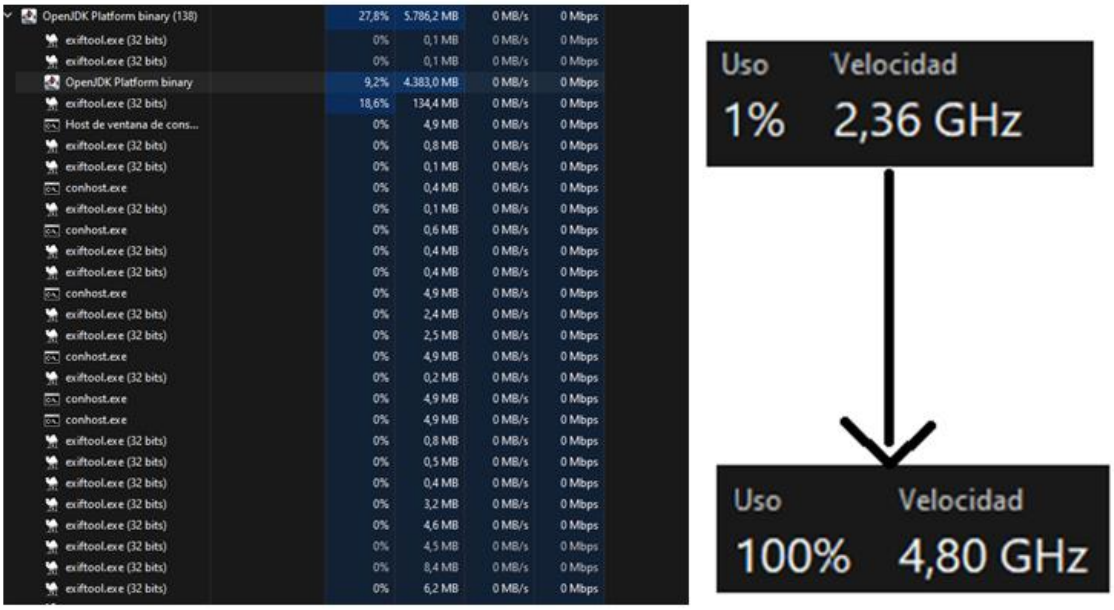

Ilustración 24: Demostración RNF1

6. Conclusiones

En este último apartado, se abordará la conclusión del trabajo, evaluando el cumplimiento de los objetivos establecidos y proyectando el futuro de la aplicación.

6.1. Objetivo

Este proyecto se propuso crear una aplicación que simplificara el análisis de metadatos en diversos contextos, ya sea para el análisis de riesgos desde la perspectiva de un atacante o para fortalecer la seguridad desde el punto de vista del usuario. En este sentido, la aplicación ha logrado cumplir su objetivo principal al proporcionar una herramienta intuitiva y eficiente para esta tarea específica.

Además, se ha enfrentado con éxito a los desafíos técnicos inherentes a la gestión de metadatos y la integración de funciones esenciales, como la eliminación de datos sensibles y el almacenamiento de análisis para su posterior referencia. La adopción de una arquitectura de tres capas ha resultado efectiva en la organización y distribución de responsabilidades, asegurando así la escalabilidad y la mantenibilidad del sistema a largo plazo. Por último, las pruebas realizadas han desempeñado un papel crucial al corregir y garantizar la funcionalidad óptima de la aplicación, añadiendo un nivel adicional de confianza en su desempeño.

Todo este proceso ha dado fruto a la primera versión de METASEC que, a pesar de haber alcanzado los objetivos principales, tiene todavía mucho camino por delante para optimizar la experiencia del usuario.

6.2. Futuras versiones

A medida que se ha ido planificando y desarrollando la aplicación, el conocimiento y el interés en los temas abordados han ido en aumento, lo que ha dado lugar a la generación de nuevas ideas para ampliar su alcance, tanto para los usuarios interesados en la parte ofensiva como para aquellos en la defensiva.

Como trabajo futuro, las pruebas de aceptación podrían ser ampliadas a profesionales, para que evalúen y comenten mejoras a realizar.

Una idea que considero prometedora para mejorar METASEC es adaptarla para que sea compatible con el protocolo FTP. Aunque esto implicaría salir del marco legal actual, sería una opción muy interesante para los pentesters, ya que les permitiría acceder a un mayor número de documentos con información relevante. Afortunadamente, la herramienta utilizada para la descarga de archivos, Wget, es compatible con dicho protocolo, lo que simplificaría el proceso. Sin embargo, sería útil considerar la posibilidad de agregar la opción de fuerza bruta para identificar usuarios.

Otra idea sería permitir el borrado individual de metadatos y la manipulación de los mismos a través de la vista de análisis. Esto aumentaría la seguridad al confundir al atacante y dificultar su tarea.

Estas son solo algunas de las ideas principales que podrían implementarse en el futuro para mejorar METASEC. Sin embargo, existen numerosas posibilidades de mejoras adicionales que podrían explorarse y desarrollarse con el tiempo, llevando así a METASEC a un nivel aún más avanzado.

7. Bibliografía

Arlow, J. (2005). *UML 2 and the Unified Process: Practical Object-oriented Analysis and Design.* Addison-Wesley.

Foundation, F. S. (s.f.). *WGET - GNU Project - Free Software Foundation*. Obtenido de https://www.gnu.org/software/wget/

Harvey, P. (s.f.). *ExifTool by Phil Harvey*. Obtenido de https://exiftool.org/

Pilone, D., & Pitman, N. (2005). *UML 2.0 in a Nutshell.* O'Reily Media, Inc.

Riley, J. (2017). *Understandig Metadata: What is Metadata, and What is it For?* Obtenido de https://digital.library.unt.edu/ark:/67531/metadc990983/m2/1/high_res_d/understanding_metadata.pdf

Sommerville, I. (2011). *Software Engineering.* Addison Wesley Longman.