enero, 2024

Informe Herramienta de Búsqueda de Escritorio
Recuperación Avanzada de la Información

Alberto García Martín

Máster en Sistemas Inteligentes
Universidad de Salamanca

Índice

1. Introducción

En este informe se describe el trabajo realizado para la asignatura de Recuperación Avanzada de la Información, en el marco del máster en Sistemas Inteligentes. El objetivo de este proyecto es desarrollar una herramienta de búsqueda de escritorio que permita a los usuarios buscar y encontrar archivos en su sistema de archivos local. La herramienta a desarrollar debe ser capaz de indexar y buscar en una amplia gama de formatos de archivos, incluyendo documentos de texto, archivos PDF, documentos web, imágenes y más.

Además, la herramienta debe proporcionar una interfaz de usuario intuitiva que permita a los usuarios especificar consultas de búsqueda y visualizar los resultados de manera eficiente, filtrando los resultados según sea necesario. El usuario también debe poder abrir los documentos directamente desde la interfaz de la herramienta, lo que facilita el acceso a los archivos buscados. El usuario podrá especificar qué directorios de su sistema desea indexar de manera sencilla.

También hay que tener en cuenta que el sistema de archivos de un usuario puede cambiar con el tiempo, con la creación, modificación y eliminación de archivos. Por lo tanto, la herramienta debe ser capaz de monitorear en tiempo real el sistema de archivos del usuario, actualizando el índice de búsqueda para reflejar los cambios más recientes. Asegurando así que los resultados de búsqueda estén siempre actualizados.

Otro objetivo importante es garantizar que la herramienta sea capaz de manejar documentos en varios idiomas. Esto requiere el uso de analizadores de idiomas específicos para mejorar la eficacia de la búsqueda en diversos contextos lingüísticos.

Para resolver el problema de búsqueda, se utilizará Apache Lucene [4]. Lucene proporciona una búsqueda rápida y precisa que se puede integrar fácilmente en un entorno local. Además, se empleará Apache Tika [5] para la extracción de texto y datos de una amplia gama de formatos de archivo. Tika facilita la identificación y extracción del contenido de los archivos, lo que permite a Lucene indexar de manera efectiva diversos tipos de documentos.

Como tanto Lucene como Tika son bibliotecas escritas en Java, se utilizará Java para el desarrollo de la herramienta, en concreto Java 21 con el JDK GraalVM [1]. La interfaz de usuario se desarrollará con JavaFX [2], con un diseño estético proporcionado por la biblioteca de componentes MaterialFX [6].

Se mantendrá una lista persistente de qué archivos se han indexado y la configuración del usuario, para ello se serializará la información en archivos JSON, empleando la biblioteca Jackson [3].

Con el objetivo de solucionar los problemas planteados anteriormente, se ha desarrollado "DeskSearX", una herramienta de búsqueda de escritorio que permite a los usuarios buscar y encontrar archivos en su sistema de archivos local. En los siguientes apartados se describirá el desarrollo de la herramienta, los criterios adoptados, los resultados y las conclusiones obtenidas.

2. Desarrollo de la herramienta

La aplicación tiene como punto de entrada la clase `SearchApplication`, aunque también existe una clase `Start` que simplemente llama a la clase `SearchApplication`, se ha utilizado para eludir un problema de ejecución al empaquetar la aplicación.

`SearchApplication` se encarga de inicializar la interfaz de usuario, así como de llamar a las clases que se encargarán de cargar los ajustes de usuarios y validar el índice de búsqueda a partir de la información almacenada en el disco. El proceso de validación y re-indexación puede demorarse unos segundos, por lo que se ha implementado una pantalla de carga que se muestra mientras se realiza esta tarea (Figura 1).

Figura 1: Pantalla de carga.

Una vez la aplicación se ha inicializado por completo, se muestra la interfaz de usuario principal, que se corresponde a la vista de búsqueda. Esta pantalla permite al usuario introducir consultas de búsqueda y visualizar los resultados en una tabla. Esta tabla está compuesta por cuatro columnas: nombre del archivo, ubicación, tipo de archivo y la última fecha de modificación. La Figura 2 muestra la vista de búsqueda con algunos resultados.

