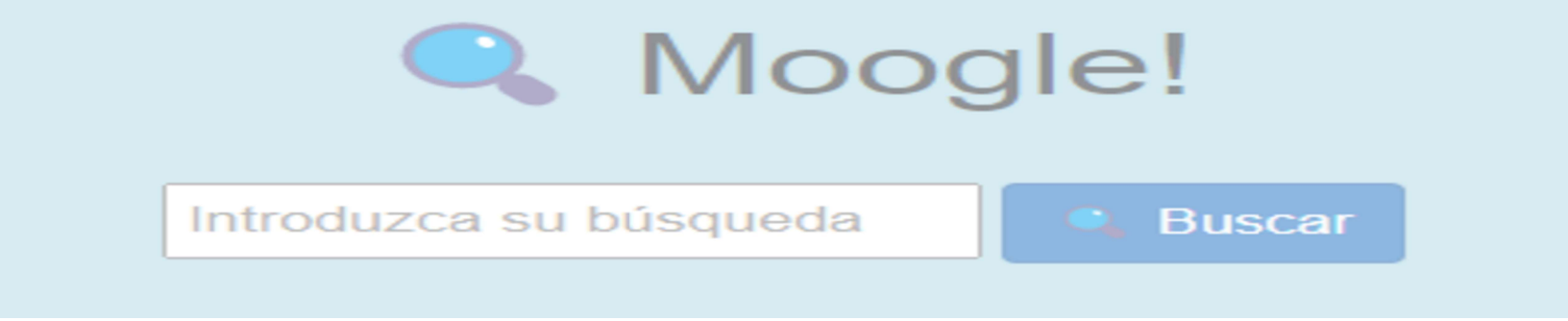


Instrucciones:

Nota: La raíz donde se pondrán todos los documentos .txt es la siguiente "moogle\MoogleServer\Content" verifique que este copiando los documentos en esta ruta.

Ejecución:

➔ Si vas a ejecutar el proyecto en **VS code** sigue estos pasos:

1. Abre Visual Studio Code y selecciona "Archivo" > "Abrir carpeta" para abrir la carpeta del proyecto.

2. Instala la extensión de C#: Si aún no lo has hecho, debes instalar la extensión de C# para Visual Studio Code. Puedes hacerlo seleccionando "Extensiones" en el menú lateral, buscando "C#" y seleccionando "Instalar".

3. Configura el entorno de ejecución: Si el proyecto requiere una configuración específica, como un archivo de configuración o una variable de entorno, asegúrate de configurar el entorno adecuadamente antes de ejecutar el proyecto.

4. Ejecuta el proyecto: Una vez que hayas instalado la extensión de C# y configurado el entorno adecuadamente, puedes ejecutar el proyecto utilizando la terminal integrada de Visual Studio Code. Abre la terminal en Visual Studio Code seleccionando "Terminal" > "Nueva terminal" en el menú. Luego, ejecuta el siguiente código desde la terminal: dotnet watch run --project MoogleServer

➔ Si vas a ejecutar el proyecto en **Visual Studio IDE** sigue estos pasos:

1. Abre Visual Studio y selecciona "Archivo" > "Abrir" > "Proyecto/Solución" para abrir el archivo de proyecto de C#.

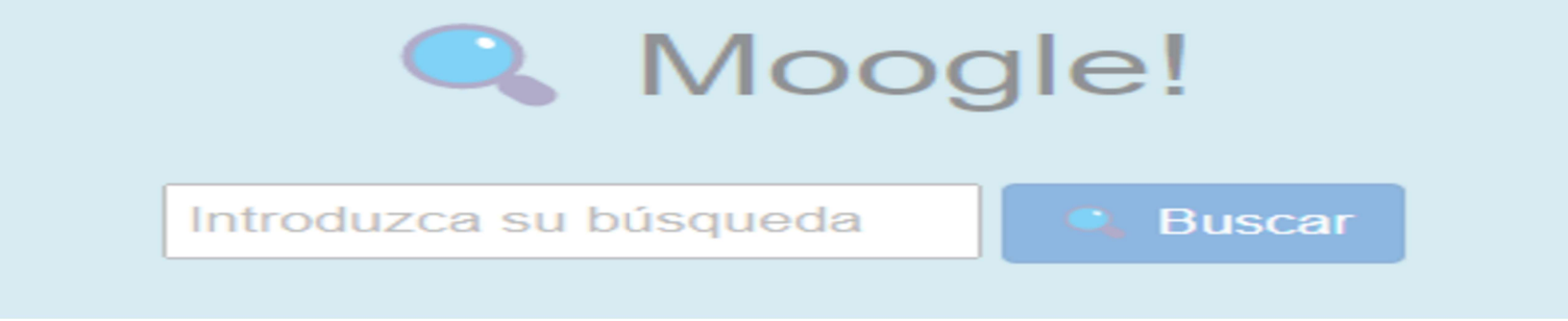
2. Antes de ejecutar el proyecto, debes compilarlo para asegurarte de que no hay errores de compilación. Puedes compilar el proyecto seleccionando "Compilar" > "Compilar solución" desde el menú.

