Diseño Técnico Simplificado

Índice

# 1. Resumen del proyecto

El sistema permite indexar un conjunto de documentos de texto y recuperar información relevante con base en consultas del usuario.

# 2. Alcance

* Se construirá un sistema básico de recuperación de la información.
* Se utilizarán documentos locales con el formato .txt.
* Se hará la búsqueda por una interfaz gráfica de escritorio.

# 3. Requisitos

## 3.1. Requisitos funcionales

* (M) El sistema debe permitir al usuario ingresar una frase de búsqueda a través de una interfaz gráfica.
* (M) El sistema debe recuperar documentos relevantes desde una base de datos local con al menos 20 archivos de texto (.txt).
* (M) El sistema debe aplicar el modelo BM25 para calcular el ranking de relevancia de los documentos.
* (M) El sistema debe mostrar los documentos ordenados según el puntaje del modelo BM25.
* (M) El sistema debe eliminar stopwords.
* (S) El sistema debe construir un índice invertido.
* (C) El sistema debe permitir ver un fragmento del contenido del documento como vista previa en los resultados de búsqueda.

## 3.2. Requisitos no funcionales

* (M) El tiempo de respuesta del sistema al realizar una búsqueda no debe superar los 2 segundos para una base de hasta 100 documentos.
* (M) La interfaz gráfica debe ser intuitiva y permitir realizar búsquedas sin necesidad de conocimientos técnicos.
* (M) El sistema debe ser compatible con Windows y Linux, ejecutándose mediante una aplicación local de escritorio.
* (S) El sistema debe registrar un log de consultas realizadas para propósitos de depuración.
* (C) El sistema debe soportar visualización de resultados en modo claro y oscuro.

# 4. Arquitectura general

El sistema está implementado por una arquitectura monolítica de escritorio, está compuesto por tres componentes principales:

1. **Procesador de Documentos**

* Carga los documentos desde el sistema de archivos.
* Aplica limpieza de texto (remoción de stopwords, tokenización).
* Construye el índice invertido utilizando MapReduce.

1. **Motor de Recuperación**

* Implementa el modelo probabilístico BM25.
* Dada una consulta del usuario, calcula la puntuación de relevancia de cada documento.
* Devuelve los resultados ordenados por relevancia.

1. **Interfaz Gráfica**

* Permite al usuario ingresar una consulta.
* Muestra los documentos rankeados.
* Muestra vista previa opcional del contenido del documento.

A continuación, se presenta el diagrama de alto nivel de estos componentes:

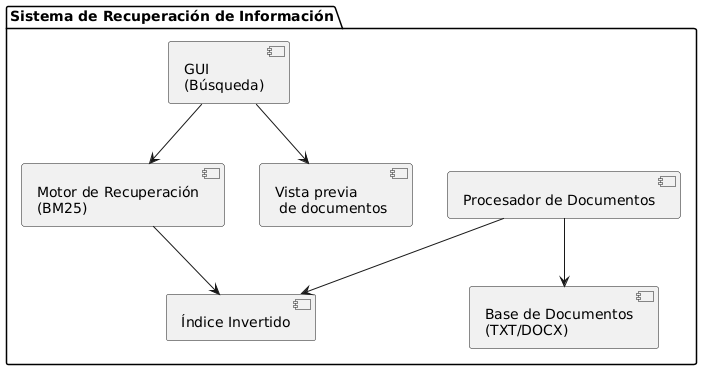


Figura 1. Diagrama de componentes de alto nivel

# 5. Componentes

Lista de los módulos principales, con su propósito.

|  |  |
| --- | --- |
| **Módulo** | **Descripción** |
| crawler.py | Carga y lee documentos desde el disco |
| index\_builder.py | Limpieza y tokenización del texto |
| inverted\_index.py | Construye y almacena el índice invertido |
| query\_processor.py | Procesa la consulta del usuario |
| retrieval\_models.py | Aplica modelo BM25 para cálculo de relevancia |
| ranking.py | Ordena resultados de acuerdo con el puntaje de relevancia |
| cli\_interface.py | Permite ingresar consultas y ver resultados |

Esta organización se puede visualizar de mejor manera en el siguiente diagrama donde se muestra la interacción entre los componentes principales en un sistema de recuperación de información,

Diagrama

El contenido generado por IA puede ser incorrecto.

Figura 2. Diagrama de componentes principales

# 6. Flujo de ejecución

A continuación, se describe paso a paso cómo interactúan los componentes del sistema según el diagrama de la Figura 2, desde que el usuario formula una consulta hasta que se muestran los resultados. Este flujo asume que el índice ya fue construido previamente.

