# Manuel Alejandro Rodríguez Peñaranda Grupo: C-121

Moogle es una aplicación web, desarrollada con tecnología .NET Core 6.0, específicamente usando Blazor como framework web para la interfaz gráfica y el lenguaje C#. La misma tiene como propósito la búsqueda inteligente de términos en un conjunto de documentos de textos. La aplicación está dividida en dos componentes fundamentales:

* MoogleServer es un servidor web que renderiza la interfaz gráfica y sirve los resultados a mostrar.
* MoogleEngine es una biblioteca de clases donde están encapsulados, los métodos y procedimientos necesarios (lógica de la aplicación) para realizar la búsqueda.

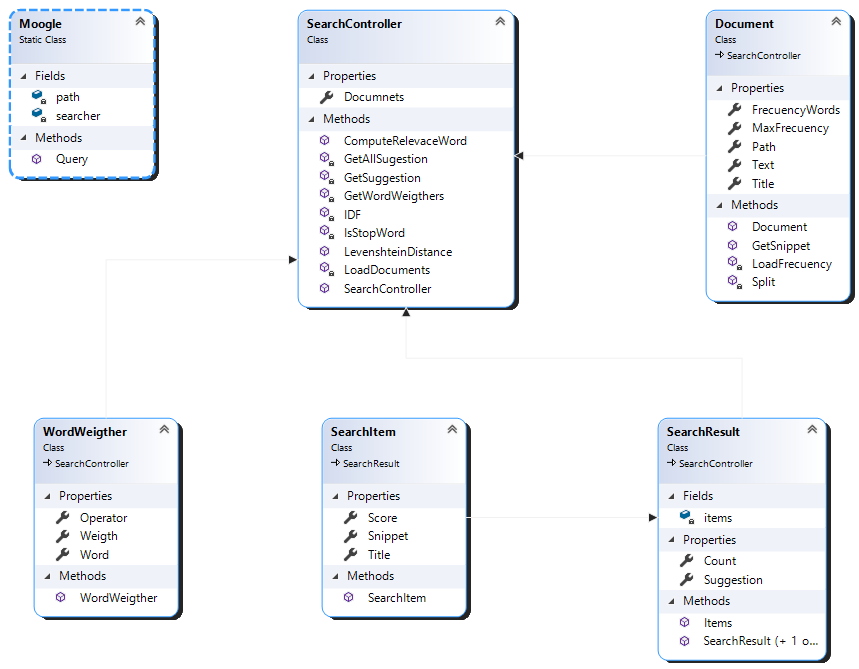
La aplicación cuenta con requisitos indispensables para la búsqueda como la búsqueda simple y la búsqueda con operadores **(! , ^ , \* , ~** ). A continuación se explican cada uno de estos operadores:

* Un símbolo `!` delante de una palabra (e.j., `"algoritmos de búsqueda !ordenación"`) indica que esa palabra \*\*no debe aparecer\*\* en ningún documento que sea devuelto.
* Un símbolo `^` delante de una palabra (e.j., `"algoritmos de ^ordenación"`) indica que esa palabra \*\*tiene que aparecer\*\* en cualquier documento que sea devuelto.
* Un símbolo `~` entre dos o más términos indica que esos términos deben \*\*aparecer cerca\*\*, o sea, que mientras más cercanos estén en el documento mayor será la relevancia.
* Cualquier cantidad de símbolos `\*` delante de un término indican que ese término es más importante, por lo que su influencia en el `score` debe ser mayor que la tendría normalmente.

Además, el buscador ofrece sugerencias de palabras al usuario, cuando su búsqueda ofrece muy pocos resultados.

## Diagrama de clases

A continuación se muestra el diagrama de clases propuesto para darle solución a los requisitos del proyecto. En él se muestran las entidades principales del negocio, relaciones, properties y métodos necesarios para su correcto funcionamiento.



**Document:** Representa la información de cada documento. Esta clase en su constructor hace la llamada a un método privado encargado de cargar un fichero, extraer su contenido y construir el diccionario de palabras contra frecuencias. Este diccionario permite la búsqueda en memoria de cualquier término o palabra de una manera rápida y reutilizable en ese documento sin tener que volver a leer el fichero.

***Title {Titulo}*** *-> Título del documento*

***Path (Path)*** *-> Dirección física donde se encuentra.*

***Text (Texto)*** *-> Texto del documento.*

***FrecuencyWords (Frecuencia de palabras)*** *-> Diccionario de cada palabra con su frecuencia absoluta. Representa las veces que aparece la palabra en el texto del documento y la Máxima frecuencia*

***MaxFrecuency*** *(Máxima frecuencia) -> Representa el valor máximo de frecuencia que tiene una palabra en el documento.*

**SearchController** (Controlador de búsqueda) -> Clase que encapsula los métodos y funcionalidades necesarias para búsqueda, contiene los cálculos de la relevancia de cada término a buscar en cada documento algoritmo(TF-IDF) así como los algoritmos para determinar la sugerencia de palabras en caso de que no ofrezca resultados la búsqueda. Además, presenta los métodos necesarios para el cálculo del score de cada documento.