3. Una vez que hayas compilado el proyecto y configurado el proyecto de inicio adecuado, puedes ejecutar el proyecto seleccionando "Depurar" > "Iniciar depuración" desde el menú. Esto compilará el proyecto si es necesario y lo ejecutará en el depurador de Visual Studio.

Fase prueba:

A) **Introducción de un término**:

1. Para buscar introduzca algún o algunos términos en la caja de texto que tiene. Se asegura de que si no se inserta nada no se realiza ninguna búsqueda y se retorna una alerta.



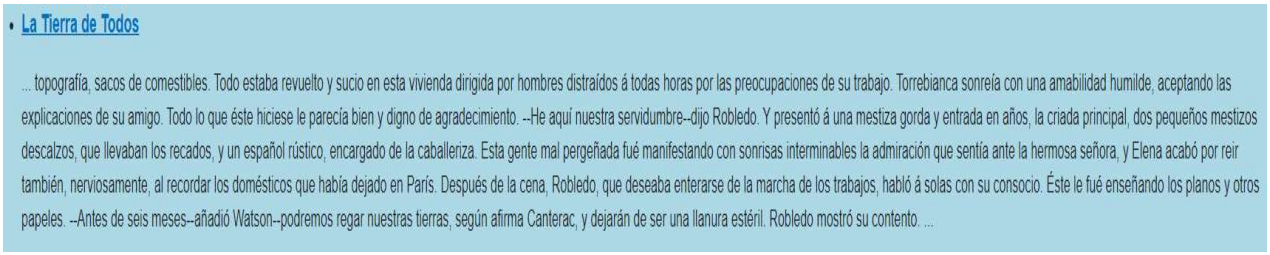
2. Cualquier término que inserte puede ser escrito con tildes o sin ellas, con mayúsculas o minúsculas, esto no afectará la búsqueda de esa palabra.

- **Retornando documentos:**

1. Una vez introducido el término o los términos haga click en el **botón buscar** y se mostrará por orden de relevancia los documentos de mayor puntuación a las menores puntuaciones.

2. Si ninguno de los términos introducidos no están contenidos en ningún documento de texto de nuestra base de datos, entonces el retorno será **nulo**.

3. Los resultados serán devueltos con el siguiente modelo, **Título** {...**Snippet**...}



4. Se garantiza para futura implementación que cada título de cada documento mostrado tenga un hipervínculo para acceder en caso de tener un **servidor local** montado.

5. Para garantizar una imagen limpia en el programa se decide mostrar los **mejores cinco documentos**, en caso de no haber cinco se garantiza que se muestren solo los que tuvieron relevancia.

6. Dese cuenta que el **Snippet** mostrado de cada documento es aleatorio, para probarlo solo basta con presionar de nuevo el botón buscar con el mismo término y notará que el texto contenido por cada documento mostrado cambiará.

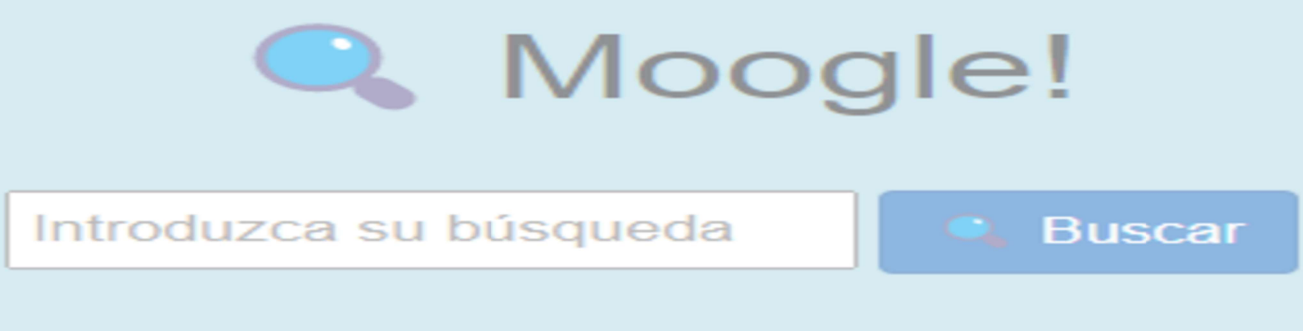
7. Este procedimiento de búsqueda lo puede repetir indefinidamente y el **tiempo de espera** entre búsquedas es lo suficientemente rápido para garantizar una buena experiencia.