**Consideraciones generales**

* **Construcción del índice**: se realiza una sola vez al inicio o cuando cambia el corpus.
* **Consultas del usuario**: reutilizan el índice invertido ya construido, evitando reprocesar los documentos innecesariamente.
* **No se reconstruye el índice por cada consulta**, lo cual mejora significativamente el tiempo de respuesta.

1. **Inicialización y construcción del índice**

1.1. El **Procesador de Documentos** (crawler.py, index\_builder.py, inverted\_index.py) recorre la carpeta de archivos .txt, extrae el contenido de cada uno, elimina stopwords y tokeniza el texto.

1.2. Con los tokens generados, se construye un **índice invertido**, que almacena, para cada término, la lista de documentos en que aparece, junto con la frecuencia de aparición y la longitud del documento.

1.3. Se calculan estadísticas **corpus-wide** necesarias para el modelo BM25: longitud de cada documento, longitud promedio del corpus, y total de documentos.

1. **Ingreso de la consulta por el usuario**

2.1. El usuario ingresa una frase de búsqueda desde la **Interfaz Gráfica de Escritorio** (cli\_interface.py o equivalente en GUI).

2.2. La **Unidad de Procesamiento de la Consulta** (query\_processor.py) recibe la cadena, aplica el mismo preprocesamiento utilizado para los documentos (minúsculas, eliminación de *stopwords*, tokenización), y genera una lista de términos normalizados.

1. **Recuperación de candidatos**

3.1. El **Motor de Recuperación** (retrieval\_models.py, ranking.py) consulta al índice invertido la lista de documentos que contienen al menos un término de la consulta.

3.2. El índice devuelve un conjunto de candidatos (pares <doc\_id, frecuencia>) para cada término.

1. **Cálculo de puntuaciones con BM25**

4.1. Para cada documento candidato, se calcula el **puntaje de relevancia** usando el modelo probabilístico **BM25**, considerando:

* Frecuencia del término en el documento.
* Longitud del documento.
* Longitud promedio del corpus.
* Total de documentos.

4.2. Los puntajes parciales de cada término se suman para obtener un **score total por documento**.

1. **Ranking de resultados**

5.1. El componente de **Ranking** ordena los documentos según su puntaje BM25, de mayor a menor.

5.2. Se seleccionan los **top-k documentos más relevantes** para mostrarse al usuario.

**🖥️ Presentación de resultados**

6.1. La **Interfaz Gráfica** recibe la lista de documentos ordenados junto con sus puntajes.

6.2. Se muestra para cada resultado:

* Título del documento.
* Puntaje BM25.
* Fragmento de vista previa.

6.3. El usuario puede hacer clic en cualquier resultado para abrir el documento completo mediante un visor externo.

# 7. Pila de tecnologías

# 8. Decisiones técnicas

**¿Por qué se eligió modelo BM25?**

El modelo fue asignado con anterioridad por el docente, el proyecto actual es una tarea que implementa un sistema de recuperación de la información con BM25.

**¿Por qué se usa un índice invertido?**

Por la facilidad en la implementación de este índice, debido al corto tiempo para la realización del proyecto.

**¿Qué librerías se usan para NLP?**

Dado que el proyecto utiliza BM25, entonces solo necesita procesamiento básico como tokenizar y quitar stopwords, por lo tanto, se utilizará únicamente la librería nltk de Python.

# 9. Limitaciones

* No incluye manejo de sinónimos ni expansión de consultas.
* No está optimizado para millones de documentos.
* Funciona únicamente para archivos de texto (.txt).
* Las búsquedas se realizan sobre un directorio simple preestablecido.
* Las búsquedas no están optimizadas para realizarse en gran variedad de idiomas (chino, francés, etc).

# 10. Posibles extensiones

* Aplicar técnicas modernas de NLP con machine learning para permitir sinónimos y expansión de consultas.
* Integrar modelos más eficientes.
* Extender la compatibilidad con mayor variedad de documentos, por ejemplo: PDF, Word, etc.
* Implementar el sistema de tal manera que el usuario pueda elegir un directorio local donde hacer sus búsquedas.
* Ampliar el conjunto de idiomas disponibles para el cual se realiza la búsqueda.
* Optimizar la actualización del índice invertido, para que solo actualice los campos de documentos específicos en lugar de actualizarse en su totalidad.
* Implementar machine learning para el aprendizaje de búsquedas personalizadas de los usuarios.
* Implementar sistema de autocompletado de búsquedas.
* Implementar historial de búsquedas.
* Implementar limpieza de tildes, caracteres especiales.
* Implementar corrección ortográfica automática de términos.
* Separar los componentes en proyectos únicos para que escalen mejor.

# 11. Instalación y uso