Introducción

SmartWord! es una aplicación web desarrollada con tecnología .NET Core 6.0 utilizando Blazor como framework y en lenguaje C#.

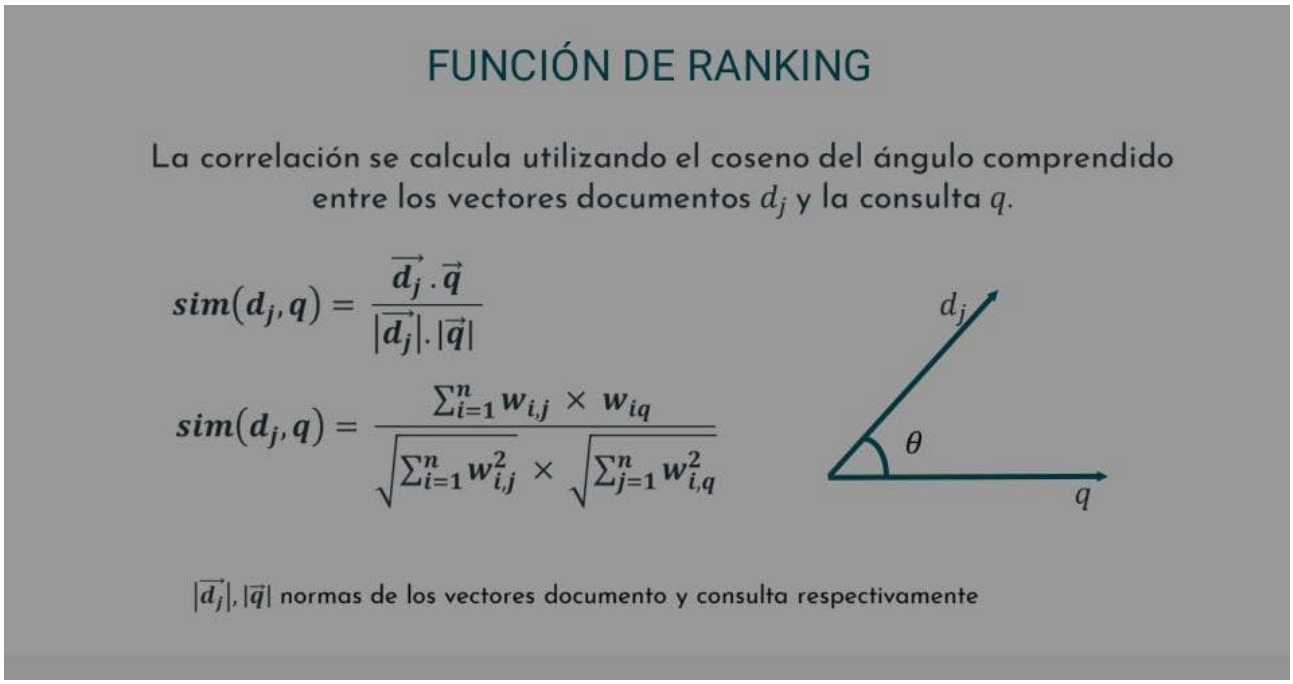
-Estructura del Proyecto

*MoogleEngine: Es una biblioteca de clases donde se encuentra la lógica asociada a la búsqueda.
*MoogleServer: Es un servidor web que renderiza la interfaz gráfica y sirve los resultados.

