Facultad
de
Ciencias

SOLUCIÓN PARA LA AUTOMATIZACIÓN DE LA REVISIÓN DE LOGS EN UN SISTEMA DE GESTIÓN DE RECLAMACIONES

(Automated Log Review Solution for a Claims Management System.)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Laura Campo Ruiz

Director: Diego García Saiz

Co-Director: Vanesa Villegas Calderón

Julio - 2025

AGRADECIMIENTOS

Este trabajo va dedicado a mi familia. A mi hermana María, por ser mi ejemplo a seguir y demostrarme que con esfuerzo y perseverancia todo es posible. A mi padre, por compartir conmigo el entusiasmo y el amor por esta profesión que ahora también compartimos. Y a ti, mamá, por cada aliento acompañado de un abrazo reconfortante que me ayudaba a seguir cuando quería rendirme.

Este logro lleva mi nombre, pero es mérito de los cuatro. GRACIAS.

RESUMEN

El objetivo de este Trabajo de Fin de Grado (TFG) es automatizar el proceso de revisión y análisis de los ficheros de *logs* de un sistema crítico de gestión de reclamaciones de usuarios, actualmente mantenido por la empresa Eviden. Este sistema, que realiza tareas esenciales, requiere de una revisión manual diaria de los *logs* para garantizar su correcto funcionamiento. Con este proyecto, se busca automatizar dicha tarea mediante el procesamiento de los *logs* y la generación de informes que faciliten la revisión de los errores de manera eficiente.

El proceso de automatización se llevará a cabo mediante la obtención de los *logs* desde una URL o desde una ruta local proporcionada por un *Backoffice*, lo que permitirá su análisis. El sistema software desarrollado se centrará en la identificación y el procesamiento de las trazas de error, clasificándolas según su tipo y proporcionando una descripción para cada uno. Se utilizará una base de datos Oracle para almacenar los errores comunes y su clasificación, y se implementará una solución tanto en el *frontend*, con JavaFX, como en el *backend*, con Java, lo que garantizará una interfaz visual y una gestión adecuada de los datos.

Al finalizar el procesamiento, se generará un informe que resumirá los errores encontrados y sus posibles soluciones. Este informe podrá ser utilizado para dar respuesta a la revisión solicitada por el cliente. El sistema contará con distintas categorías de errores, como errores ya conocidos, problemas de comunicación con servicios externos o errores relacionados con la modificación de archivos XML, lo que permitirá un tratamiento adecuado y personalizado de cada caso.

Abstract

The aim of this Bachelor's Thesis (TFG) is to automate the process of reviewing and analysing the log files of a critical user claims management system, currently maintained by Eviden. This system, which performs essential tasks, requires daily manual review of the logs to ensure its proper functioning. The goal of this project is to automate this task by processing the logs and generating reports that facilitate efficient error review.

The automation process will be carried out by obtaining the logs from a URL or a local path provided by a Backoffice, allowing for their analysis. The system will focus on identifying and processing error traces, classifying them by type, and providing a description for each one. An Oracle database will be used to store common errors and their classification, and a solution will be implemented both on the frontend, using JavaFX, and on the backend, using Java, ensuring a visual interface and proper data management.

Upon completing the processing, a report will be generated summarising the errors found and their potential solutions. This report can be used to respond to the client's review request. The system will include various categories of errors, such as known errors, communication issues with external services, or errors related to XML file modification, enabling tailored and appropriate handling of each case.

ÍNDICE

1. INTRODUCCIÓN

En este apartado se introduce el trabajo de fin de grado, justificando la selección del tema, así como los objetivos que se quieren alcanzar con su implementación.

1.1. Contexto y justificación del trabajo

En el ámbito empresarial, resulta fundamental llevar a cabo análisis periódicos sobre el funcionamiento de los procesos internos de sus sistemas con la finalidad de evitar pérdidas de información o fallos críticos que puedan comprometer la estabilidad del sistema. De la necesidad de monitorizar estos procesos nacen los ficheros *logs*, que desempeñan un papel esencial. Estos ficheros recogen información detallada acerca de las operaciones realizadas por el sistema, permitiendo así un seguimiento exhaustivo de su comportamiento [1].

No obstante, el mero almacenamiento de estos registros no es suficiente, sino que es necesario un análisis detallado que permita identificar el origen de los errores y el impacto que los mismos pueden suponer. Esta labor de análisis forma parte de la rutina diaria de muchas organizaciones permitiendo detectar si los errores son aislados o si pueden derivar en fallos a mayor escala en el proyecto.

A pesar de que gran parte de los errores registrados suelen ser recurrentes y conocidos por los profesionales del área, el análisis manual de los *logs* continúa siendo una tarea que demanda un elevado esfuerzo temporal y operativo. Esta situación representa una inversión significativa de tiempo, el cual podría ser dedicado a otras tareas.

Eviden, la empresa con la que colaboré durante el desarrollo de este proyecto se especializa en el diseño y mantenimiento de sistemas para compañías del sector energético. En mi caso, me integré en el equipo encargado del mantenimiento de los sistemas de EDP, cuya labor incluye, entre otras funciones, el monitoreo del sistema *Gredos* —una plataforma de gestión de reclamaciones— para garantizar su correcto funcionamiento y detectar posibles incidencias.

Es por todo esto que nace este TFG, cuya función es automatizar la revisión y análisis de los *logs* con la finalidad de aumentar la eficiencia del proceso, además de reducir el tiempo necesario para su ejecución.

1.2. Desglose de objetivos

El objetivo final es desarrollar una herramienta capaz de generar informes completos y precisos sobre los errores detectados, de forma rápida y eficiente, reduciendo así la carga operativa del análisis manual y mejorando la capacidad de

respuesta ante posibles incidencias. Para alcanzar este objetivo general, se plantean además una serie de objetivos específicos complementarios:

- **La implementación de interfaces interactivas e intuitivas**, que faciliten el análisis de los errores, con el objetivo de facilitar al profesional el procedimiento.
- **Extracción y recopilación exhaustiva de datos históricos**, con el fin de construir una base de datos lo más completa posible a partir de los errores registrados en los logs a lo largo del tiempo. Esto permitirá minimizar la aparición de errores no identificados previamente.
- **Implementación de un diseño escalable y fácilmente mantenible**, que permita incorporar nuevas funcionalidades de forma sencilla, sin requerir modificaciones estructurales significativas en el sistema.

En conclusión, el desarrollo de esta herramienta no solo busca automatizar la generación de informes de errores, sino también optimizar el proceso de análisis mediante interfaces intuitivas, el aprovechamiento de datos históricos y un diseño escalable.

2. MARCO TEÓRICO Y TECNOLOGÍAS UTILIZADAS

Este apartado permite comprender y conocer el contexto sobre el que se ha desarrollado el TFG, detallando las tecnologías utilizadas para la implementación del sistema.

Uno de los elementos centrales sobre los que se articula el proyecto son los *logs* del sistema. En el ámbito de la informática, un *log* es un registro secuencial de los eventos, acciones o mensajes generados por un sistema, aplicación o proceso [2]. Estos registros resultan esenciales para tareas como la depuración o el monitoreo, ya que permiten rastrear el funcionamiento interno del sistema y detectar posibles fallos [3]. En el contexto de este proyecto, los errores recopilados en los *logs* constituyen la base sobre la que se genera el informe final del sistema, cuyo objetivo es ofrecer una visión detallada de los fallos detectados y su posible causa.

Asimismo, el proyecto se apoya en tecnologías y paradigmas propios de la programación orientada a objetos, característica principal del lenguaje Java, lo que facilita el desarrollo modular y el mantenimiento del sistema. Además, el proyecto ha contemplado la importancia de las pruebas mediante *frameworks* como *JUnit* y *Mockito*, fundamentales para garantizar la fiabilidad de los componentes desarrollados.

2.1. Metodología de desarrollo software

La metodología de desarrollo software empleada en el proyecto se ha basado en un enfoque ágil e incremental, que ha permitido construir el sistema de forma progresiva mediante la implementación de una primera versión básica y la posterior incorporación de nuevas funcionalidades en sucesivos incrementos. Este planteamiento ha favorecido la adaptación del desarrollo a los resultados obtenidos en cada fase y ha facilitado la detección temprana y la corrección de posibles errores. [4]

La organización y planificación de las tareas se ha gestionado mediante Jira, herramienta que ha permitido dividir el trabajo en actividades y subtareas, priorizar los elementos más relevantes y monitorizar el avance del proyecto. Aunque no se ha seguido de forma estricta un marco de trabajo ágil concreto como *Scrum*, el proyecto ha adoptado varias de sus prácticas y principios, adaptados al contexto de un desarrollo individual.

2.2. Tecnologías utilizadas

Para el desarrollo del sistema propuesto se han empleado diversas tecnologías, seleccionadas en función de criterios de familiaridad, funcionalidad y adecuación a los objetivos del proyecto.

Oracle SQL Developer: La base de datos del sistema se ha gestionado mediante Oracle SQL Developer, un entorno de desarrollo integrado proporcionado por Oracle que permite la administración y consulta de bases de datos relacionales. Esta herramienta ha sido especialmente útil para trabajar sobre la base de datos existente del proyecto Gredos, facilitando la creación y modificación de esquemas, la ejecución de consultas SQL y el almacenamiento de un listado de errores comunes junto con sus descripciones y tipologías.

Java y Eclipse: El lenguaje de programación empleado ha sido Java, debido a su robustez, portabilidad y a la experiencia previa adquirida durante el grado. El desarrollo del código se ha llevado a cabo en el entorno de desarrollo integrado Eclipse, una herramienta ampliamente utilizada en el ámbito profesional que destaca por su soporte para proyectos complejos, su integración con herramientas externas y sus capacidades avanzadas de depuración. Con Java y Eclipse se ha implementado la lógica encargada del procesamiento de los logs, la clasificación automática de errores y la inserción de nuevos registros en la base de datos.

JavaFX: Para el diseño de la interfaz gráfica se ha utilizada JavaFX, una biblioteca que, pese a ser inicialmente desconocida, se ha seleccionado por la versatilidad de sus componentes y su capacidad para crear interfaces interactivas y modernas. JavaFX ha permitido dotar al sistema de una interfaz que facilita al usuario la consulta de errores, la generación de informes y la inserción de nuevos datos de forma visual e intuitiva. [5]

JUnit y Mockito: Para la implementación de las pruebas unitarias del sistema se han utilizado *JUnit* y *Mockito*, dos herramientas ampliamente reconocidas en el desarrollo de software en Java. *JUnit* ha permitido verificar el correcto funcionamiento de los distintos módulos mediante la definición y ejecución de pruebas automáticas [6]. Por su parte, *Mockito* ha facilitado la creación de objetos simulados (*mocks*), lo que ha hecho posible aislar los componentes en prueba y comprobar su comportamiento de forma independiente. El uso de estas herramientas ha contribuido a garantizar la fiabilidad y calidad del sistema desarrollado.

Jira: Para la organización, planificación y seguimiento de las tareas del proyecto se ha empleado Jira, una herramienta ampliamente utilizada en la gestión de proyectos de software. Gracias a su uso, ha sido posible dividir el trabajo en tareas y subtareas, asignar prioridades, lo que ha facilitado el control del progreso del proyecto. Además, Jira ha permitido registrar avances y mantener una visión global del estado del desarrollo.

3. REQUISITOS

En este apartado se explican los requisitos funcionales y no funcionales que debe cumplir el sistema para cubrir todas sus funcionalidades correctamente. Además, en base a estos se han especificado los casos de uso necesario para cubrir todos los requisitos.

3.1. Requisitos funcionales

A continuación, en la Tabla 3.1.1 se describen los requisitos funcionales del sistema, los cuales definen las funcionalidades específicas que este debe ser capaz de implementar el sistema para cumplir con sus objetivos:

Identificador	Descripción
RF1: Entrada de datos (por URL o archivo *Excel*)	El sistema debe permitir al usuario proporcionar los datos de entrada necesarios para el análisis de *logs*. Esta entrada podrá realizarse mediante dos mecanismos principales, carga de archivos *Excel* o vía URL.
RF2: Procesamiento de *logs* (extracción de trazas de errores)	Una vez que los archivos de log han sido cargados correctamente, el sistema deberá comenzar con su procesamiento. El objetivo principal de esta etapa es extraer las trazas de error que se encuentren registradas en los *logs*.
RF3: Almacenamiento en base de datos	El sistema debe integrarse con una base de datos que permita tanto la consulta como la inserción de información relacionada con los errores.
RF4: Generación de informes	Tras el análisis de los *logs* y la clasificación de los errores encontrados, el sistema debe generar un informe resumen que contenga una lista detallada de los errores detectados, junto con sus respectivas descripciones que a su vez deberán estar clasificado en función de la categoría (*switchingReclamaciones*, *AutomatizacionReclamaciones* o *GeneraciónCartas*) El informe generado debe ser accesible para el usuario mediante una interfaz que permita varias acciones como puede ser “Guardar”, “Visualizar” o “Editar”.
RF5: Interfaz gráfica de usuario (GUI)	El sistema debe disponer de una interfaz gráfica de usuario en este caso desarrollada utilizando *JavaFX*.
RF6: Soporte de múltiples errores	El sistema debe ser capaz de clasificar automáticamente los errores detectados durante el análisis de los *logs* en función de su tipología.

Tabla 3. 1. 1 – Requisitos funcionales.

3.2. Requisitos no funcionales

Los requisitos no funcionales reflejados en la Tabla 3.2.1 describen las características del sistema que no están directamente relacionadas con las funcionalidades específicas, pero que son cruciales para asegurar su correcto funcionamiento, rendimiento y la calidad de la experiencia del usuario. Los requisitos no funcionales definidos para la automatización de los *logs* son los siguientes:

Identificador	Descripción
RNF1: Usabilidad	El sistema debe ser intuitivo y fácil de usar para cualquier usuario, independientemente de su nivel de conocimientos técnicos. Por ello el diseño de las interfaces está centrado en el usuario, de manera que este pueda realizar todas las operaciones de manera sencilla. Por ello las interfaces son claras e intuitivas, con botones, menús y etiquetas que guían al usuario de forma eficiente a través del proceso.
RNF2: Navegabilidad	El sistema debe permitir al usuario navegar de manera clara, directa y eficiente a través de todas las funcionalidades disponibles. Los requisitos de navegabilidad incluyen: • Accesibilidad directa: Las principales funcionalidades del sistema deben ser fácilmente accesibles desde la interfaz principal, evitando la necesidad de realizar múltiples pasos o búsquedas complicadas. • Estructura de menús lógica: La disposición de los menús y las opciones debe seguir un flujo natural que se alinee con las expectativas del usuario, asegurando que este pueda acceder a cualquier funcionalidad con el menor número de clics posibles.
RNF3: Interactividad	El sistema debe proporcionar una retroalimentación visual clara sobre las acciones realizadas por el usuario, haciendo que el sistema sea más fácil de usar y comprendido de manera eficiente. Esto incluye: • Indicadores visuales: El sistema debe mostrar visualmente el estado de las operaciones en tiempo real, como la carga de datos, el procesamiento de *logs* y la generación de informes. • Confirmaciones claras: El sistema debe ofrecer mensajes de confirmación de modo que el usuario asegure que desea realizar la acción (inserciones en base de datos, asignación de una descripción)

	• Errores y advertencias: En caso de que se produzca un problema, el sistema proporciona mensajes de error claros y fáciles de entender, que explican la causa del problema, para que este sea solventado.
RNF4: Rendimiento	El sistema debe ser capaz de procesar grandes volúmenes de datos de manera eficiente, sin afectar la experiencia del usuario. Es decir, debe proporcionar un tiempo de respuesta rápido y un procesamiento eficiente.
RNF5: Escalabilidad	El sistema debe poder manejar incrementos en la cantidad de datos sin que su rendimiento se vea afectado significativamente.
RNF6: Seguridad	El sistema debe garantizar la seguridad de los datos que maneja, especialmente si estos contienen información sensible o confidencial. Las medidas de seguridad necesarias incluyen protección de datos (Solo quien tenga acceso a la VPN de Viesgo podrá acceder a los *logs*) y autenticación y autorización el acceso a tablas de la base de datos tendrá permisos limitados.
RNF7: Compatibilidad	El sistema debe ser compatible con diversos entornos y plataformas. Por lo que el sistema debe ser capaz de ejecutarse en los principales sistemas operativos como Windows, macOS y Linux, sin que su funcionamiento se vea afectado por el sistema operativo utilizado.
RNF8: Mantenimiento	El sistema debe ser diseñado de manera que sea fácil de mantener y actualizar, con código modular y bien documentado.