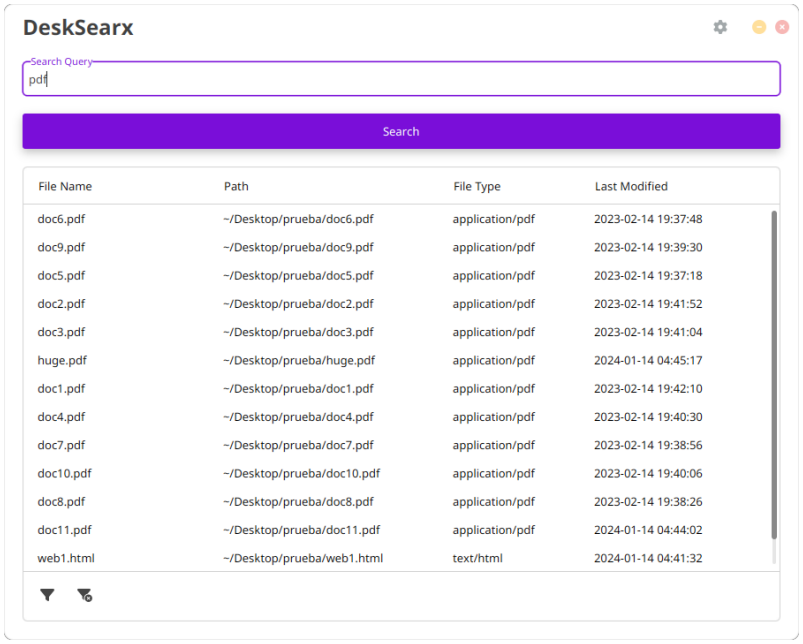

Figura 2: Vista de búsqueda.

La funcionalidad de esta vista está controlada por la clase `SearchController`. Esta clase se encarga de inicializar el modelo de vista para mostrar los resultados de búsqueda, que se corresponden con la clase `Result`, que forma parte del modelo de datos de la aplicación y almacena los datos de cada resultado de búsqueda. En `SearchController` también se añaden los controladores de eventos los distintos elementos de la interfaz de usuario. Para mover y cambiar de tamaño la ventana se usa la clase auxiliar `MoveWindow` que se encarga de gestionar estos eventos, moviendo la ventana cuando se arrastra por la parte superior y cambiando el tamaño al arrastrar por la esquina inferior derecha.

El componente de vista de tabla utilizado es de MaterialFX, y ya incorpora herramientas de filtrado que se pueden utilizar para filtrar los resultados de búsqueda, como se muestra en la Figura 3. También se pueden ordenar los resultados haciendo clic en el encabezado de la columna correspondiente. El usuario también puede abrir los archivos directamente desde la interfaz de la herramienta, haciendo doble clic en la fila correspondiente. Esto abrirá el archivo con la aplicación predeterminada del sistema operativo.