Desarrollo

-Métodos de Búsqueda

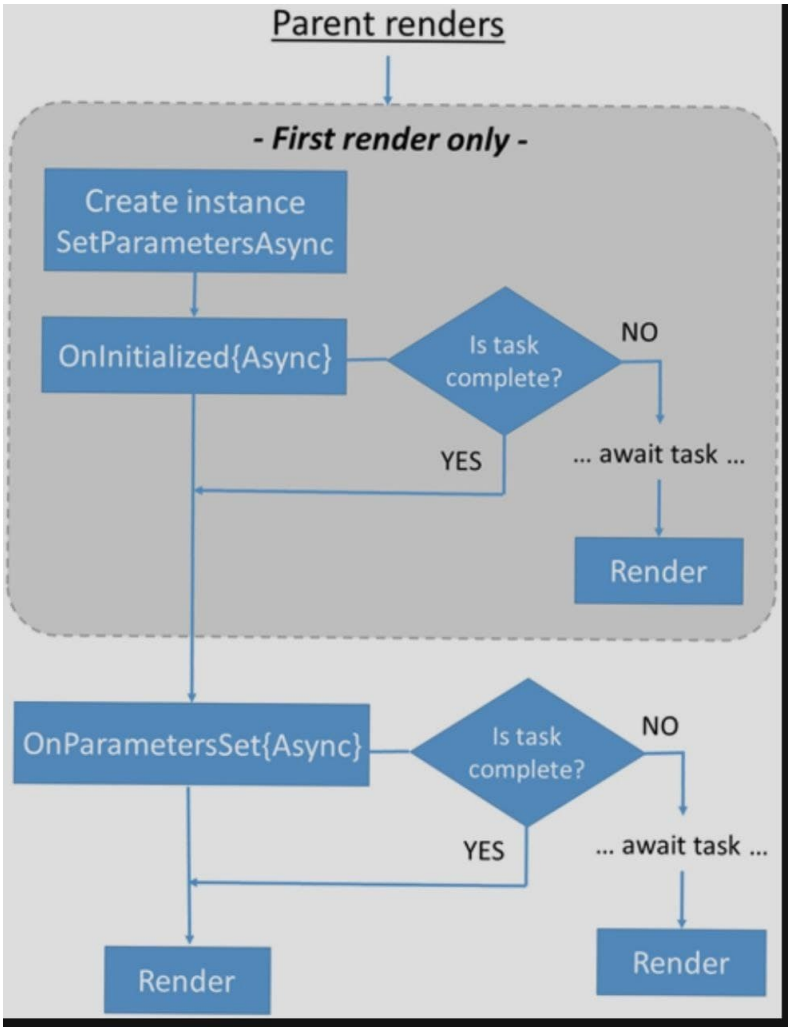
Se realiza la búsqueda introduciendo una frase o palabra en el campo de texto de la pagina principal de SmartWord!
SmartWord! consta de un método DevelopWord el cual se encarga de devolver la mejor sugerencia posible de la cadena de caracteres introducida por el usuario. El mismo recibe un método denominado Levenshtein que se enfoca en devolver el numero mínimo de operaciones requeridas que se necesita hacerle a una cadena de caracteres para que esta se transforme en alguna que se encuentre en los documentos existentes.
Esta aplicación realiza una búsqueda utilizando lo denominado TF(Term-Frequency[Frecuencia del Término]) e IDF(Inverse-Document-Frequency[Frecuencia del Documento Inverso]) basado en un "modelo vectorial", el cual brinda un ranking de los documentos de acuerdo con su grado de similitud de consulta. Además para la recuperación de los documentos puede establecerse un umbral de similitud y recuperar los documentos cuyo grado de similitud sea mayor que este umbral.



Breve ejemplo de la Función Ranking

-Interfaz Gráfica

Para la interfaz grafica de esta aplicación se utilizó Blazor como framework. En la misma se hizo que la página aceptara dos rutas: la primera sin parámetros y la siguiente con parámetros, para ello utilizamos la directiva @page.
En resumen se modificó el ciclo de vida o ciclo de aplicación de la página cambiando los eventos OnInitialized y OnParameterSet, OnInitialized se aplica en el primer Render y luego se llama al evento OnParameterSet como se puede observar en la siguiente imagen.



En estos eventos se obtuvo un problema el cual fue que el evento OnInitialized no se llamaba si ya la ruta estaba y para engañar eso se realizo un hack a una variable Counter que va aumentando y haciendo que la ruta tenga siempre un parámetro distinto, luego esto se introdujo en el href del link.