Tabla 3. 2. 1 – Requisitos no funcionales.

3.3. Casos de uso

Con el fin de definir y comprender mejor las funcionalidades del sistema propuesto, se han modelado una serie de casos de uso que describen las interacciones entre los actores y el sistema. Estos casos de uso permiten representar, de forma estructurada, los requisitos funcionales, facilitando tanto el diseño como el desarrollo posterior [7].

Cada caso de uso describe un escenario específico en el que uno o varios actores interactúan con el sistema para alcanzar un objetivo concreto. La elaboración de estos casos se ha basado en los requisitos definidos en los apartados anteriores y permite identificar claramente las funcionalidades clave que el sistema debe ofrecer.

La siguiente figura corresponde al diagrama UML donde se representan los casos de uso y actores:

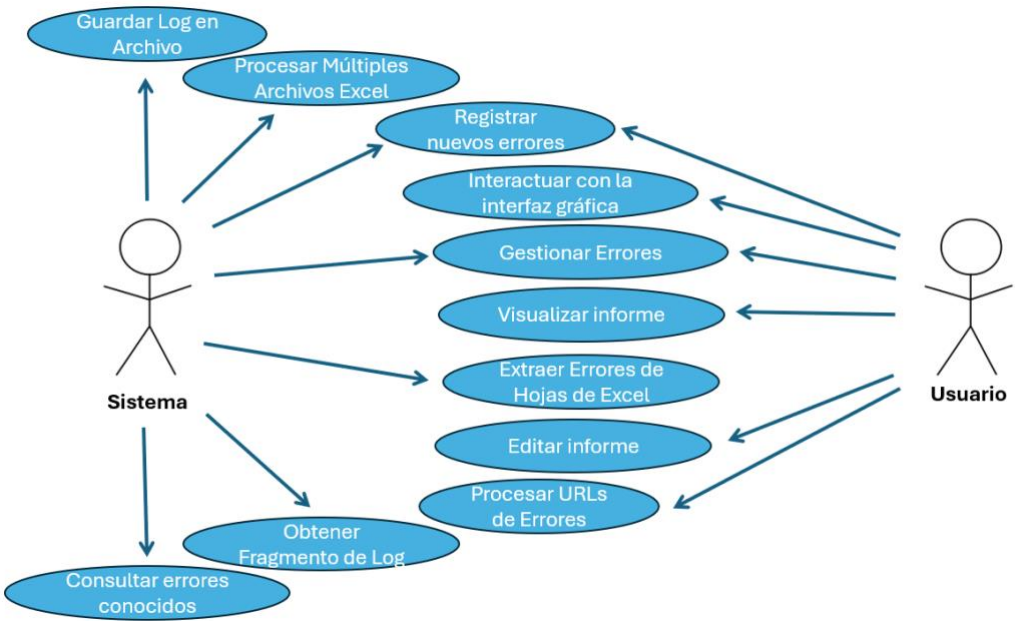


Figura 3. 3. 1 – Diagrama UML sobre los casos de uso

A continuación, se especifican los casos de uso representados en el diagrama UML:

CASO DE USO N.º 1	***Guardar Log en Archivo***
Descripción	El sistema permite guardar el contenido de un *log* en un archivo de texto local.
Actores	Sistema
Precondiciones	Se debe proporcionar una URL válida que apunte a un *log*.
Flujo de eventos	1. El sistema llama al método logEnTxt(urlString). 2. El sistema intenta abrir la URL proporcionada. 3. Si la URL es válida, el sistema lee el contenido línea por línea. 4. Cada línea se escribe en el archivo logParaFragmentos.txt. 5. El sistema confirma que la información se ha guardado correctamente.
Resultado	El archivo logParaFragmentos.txt contendrá el contenido del *log* descargado.

Tabla 3. 3. 1 – Caso de uso para guardar el log en un archivo

CASO DE USO N.º 2	*Obtener Fragmento de Log*
Descripción	El sistema permite extraer un fragmento de *log* que contenga información sobre un error específico.
Actores	Sistema
Precondiciones	El archivo logParaFragmentos.txt debe existir y contener datos.
Flujo de eventos	1. El sistema llama al método fragmentoLog(mensaje). 2. El sistema abre y lee el archivo logParaFragmentos.txt. 3. Se busca la primera aparición del mensaje de error. 4. Si se encuentra el mensaje, se guardan las líneas subsecuentes hasta encontrar un marcador como [INFO] o [DEBUG]. 5. El sistema devuelve el fragmento encontrado.
Resultado	Se devuelve el fragmento de log correspondiente al error, o un mensaje indicando que no se encontró información.

Tabla 3. 3. 2 – Caso de uso para obtener un fragmento concreto del log

CASO DE USO N.º 3	***Procesar Múltiples Archivos Excel***
Descripción	Permite procesar una lista de archivos de Excel y extraer errores únicos de cada uno.
Actores	Sistema
Precondiciones	El usuario debe tener acceso a uno o más archivos de Excel. Los archivos de Excel deben estar en un formato compatible (XLSX).
Flujo de eventos	1. El usuario invoca el método procesandoExcel pasando una lista de rutas de archivos de Excel. 2. El sistema verifica cada archivo en la lista (Si el archivo no existe, se registra un mensaje de error en la consola.) 3. Para cada archivo existente, se llama al método procesarArchivoExcel. 4. Dentro de procesarArchivoExcel, el sistema: - Abre el archivo y lee cada hoja. - Extrae los errores únicos encontrados en las celdas específicas. - Escribe los errores en el archivo Errores.txt. 5. El sistema finaliza el procesamiento y cierra el archivo de salida.
Resultado	Se crea o se actualiza el archivo Errores.txt con errores únicos de todos los archivos procesados.

Tabla 3. 3. 3 – Caso de uso para procesar varios archivos excel

CASO DE USO N.º 4	*Gestionar Errores*
Descripción	Controla todo el proceso de gestión de errores, desde la lectura del archivo de errores hasta la escritura de un informe procesado.
Actores	Usuario (Interactúa con la interfaz de la aplicación.) Sistema (La aplicación que ejecuta la clase ProcesarErrores.)
Precondiciones	- El archivo de errores debe estar disponible en la ruta especificada. - La base de datos debe ser accesible y contener los mensajes de error necesarios.
Flujo de eventos	1. El sistema inicia el método GestionErrores. 2. Se leen las líneas del archivo de errores desde la ruta especificada. 3. Se obtienen los mensajes de error de la base de datos. 4. El sistema procesa cada línea del archivo: - Si la línea indica un error conocido, se determina el tipo de error. - Dependiendo del tipo de error, se realizan diferentes acciones: Tipo 0: Añadir descripción debajo de la línea. Tipo 1: Eliminar líneas duplicadas y agregar descripción. Tipo 2: Mostrar una interfaz para interacción del usuario. Tipo 3: Mostrar una interfaz para personalización de descripciones. - Si la línea no corresponde a un error conocido, se gestiona como un nuevo error. 5. Se escriben las líneas procesadas en un archivo de informe. 6. Se muestra una interfaz final al usuario.
Resultado	El archivo de informe es creado o actualizado con las líneas procesadas.

Tabla 3. 3. 4 – Caso de uso para gestionar los errores detectados

CASO DE USO N.º 5	Procesar URLs de Errores
Descripción	El sistema permite procesar URLs de *logs* para extraer y almacenar los errores encontrados en un archivo de texto.
Actores	Usuario (interactúa a través de la interfaz gráfica)
Precondiciones	La URL proporcionada debe estar en el formato correcto. La conexión a la red debe estar activa. Debe estar establecida la conexión a la VPN
Flujo de eventos	1. El usuario proporciona una URL y elige un modo de procesamiento (modo 0 o modo 1). 2. El sistema valida la URL proporcionada. (Si la URL es inválida, se muestra un mensaje de error y se termina el proceso) 3. El sistema crea una lista de URLs predeterminadas. 4. El sistema verifica si la fecha en la URL es del día anterior y si la hora actual es menor o igual a las 21:00. (Si es así, se elimina la URL de generación de cartas de la lista.) 5. Se sobrescriben los archivos de errores y log para almacenar nuevos datos. 6. Dependiendo del modo seleccionado: - Modo 1: Se procesan todas las URLs de la lista. - Modo 0: Se procesa solo la URL proporcionada por el usuario. 7. Se extraen los errores de los *logs* y se almacenan en el archivo de errores. 8. El sistema informa al usuario que el procesamiento se ha completado.
Resultado	Los errores encontrados en los *logs* se almacenan en Errores.txt. Un log con la información de la URL se guarda en log.txt.

Tabla 3. 3. 5 – Caso de uso para obtener el listado de errores a través de una url

CASO DE USO N.º 6	Extraer Errores de Hojas de Excel
Descripción	Extrae errores de cada hoja de un archivo de Excel y los almacena de manera única.
Actores	Sistema
Precondiciones	Se ha abierto un archivo de Excel válido.
Flujo de eventos	1. El sistema itera sobre cada hoja del libro de Excel. 2. Para cada hoja: - Lee el nombre de la hoja y lo escribe en el archivo de salida. - Itera sobre las filas de la hoja. - Dependiendo del número de columnas, extrae el texto de error de la última columna. - Si se encuentra un error único, se agrega al conjunto de errores y se escribe en el archivo de salida. 3. Se imprime un mensaje en la consola indicando que la extracción se ha completado para la hoja.
Resultado	Los errores únicos extraídos se escriben en el archivo Errores.txt bajo el nombre de la hoja correspondiente.

Tabla 3. 3. 6 – Caso de uso para extraer el listado de errores de archivo Excel

CASO DE USO N.º 7	Interactuar con la interfaz gráfica
Descripción	Todas las operaciones se realizan a través de una interfaz desarrollada con *JavaFX*, permitiendo al usuario realizar todas las tareas de forma visual.
Actores	Usuario
Precondiciones	Se debe detectar un error que requiere interacción.
Flujo de eventos	1. El sistema determina el tipo de error. 2. Se invocan las interfaces correspondientes (InterfazTipo2, InterfazTipo3, InterfazTipoNuevoError). 3. El usuario interactúa con la interfaz y proporciona la información necesaria. 4. La descripción del error se actualiza según la interacción del usuario.
Resultado	Las descripciones de los errores son actualizadas en el informe y en caso de InterfazTipoNuevoError se almacena la nueva información en base de datos.

Tabla 3. 3. 7 – Caso de uso para la interacción con las interfaces gráficas.

CASO DE USO N.º 8	Consultar errores conocidos
Descripción	Durante el análisis, el sistema consulta la base de datos para comprobar si los errores ya están registrados, y si es así, les asigna su descripción predefinida.
Actores	Sistema
Precondiciones	La conexión a la base de datos debe estar establecida.
Flujo de eventos	1. Llamar al método obtenerListadoErrores. 2. Ejecutar la consulta para recuperar todos los errores. 3. Almacenar los resultados en un mapa y devolverlo.
Resultado	Se devuelve un mapa con los mensajes de error y sus descripciones.

Tabla 3. 3. 8 – Caso de uso para consultar los errores almacenados en base de datos.

CASO DE USO N.º 9	Registrar nuevos errores
Descripción	Si el sistema detecta un error no registrado, ofrece al usuario la opción de completarlo con la descripción y tipo de error, y lo almacena en la base de datos.
Actores	Usuario Sistema
Precondiciones	La conexión a la base de datos debe estar establecida.
Flujo de eventos	1. Detectar un error desconocido 2. El usuario introduce la información del nuevo error (mensaje, descripcion, y tipo.) 3. Llamar al método nuevaEntradaBBDD con los parámetros mensaje, descripcion, y tipo. 4. Ejecutar la inserción SQL.
Resultado	Se inserta una nueva entrada en la base de datos.

Tabla 3. 3. 9 – Caso de uso para añadir un nuevo error en base de datos.

CASO DE USO N.º 10	Editar informe
Descripción	El usuario puede modificar directamente el contenido del informe desde la aplicación.
Actores	Usuario
Precondiciones	Se debe haber cargado InterfazFinal con el documento Informe.txt
Flujo de eventos	1. El usuario accede a la pantalla de visualización del informe. 2. El usuario selecciona el botón "Editar". • El sistema habilita el área de texto para permitir la edición. • El botón "Guardar cambios" se hace visible. • El botón "Cerrar modo edición" se hace visible. 3. El usuario modifica el contenido del informe en el área de texto. 4. El sistema detecta cambios en el área de texto. • El botón "Guardar cambios" cambia de color a rojo para indicar que hay cambios no guardados. 5. El usuario selecciona el botón "Guardar cambios". • El sistema guarda los cambios realizados en el archivo del informe. • Se muestra un mensaje confirmando que el informe ha sido guardado correctamente. 6. El usuario puede elegir cerrar el modo de edición seleccionando el botón "Cerrar modo edición". • El sistema deshabilita la edición en el área de texto. • Los botones de guardar y cerrar edición se ocultan.
Resultado	Si el usuario guardó los cambios, el informe se actualiza con el nuevo contenido. Si el usuario cerró el modo de edición sin guardar, el informe permanece sin cambios.

Tabla 3. 3. 10 – Caso de uso para editar el informe.

CASO DE USO N.º 11	Visualizar informe
Descripción	El usuario puede consultar el informe directamente desde la interfaz gráfica de la aplicación.
Actores	Usuario
Precondiciones	Se debe haber generado el informe en “Informe.txt”
Flujo de eventos	1. Finalizar el proceso de creación de informe 2. Se invoca la interfaz “InterfazFinal” donde se carga el contenido de “Informe.txt” 3. El usuario puede elegir entre las siguientes opciones: • Guardar: Guardar el contenido actual del informe. • Editar: Activar el modo de edición para modificar el contenido. • Cerrar: Cerrar la aplicación o regresar al menú principal.
Resultado	Se previsualiza el informe generado en el proceso.

Tabla 3. 3. 11 – Caso de uso para visualizar el informe generado en el proceso.

4. DISEÑO Y DESARROLLO DEL SISTEMA

El trabajo está diseñado siguiendo el modelo de arquitectura software en tres capas, el cual se basa en dividir la aplicación en tres niveles lógicos separados: presentación, negocio y acceso a datos. Cada uno de estos niveles asume unas funcionalidades específicas que, al ser combinadas, completan el funcionamiento del sistema. En concreto, la capa de presentación se encarga de gestionar la interfaz de usuario y la interacción con este; la capa de negocio implementa los métodos y reglas que definen el comportamiento del sistema según los requerimientos funcionales; y la capa de acceso a datos se responsabiliza de la comunicación con la base de datos, incluyendo operaciones de lectura y escritura de información. Esta estructuración permite una mejor organización del código, facilitando tanto el mantenimiento como la escalabilidad del software. [8]

Como resultado de esta arquitectura, en los siguientes subapartados se observará que las clases pertenecientes a una misma capa no presentan muchas relaciones entre sí. Esto se debe a que el sistema ha sido desarrollado de forma que las interacciones entre componentes se producen principalmente entre capas distintas, y no dentro de una misma capa. En particular, las clases de la capa de negocio no interactúan directamente entre ellas, sino que lo hacen a través de la capa de presentación, que actúa como intermediaria en la coordinación de las operaciones del sistema. Esta separación favorece un diseño modular, en el que cada funcionalidad está claramente definida dentro de su contexto correspondiente.