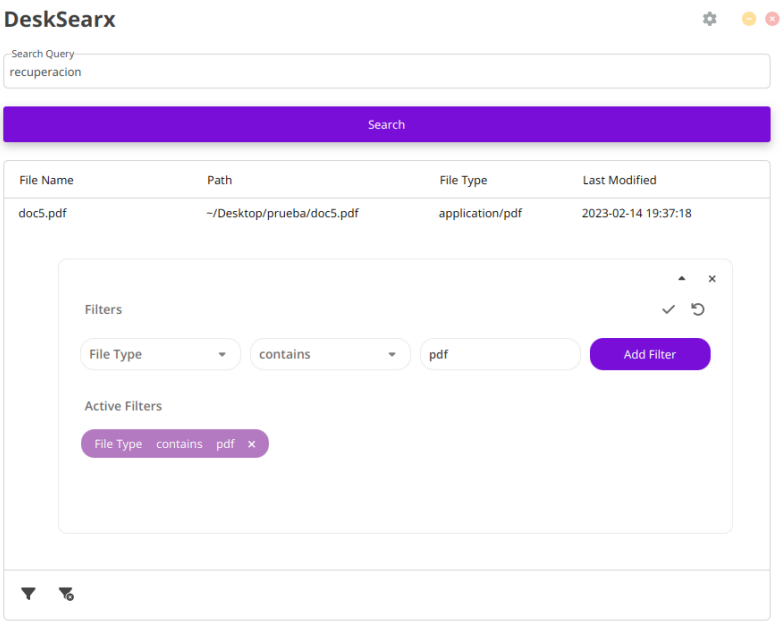

Figura 3: Filtrado de resultados.

Al hacer clic sobre el botón de ajustes de la parte superior, se muestra la vista de ajustes, que permite al usuario especificar qué directorios de su sistema desea indexar entre otras opciones. Esta vista se muestra en la Figura 4. La funcionalidad de esta vista está controlada por la clase `SettingsController`, que se encarga del modelo de vista para mostrar los directorios que se han seleccionado para la indexación, así como de ordenar la re-indexación de contenido cuando sea adecuado. La información de la configuración se almacena en la clase `Settings`, que almacena los directorios seleccionados por el usuario. Para poder acceder a esta información desde cualquier parte de la aplicación, se ha implementado una clase Singleton, `ConfigManager`, que se encarga de gestionar y almacenar la información de la configuración en el disco.

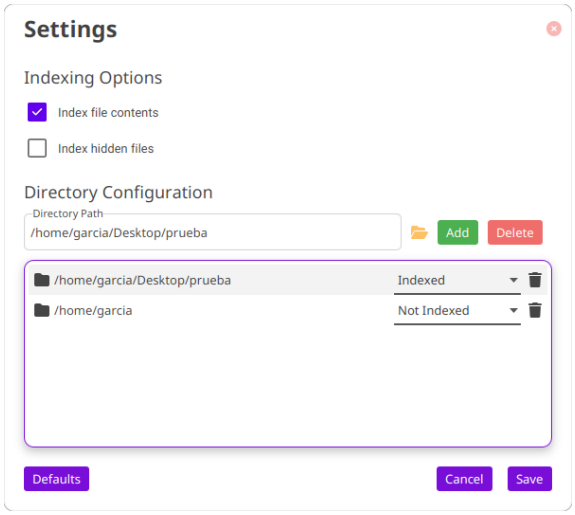

Figura 4: Vista de ajustes.

La clase que realiza todo el trabajo de indexación y búsqueda es `IndexService`. Esta clase implementa todos los métodos necesarios para indexar y buscar en el sistema de archivos local. Todos los métodos relacionados con Apache Lucene se encuentran aquí, por lo que se llamará a esta clase desde distintos puntos de la aplicación, así que también se ha implementado el patrón de diseño Singleton en esta clase. `IndexService` también instancia la clase `FileWatcher`, que se encarga tanto de mantener una lista de los archivos que se han indexado en disco, como de monitorizar en un hilo separado los cambios en el sistema de archivos local, actualizando el índice de búsqueda cuando sea necesario. Para comunicarse con `IndexService`, `FileWatcher` implementa una interfaz, `FileChangeListener`, que se utiliza para notificar a `IndexService` de los cambios en el sistema de archivos.

Por último, la clase `ParseService` se encarga de extraer el contenido de diversos tipos de archivos, utilizando Apache Tika. Esta clase se utiliza para extraer el contenido en texto plano de los archivos que se van a indexar, para detectar el tipo de archivo indexado y para detectar el idioma de tanto el contenido de los archivos, como de la consulta de búsqueda introducida por el usuario.

