Mi Proyecto se divide en dos partes fundamentales:

* Creación de la base de datos (TF-IDF de los documentos)
* Realización de la búsqueda

**Creación de la base de datos:**

Esta parte del código se ejecuta cuando se inicia el proyecto y solo ocurre una vez. Cuando inicializa el proyecto se instancia una clase llamada *GlobalVariables* en el archivo Program.cs, la cual se encarga de inicializar todas las variables q se van a utilizar durante la ejecución del programa.

Para no afectar la creación del servidor utilicé *System.Thread* para separar la creación de la base de datos del hilo principal, esto se ejecuta cuando llamo a inicializar a *GlobalVariables*. Por tanto, en lo que carga el proyecto se ejecuta a la vez mi método Start () que se encuentra en SearchMethod.cs.

(\*la variable global path existe porque tenía distintas carpetas con diferentes bases de datos para realizar comprobaciones de funcionalidad)

Start () se encarga de llamar a:

* SetFilesNames (): Se encarga de acceder a la carpeta predefinida y se obtiene la dirección de todos los documentos y sus nombres con su respectiva extensión.
* ReadInside (): Se encarga de leer el contenido de cada documento y su título. La lectura la realiza caracter por caracter para facilitar el procesado de las palabras (eliminar tildes, símbolos y mayúsculas). Luego se guarda la palabra procesada y la original en listas, la primera para obtener un vocabulario (es en donde almaceno todas las palabras sin repetición), además de en una lista para obtener el índice de las palabras más adelante y en un diccionario creado por documento en donde se va a guardar la cantidad de veces que aparece en él. La palabra original se guarda en una lista para obtener el snipped.
* Calculate\_TF\_IDF (): Se encarga de calcular el tf\_idf de todos los documentos. Como en los diccionarios ya está guardada la cantidad de veces q aparece la palabra en cada documento (TF), ese valor lo dividimos entre el mayor TF que aparece en el mismo para normalizar los valores. Luego obtenemos en cuantos documentos aparece la palabra y calculamos el logaritmo de la división de la cantidad de documentos total entre la cantidad de documentos en los q aparece la palabra (IDF). Realiza la operación para cada palabra en todos los documentos y para cada título, luego los valores (solo los de los documentos) los multiplica por una constante menor que 1 para disminuir su valor. Después verifica que si una palabra existe en el título del documento le reescribe su valor del cuerpo del documento (para darle mayor importancia al documento si la palabra buscada aparece en su título).
* CreateMatrix (): Mueve todos los tf\_idf a un array de arrays (filas = cantidad de documentos, columnas = cantidad de palabras en mi vocabulario) para facilitar su posterior uso.

**Realización de la búsqueda:**

Esta parte del código se ejecuta una vez que se pida realizar una búsqueda.

Al tocar el botón de búsqueda (o solamente con presionar “ENTER” 😎👌) primero se espera a que haya terminado la creación de la base de datos con Thread.Join() para evitar posibles problemas, luego se llama a Moogle.Query (), lugar donde se procesa el resultado de la búsqueda para q sea mostrado en la página. En este método se hace el llamado a SearchMethod.MakeQuery () que es en donde se realiza la búsqueda en sí.

Primero se eliminan espacios innecesarios introducidos por el usuario, se obtienen las posiciones de los operadores que modifican la búsqueda y el query limpio (solamente las palabras q lo conforman).

Luego se añaden los sinónimos de cada palabra de la query en el mismo con AddSynonyms () para que aparezcan en la búsqueda.

(\*Tengo dos Stemmers, uno en español y otro en inglés, el idioma predeterminado es inglés, pero se puede cambiar modificando el enum q está en *GlobalVariables* llamado language)

AddSynonyms (): Primero saca la raíz de la palabra y busco en mi base de datos de sinónimos palabras cuya raíz la contenga (o viceversa) y las agrego en mi query.

Luego busco una sugerencia a mi query con SuggestQuery () y le elimino las palabras que no están en mi vocabulario con ChangeQuery ().

SuggestQuery (): Obtiene todas las palabras similares a las que no aparezcan y literalmente crea todas las variaciones posibles de query con esas palabras y devuelve la variación que obtuvo un resultado con la mayor similitud a un documento, garantizando una búsqueda la cual es muy precisa.

ChangeQuery (): Elimina todas las palabras del query q no estén en el vocabulario.

Al terminar estas operaciones entonces se calcula el TF\_IDF del query limpio con Calculate\_TF\_IDF\_Query () el cual realiza prácticamente las mismas operaciones que Calculate\_TF\_IDF (). Luego lleva sus valores a un array para realizar su multiplicación con los arrays que conforman los documentos. Para esto creé una clase llamada Vector en la cual redefiní su operador multiplicación para realizar una multiplicación de vectores en GetCoisineSim (). Luego organiza los resultados y elimina a los que tengan un valor menor a 0,001 y guarda sus nombres.

Con los resultados ya obtenidos se aplican los operadores de búsqueda.

“!”: elimina los resultados que tengan a la palabra.

“^”: elimina los resultados q no contengan la palabra

“~”: dependiendo de cuan cercanas estén las palabras le aumenta su tf\_idf en cada documento y luego vuelve a llamar a MakeQuery () con sus valores modificados (el código funciona trabajando con pares de palabras).

“\*”: Se pasa como parámetro a Calculate\_TF\_IDF\_Query () y le aumenta en 1 a la cantidad de veces que aparece en el query.

El método MakeQuery () devuelve los archivos , el snipped de cada uno (con SearchSnipped ()), y la sugerencia.

SearchSnipped (): Obtiene la palabra más valiosa (mayor TF\_IDF) por documento con GetMostValueWord () y devuelve un fragmento donde está la primera vez que aparece la palabra.

\*(En Tools.cs aparecen métodos q no están usándose porque fueron creados durante el transcurso del desarrollo del proyecto, pero luego se encontró otras alternativas y por motivos de poder necesitarlos luego no se eliminaron)

**Extras:**

En Index.razor se implementó la opción de que al tocar la sugerencia se cambie el query por esta y realice su búsqueda.

Se implementó que al tocar el enlace de cualquier documento devuelvo en la búsqueda se cargue una nueva pagina con un botón, el cual al accionarlo ejecute el documento con el lector predeterminado del ordenador