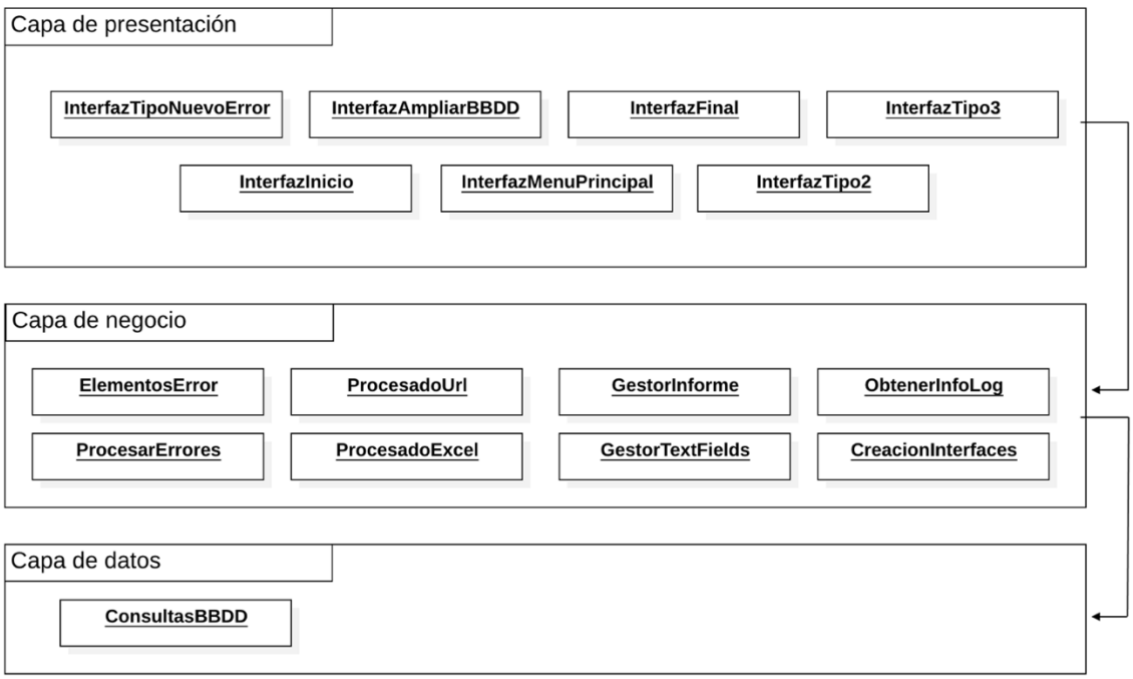


Figura 4.1 – Arquitectura en tres capas del sistema

La figura 4.1 representa el modelo de arquitectura en tres capas del sistema desarrollado, donde se pueden identificar las clases que conforman el proyecto, así como su distribución y nexo entre la capa de presentación, capa de negocio y capa de datos.

4.1. Capa de presentación

La capa de presentación es la encargada de mostrar datos al usuario y facilitar la interacción entre la persona y el sistema. En otras palabras, esta capa se corresponde a las interfaces de usuario.

El diseño de dichas interfaces se basó en tres objetivos claves: debían ser intuitivas permitiendo que cualquier profesional que haga uso del sistema no necesite conocimientos previos; debían mantener un estilo visual coherente y profesional; y, por último, debía haber coherencia entre el funcionamiento del sistema y la información que presentan las interfaces.

Para mejorar la intuición al utilizar el sistema, se añadió una barra de progreso en la parte inferior de las interfaces, que muestra los pasos a seguir y resalta en qué punto del proceso se encuentra el usuario. Además, se utilizan ventanas emergentes que notifican posibles fallos durante el procedimiento como problemas de conexión con la VPN, formatos incorrectos, etc.), así como ventanas de confirmación para evitar que el usuario realice acciones por error como añadir nueva información a la base de datos o, por el contrario, saltarse un paso por accidente.

El uso de un estilo unificado en todas las pantallas busca transmitir seriedad a través de una estética cuidada, es por ello, que todas las interfaces cuentan con un encabezado y un pie fijos. Es por el mismo motivo que se emplea una paleta de colores y tipografía coherentes en todo el sistema.

Para garantizar la coherencia entre el funcionamiento del sistema y la información proporcionada por las interfaces, el primer paso fue pensar que interacciones eran realmente necesarias para el correcto desarrollo del sistema. Por ello desarrollaron las siguientes siete interfaces que estructuran su funcionamiento.

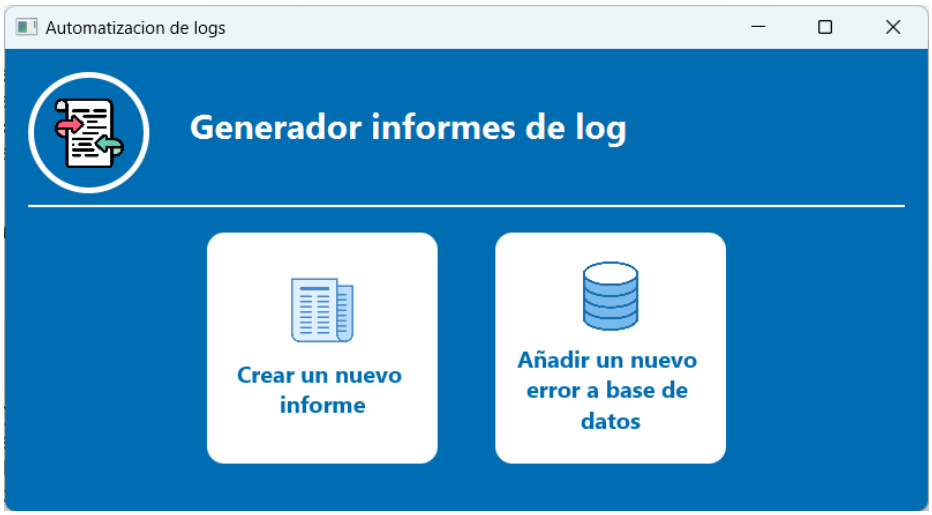


Figura 4.1.1 – Interfaz menú principal 1

En la figura 4.1.1 se muestra la primera pantalla que visualiza el usuario, permite seleccionar la acción que se quiere realizar.

- Crear un nuevo informe
- Añadir un nuevo error a base de datos

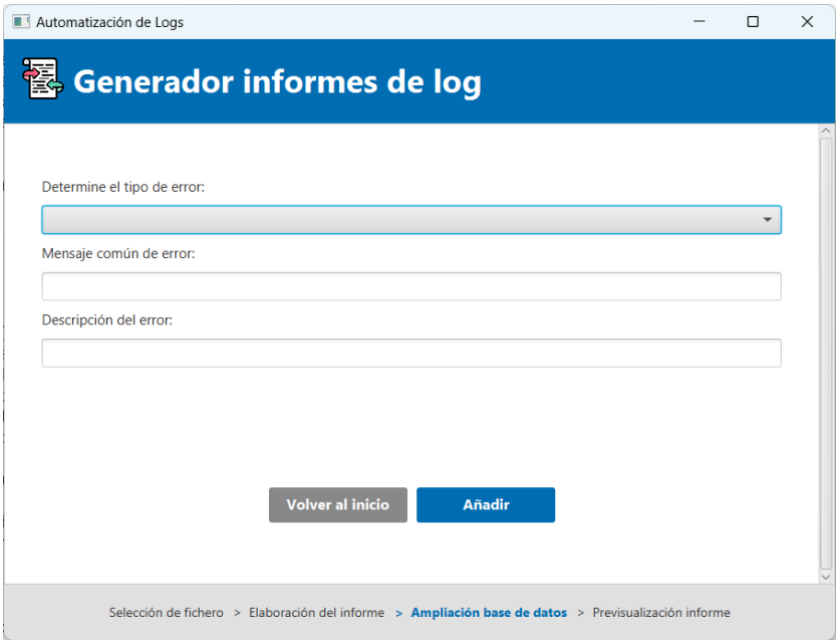


Figura 4.1.2 - Interfaz ampliar base de datos

La figura 4.1.2 muestra la interfaz que se visualiza en caso de que el usuario seleccione "añadir un nuevo error a base a datos".

En ella el usuario debe categorizar el error, así como determinar el mensaje y la descripción correspondiente al error a agregar. Deberá pulsar añadir para que la acción se complete.

Además, en caso de que el nuevo error sea de tipo 2 “Multirrespuesta” la interfaz mostrará nuevos campos de texto para tantas descripciones como se quieran añadir con un mínimo de dos y hasta un máximo de cinco. Indistintamente de que el usuario opte por “volver al inicio” o “añadir” aparecerá una ventana para confirmar la acción.

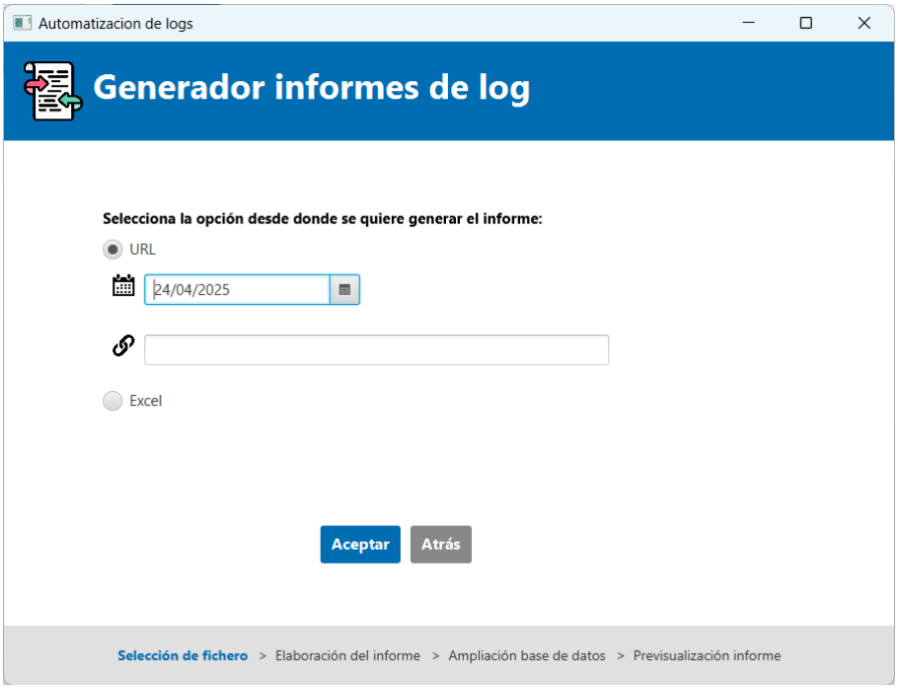


Figura 4.1.3 – Interfaz de inicio

En la figura 4.1.3 se puede ver la pantalla que se mostrará en caso de que el usuario opte por crear un nuevo informe. Permite seleccionar el método desde el que se quiere obtener los errores de un día.

- Por URL: Seleccionando una fecha / Especificando una url.
- Mediante Excel [anexo 1].

En caso de que el usuario no introduzca ningún tipo de información, el Excel no cuente con la extensión correcta o la url no cumpla con el formato se notifica en una ventana emergente reportando dicha información [Figura 4.1.4]. Lo mismo si la VPN no está conectada.

Figura 4.1.4 – Ventana notificación de error.

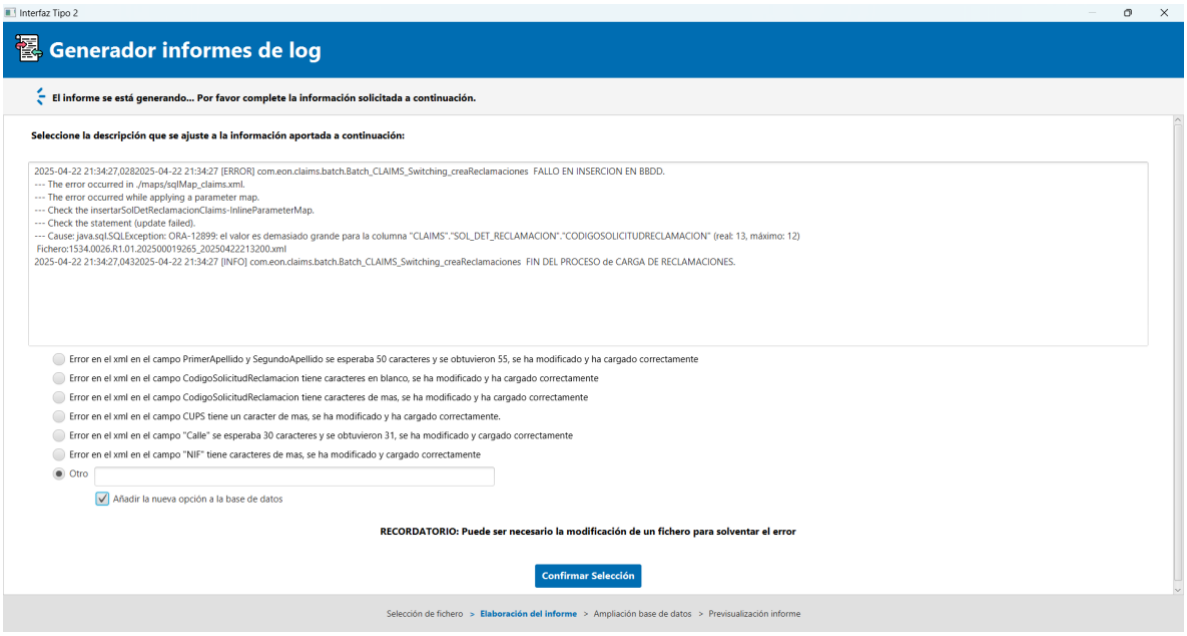


Figura 4.1.5 – Interfaz Tipo 2

En la figura 4.1.5 se puede ver la interfaz que se mostrará si, durante la generación del informe, se detecta un error del tipo 2, "multirrespuesta".

El sistema proporcionará al usuario más detalles sobre el error, incluyendo múltiples descripciones para un mismo este. Con la ayuda del fragmento de log reportado, el profesional podrá determinar la opción más adecuada. En caso de que ninguna descripción sea aplicable, se podrá agregar una nueva respuesta, la cual se almacenará en la base de dato si así se determina.

Cuando se selecciona la opción “otro” solo se añadirá a base de dato la nueva descripción en caso de que se seleccione esta opción. Además, no se permitirá continuar si no se selecciona ninguna opción o se selecciona “otro” y no se introduce la descripción.