Implementación:

- **Clases utilizadas:**

1. class **Indexer**: Esta clase contiene el cuerpo principal de la base de datos dada, algunos de sus principales métodos son:

   a) El método `private void ReturnWords(string[] FilesPath)` es el encargado de hacer una lista de diccionarios donde se almacenan todas las palabras sin repetir por documento de todos los documentos.



Nota: Este método aprovecha la **iteración principal** para calcular el **TF** de cada palabra de un documento en cada ciclo de la iteración principal, y dentro de la **iteración secundaria** se aprovecha para ir haciendo **DF** de cada palabra de la siguiente manera:

```
for(int j = 0; j < wordsContent.Length; j++)
{
    if(WordsInFiles[i].ContainsKey(wordsContent[j]))
    {
        WordsInFiles[i][wordsContent[j]]++;
    }
    else
    {
        WordsInFiles[i].Add(wordsContent[j], 1);
        //Aprovechamos para adicionar la palabra a la lista de palabras sin repetir en un diccionario
        if(!(DocumentFrequency.ContainsKey(wordsContent[j])))
        {
            DocumentFrequency.Add(wordsContent[j], 1);
        }
        //En el caso que la palabra este ya en el diccionario le adicionamos 1 al value, y vamos obteniendo
        // la cantidad de documentos en que aparece la palabra.
        else{
            DocumentFrequency[wordsContent[j]]++;
        }
    }
}
```

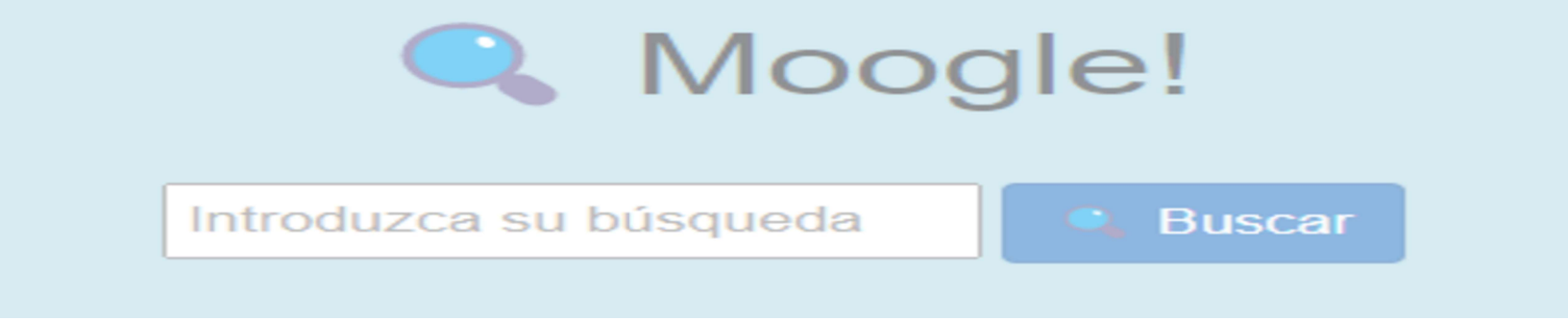
b) El método `public static string NormalizeTheText(string fileText)` normaliza el texto que recibe quitando los signos de puntuación y se declara estático para que este pueda ser usado fuera de la clase, por ejemplo en el query del usuario.

c) Este método `public static string GetDocumentSnippet(string file, int snippetLength = cant_lineas_fragmento)` toma un **fragmento aleatorio** del documento y lo devuelve para que luego pueda ser mostrado como **Snippet**.

2. class **Vector**: Esta clase obtiene el **score final** de los vectores de los Documentos por el vector Query y algunos de sus principales métodos son:

a) El método `private double VectorDotProduct(double[] vectorA, double[] vectorB)` realiza el producto punto entre dos vectores y devuelve el valor expresado en double. Se asegura que los vectores tengan el mismo tamaño y no sean nulos.

b) El método `private double VectorMagnitudeProduct(double[] vectorA, double[] vectorB)` calcula el producto de la magnitud entre dos vectores y devuelve el valor expresado en double. Se asegura que los vectores tengan el mismo tamaño y no sean nulos.



c) El método `public static int[] GetTopFiveDocumentScore(double[] documentScore)` Este método obtiene los cinco documentos con mayor relevancia en cuanto al score, se asegura que en caso de que la cantidad de documentos en la base de datos es menor que cinco se cree un array de tamaño del mínimo entre documentScore.Lenght o 5, asegurando que no se salga de los límites.

Muchas Gracias!