Moogle! es una aplicación "totalmente original" cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos.

Es una aplicación web, desarrollada con tecnología .NET Core 6.0, específicamente usando Blazor como "framework" web para la interfaz gráfica, y en el lenguaje de visual stuio code. La aplicación está dividida en dos componentes fundamentales:

- 'MoogleServer' es un servidor web que renderiza la interfaz gráfica y sirve los resultados. - 'MoogleEngine' es una biblioteca de clases donde está... ehem... casi implementada la lógica del algoritmo de búsqueda.

En primer lugar, el usuario puede buscar no solo una palabra sino en general una frase cualquiera. - Si no aparecen todas las palabras de la frase en un documento, pero al menos aparecen algunas, este documento también queremos que sea devuelto, pero con un (score) menor mientras menos palabras aparezcan. - El orden en que aparezcan en el documento los términos del (query) en general no debe importar, ni siquiera que aparezcan en lugares totalmente diferentes del documento. - Si en diferentes documentos aparecen la misma cantidad de palabras de la consulta, (por ejemplo, 2 de las 3 palabras de la consulta "algoritmos de ordenación"), pero uno de ellos contiene una palabra más rara (por ejemplo, "ordenación" es más rara que "algoritmos" porque aparece en menos documentos), el documento con palabras más raras debe tener un ' "score" más alto, porque es una respuesta más específica. - De la misma forma, si un documento tiene más términos de la consulta que otro, en general debería tener un "score" más alto (a menos que sean términos menos relevantes). - Algunas palabras excesivamente comunes como las preposiciones, conjunciones, etc., deberían ser ignoradas por completo ya que aparecerán en la inmensa mayoría de los documentos (esto queremos que se haga de forma automática, o sea, que no haya una lista cableada de palabras a ignorar, sino que se computen de los documentos).

La idea original del proyecto es buscar en un conjunto de archivos de texto (con extensión '.txt') que estén en la carpeta 'Content'

Lo primero que tendrás que hacer para poder trabajar en este proyecto es instalar .NET Core 6.0 . Luego, solo te debes parar en la carpeta del proyecto y ejecutar en la terminal de Linux:

"'bash make dev "'

Si estás en Windows, debes poder hacer lo mismo desde la terminal del WSL (Windows Subsystem for Linux). Si no tienes WSL ni posibilidad de instalarlo, deberías considerar seriamente instalar Linux, pero si de todas formas te empeñas en desarrollar el proyecto en Windows, el comando *ultimate* para ejecutar la aplicación es (desde la carpeta raíz del proyecto):

"'bash dotnet watch run –project MoogleServer "'

* Explicación de mi proyecto

En primer lugar utilicé la función Query, esta función recibe una cadena de búsqueda (query) y devuelve un objeto de tipo SearchResult. Este objeto contiene una lista de resultados de búsqueda (SearchItem) y una cadena de texto vacía. Primero, verifica si la variable booleana estática (cargado) es falsa. Si es así, significa que los archivos de texto aún no se han cargado. Luego den-

tro del condicional, se obtiene un array de rutas de archivos en el directorio (content) utilizando el método (Directory.EnumerateFiles(content).ToArray()), después se inicializa un array llamado (archivos) de objetos (TFIDF) con la misma longitud que el array de rutas de archivos. A continuación, se recorre el array de rutas de archivos y se crea un objeto (TFIDF) correspondiente para cada ruta, que se almacena en el array (archivos). Después de cargar los archivos, la función divide la cadena de búsqueda en palabras individuales, utilizando el espacio como separador, y almacena las palabras en un array llamado (palabrasQuery), luego se inicializa un array de objetos (SearchItem) llamado (items) con la misma longitud que el array (archivos). A continuación, se itera sobre el array (archivos) y se crea un objeto (SearchItem) para cada archivo. El objeto (SearchItem) contiene el nombre del archivo, un fragmento de texto (Snippet) y un puntaje (Score) calculado utilizando la función (Score). Después de crear todos los objetos (SearchItem), se llama a la función (Ordenar) para ordenar los elementos en el array (items) de mayor a menor según el puntaje. Luego, se crea una nueva lista de objetos (SearchItem) llamada (arr). Luego se itera sobre el array (items) y se agregan los objetos (SearchItem) que tienen un puntaje diferente de cero a la lista (arr).Después se copian los elementos de la lista (arr) al array (answer). Y finalmente, se crea un nuevo objeto (SearchResult) con el array (answer) y una cadena de texto vacía como argumentos y se devuelve como resultado de la función (Query). En resumen, este código realiza una búsqueda utilizando una implementación del algoritmo de Ranking TF-IDF. Carga los archivos de texto una vez (si no se han cargado) y calcula el puntaje de cada archivo según la similitud con la consulta de búsqueda. Luego, ordena los resultados por puntaje y devuelve una lista de objetos ‘SearchItem‘ que contienen información relevante del archivo correspondiente a la búsqueda.