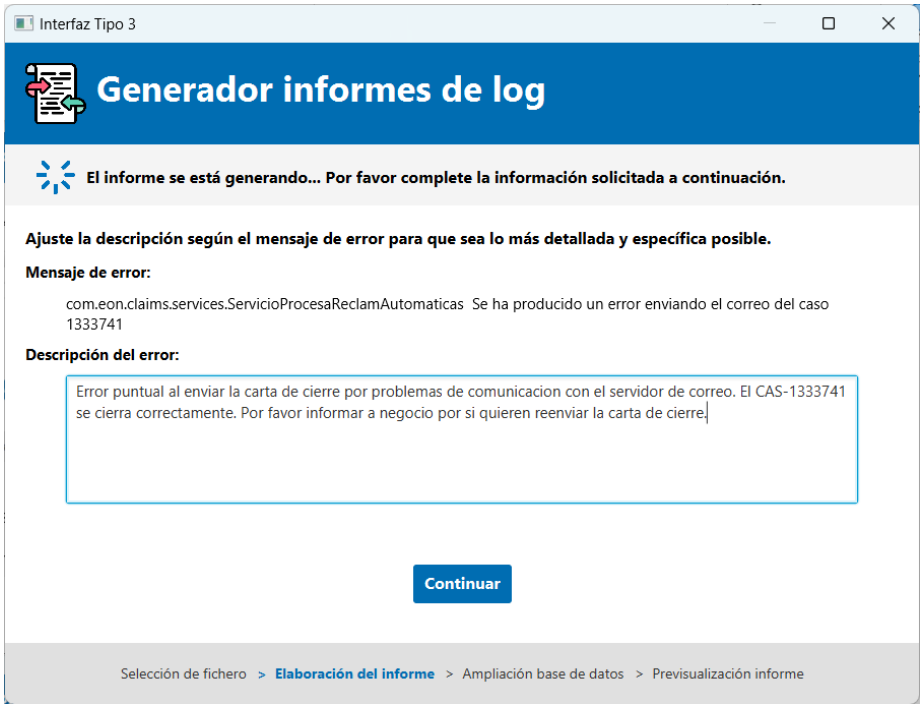


Figura 4.1.6 - Interfaz Error Tipo 3

La figura 4.1.6 muestra la interfaz en caso de que se detecte un error de tipo 3 ("Descripciones personalizadas") durante la generación del informe. El sistema proporcionará al usuario el mensaje de error, así como su respectiva descripción la cual el usuario deberá modificar para personalizarlo en base al error concreto que se ha producido. Para poder continuar será obligatorio modificar la descripción.

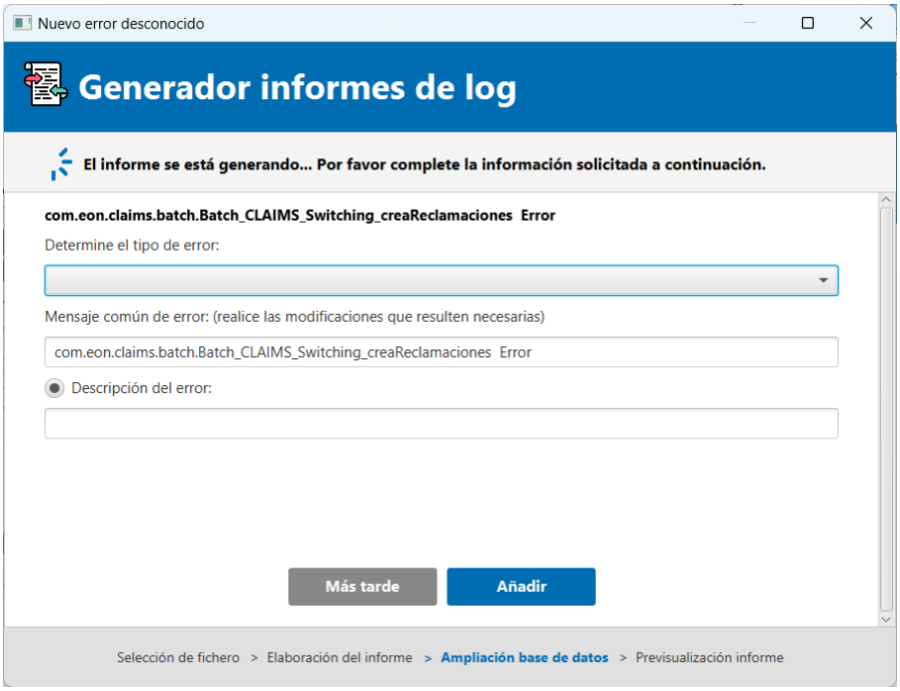


Figura 4.1.7 – Interfaz tipo nuevo error

En la figura 4.1.7 muestra la interfaz en caso de que se haya identificado un error durante el análisis de logs, el cual no se encuentra almacenado en la base de datos.

A simple vista, la estética de este componente guarda similitudes con la interfaz "InterfazAmpliarBBDD" a diferencia de que en la parte superior de la interfaz se presenta un mensaje de error, solicitando al usuario que ingrese la información correspondiente al mismo.

En caso de que el nuevo error sea de tipo 2 la interfaz mostrará nuevos campos de texto para tantas descripciones como se quieran añadir con un mínimo de dos y hasta un máximo de cinco.

Indistintamente de que el usuario opte por “Más tarde” (no se añadirá el nuevo error) o “Añadir” aparecerá una ventana para confirmar la acción.

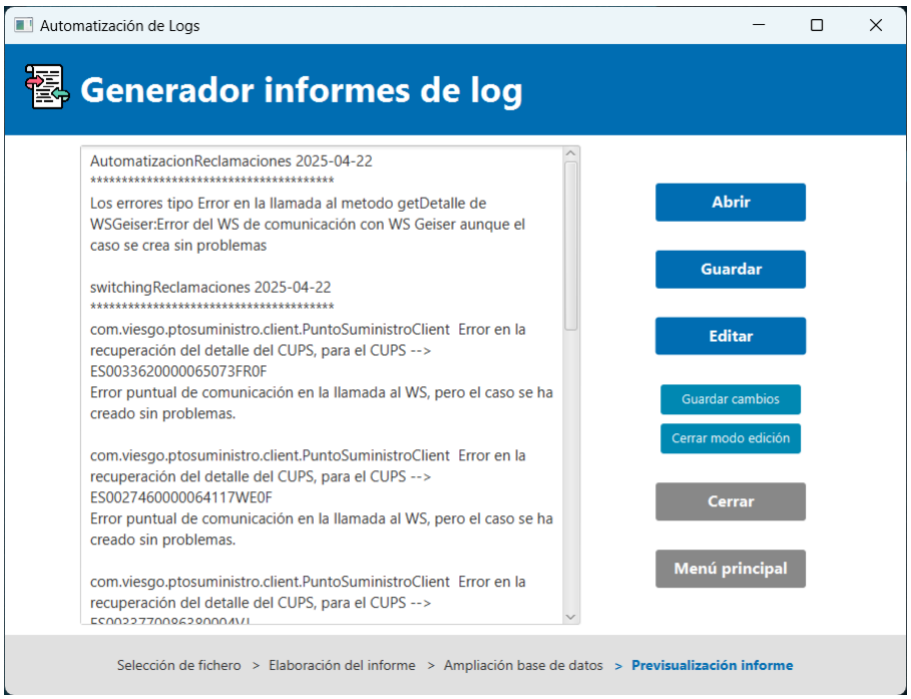


Figura 4.1.8 – Interfaz final

La figura 4.1.8 muestra la pantalla donde se visualiza el informe generado tras completar todo el proceso. Permite acciones como abrir, guardar o editar el archivo. Además, si fuese necesario permite volver al inicio del sistema.

Cuando se edita el informe es indispensable pulsar el botón “Guardar cambios” para no perder las modificaciones.

Con el propósito de obtener una visión general más clara y estructurada del sistema, se ha elaborado un diagrama de clases [Figura 4.1.9]. En este se representan las clases que lo componen, sus respectivos métodos, así como las relaciones que existen entre ellas. Este tipo de diagramas facilita la comprensión del diseño del sistema al ofrecer una representación visual precisa y esquematizada de su estructura interna.

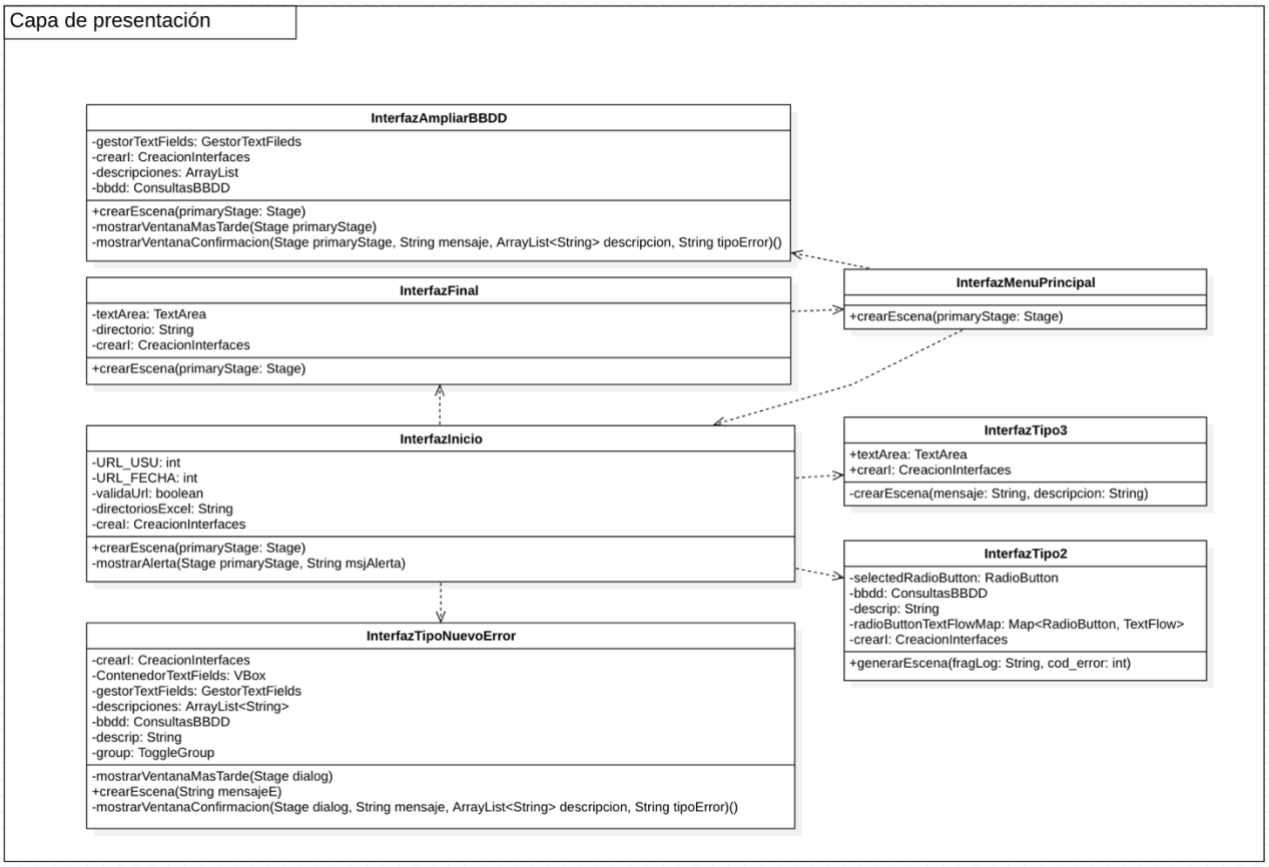


Figura 4. 1. 9 – Diagrama de clases capa de presentación

A continuación, se detalla lo representado en el diagrama de clases de las interfaces, justificando su estructura:

- **InterfazTipo2, InterfazTipo3 e InterfazTipoNuevoError:** Estas clases están asociadas a los distintos tipos de errores que pueden producirse durante la ejecución del sistema. Cada una de ellas implementa la interfaz gráfica que se mostrará al usuario en función del tipo de error detectado. En el caso de errores no contemplados previamente, se recurre a la clase InterfazTipoNuevoError. Además de definir la apariencia de la pantalla, estas clases gestionan el comportamiento del sistema en función de las interacciones del usuario con la interfaz. Para construir de forma coherente y reutilizable los elementos visuales de estas pantallas, hacen uso de la clase CreacionInterfaces, lo que contribuye a mantener una estructura modular y facilita la evolución del sistema.

- **InterfazFinal:** Esta clase se encarga de implementar la interfaz gráfica final del sistema, en la cual se presenta al usuario el informe generado durante el proceso de tratamiento de errores. Además de la visualización del informe, InterfazFinal incorpora funcionalidades clave como abrir, guardar y editar el documento generado. También proporciona al usuario opciones para cerrar la aplicación o regresar al menú principal (InterfazInicio) con el fin de iniciar la creación de un nuevo informe. Esta clase representa el punto final del flujo de trabajo.
- **InterfazMenuPrincipal:** Esta clase tiene como función principal permitir al usuario seleccionar la acción que desea realizar: “generar un nuevo informe” o “añadir un nuevo error a la base de datos”. Según la opción escogida, el sistema dirigirá el flujo de ejecución hacia el proceso correspondiente, adaptando su comportamiento en función de la elección del usuario.
- **InterfazAmpliarBBDD:** Esta clase permite al usuario añadir un nuevo error a la base de datos en cualquier momento, de forma independiente al proceso de generación de informes. A diferencia de InterfazNuevoError, que se ejecuta exclusivamente cuando durante la creación de un informe se detecta un error desconocido (es decir, un error no registrado previamente en la base de datos), InterfazAmpliarBBDD ofrece una vía directa y manual para ampliar la base de datos de errores en cualquier momento.
- **InterfazInicio:** se encarga de generar y gestionar la interfaz inicial de la aplicación, permitiendo al usuario seleccionar la fuente de los datos (URL, Excel o fecha) para el análisis de errores. Integra componentes gráficos como botones, campos de texto, calendarios y notificaciones, facilitando la interacción y validación de las entradas. Además, coordina el inicio de los procesos de lectura y análisis según la opción elegida y gestiona los mensajes de alerta en caso de errores o entradas no válidas.

4.2. Capa de negocio

La capa de negocio tiene como principal función el procesamiento de los datos recibidos desde la capa de presentación, así como la comunicación con la capa de acceso a datos. En ella se gestionan los *logs* del sistema, se genera un listado de errores detectados para finalmente, elaborar el informe correspondiente.

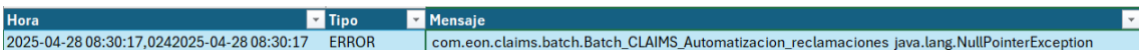
La generación del informe final requiere la ejecución previa de varias etapas. En primer lugar, es necesario obtener el listado de errores. A continuación, cada error debe ser analizado individualmente para definir su tipología y asignarle una descripción adecuada. Finalmente, con toda esta información procesada, se genera el informe definitivo.

El listado de errores puede obtenerse mediante dos métodos diferentes: directamente a partir del procesamiento de los *logs*, o bien a partir de un o más archivos Excel proporcionados por el cliente, quien previamente ha extraído los mensajes de error de los *logs*. En este segundo caso, el sistema emplea la clase *ProcesadoExcel.java*, la cual contiene diversas funcionalidades orientadas al tratamiento de este tipo de archivos.

En una primera etapa, *ProcesadoExcel.java* genera un archivo denominado "*Errores.txt"*, que almacenará todos los errores detectados. Posteriormente, se realiza una iteración sobre todos los archivos Excel proporcionados (pueden ser uno o varios), comprobando tanto su existencia como que poseen la extensión adecuada. Una vez verificados estos aspectos, se procede al procesamiento de cada archivo Excel y de cada una de sus hojas de cálculo. Por cada hoja procesada, se añade un título identificativo en *Errores.txt*, lo que permite categorizar adecuadamente los errores ("switchingReclamaciones", "AutomatizacionReclamaciones" o "GeneraciónCartas"). Además, se incorporan los errores registrados en la hoja, asegurando que no se introduzcan duplicados.

Los archivos Excel suministrados por el cliente pueden presentar dos formatos distintos, los cuales han sido contemplados en el diseño de *ProcesadoExcel*:

- Formato 1: Tres columnas diferenciadas que contienen información relativa a cada error (fecha y hora, tipo, y mensaje) como se muestra en la figura 4.2.1. En este caso, el sistema únicamente extrae el contenido de la tercera columna, ya que es la que almacena el mensaje de error propiamente dicho.



