¿Qué es Moogle!?
"Moogle!" es una aplicación web cuyo objetivo es realizar búsquedas en una base de datos de archivos .txt y devolver los resultados en el menor tiempo posible. Para esto, se utilizan algoritmos de TF-IDF y otros conocimientos de álgebra lineal que hacen más eficiente el buscador.

¿Cómo ejecutar el proyecto?

En la carpeta raíz del proyecto, ejecute el comando ´dotnet watch run --project MoogleServer´. Este comando inicializará el servidor local de la aplicación. Luego (en caso de que de forma automática no se haya lanzado) abra un navegador y ingrese la dirección 'http://localhost:5285/' para ver el contenido de la web

¿Cómo funciona?

El objetivo de Moogle! es realizar búsquedas en el interior de varios archivos .txt y en función del contenido de los mismos, mostrar los resultados más relevantes de acuerdo a la búsqueda que usted haya realizado. Para esto, usted debe copiar los archivos .txt a los cuales quiera realizarle la búsqueda en la carpeta Content que aparece en la raiz del proyecto. La cantidad mínima de archivos .txt que el proyecto debe tener en la carpeta Content para funcionar de manera correcta es de 2 archivos. Los cuales ya se encuentran en dicha carpeta. (Siéntase libre de borrarlos y copiar sus propios archivos .txt, el código está preparado para trabajar con cualquier archivo .txt que usted provea, mientras la cantidad mínima de estos sean 2). En cuanto a la cantidad máxima no debería tener ningún problema.

Desde el terminal, una vez ejecutado e proyecto, podrá ver este proceso de carga de la base de datos. El cual no debería demorar mas de unos pocos segundos. Claro, todo dependerá de la cantidad de archivos .txt que usted tenga en la carpeta Content. Pero para una base de datos de 50 mb de archivos .txt (alrededor de 30 libros con miles de palabras) el algoritmo solo demora entre 4 a 6 segundos en cargar estos. La carga de la base de datos solo se realiza en la primera búsqueda que usted realice, luego se almacena en memoria, por lo que esta primera búsqueda demorará unos pocos segundos más que las siguientes que usted realice.

Estructura de clases del proyecto

Clase txtData
La clase txtData tiene como objetivo cargar los datos de los archivos de texto en una ruta específica. La clase tiene un constructor que toma una ruta como parámetro y utiliza esta ruta para obtener todas las rutas de archivos en la carpeta. Luego, filtra las rutas para obtener solo las rutas de archivos de texto y las almacena en un array llamado Paths. La clase también tiene un array llamado Texts que contiene todo el texto de los archivos de

texto en la ruta especificada. Además, tiene un array llamado Names que contiene los nombres de los archivos de texto sin la extensión ".txt".
La clase también tiene un método llamado GetPaths que devuelve el array de rutas actualizada, un método llamado GetNames que devuelve los nombres y un método llamado GetTexts que devuelve todos los textos en los archivos de texto.

Clase tfidf
La clase llamada tfidf se utiliza para calcular el valor TF-IDF de las palabras en los documentos y la similitud entre los documentos. La clase tiene varios diccionarios y arreglos que se utilizan para almacenar los valores de TF, IDF, TF-IDF y similitud.
Los diccionarios Name_Words_TF, Name_Words_IDF y Name_Words_TFIDF se utilizan para almacenar los valores de TF, IDF y TF-IDF para cada palabra en cada documento. El diccionario NameVSWorsd se utiliza para almacenar las palabras en cada documento, mientras que el diccionario NameVSUnrepeatedWords se utiliza para almacenar las palabras únicas en cada documento. El arreglo Names se utiliza para almacenar los nombres de los documentos.
Los diccionarios Query_TF, Query_IDF y Query_TFIDF se utilizan para almacenar los valores de TF, IDF y TF-IDF para cada palabra en la consulta. La propiedad Query se utiliza para almacenar la consulta, mientras que la propiedad QueryWords se utiliza para almacenar las palabras en la consulta.
Los diccionarios Documents_Similitude y Documents_SimilitudeCos se utilizan para almacenar la similitud entre los documentos. El diccionario Results se utiliza para almacenar los resultados de la búsqueda, mientras que el diccionario Snippet se utiliza para almacenar un fragmento de texto de cada documento que contiene la consulta.
El constructor de la clase toma los diccionarios y arreglos mencionados anteriormente como parámetros. El método Counter cuenta la cantidad de veces que una palabra aparece en un arreglo de palabras. Y por último el método WhereExist cuenta la cantidad de documentos en los que aparece una palabra.

Clase SearchResult
La clase SearchResult tiene como objetivo representar los resultados de una búsqueda. Tiene un constructor que toma un arreglo de objetos SearchItem y una sugerencia de búsqueda como parámetros. Si el arreglo de SearchItem es nulo, se lanza una excepción. La clase también tiene un constructor sin parámetros que crea un arreglo vacío de SearchItem.
La clase tiene una propiedad de solo lectura llamada Suggestion que almacena la sugerencia de búsqueda. Además tiene un método llamado Items que devuelve una enumeración de los objetos SearchItem en el arreglo items. Y por último tiene una propiedad de solo lectura llamada Count que devuelve la cantidad de objetos SearchItem en el arreglo items.

Clase SearchItem
La clase SearchItem representa un elemento de búsqueda. Tiene un constructor que toma tres parámetros: el título del elemento, un fragmento de texto que contiene la consulta y la puntuación del elemento. Tiene tres propiedades de solo lectura: Title, Snippet y Score,

que almacenan el título del elemento, el fragmento de texto que contiene la consulta y la puntuación del elemento, respectivamente.

Clase Moogle

La clase `Moogle` es una clase estática que contiene dos métodos: `Cargar` y `Query`. El método `Cargar` carga los datos de la base de datos y devuelve un diccionario que contiene los resultados de la búsqueda. El método `Query` realiza una búsqueda en la base de datos y devuelve los resultados en forma de objeto `SearchResult`.

La clase tiene una variable estática llamada `DocumentvsSnipet` que almacena un diccionario que relaciona los nombres de los documentos con un fragmento de texto que contiene la consulta. También tiene una variable estática llamada `objeto1` que almacena una instancia de la clase `txtData`.

El método `Cargar` utiliza la instancia de `txtData` para cargar los datos de la base de datos y crear una instancia de la clase `tfidf`. Luego, utiliza la instancia de `tfidf` para calcular el TFIDF de los documentos y la similitud coseno entre los documentos y la consulta. Finalmente, agrega los snippets de los documentos a `DocumentvsSnipet` y devuelve un diccionario que contiene los resultados de la búsqueda.

El método `Query` utiliza el método `Cargar` para cargar los resultados de la búsqueda en un diccionario y los ordena por score. Luego, crea una lista de objetos `SearchItem` para almacenar los resultados y los agrega a la lista. Finalmente, convierte la lista de objetos `SearchItem` en un arreglo de objetos `SearchItem` y devuelve un objeto `SearchResult` que contiene los resultados de la búsqueda.