3. Criterios adoptados

Durante el desarrollo de la herramienta se han tenido en cuenta una serie de criterios para lograr un rendimiento óptimo y unos resultados precisos. Con respecto al soporte multilingüe, se ha decidido solamente ofrecer soporte para los idiomas inglés y español. El razonamiento detrás de esta decisión es que no por ofrecer soporte para más idiomas se va a mejorar la precisión de los resultados, ya que para obtener resultados óptimos tanto el documento indexado como la consulta de búsqueda deben estar en el mismo idioma. Si el idioma detectado para un texto no tiene un factor de confianza superior al 50 %, se utiliza el analizador estándar de Lucene.

Además, aunque existan tanto analizadores en Lucene como detectores en Tika para una gran cantidad de idiomas, el añadir soporte para más idiomas disminuiría la confianza y precisión en la detección del idioma. Por estos motivos, he decidido centrarme en los idiomas que personalmente conozco, para poder realizar pruebas y comprobar que los resultados son precisos. Idealmente, se añadiría un menú en los ajustes de la aplicación para que el usuario pueda seleccionar los idiomas que desea soportar, pero por falta de tiempo no se ha implementado.

También se han decidido soportar todos los formatos de archivos para los que Tika ofrece un *parser*. La lista completa se puede ver en el repositorio del proyecto [1]. Esto incluye formatos de archivos de texto, documentos de Microsoft Office, documentos PDF, documentos web, detección de texto en imágenes, archivos comprimidos, etc. En retrospectiva se podría haber decidido no soportar algunos formatos de archivos, ya que algunos son característicos de archivos de gran tamaño que impactan negativamente en el tiempo de indexación. También se podría haber implementado un menú en los ajustes para que el usuario pueda seleccionar los formatos de archivos que desea soportar.

Otro criterio adoptado es que se ha cambiado el límite máximo de Tika para la extracción de contenido. Por defecto Tika extrae un máximo de 100000 caracteres de cada archivo, pero para este trabajo se ha eliminado este límite por completo, con el objetivo de poner a prueba la capacidad de la herramienta para manejar archivos de gran tamaño. Se ha comprobado que la herramienta es capaz de manejar archivos de más de 50 MB sin problemas, aunque el tiempo de indexación aumenta considerablemente. En un entorno de producción, se podría establecer un límite de 100000 caracteres para la extracción de contenido, ya que la mayoría de los archivos no superan este límite.

Con respecto al índice de búsqueda construido por Lucene, se han decidido implementar los siguientes campos para cada documento indexado:

- **Ruta del archivo**: Un `StringField` que almacena la ruta absoluta del archivo, es imprescindible que sea un `StringField` y se almacene en el índice para poder obtener el valor exacto del campo, ya que se usa para determinar si un archivo ya ha sido indexado.

- **Tipo de archivo**: Otro `StringField` que almacena el tipo de archivo, obtenido a partir de la función `detect` de Tika. Se almacena en el índice para devolverlo junto con el resultado de búsqueda.

- **Contenido del archivo**: Un `TextField` que almacena el contenido del archivo, obtenido a partir de la función `parseToString` de Tika. Es parseado con el analizador correspondiente al idioma detectado para el contenido del archivo. No se almacena el contenido en el índice, ya que no es necesario y aumentaría considerablemente el tamaño del índice.

[1] `https://github.com/apache/tika/tree/main/tika-parsers/tika-parsers-standard/tika-parsers-standard-modules`

Para la búsqueda de documentos, se han construido distintas consultas para cada tipo de campo. Para tanto la ruta del archivo como el tipo de archivo se utiliza una consulta de tipo `FuzzyQuery` junto con otra consulta de tipo `WildcardQuery` que añade un comodín al principio y al final de la consulta de búsqueda introducida por el usuario. Estas consultas tienen como objetivo encontrar el archivo buscado en el caso de que el usuario introduzca el nombre del archivo o parte de él.