Hora	Tipo	Mensaje
2025-04-28 08:30:17,0242025-04-28 08:30:17	ERROR	com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException

Figura 4.2.1 – Ejemplo de error en Formato 1 de excel

- Formato 2: Una única columna donde toda la información del error se encuentra combinada en una sola cadena [Figura 4.2.2]. Este formato requiere que el sistema filtre y extraiga únicamente el mensaje de error, desechando cualquier información adicional que no sea relevante para el informe. Habiendo analizado la estructuración de este formato, se concluyó que la parte correspondiente empieza a partir de "[ERROR]".

Figura 4.2.2 – Ejemplo de error en Formato 2 de excel

En cuanto al segundo método de obtención de errores, este se gestiona a través de la clase *ProcesadoUrl.java*, la cual permite procesar los logs directamente a partir de una URL. La dirección del log se obtiene desde el *frontend*, ya sea seleccionándola mediante una fecha o introduciendo directamente la URL completa. Cada URL es sometida a una verificación de su formato antes de proceder con a guardarlo en un fichero ".txt".

Una vez validada la URL, el log se descarga y guarda en un archivo denominado *log.txt*. A continuación, se analizan sus líneas en busca de aquellas que contengan la etiqueta "[ERROR]". Cuando se detecta una línea con esta característica, se buscan patrones que permitan identificar un error. Si se encuentra una coincidencia, el mensaje se añade al archivo *Errores.txt*. Al igual que con el procesamiento desde Excel, se evita la duplicación de mensajes, garantizando que cada error se registre una sola vez.

Una vez obtenidos y almacenados en "*Errores.txt*" el siguiente paso es el procesado de cada error. A través de *ProcesarErrores.java*, esta es la clase principal encargada de leer línea por línea el fichero "*Errores.txt*" y probando si alguno de los errores almacenados en la base de datos coincide con la línea que almacena el error producido. En caso negativo se le permite al usuario añadir el nuevo error a base de datos de manera que la base de datos se enriquezca y se complete con cada nuevo error. y en caso de acierto, es decir, el error coincide lo que se hace es comprobar la tipología del error (tipo 0, tipo 1, tipo 2, tipo 3) y actuar en función de este valor

Se debe tener en cuenta que la categorización se ha desarrollado en base a la revisión de los *logs* en Gredos, pero no supondría grandes modificaciones para adaptarlo a otros sistemas. Se han clasificado los errores en las siguientes categorías:

- **TIPO = 0** → El error ya está almacenado en la base de datos y tiene asociada una descripción única para ese mensaje de error.

  Mensaje com.eon.claims.batch.Batch_CLAIMS_Switching_creaReclamaciones Fichero: no descomprimido, no es XML: 1403-0027_R1.zip. No será procesado.

  Descripción Error puntual en la descompresion al no ser un XML

- **TIPO = 1** → El error ya está almacenado en la base de datos y corresponde a errores de comunicación con un web servicie de terceros (WS Geiser), un error común. En este caso, no solo se debe añadir la descripción, sino también eliminar todas las líneas relacionadas con este error para evitar que se repita varias veces en el informe.

  Mensaje com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException

  Descripción Los errores tipo Error en la llamada al metodo getDetalle de WSGeiser:Error del WS de comunicación con WS Geiser aunque el caso se crea sin problemas
- **TIPO = 2** → El error ya está almacenado en la base de datos, pero tiene múltiples posibles descripciones asociadas a este error. Para la asignación de una de las opciones el usuario contará con una lista de todas las descripciones posibles, además se le proporciona la parte del *log* correspondiente al mensaje facilitando un mayor contexto de la situación.
- **TIPO = 3** → El error ya está almacenado en la base de datos junto a una descripción única, pero esta requiere de una modificación mínima personalizando la descripción con información más específica del error.

  Mensaje com.eon.claims.batch.Batch_CLAIMS_GeneracionCartas No se ha podido generar la carta asociada al caso sin CODIGO POSTAL CAS-13455997

  Descripción Error puntual al generar la carta para el CAS-XXXXX.

  La modificación que debería hacer sería sustituir “CAS-XXXXX” por el número de caso sobre el que se ha producido el error, es decir, CAS-13455997.

Cuando el error es categorizado se añade la descripción a continuación del mensaje de error en el fichero “*Informe.txt*”.

Como para el tipo 2 se proporciona un fragmento del *log* con el fin de ampliar la información correspondiente al error, se ha implementado la clase *ObtenerInfoLog.java* cuya funcionalidad se basa en proporcionar toda la información relativa al mensaje de error indicado, para ello se requerirá obtener el *log* original, buscar el error y obtener toda la información desde la línea que contiene el mensaje hasta encontrar la cadena de caracteres “[INFO]” o “[DEBUG]”.

Gran parte del trabajo de la capa de negocio es la lectura y escritura de ficheros con los que trabaja el sistema. En este caso estos ficheros son:

- Errores.txt → almacena el listado de los errores obtenidos sin duplicados.
- Log.txt → guarda una copia del log desde donde se obtendrá el listado.

- logParaFragmentos.txt → encargado de almacenar el log del que se extrae el fragmento requerido para el tipo 2.
- Informe.txt → archivo con el informe final obtenido del análisis de los errores.

Con el objetivo de garantizar un correcto comportamiento del sistema y minimizar posibles fallos durante su ejecución, se han implementado diversos mecanismos de control y notificación de errores. Estos mecanismos permiten identificar, informar y explicar de forma clara las incidencias que puedan surgir durante el desarrollo del proceso, mejorando así la experiencia del usuario y la fiabilidad del sistema. Estas validaciones son:

- Validación de la fecha de generación del informe: El sistema exige que la fecha introducida por el usuario sea anterior a la fecha actual y no exceda los treinta días de antigüedad, ya que únicamente se almacenan los *logs* correspondientes al último mes. En caso de que la fecha no cumpla con estos criterios, se muestra una ventana emergente notificando al usuario del error.
- Conexión a la red privada (VPN): Para acceder a los archivos de *log*, es imprescindible que el usuario esté conectado a la red privada virtual (VPN) de Viesgo. Si el sistema detecta que no está establecida la conexión, se lanza una ventana emergente indicando el fallo de conexión y recordando que la VPN debe estar habilitada para continuar con el proceso.
- Verificación del formato de la URL: En aquellos casos en los que el usuario introduce manualmente una URL para acceder a un *log* determinado, el sistema comprueba que dicha URL cumpla con el formato requerido. En caso contrario, se genera una notificación informando del error y solicitando una URL válida.
- Validación de campos al añadir nuevos errores: Cuando el usuario desea registrar manualmente un nuevo error, es obligatorio que todos los campos del formulario estén correctamente cumplimentados. Si falta alguno de los datos requeridos, el sistema bloquea la continuación del proceso y muestra un mensaje explicativo, indicando que es necesario completar toda la información para poder continuar.

Estas medidas de validación no solo previenen errores durante la ejecución del sistema, sino que también contribuyen a lo explicado en el punto anterior sobre la búsqueda de una mayor claridad en la interacción con el usuario, permitiéndole entender y corregir de manera rápida y sencilla cualquier entrada de datos errónea.

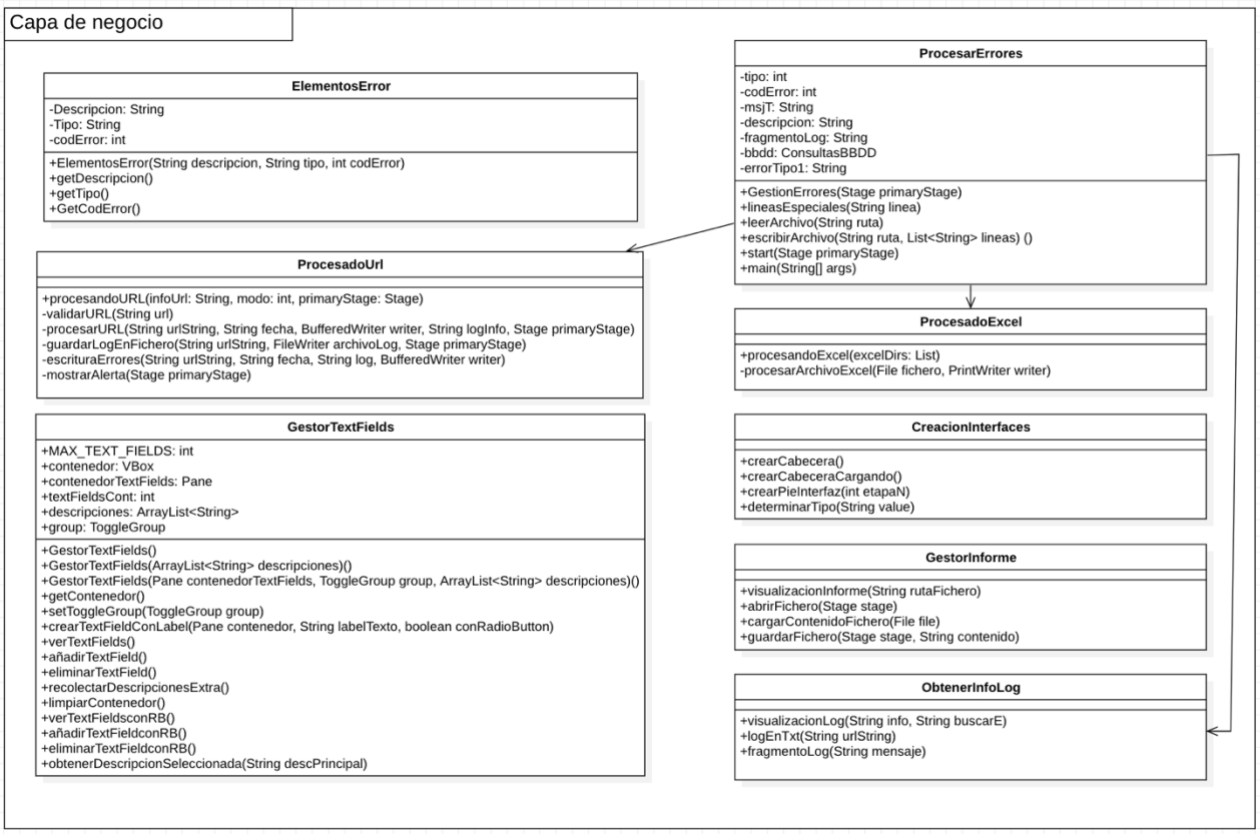


Figura 4.2.3 – Diagrama de clases capa de negocio

A continuación, se detalla la funcionalidad de las clases que componen el diagrama de clases de la capa de negocio correspondiente a la figura 4.2.3:

- **ElementosError:** Clase que almacena la información esencial de un error concreto, incluyendo su descripción, tipo y código identificativo. Esta estructura facilita el manejo organizado de los errores detectados. Además, ElementosError es indispensable para la clase ConsultasBBDD, ya que permite obtener y gestionar el listado de errores almacenados en la base de datos.
- **ProcesarErrores:** Esta clase constituye el núcleo del proceso de generación de informes. Una vez que, desde la clase InterfazInicio, se define la funcionalidad sobre la cual se desea crear el informe, ProcesarErrores se encarga de analizar y tratar cada error de forma individual, en función de su tipología. Para ello, interactúa con las clases correspondientes según el tipo de error detectado (InterfazTipo2, InterfazTipo3, InterfazTipoNuevoError). La gestión de las distintas tipologías de errores se realiza mediante una estructura *switch*, lo cual facilita la escalabilidad del sistema. De este modo, ante la aparición de un nuevo tipo de error, será suficiente con añadir un nuevo caso (*case*) en dicha estructura y desarrollar la interfaz asociada, sin necesidad de modificar el funcionamiento global del sistema.

Una vez finalizado el procesamiento de todos los errores, esta clase se encarga de mostrar la interfaz InterfazFinal, que actúa como pantalla de cierre del proceso y proporciona al usuario un resumen o confirmación del resultado obtenido.

- **ProcesadoExcel y ProcesadoUrl:** Estas clases son utilizadas por InterfazInicio para determinar la fuente desde la cual se obtendrán los errores generados, ya sea a través de un archivo Excel o mediante una URL. Su función principal es extraer y estructurar los datos de error, que posteriormente serán procesados por la clase ProcesarErrores.
- **ObtenerInfoLog:** Esta clase se utiliza exclusivamente cuando se detecta un error de tipo 2, denominado "multirrespuesta". Para determinar la descripción más adecuada del mensaje de error, es necesario acceder a la mayor cantidad de información posible relacionada con dicho error. En este contexto, ObtenerInfoLog interviene accediendo al archivo de registro (*log*) que se está procesando, localizando el error específico y extrayendo toda la información relevante asociada a ese tipo de error. En resumen, esta clase proporciona el fragmento del log correspondiente al error detectado, facilitando así al usuario la selección de la descripción más precisa a partir de los datos disponibles.
- **CreacionInterfaces:** Esta clase tiene como propósito centralizar los métodos responsables de definir la estética de las interfaces. Su objetivo es evitar la repetición de código en las distintas clases que implementan una interfaz, específicamente en lo referente a la generación de la cabecera y el pie de esta. De este modo, se reduce la redundancia y se facilita el mantenimiento y la implementación de futuras modificaciones en el diseño de las interfaces.
- **GestorInforme:** Proporciona funcionalidades para la gestión de informes en formato de texto dentro de la aplicación. Incluye métodos para la visualización del contenido de un fichero a partir de su ruta, la apertura de archivos mediante un selector gráfico, así como la lectura y carga del contenido de dichos archivos. Además, permite guardar información en ficheros de texto, asegurando la interacción fluida con el sistema de archivos y facilitando al usuario la manipulación de los informes generados. Esta clase es fundamental para la correcta gestión y presentación de los datos derivados del análisis de errores.
- **GestorTextFields:** gestiona dinámicamente un conjunto de campos de texto (*TextFields*) en una interfaz gráfica, permitiendo añadir, eliminar y visualizar hasta un máximo de cinco campos con sus respectivas descripciones. Además, ofrece la posibilidad de asociar estos campos con botones de opción (*RadioButtons*) para seleccionar una descripción específica. Internamente,

mantiene una lista de descripciones que se actualiza en tiempo real conforme el usuario escribe en los campos. Esta clase facilita la interacción y organización de los datos introducidos por el usuario, simplificando la gestión visual y lógica de múltiples entradas relacionadas en la interfaz.

4.3. Capa de datos

Por último, se encuentra la capa de datos cuya funcionalidad es comunicarse con la base de datos permitiendo hacer consultas, actualizaciones o inserciones de nuevas entradas.

Para profundizar en las consultas implementadas es necesario previamente conocer las tablas y campos creados para el análisis y procesamiento de los *logs*. En este caso se han establecido dos nuevas tablas.