Clases Creadas
MoogleEngine.Constant: Guarda las constates a utilizar en la aplicación.
MoogleEngine.Corrector: Si existe algún error introducido por el usuario esta se encarga de corregirlo.
MoogleEngine.Document: Guarda todos los datos de los documentos.
MoogleEngine.MyMatrix: Guarda los datos necesarios de los documentos en una matriz.
MoogleEngine.QueryDocument: Guarda todos los datos de la cadena de caracteres introducida por el usuario.
MoogleEngine.Vector: Guarda los datos necesarios de la cadena de caracteres introducida por el usuario en una matriz.

Flujo del algoritmo de Levenshtein:

Este algoritmo también es conocido como distancia de edición. La similaridad entre dos cadenas de texto h1 y h2 se basa en el conjunto mínimo de operaciones de edición necesarias para transformar h1 en h2, o viceversa. Hay tres operaciones de edición, las cuales son destrucción, inserción y substitución. Entre más cerca de cero es la distancia de Levenshtein más parecidas son las hileras.

Ejemplo:

	0	1	2	3	4
0					
1					
2					
3					
4					

	0	1	2	3	4
0		C	O	M	A
1	C				
2	E				
3	N				
4	A				

	0	1	2	3	4
0		C	O	M	A
1	C	0	1	1	1
2	E	1	1	1	1
3	N	1	1	1	1
4	A	1	1	1	0

	0	1	2	3	4
0		C	O	M	A
1	C	0	1	2	3
2	E	1	1	1	1
3	N	1	1	1	1
4	A	1	1	1	0

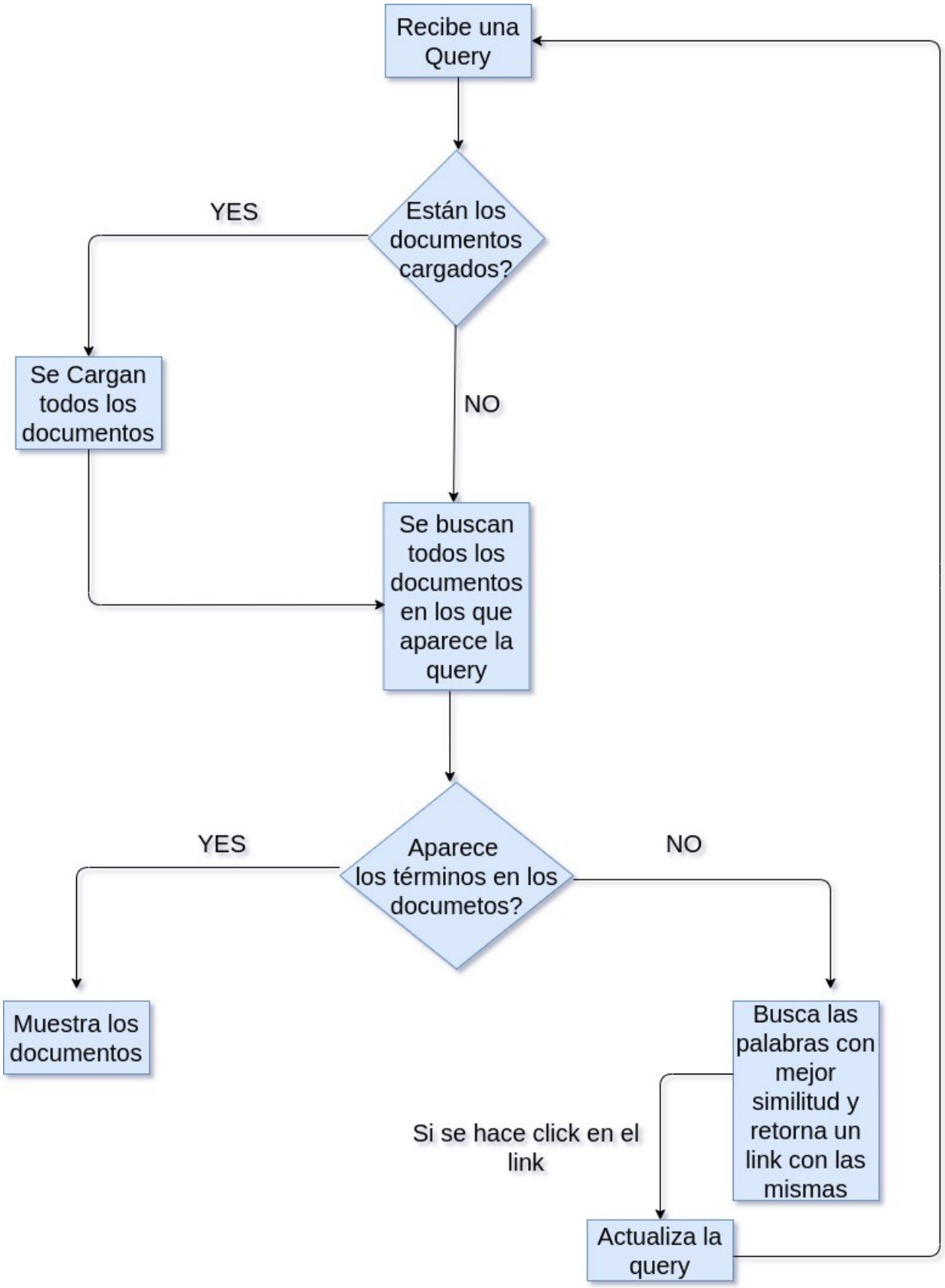
	0	1	2	3	4
0		C	O	M	A
1	C	0	1	2	3
2	E	1	1	2	3
3	N	1	1	1	1
4	A	1	1	1	0