Para el contenido del archivo se utiliza una consulta de tipo `QueryParser`, que parsea la consulta de búsqueda introducida por el usuario con el analizador correspondiente al idioma detectado para la consulta del usuario. Esta consulta se utiliza para encontrar los archivos que contienen las palabras introducidas por el usuario en el contenido del archivo. Para combinar las consultas de los distintos campos se utiliza una consulta de tipo `BooleanQuery`, que combina las consultas de los distintos campos con un operador `SHOULD`.

La búsqueda devuelve los 100 documentos más relevantes, ordenados por relevancia. Esto se disminuye del máximo de 1000 que garantiza Lucene para mantener el orden de relevancia, ya que 100 resultados se consideran más que suficientes para una búsqueda de escritorio.

Por último, un criterio adoptado para mantener el tiempo de indexación y tamaño del índice bajo control, es que se ha decidido no indexar recursivamente los directorios seleccionados por el usuario. Esto significa que si el usuario selecciona un directorio para la indexación, solo se indexarán los archivos que se encuentren en ese directorio, no los que se encuentren en los subdirectorios. Y tampoco se vigilan los cambios realizados en los subdirectorios.

4. Resultados

La herramienta desarrollada ha logrado integrar con éxito todas las funcionalidades clave definidas como objetivos. Con un número moderado de directorios y archivos a indexar, la herramienta es capaz de indexar el contenido de los archivos en un tiempo aceptable. Aunque es cierto que cuando el número o tamaño de los archivos a indexar aumenta, el tiempo de indexación puede ser considerable. Esto se podría remediar en un futuro aplicando una heurística para decidir si un archivo debe ser indexado o no, basándose en su tamaño o tipo de archivo. También se podría hacer una implementación multi-hilo para la indexación, ya que actualmente se realiza en un solo hilo.

En cuanto al tiempo de búsqueda, se ha comprobado que es prácticamente instantáneo, incluso con un número muy alto de archivos indexados. Esto demuestra la robustez y eficacia de las estructuras de datos y algoritmos utilizados por Lucene tanto a la hora de construir el índice como a la hora de realizar las búsquedas. Quizás se podría considerar la posibilidad de añadir términos de búsqueda que puedan mejorar aún más la calidad de los resultados, aprovechando la velocidad de búsqueda.

Si bien al inicio del desarrollo los resultados obtenidos no eran totalmente satis-

factorios, especialmente al realizar la búsqueda en español debido a los acentos y caracteres especiales, la mejora al implementar analizadores de idiomas específicos ha sido notable. En la versión final, la calidad de los resultados es muy alta, especialmente si se compara con otras herramientas de búsqueda de escritorio, como la que viene integrada en Windows. La herramienta es capaz de encontrar los archivos buscados en todo tipo de archivos, como se puede ver en la Figura 5, donde se ha buscado la frase "recuperacion de la información" en un directorio de prueba con archivos de texto, PDF, documentos de Microsoft Office, documentos web, imágenes, etc., y se han encontrado todos los archivos que tratan del tema, independientemente del tipo, siendo el resultado más relevante el archivo PDF de la Wikipedia que trata sobre el tema.