"LOG_ERRORES_COMUNES" almacena el listado de errores comunes obtenidos tras un exhaustivo análisis de *logs* de los últimos meses. Esta tabla cuenta con las siguientes columnas:

- COD_ERROR → Código identificativo único del error, que se autogenera a partir de la secuencia "CLAIMS.SEQ_LISTADO_ERRORES".
- MENSAJE → Fragmento del mensaje de error que permite identificarlo.
- DESCRIPCION → Explicación de la causa del error
- TIPO → Permite clasificar el error entre los tipos categorizados (0,1, 2 o 3) para su posterior procesamiento.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	COD_ERROR	NUMBER(38,0)	No	(null)	1	(null)
2	MENSAJE	VARCHAR2(500 BYTE)	No	(null)	2	(null)
3	DESCRIPCION	VARCHAR2(500 BYTE)	No	(null)	3	(null)
4	TIPO	VARCHAR2(3 BYTE)	Yes	(null)	4	(null)

Figura 4.3.1 – Representación de la tabla LOG_ERRORES_COMUNES en SQL

Por otro lado, cuando se categorizaron lo errores se concluyó que era necesaria una segunda tabla cuya función es almacenar las descripciones de aquellos mensajes de error que tienen múltiples descripciones para un mismo error y que requiere de información más detallada para saber cuál es la respuesta mejor se adecua en cada caso. Por ello se requiere de la tabla **"LOG_ERRORES_MULTIOPCION"**, con las siguientes columnas:

- ID → Código identificativo de la descripción, que se autogenera a partir de la secuencia "CLAIMS.SEQ_LISTADO_ERRORES_TIPO2".
- FK_COD_ERROR → Código identificativo del error al que corresponde la descripción, es una *foreing key* a la tabla LOG_ERRORES_COMUNES.
- DESCRIPCION → Explicación de la causa del error.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	ID	NUMBER(38,0)	No	CLAIMS.S...	1	(null)
2	FK_COD_ERROR	NUMBER(38,0)	Yes	(null)	2	(null)
3	DESCRIPCION	VARCHAR2(500 BYTE)	No	(null)	3	(null)

Figura 4.3.2 – Representación de la tabla LOG_ERRORES_MULTIOPCION en SQL

Para la asignación de identificativos únicos para COD_ERROR en **"LOG_ERRORES_COMUNES"** y para ID en **"LOG_ERRORES_MULTIOPCION",** se han creado dos nuevas secuencias que asignan un valor autogenerado y único. Estas nuevas secuencias son respectivamente **"CLAIMS.SEQ_LISTADO_ERRORES"** y **"CLAIMS.SEQ_LISTADO_ERRORES_TIPO2**", que incrementan su valor en uno cada vez que se hace una nueva entrada.

Al tratarse de una base de datos ya existente es importante adaptarse a los requisitos y normativas ya impartidos en las misma. En este caso Gredos cuenta con un estricto régimen de seguridad donde los usuarios requieren de permisos específicos para poder crear, borrar, consultar o actualizar cada tabla. Por todo esto, para poder llevar a cabo estas acciones sobre las nuevas tablas fue necesario asignarle permisos sobre ellas a los usuarios correspondientes. Para el sistema solo se requiere hacer consultas e inserciones a base de datos por lo que al usuario solo se le ha asignado este tipo de permisos.

Una vez creadas las tablas, secuencias y asignados los permisos necesarios, el siguiente paso consistió en establecer la conexión con la base de datos e implementar las consultas, inserciones y actualizaciones requeridas para el correcto funcionamiento del sistema. Para ello, se desarrolló la clase **ConsultasBBDD.java**, que agrupa todos los métodos relacionados con la interacción con la base de datos de Gredos. Esta clase es utilizada tanto por la clase principal **ProcesarErrores** como por otras clases encargadas de gestionar nuevos errores, como **InterfazNuevoError** e **InterfazAmpliarBBDD**.

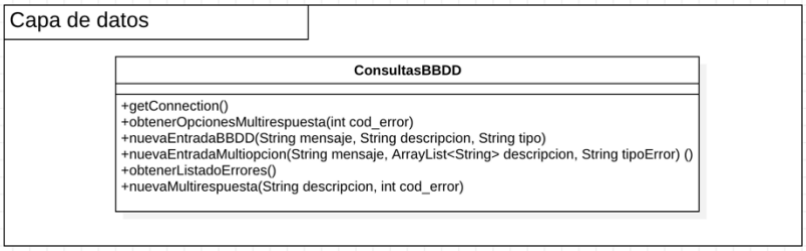


Figura 4.3.3 – Diagrama de clases capa de datos

Entre los principales métodos desarrollados en ConsultasBBDD.java se encuentran:

- **getConnection():** método encargado de establecer la conexión con la base de datos.
- **nuevaEntradaBBDD(String mensaje, String descripcion, String tipo):** realiza la inserción de un nuevo error en la tabla LOG_ERRORES_COMUNES.
- **nuevaEntradaMultiopcion(String mensaje, ArrayList<String> descripcion, String tipoError):** inserta un nuevo error de tipo 2 mediante una operación en ambas tablas: LOG_ERRORES_COMUNES y LOG_ERRORES_MULTIOPCION.
- **nuevaMultirespuesta(String descripcion, int cod_error):** añade una nueva descripción asociada a un error ya existente en la tabla LOG_ERRORES_MULTIOPCION.
- **obtenerListadoErrores():** recupera todos los errores almacenados en la tabla LOG_ERRORES_COMUNES para su comparación con los errores reportados en los *logs*.
- **obtenerOpcionesMultirespuesta(int cod_error):** consulta en la base de datos todas las descripciones asociadas al código identificativo del error indicado, obtenidas de la tabla LOG_ERRORES_MULTIOPCION.

La figura 4.3.4 muestra el diagrama relacional, en el cual se representan las relaciones entre las dos nuevas tablas descritas previamente. Cabe destacar que la base de datos de Gredos está formada por múltiples tablas adicionales; sin embargo, como estas no interactúan con el sistema de automatización, no se incluyen en el diagrama relacional.

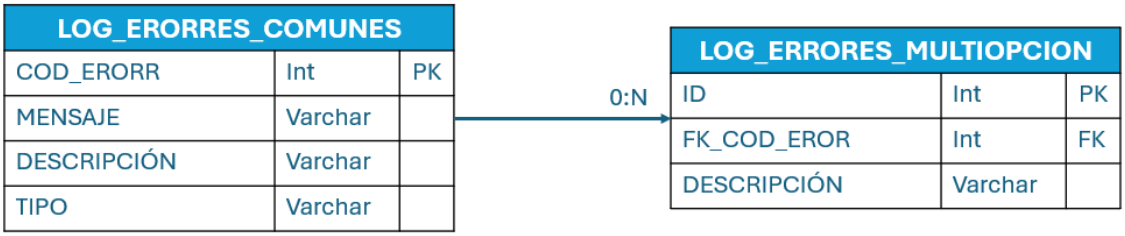


Figura 4.3.4 – Diagrama relacional

La relación 0:N determina que para una entidad A (“LOG_ERRORES_COMUNES”) puede no tener ninguna entidad B (“LOG_ERRORES_MULTIOPCION”) relacionada. Dicho de otro modo, una entrada de la entidad A puede estar relacionada con cero, uno o muchos registros de la entidad B. Pero una instancia de la entidad B debe estar relacionada con exactamente una instancia de la entidad A.

4.4. Flujo del proceso.

El sistema implementado tiene como objetivo automatizar el proceso de análisis de *logs* en el contexto de un sistema de gestión de reclamaciones. Su funcionamiento se estructura en dos posibles rutas iniciales: la ampliación de la base de datos de errores o la generación de un informe de análisis.

1. Ampliación de la Base de Datos de Errores

En esta modalidad, el usuario puede introducir manualmente nuevos mensajes de error junto con su correspondiente descripción. Esta información es incorporada a la base de datos, permitiendo su reutilización en análisis futuros y facilitando la identificación automatizada de errores ya conocidos.

2. Generación de Informe de Errores

Para esta opción, el sistema solicita al usuario una entrada inicial, que puede adoptar una de las siguientes formas: una fecha concreta, una URL específica o un archivo en formato Excel. El tratamiento varía según el tipo de entrada proporcionada:

- Entrada por Fecha: El sistema accede automáticamente a los logs correspondientes a la fecha indicada, relativos a los tres procesos principales del sistema ("*switchingReclamaciones*", "*AutomatizacionReclamaciones*" y "*GeneraciónCartas*"). Estos archivos se descargan y filtran para extraer únicamente los mensajes de error relevantes.
- Entrada por URL: Se obtiene únicamente el log vinculado a la URL proporcionada, al cual se aplica el mismo procedimiento de extracción de errores.
- Entrada mediante Archivo Excel: El sistema procesa el archivo para extraer exclusivamente los mensajes de error, omitiendo metadatos como fechas u horas.

Una vez recopilados los mensajes de error, estos se agrupan evitando duplicidades en un archivo temporal. A partir de este archivo se inicia el proceso de análisis automatizado.

Proceso de Análisis y Clasificación

Durante el análisis, el sistema recorre cada línea del archivo temporal y la compara con los registros almacenados en la base de datos de errores conocidos. En caso de coincidencia, se añade automáticamente la descripción asociada al mensaje.

Para aquellos errores que no tienen una coincidencia directa o que requieren una intervención por parte del usuario, se contemplan los siguientes casos:

- Error Desconocido: Se clasifica como nuevo y el sistema permite al usuario introducir manualmente una descripción, ampliando así la base de datos.
- Tipo 2 – Error con Posibles Interpretaciones: El sistema proporciona información adicional del error, incluyendo fragmentos relevantes del log (utilizando, por ejemplo, la clase ObtenerInfoLog). Además, ofrece al usuario un conjunto de descripciones sugeridas entre las que puede elegir o, si lo prefiere, introducir una nueva.
- Tipo 3 – Error Parcialmente Descrito: Se presenta el mensaje de error junto con una descripción incompleta, la cual debe ser completada por el usuario (por ejemplo, introduciendo un número de caso o un identificador específico).
- Tipo 0 y Tipo 1: Estos casos no requieren intervención del usuario, ya que la descripción asociada en la base de datos es suficiente para su interpretación.

Generación del Informe Final

Una vez finalizado el análisis y asignadas las descripciones correspondientes a todos los errores detectados, el sistema genera una previsualización del informe final. Desde esta vista, el usuario tiene las siguientes opciones:

- Guardar el informe generado.
- Realizar modificaciones manuales sobre los datos presentados.
- Volver al menú principal para iniciar un nuevo proceso.

La figura 4.4.1 proporciona una visión más esquemática del flujo y funcionamiento del sistema

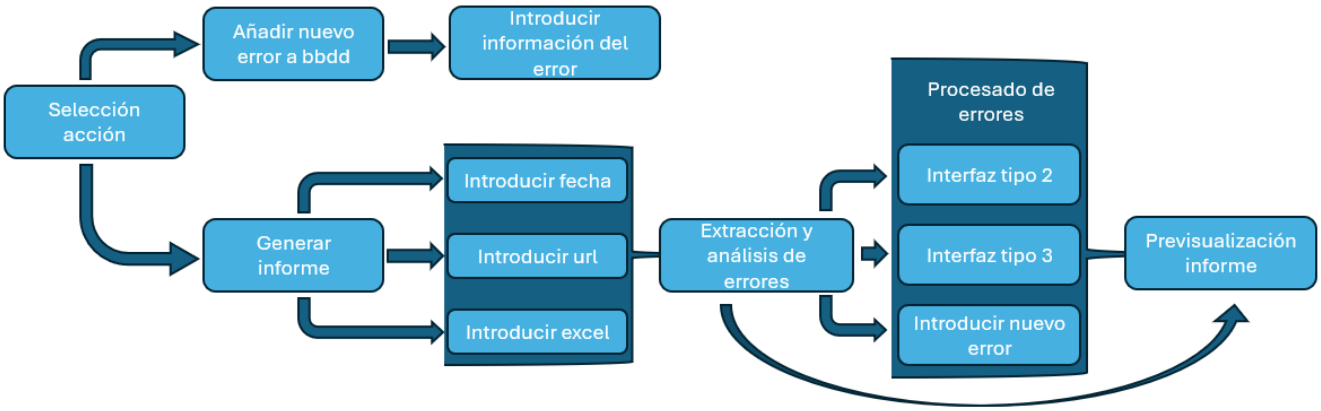


Figura 4. 4. 1 – Esquema representativo del flujo del proceso

5. RESULTADOS

Se han realizado múltiples pruebas con el objetivo de verificar si el sistema cumple con el funcionamiento deseado. Para que el testeo sea lo más exhaustivo posible se han realizado los siguientes tres tipos de pruebas:

5.1. Pruebas unitarias

Las pruebas unitarias se encargan de verificar el funcionamiento correcto de unidades individuales de código, es decir, que cada componente haga exactamente lo que se espera.

Para garantizar el adecuado funcionamiento del sistema en su conjunto, es fundamental que los métodos que interactúan directamente con la base de datos operen correctamente. En este sentido, se ha prestado especial atención a las funciones desarrolladas en la clase ConsultasBBDD.java. Por ello, se han implementado pruebas unitarias específicas para los métodos contenidos en esta clase, con el fin de validar su correcto desempeño.

ObtenerOpcionesMultirespuestaTest [Tablas 5.1.1 y 5.1.2]

Clase	ConsultasBBDD.java
Método	obtenerOpcionesMultirespuesta(int cod_error)
ID	**DESCRIPCIÓN**
U1. a	Devuelve las múltiples descripciones del error indicado
U1. b	No hay descripciones para el caso indicado

Tabla 5.1. 1 – Casuísticas a probar para obtenerOpcionesMultirespuesta

ID	**Entrada**	**Valor esperado**
U1. a	Cod_error = 123	["Descripcion 1", "Descripcion 2"]
U1. b	Cod_error = 123	[ ]

Tabla 5.1. 2 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.1.2

NuevaEntradaBBDDTest: [Tablas 5.1.3 y 5.1.4]

Clase	ConsultasBBDD.java
Método	nuevaEntradaBBDD(String mensaje, String descripcion, String tipo)
ID	**DESCRIPCIÓN**
U2. a	Añade un nuevo error a bbdd

Tabla 5.1. 3– Casuísticas a probar para nuevaEntradaBBDD

ID	**Entrada**	**Valor esperado**
U2. a	Mensaje: Mensaje error BBDD" Descripción: Error descripcion"	Actualización de la base de datos

	Tipo: “TIPO 2"	

Tabla 5.1.4 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.1.3

NuevaEntradaMultiopcionTest: [Tablas 5.1.5 y 5.1.6]

Clase	ConsultasBBDD.java
Método	nuevaEntradaBBDD(String mensaje, ArrayList<String> descripciones, String tipo)
ID	**DESCRIPCIÓN**
U3. a	Añade una nueva descripción a un error de bbdd
U3. b	No existe el error al que se quiere añadir la descripción

Tabla 5.1.5– Casuísticas a probar para nuevaEntradaBBDD

ID	**Entrada**	**Valor esperado**
U3. a	cod_error: previamente creado, por ejemplo, el 3 Mensaje: msg Descripciones: "Descripcion A", "Descripcion B" Tipo: TIPO 2	Actualización de la base de datos
U3. b	cod_error: no hay un error con ese identificador. Mensaje: msg Descripciones: "Descripcion A", "Descripcion B" Tipo: TIPO 2	Excepción

Tabla 5.1.6 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.1.5

NuevaMultirespuestaTest: [Tablas 5.1.7 y 5.1.8]

Clase	ConsultasBBDD.java
Método	nuevaMultirespuesta(String descripcion, int cod_error)
ID	**DESCRIPCIÓN**
U5. a	Añade una nueva descripción a un error existente

Tabla 5.1.7 – Casuísticas a probar para nuevaMultirespuesta

ID	**Entrada**	**Valor esperado**
U5. a	Descripcion:"Nueva descripcion" cod_error: 200	Actualización de la base de datos

Tabla 5.1.8 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.1.7

ObtenerListadoErroresTest: [Tablas 5.1.9 y 5.1.10]

Clase	ConsultasBBDD.java
Método	obtenerListadoErrores()
ID	DESCRIPCIÓN
U4. a	No existe el error al que se quiere añadir la descripcion
U4. b	Devuelve listado de errores almacenados en base de datos

Tabla 5.1.9 – Casuísticas a probar para obtenerListadoErrores

ID	Entrada	Valor esperado
U4. a	-	[ ]
U4 b	-	[ElementosError(ERR_1, Desc 1, Tipo 0), ElementosError(ERR_2, Desc 2, Tipo 1), ElementosError(ERR_3, Desc 3, Tipo 2)]

Tabla 5.1.10 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.1.9

La implementación de estas pruebas unitarias ha sido clave para garantizar la robustez de las operaciones de lectura y escritura sobre la base de datos. Cada prueba ha sido diseñada para abarcar tanto los escenarios de uso habituales como aquellos poco frecuentes. Esto permite verificar que los métodos reaccionan adecuadamente ante entradas válidas e inválidas, preservando la consistencia del sistema en todo momento.