Esta clase almacena un listado de Document, cuando se corre la aplicación se llama al constructor y se ejecuta un método encargado de recorrer cada fichero y crear por cada fichero un documento con toda su información correspondiente. Todo esto se logra de manera independiente delegando responsabilidades a cada clase que le corresponde.

**Moogle** -> Clase referenciada por el cliente de la aplicación para ejecutar la búsqueda. El método Query es el encargado de dicho proceso.

**SearchResult** -> Este objeto contiene los resultados de la búsqueda realizada por el usuario se relaciona con la clase SearchItem. Este recibe en su constructor dos argumentos: items y suggestion. El parámetro items es un array de objetos de tipo SearchItem. Cada uno de estos objetos representa un posible documento que coincide al menos parcialmente con la consulta en query.

**SearchItem** -> Objeto que recibe 3 argumentos en su constructor: title, snippet y score. El parámetro title es el título del documento (el nombre del archivo de texto correspondiente). El parámetro snippet contiene una porción del documento donde se encontró el contenido del query. El parámetro score tendrá un valor de tipo float que será más alto mientras más relevante sea este item.

## Algoritmo (TF-IDF)

Este algoritmo se implementó para medir la [relevancia](https://es.wikipedia.org/wiki/Relevancia) de un documento dada una [consulta](https://es.wikipedia.org/wiki/Consulta) del usuario, estableciendo así una ordenación o ranking (Score) de los mismos.

Las siglas **TF** provienen de la expresión en inglés *Term Frequency*, **Frecuencia de término** en español, y determina la frecuencia relativa de un término específico, una palabra o una combinación de palabras, en un documento. Este valor se compara con la frecuencia relativa de todos los demás términos de un texto, documento o sitio web.



El término IDF proviene del inglés *Inverse Document Frequency* y significa Frecuencia Inversa de Documento. Esta segunda parte de la fórmula completa el análisis de evaluación de los términos y actúa como el *corrector del TF*. La Frecuencia Inversa de Documento es muy importante ya que incluye en el cálculo la frecuencia de documento de términos específicos: compara el número de todos los documentos disponibles con el número de documentos que contienen el término.



***Score del documento = tf\*idf***

Mediante el cálculo del TF-IDF se puedo utilizar una especie de filtrado de las palabras denominadas *stop-words* (palabras que suelen usarse en casi todos los documentos como preposiciones).

## Operadores de búsqueda y tratamiento de pesos

Una de las funcionalidades indispensables del proyecto son los operadores de búsqueda. De manera general el valor de score de cada documento depende de ellos y debe corresponder a cuán relevante es el documento devuelto para la búsqueda realizada, por lo que hay muchos factores que aumentan o disminuyen esta relevancia.

Como todos estos factores están en oposición unos con otros, se encontró una forma de balancearlos mediante un sistema de puntuación que permita evaluar todo documento con respecto a toda consulta posible. Si un documento no tiene ningún término de la consulta, y no es para nada relevante, entonces su score es 0 como mínimo. Por tanto, la relevancia de los documentos (score) esta conformada por dos elementos: el valor de TF-IDF de la palabra en el documento y su peso.

**Score** = ***valor de TF-IDF del documento*** ***\* Peso de la palabra que se está buscando.***

Las asignaciones de peso para cada operador fueron las siguientes:

Búsqueda simple = 1, Operador ! = 0, Operador ^ = 1, Operador \* = cantidad de veces que se repite el \* en la palabra.

## Cálculo del suggestion

La sugerencia (suggestion) de términos ofrece al usuario una palabra similar a la que se desea buscar y que está contenida en los documentos cuando su búsqueda da muy pocos resultados. Para implementar esta funcionalidad requerida se realizó un estudio de funciones de disimilitud entre cadenas, destacándose el algoritmo de Distancia de Levenshtein. Este algoritmo determina el número mínimo de operaciones requeridas para transformar una [cadena de caracteres](https://es.wikipedia.org/wiki/Cadena_de_caracteres) en otra, se usa ampliamente en [teoría de la información](https://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_informaci%C3%B3n).

Por ejemplo, la distancia de Levenshtein entre "casa" y "calle" es de 3 porque se necesitan al menos tres ediciones elementales para cambiar uno en el otro.

casa → cala (sustitución de 's' por 'l')

cala → calla (inserción de 'l' entre 'l' y 'a')

calla → calle (sustitución de 'a' por 'e')

Se le considera una generalización de la [distancia de Hamming](https://es.wikipedia.org/wiki/Distancia_de_Hamming), que se usa para cadenas de la misma longitud y que solo considera como operación la sustitución.

En la clase SearchController se encuentra este método implementado que recibe dos cadenas y devuelve su grado de disimilitud. Mientras menor sea ese valor más parecido será. Por cada término de búsqueda, se calcula la distancia entre todas las palabras que contienen los documentos y se selecciona el menor valor.