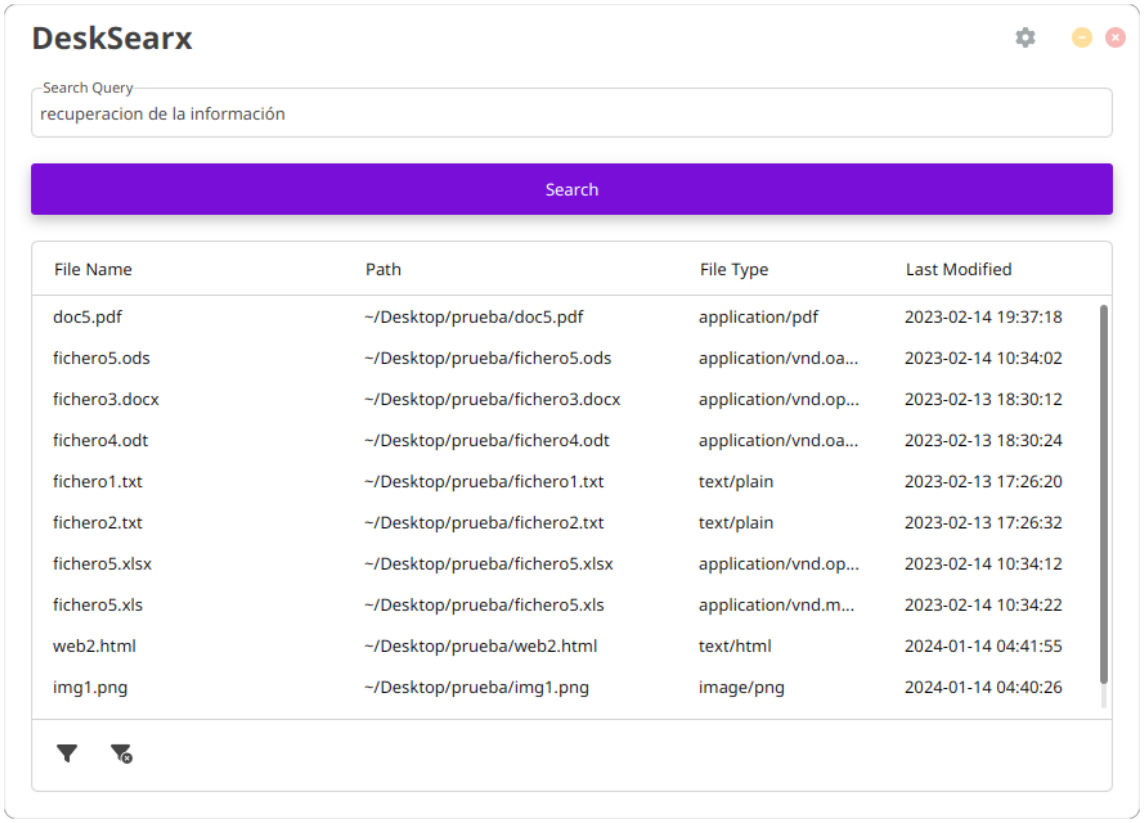

Figura 5: Resultados de búsqueda.

5. Conclusiones

El proceso de desarrollo de la herramienta ha sido muy interesante y enriquecedor. He aprendido sobre el funcionamiento de los sistemas de búsqueda, así como sobre las herramientas y tecnologías utilizadas para su desarrollo, logrando integrar con éxito todas las funcionalidades clave definidas como objetivos. La herramienta, aun siendo mejorable, puede ser utilizada en el día a día para buscar archivos en un conjunto de directorios locales. Me gustaría continuar en el futuro con su desarrollo, añadiendo nuevas funcionalidades y mejorando las ya existentes.

Junto con este informe se ha entregado el código fuente de la herramienta, que puede ser compilado en cualquier sistema operativo con Java 21 ejecutando el comando `./gradlew run`. También se ha entregado una carpeta con el ejecutable de la aplicación para sistemas Linux, localizado en el directorio `desksearx/bin`, pero al ser una aplicación Java, se puede ejecutar en cualquier otro sistema operativo ejecutando el archivo `desksearx-1.0.0-all.jar` localizado en el directorio `desksearx/lib/app`. Se ha comprobado el correcto funcionamiento de la aplicación en un sistema con Fedora 39, otro con Ubuntu 20.04 y en una máquina virtual con Windows 11.

Referencias

[1] Oracle Corporation. GraalVM, 2023. URL: `https://www.graalvm.org/`.

[2] Oracle Corporation. JavaFX, 2023. URL: `https://openjfx.io/`.

[3] FasterXML. Jackson, 2023. URL: `https://github.com/FasterXML/jackson`.

[4] Apache Software Foundation. Apache Lucene, 2023. URL: `https://lucene.apache.org/`.

[5] Apache Software Foundation. Apache Tika, 2023. URL: `https://tika.apache.org/`.

[6] palexdev. MaterialFX, 2023. URL: `https://github.com/palexdev/MaterialFX`.