Estas pruebas no solo detectan errores tempranos, sino que, gracias a su cobertura, se ha conseguido reducir significativamente el riesgo de fallos durante el funcionamiento real de la aplicación, contribuyendo así a una mayor calidad, fiabilidad y mantenibilidad del software.

5.2. Pruebas de integración

En el caso de las pruebas de integración tienen como objetivo asegurar el correcto funcionamiento del sistema cuando diferentes componentes de este interactúan entre sí. Dicho de otro modo, se prueba la interacción entre módulos.

A continuación, se especifica el plan de pruebas que se ha implementado en base a las funcionalidades de las clases “ObtenerInfoLog.java”, “ProcesadoExcel.java” y “ProcesadoURL.java”.

ObtenerInfoLogTest.java [Tablas 5.2.1, 5.2.2, 5.2.3, 5.2.4, 5.2.5, 5.2.6]

Clase	ObtenerInfoLog.java
Método	visualizacionLog(String info, String buscarE);
ID	DESCRIPCIÓN DEL CASO
I1. a *	Error categoría AutomatizacionReclamaciones fecha antigua por lo que no hay registros del log.
I1. B	Error categoría GeneracionCartas fecha del día de ayer antes de las 21 horas de hoy.
I1. c *	Error categoría AutomatizacionReclamaciones fecha válida y encuentra el error.
I1. d *	Error categoría SwitchingReclamaciones fecha valida, pero error no encontrado.
I1. e	Categoría no valida, con fecha valida.

*Estos casos deberían ser probados para las tres categorías (“AutomatizacionReclamaciones”, “switchingReclamaciones” y “GeneraciónCartas”)

Tabla 5.2.1 – Casuísticas a probar para visualizaciónLog

ID	Entrada	Valor esperado
I1. a	“AutomatizacionRecl2025-01-10” "com.eon.claims.batch.Batch_CLAIMS_ Automatizacion_reclamaciones java.lang.Exception"	“No ha sido posible proporcionar más información sobre el error"
I1. b	“GeneracionCartas + (fecha de ayer)” “com.eon.claims.batch.Batch_CLAIMS _Automatizacion_reclamaciones java.lang.Exception”	"No ha sido posible proporcionar más información sobre el error"
I1. c	“AutomatizacionRecl + (fecha de ayer)” “com.eon.claims.batch.Batch_ CLAIMS_Automatizacion_reclamacione s java.lang.Exception”	“ java.lang.NullPointerException: null at com.eon.claims.services.Servicio ProcesaReclamAutomaticas. obtenerInfoBDF1_E_REDES…”
I1. d	"SwitchingRecl + (fecha de ayer)” “Este error no estará en el log”	"No ha sido posible proporcionar más información sobre el error"
I1. e	"CategoriaNoValida + (fecha de ayer)” “com.eon.claims.batch.Batch_CLAIMS _Automatizacion_reclamaciones java.lang.Exception”	"No ha sido posible proporcionar más información sobre el error"

Tabla 5.2. 2 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.2.1

Clase	ObtenerInfoLog.java
Método	fragmentoLog(String mensaje)
ID	DESCRIPCIÓN DEL CASO
I2. a	Se encuentra el error dentro del fichero donde se ha almacenado el log
I2. b	El error que se busca NO se encuentra en el log

*Para probar este método, se ha simulado un fichero forzando la escritura de determinadas líneas en el fichero "logParaFragmentos.txt"

Tabla 5.2.3 – Casuísticas a probar para fragmentoLog

ID	Entrada	Valor esperado
I2. a	"com.viesgo.ptosuministro.client.PuntoSuministroClient Error en la recuperación del detalle del CUPS, para el CUPS --> ES0033770524881002AP"	"com.viesgo.ptosuministro.client.PuntoSuministroClient Error en la recuperación del detalle del CUPS, para el CUPS --> ES0033770524881002AP Mas líneas de prueba Hasta no encontrar los caracteres correctos debe seguir mostrando estas líneas [INFO] Cadena de caracteres encontrada"
I2. b	"EsteErrorNoEstaEnLog"	"No ha sido posible proporcionar más información sobre el error"

Tabla 5.2.4 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.2.3

Clase	ObtenerInfoLog.java
Método	logEnTxt(String urlString)
ID	DESCRIPCIÓN DEL CASO
I3. a	Se introduce una URL con un formato no válido

Tabla 5.2.5 – Casuísticas a probar para logEnTxt

ID	Entrada	Valor esperado
I3. a	"htp://invalid-url"	MalformedURLException

Tabla 5.2.6 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.2.5

Figura 5.2.1 – Captura que muestra que los test Junit se ejecutan con éxito

ProcesadoExcelTest.java: [Tablas 5.2.7 y 5.2.8]

Clase	ProcesadoExcel.java
Método	procesandoExcel(List<String> excelDirs)
ID	DESCRIPCIÓN DEL CASO
I4. a	Excel introducido es válido.
I4. b	Documento introducido no existe.
I4. c	Se introduce un documento Excel correcto, pero está vacío.
I4. d	Se introduce más de un documento Excel.

Tabla 5.2.7 – Casuísticas a probar para procesandoExcel

ID	Entrada	Valor esperado
I4. a	Directorio del fichero “Excel_valido.xlsx” con el siguiente listado de errores: [Error1, Error2, Error3, Error1]	[Error1, Error2, Error3]
I4. b	Directorio del fichero "excelInexistente.xlsx" (no existe este Excel)	No procesa el Excel.
I4. c	Directorio del fichero “ExcelVacio.xlsx” (no tiene información) []	[]
I4. d	Directorio de los ficheros “Excel_valido.xlsx” y “Excel_valido2.xlsx”, con el siguiente listado de errores: [Error1, Error2, Error3, Error1] [Error1, Error2, Error4, Error2]	[Error1, Error2, Error3, Error4]

Tabla 5.2. 8 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.2.7

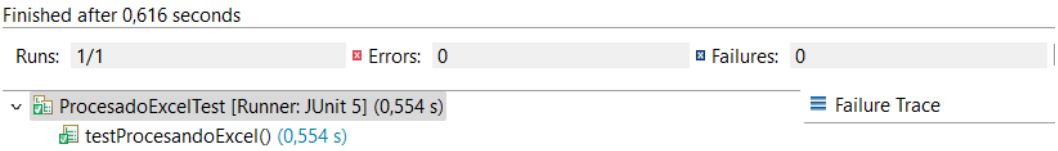


Figura 5.2.2 – Captura que muestra que los test Junit se ejecutan con éxito

ProcesadoURLTest.java: [Tablas 5.2.9 y 5.2.10]

Clase	ProcesadoExcel.java
Método	procesandoURLS(String infoUrl, int modo, Stage primaryStage)
ID	DESCRIPCIÓN DEL CASO
I5. a	Usuario introduce una URL válida
I5. b	Usuario introduce una URL invalida
I5. c	Usuario introduce una fecha

Tabla 5.2. 9 – Casuísticas a probar para procesandoURLS

ID	Entrada	Valor esperado
I5. a	http://10.120.71.119/logs/batch/GRE2/AutomatizacionReclamaciones/CLAIMS_AutomatizacionReclamaciones.log." + fechaAyer 0 mockStage	[True Se crea el fichero “Errores.txt” con los errores del día introducido
I5. b	http://url_invalida.log.2024-05-01 0 mockStage	False
I5. c	fechaAyer 1 mockStage	True Se crea el fichero “Errores.txt” con los errores del día introducido

Tabla 5.2. 10 – Valores de entrada y salida esperada para las casuísticas de la tabla 5.2.9

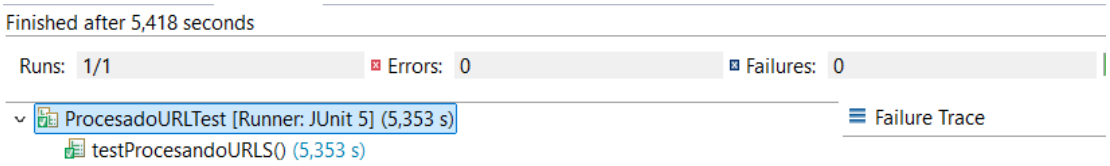


Figura 5.2.3 – Captura que muestra que los test Junit se ejecutan con éxito

El plan de pruebas implementado en este apartado responde al enfoque de pruebas de integración, ya que han sido diseñadas específicamente para verificar la correcta interacción entre múltiples componentes del sistema. En particular, se validan procesos completos que abarcan desde la lectura de archivos externos (como *logs* o ficheros Excel), el tratamiento de fechas y rutas, hasta la escritura de resultados en archivos de salida. Por ejemplo, en los test de la clase ObtenerInfoLog se comprueba cómo se recupera información de un log real a partir de una URL, se guarda en disco y luego se analiza su contenido, lo que implica una interacción directa con el sistema de archivos y con estructuras de datos externas. De manera similar, en ProcesadoExcelTest y ProcesadoURLTest se evalúan flujos de procesamiento donde intervienen datos de entrada reales, validaciones lógicas, y escritura de resultados, comprobando que todo el conjunto funcione de forma coherente.

Las pruebas desarrolladas han permitido validar el comportamiento del sistema en condiciones reales, comprobando que los distintos módulos colaboran correctamente y que los datos se procesan de manera fiable desde la entrada hasta la salida. Aunque se ha optado por un enfoque de integración que prioriza la verificación de flujos completos, esto ha sido especialmente útil para detectar errores relacionados con archivos, conexiones y lógica intermedia.

5.3. Pruebas de aceptación

Para la realización de este apartado, ha sido necesaria la colaboración de usuarios externos, dado que las pruebas llevadas a cabo tienen como objetivo validar que el sistema cumple con los requisitos establecidos por el cliente. Estas pruebas se centran en evaluar el comportamiento del sistema desde una perspectiva funcional, verificando que el software responde adecuadamente a las necesidades especificadas.

Con el fin de maximizar la eficacia de este tipo de validaciones, se proporcionó el sistema desarrollado a profesionales con experiencia en la revisión manual de *logs*. De este modo, se pudo someter el software a una evaluación rigurosa y especializada.

Del análisis de las valoraciones emitidas por estos profesionales, destaca de forma notable la reducción en el tiempo necesario para la elaboración de los informes.

Asimismo, señalaron como aspecto positivo la posibilidad de realizar otras tareas en paralelo durante el proceso de análisis. Otro elemento ampliamente valorado fue la intuitividad de la aplicación, gracias a la incorporación de mensajes informativos y aclaratorios que minimizan la posibilidad de confusión durante su uso.

Además, se subrayó como fortaleza la extensa base de datos de errores comunes que el sistema consulta. Permitiendo que, en la mayoría de los casos, los informes puedan generarse de forma totalmente autónoma, siendo poco frecuente que un error detectado no esté previamente registrado.

Si bien las evaluaciones generales fueron muy positivas, también surgieron propuestas de mejora. Entre ellas, se sugiere la posibilidad de ampliar las funcionalidades sobre la base de datos, como la opción de eliminar entradas obsoletas, circunstancia que podría darse teniendo en cuenta que el sistema Gredos está en constante evolución y mantenimiento. Otras sugerencias incluyen la posibilidad de realizar acciones adicionales sobre los informes, como enviarlos por correo directamente desde la aplicación una vez validados, o permitir su exportación en distintos formatos, más allá del actual formato ".txt".

En conclusión, las pruebas realizadas con la colaboración de usuarios externos han permitido validar de forma efectiva el cumplimiento de los requisitos funcionales del sistema. La retroalimentación obtenida no solo confirma que el software responde a las expectativas del cliente, sino que también destaca mejoras sustanciales en términos de eficiencia, usabilidad y autonomía en la generación de informes. Además, las sugerencias planteadas por los profesionales participantes abren nuevas oportunidades de mejora y evolución para el sistema, especialmente en lo relativo a la gestión dinámica de la base de datos y la ampliación de funcionalidades en el tratamiento de informes. En conjunto, los resultados obtenidos refuerzan la solidez de la solución propuesta y confirman que el objetivo principal del desarrollo ha sido alcanzado con éxito.

6. CONCLUSIÓN

En este apartado se hace un análisis crítico sobre el sistema desarrollado teniendo en cuenta no solo el producto final sino también su proceso, haciendo autocritica de los puntos fuertes y de las mejores y escalabilidad que puede alcanzar el proyecto.

6.1. Comparativo procedimiento manual vs automatizado.

Para entender la importancia de las mejoras que han supuesto la automatización del proceso lo primero ha sido hacer una comparativa entre ambos procedimientos.

En el enfoque tradicional o manual, la generación de informes de errores parte de un archivo Excel proporcionado por el usuario. Este documento contiene información básica sobre cada error, como la fecha, la hora y el mensaje generado por el sistema. El análisis de los errores se realiza de forma completamente manual: el usuario debe revisar uno por uno los errores listados, buscar cada caso en los registros (*logs*) del sistema, identificar el fragmento correspondiente y analizarlo para determinar la causa. A continuación, debe redactar una descripción explicativa que acompañe al mensaje original. Este procedimiento debe repetirse para cada entrada del Excel, lo cual representa una carga significativa de trabajo, especialmente en entornos con una alta frecuencia de errores o registros extensos. Además, este método está expuesto a errores de interpretación y falta de uniformidad en los criterios empleados por distintos analistas.

El proceso automatizado, desarrollado como objetivo principal del Trabajo Fin de Grado, transforma y optimiza por completo esta dinámica. El usuario solo necesita proporcionar alguno de los siguientes elementos: el archivo Excel con los errores o la URL (bien especificando la URL de un log o a través de la fecha acceder a las URLs de los *logs* del día especificado). A partir de esta información, el sistema extrae automáticamente los mensajes de error, los procesa uno a uno y los compara con los registros almacenados en una base de datos que contiene errores ya clasificados junto a sus descripciones encargadas de la notificar la causa del error. Cuando se detecta una coincidencia, se inserta automáticamente tanto el mensaje original como su descripción en el documento del informe que se está generando. Si bien es importante recalcar que el proceso no puede ser al cien por ciento automático pues para determinados errores se requiere la intervención por parte del usuario.

6.2. Análisis de los resultados

La anterior comparación entre el análisis manual del proceso y su correspondiente versión automatizada ha sido la base para realizar un análisis crítico de los resultados obtenidos. Esta comparación ha permitido identificar mejoras

significativas, especialmente en lo que respecta a la eficiencia y la reducción del tiempo necesario para completar el proceso.

Uno de los aspectos más destacables es la notable disminución en el tiempo requerido para la generación de informes. Mientras que el análisis manual podía requerir varias horas, la versión automatizada ha conseguido completar la misma tarea en cuestión de minutos, gracias a la elevada velocidad de procesamiento del sistema desarrollado. Esta mejora no solo implica un ahorro de tiempo considerable, sino que también representa una optimización importante en la gestión de recursos y en la operatividad general del procedimiento.

Aunque los resultados han sido muy positivos, cabe señalar que aún existe margen de mejora. Con el uso de técnicas más avanzadas, especialmente en lo referente a la extracción y clasificación de errores, es factible que los tiempos de procesamiento puedan reducirse aún más.