	0	1	2	3	4
0		C	O	M	A
1	C	0	1	2	3
2	E	1	1	2	3
3	N	2	2	2	3
4	A	1	1	1	0

	0	1	2	3	4
0		C	O	M	A
1	C	0	1	2	3
2	E	1	1	2	3
3	N	2	2	2	3
4	A	3	3	3	2

	0	1	2	3	4
0		C	O	M	A
1	C	0	1	2	3
2	E	1	1	2	3
3	N	2	2	2	3
4	A	3	3	3	**2** ← *LD(h1,h2)*

Flujo del Programa:



Problemas Detectados

Después de varias pruebas realizados en el proyecto a tratar se pudieron encontrar varias dificultades en el mismo como que era imposible realizar la búsqueda de mas de una cierta cantidad de palabras esto era debido a que cada vez que se tenia que realizar la búsqueda de una palabra o frase el mismo cargaba todas los textos nuevamente lo cual hacía que la velocidad del programa fuera demasiada lenta.

-Tras la necesidad de contar la cantidad de documentos en que se encontraba la palabra a buscar se creó un contador pero este siempre se estaba revisando por todos los documentos sin obviar los que ya había revisado

Nuevas Implementaciones

Acciones realizadas para contrarrestar dicho problema

-Se guardaron todos los datos necesarios y no modificables una solo vez(o sea esta acción se ejecutaba solo si no teníamos ningún dato guardado).

-Se creo una mascara booleana la cual controlaba toda la búsqueda de palabra o frase por todos los documentos.

-Se realizó el operador de búsqueda “!” el cual indica que la palabra no puede existir en ningún documento.

-Se realizó el operador de búsqueda “ ^ ” el cual indica que la palabra tiene que existir en los documentos.

-Se realizó el operador de búsqueda “ * ” el cual aumenta el score del documento en que se encuentra la palabra a que se hace referencia

Bibliografía:

-Wikipedia para la implementación del algoritmo de Levenshtein:
https://es.wikipedia.org/wiki/Distancia_de_Levenshtein
-Pagina oficial de Microsoft para varias tareas entre ellas el trabajo con el framework Blazor:
https://docs.microsoft.com