Desde una perspectiva crítica, se puede afirmar que el sistema cumple los objetivos planteados al inicio del trabajo. La automatización del proceso ha mejorado sustancialmente tanto en términos de eficiencia como de rendimiento. Sin embargo, también se ha constatado que no es posible una automatización completa del 100%, ya que el sistema todavía requiere de ciertas interacciones mínimas por parte del usuario. Aun así, estas intervenciones son puntuales y no afectan de forma significativa al ahorro global de tiempo conseguido.

En definitiva, los resultados alcanzados validan la hipótesis inicial del trabajo: que la automatización del proceso puede mejorar significativamente su eficiencia sin comprometer la calidad del análisis. Este proyecto no solo demuestra la viabilidad técnica de la propuesta, sino que también sienta las bases para futuras mejoras que podrían llevar el sistema a un nivel aún más alto de autonomía y rendimiento.

6.3. Limitaciones del sistema

A pesar de los resultados positivos obtenidos, el sistema desarrollado presenta algunas limitaciones que es importante tener en cuenta para futuras fases de mejora.

En primer lugar, aunque actualmente el sistema gestiona de forma eficaz cuatro tipos de errores, su alcance puede verse limitado en cuanto a la detección y tratamiento de otros tipos de incidencias que puedan surgir a medida que evoluciona la plataforma Gredos. No obstante, el sistema ha sido diseñado con una arquitectura modular y escalable, lo cual facilita su adaptación y actualización para incorporar nuevos tipos de errores en el futuro, en caso de que estos aparezcan.

Otra limitación significativa es que el sistema no permite generar informes agrupados opción interesante para los análisis del fin de semana, es decir, para viernes, sábado y domingo. En su estado actual, el análisis debe realizarse de forma individual para cada uno de esos días, lo cual puede resultar ineficiente.

También se debe mencionar que la herramienta solo permite analizar datos correspondientes a los últimos 30 días, ya que no se dispone de registros históricos (logs) con una antigüedad mayor. Esto restringe el análisis a un rango temporal relativamente corto, limitando la capacidad para detectar patrones a largo plazo.

Por último, cuando el usuario introduce una URL con un formato incorrecto, el sistema únicamente informa del error sin proporcionar opciones adicionales para corregirlo. Sería deseable que, en estos casos, el sistema ofreciera una ventana emergente que permitiera al usuario introducir manualmente la fecha y categoría correspondiente, facilitando así una recuperación más rápida y cómoda ante errores de entrada.

6.4. Posibles mejoras

Con el objetivo de continuar optimizando el sistema y abordar las limitaciones descritas en el apartado anterior, se proponen a continuación una serie de mejoras:

- **Integración con *ServiceNow*:** Una mejora clave sería permitir la publicación directa del informe generado en la plataforma *ServiceNow*. Esta funcionalidad facilitaría el flujo de trabajo y aumentaría la utilidad práctica del sistema en entornos corporativos. Alternativamente, se podría habilitar el envío automático del informe por correo electrónico a los destinatarios correspondientes.
- **Generación de estadísticas e informes analíticos:** Incorporar funcionalidades que permitan elaborar estadísticas sobre los errores más frecuentes facilitaría la identificación de patrones, irregularidades o áreas críticas que requieren atención. Esto dotaría al sistema de un componente analítico que podría ser muy útil para la toma de decisiones.
- **Agrupación de informes durante fines de semana:** Implementar una opción que permita agrupar los informes de viernes, sábado y domingo en un único análisis facilitaría la revisión del rendimiento durante el fin de semana y optimizaría el trabajo de los usuarios.
- **Mejora en el manejo de URLs incorrectas:** Diseñar un sistema más interactivo para la validación de URLs, que ofrezca al usuario alternativas o formularios de corrección en caso de error de formato, mejoraría notablemente la experiencia de uso y reduciría errores manuales.

La implementación de estas mejoras consolidaría la robustez del sistema y aumentaría significativamente su grado de automatización, usabilidad y valor añadido.

Como conclusión podemos determinar que la automatización no solo reduce de forma significativa el tiempo de elaboración del informe, sino que también mejora la precisión, la trazabilidad y la estandarización de los resultados. Esta solución se adapta especialmente bien a contextos donde se manejan grandes volúmenes de datos y donde la agilidad y la consistencia son factores clave para la eficiencia operativa.

7. REFERENCIAS

[1] F. Knupfer, "¿Qué es la gestión de logs? Guía experta y pasos clave en el proceso de gestión de logs", *New Relic*, 15 de noviembre de 2023. [En línea]. Disponible en: https://newrelic.com/es/blog/how-to-relic/what-is-log-management

[2] Definición de *log*: https://areaf5.es/blog/logs-que-son-y-para-que-sirven

[3] Definición y benecifios de la metodología agil: https://proyectosagiles.org/desarrollo-iterativo-incremental/

[4] Importancia y beneficios del monitoreo de *logs*: https://www.elastic.co/es/what-is/log-monitoring

[5] Documentación JavaFX: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[6] Libro sobre el testing JUnit 5: B. Kolaczyk, *Mastering Software Testing with JUnit 5*. Birmingham, UK: Packt Publishing, 2017.

[7] Definición caso de uso: https://www.ibm.com/docs/es/product-master/12.0.0?topic=processes-defining-use-cases

[8] Definición arquitectura de tres niveles: https://www.ibm.com/es-es/topics/three-tier-architecture

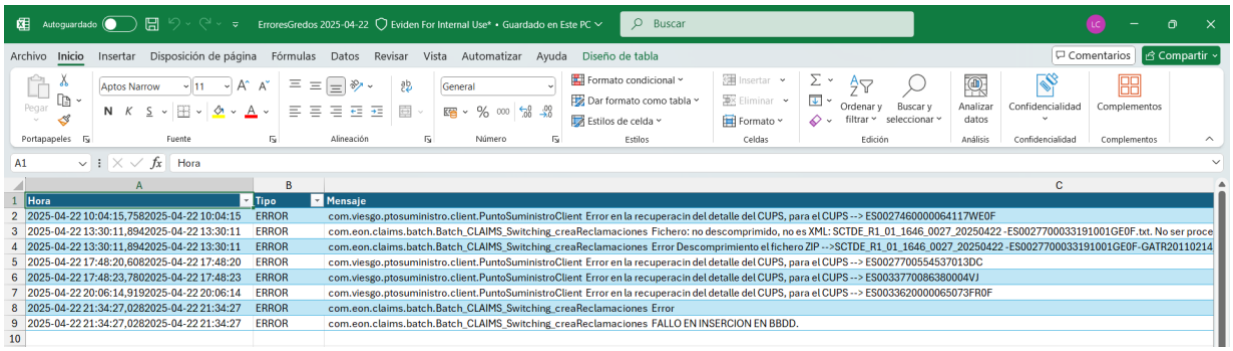
8. ANEXOS

Anexo I: Ejemplo formatos documento Excel donde se registran lo errores.

Estos ficheros reportan tres aspectos del fallo: fecha y hora a la que se ha producido, **tipo y el mensaje que ha reportado el error.**

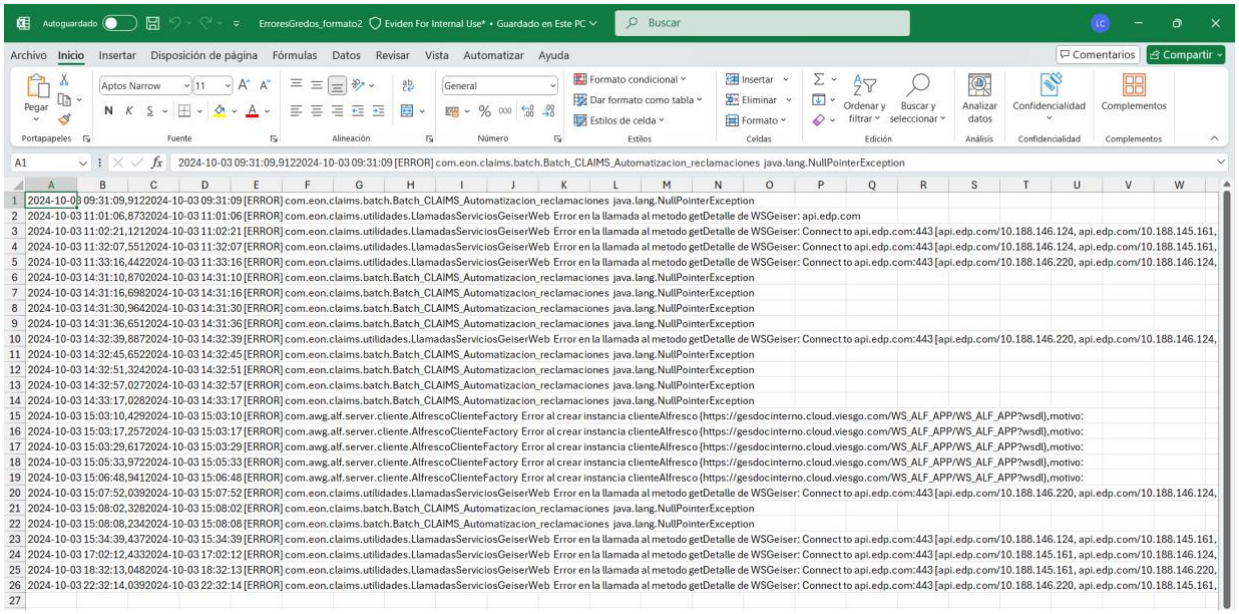
El sistema está preparado para gestionar los dos siguientes formatos:

Formato 1: tres columnas y solo utiliza la correspondiente a “mensaje”



Hora	Tipo	Mensaje
2025-04-22 10:04:15,7582025-04-22 10:04:15	ERROR	com.viesgo.ptosuministro.client.PuntoSuministroClient Error en la recuperacin del detalle del CUPS, para el CUPS --> ES0027460000064117WE0F
2025-04-22 13:30:11,8942025-04-22 13:30:11	ERROR	com.eon.claims.batch.Batch_CLAIMS_Switching_creaReclamaciones Fichero: no descomprimido, no es XML: SCTDE_R1_01_1646_0027_20250422 -ES0027700033191001GE0F.txt. No ser proce
2025-04-22 13:30:11,8942025-04-22 13:30:11	ERROR	com.eon.claims.batch.Batch_CLAIMS_Switching_creaReclamaciones Error Descomprimiento el fichero ZIP -->SCTDE_R1_01_1646_0027_20250422 -ES0027700033191001GE0F-GATR20110214
2025-04-22 17:48:20,6082025-04-22 17:48:20	ERROR	com.viesgo.ptosuministro.client.PuntoSuministroClient Error en la recuperacin del detalle del CUPS, para el CUPS --> ES0027700554537013DC
2025-04-22 17:48:23,7802025-04-22 17:48:23	ERROR	com.viesgo.ptosuministro.client.PuntoSuministroClient Error en la recuperacin del detalle del CUPS, para el CUPS --> ES0033770086380004VJ
2025-04-22 20:06:14,9192025-04-22 20:06:14	ERROR	com.viesgo.ptosuministro.client.PuntoSuministroClient Error en la recuperacin del detalle del CUPS, para el CUPS --> ES0033620000065073FR0F
2025-04-22 21:34:27,0282025-04-22 21:34:27	ERROR	com.eon.claims.batch.Batch_CLAIMS_Switching_creaReclamaciones Error
2025-04-22 21:34:27,0282025-04-22 21:34:27	ERROR	com.eon.claims.batch.Batch_CLAIMS_Switching_creaReclamaciones FALLO EN INSERCION EN BBDD.

Formato 2: una única fila por lo que el sistema se encarga de extraer solo lo relativo al mensaje, es decir, de “[ERROR]” en adelante.



1	2024-10-03 09:31:09,9122024-10-03 09:31:09 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
2	2024-10-03 11:01:06,8732024-10-03 11:01:06 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: api.edp.com
3	2024-10-03 11:02:21,1212024-10-03 11:02:21 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.124, api.edp.com/10.188.145.161,
4	2024-10-03 11:32:07,5512024-10-03 11:32:07 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.124, api.edp.com/10.188.145.161,
5	2024-10-03 11:33:16,4422024-10-03 11:33:16 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.220, api.edp.com/10.188.146.124,
6	2024-10-03 14:31:10,8702024-10-03 14:31:10 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
7	2024-10-03 14:31:16,6982024-10-03 14:31:16 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
8	2024-10-03 14:31:30,9642024-10-03 14:31:30 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
9	2024-10-03 14:31:36,6512024-10-03 14:31:36 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
10	2024-10-03 14:32:39,8872024-10-03 14:32:39 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.220, api.edp.com/10.188.146.124,
11	2024-10-03 14:32:45,6522024-10-03 14:32:45 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
12	2024-10-03 14:32:51,3242024-10-03 14:32:51 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
13	2024-10-03 14:32:57,0272024-10-03 14:32:57 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
14	2024-10-03 14:33:17,0282024-10-03 14:33:17 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
15	2024-10-03 15:03:10,4292024-10-03 15:03:10 [ERROR] com.awg.alf.server.cliente.AlfrescoClienteFactory Error al crear instancia clienteAlfresco [https://gesdocinterno.cloud.viesgo.com/WS_ALF_APP/WS_ALF_APP?wsdl],motivo:
16	2024-10-03 15:03:17,2572024-10-03 15:03:17 [ERROR] com.awg.alf.server.cliente.AlfrescoClienteFactory Error al crear instancia clienteAlfresco [https://gesdocinterno.cloud.viesgo.com/WS_ALF_APP/WS_ALF_APP?wsdl],motivo:
17	2024-10-03 15:03:29,6172024-10-03 15:03:29 [ERROR] com.awg.alf.server.cliente.AlfrescoClienteFactory Error al crear instancia clienteAlfresco [https://gesdocinterno.cloud.viesgo.com/WS_ALF_APP/WS_ALF_APP?wsdl],motivo:
18	2024-10-03 15:05:33,9722024-10-03 15:05:33 [ERROR] com.awg.alf.server.cliente.AlfrescoClienteFactory Error al crear instancia clienteAlfresco [https://gesdocinterno.cloud.viesgo.com/WS_ALF_APP/WS_ALF_APP?wsdl],motivo:
19	2024-10-03 15:06:48,9412024-10-03 15:06:48 [ERROR] com.awg.alf.server.cliente.AlfrescoClienteFactory Error al crear instancia clienteAlfresco [https://gesdocinterno.cloud.viesgo.com/WS_ALF_APP/WS_ALF_APP?wsdl],motivo:
20	2024-10-03 15:07:52,0392024-10-03 15:07:52 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.220, api.edp.com/10.188.146.124,
21	2024-10-03 15:08:02,3282024-10-03 15:08:02 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
22	2024-10-03 15:08:08,2342024-10-03 15:08:08 [ERROR] com.eon.claims.batch.Batch_CLAIMS_Automatizacion_reclamaciones java.lang.NullPointerException
23	2024-10-03 15:34:39,4372024-10-03 15:34:39 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.124, api.edp.com/10.188.145.161,
24	2024-10-03 17:02:12,4332024-10-03 17:02:12 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.145.161, api.edp.com/10.188.146.124,
25	2024-10-03 18:32:13,0482024-10-03 18:32:13 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.145.161, api.edp.com/10.188.146.220,
26	2024-10-03 22:32:14,0392024-10-03 22:32:14 [ERROR] com.eon.claims.utilidades.LlamadasServiciosGeiserWeb Error en la llamada al metodo getDetalle de WSGeiser: Connect to api.edp.com:443 [api.edp.com/10.188.146.220, api.edp.com/10.188.145.161,
27	