

Richard Matos Arderí C111



Moog!e!

Introduzca su búsqueda



Buscar

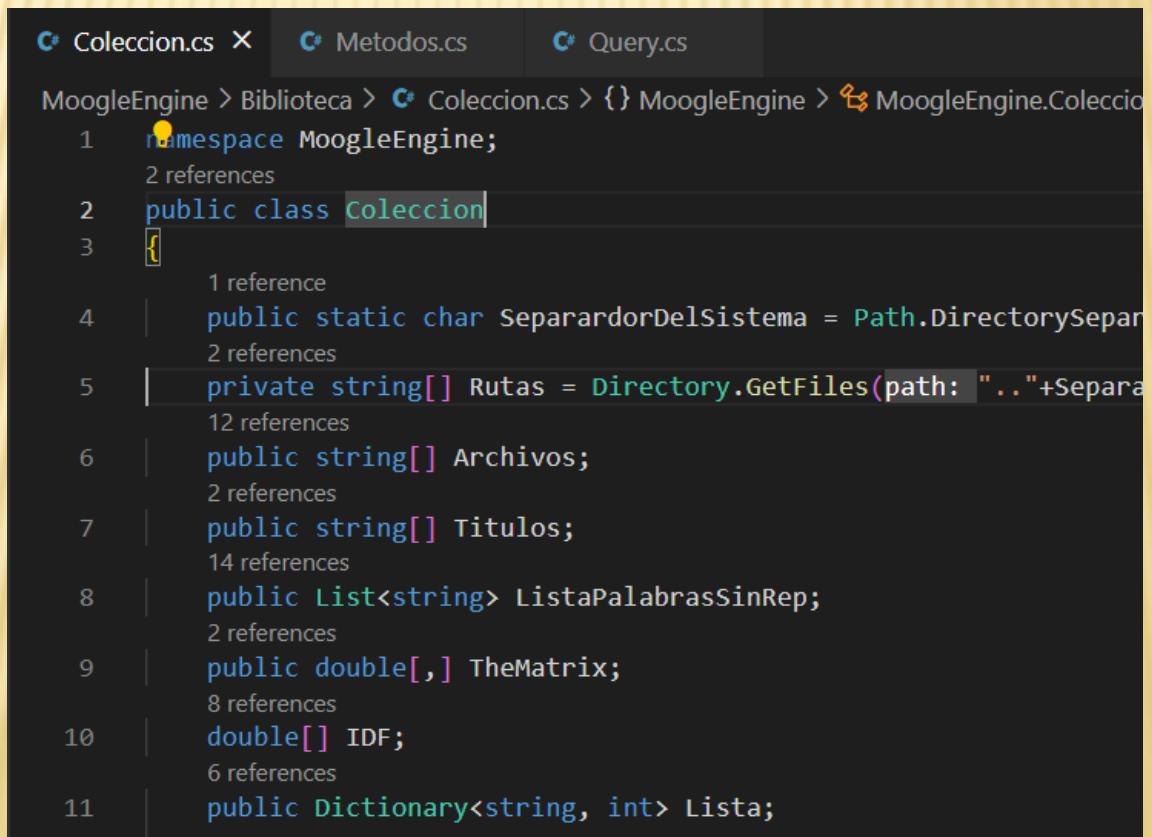
Moog!e!

Descripción

La estructura del proceso algebraico del proyecto recae en las clases Colección, Metodos, y Query.

En la primera de estas se encuentra todo lo relativo al trabajo con los archivos de la colección de archivos (.txt) que utiliza Moogles como materia prima, de ahí el nombre. Entre sus propiedades más significativas está TheMatrix, una matriz, que utilizando un diccionario (`Dictionary<string, int>`) que asocia a cada palabra presente en el documento su frecuencia en el mismo, una lista (`List<string>`) que contiene las palabras únicas de toda la colección y por supuesto la cardinalidad del conjunto de archivos, implementa los conceptos de TF-IDF para la posterior respuesta a la interacción con el usuario. En esta clase se encuentra también el método `VectQuerys`, que sobre la misma base conceptual devuelve el vector TF-IDF de la cadena introducida por el usuario. La clase descansa en la clase `Metodos`, aquí se encuentra cada una de las funcionalidades más generales que demandan las otras clases, una de ellas es el método `Similitud`, que utiliza el procedimiento de similitud por coseno para devolver un array de `double` que contiene de forma ordenada la puntuación de cada documento de acuerdo a su parecido con el query (entrada del usuario al programa), esta puntuación va de 0 a 1 y mientras mayor sea, mayor será la similitud y por ende la relevancia del documento para ser devuelto.

Lo más interesante de la clase es quizás el método ObtenerSearchItems, que además de organizar los documentos por su relevancia llamando a métodos de la clase Metodos, (MejoresTitulos), y devolver un array de este tipo e objeto, incorpora a estos un fragmento del documento donde aparece al menos una palabra de la cadena introducida por el usuario, precisamente la de mayor relevancia por su valor de TF-IDF de acuerdo a su aparición en el query, para esto se implementan métodos de la propia clase y otros de la clase Metodos, con el objetivo de organizar el query de acuerdo al TF-IDF de las palabras y especialmente el método Snipet que busca la primera aparición de la palabra seleccionada en el texto y nos devuelve una vecindad de la misma en string.



```
1 namespace MooglesEngine;
2 references
2 public class Coleccion
3 {
4     1 reference
4     public static char SeparadorDelSistema = Path.DirectorySeparatorChar;
5     2 references
5     private string[] Rutas = Directory.GetFiles(Path.Combine("..", SeparadorDelSistema));
6     12 references
6     public string[] Archivos;
7     2 references
7     public string[] Titulos;
8     14 references
8     public List<string> ListaPalabrasSinRep;
9     2 references
9     public double[,] TheMatrix;
10    8 references
10    double[] IDF;
11    6 references
11    public Dictionary<string, int> Lista;
```

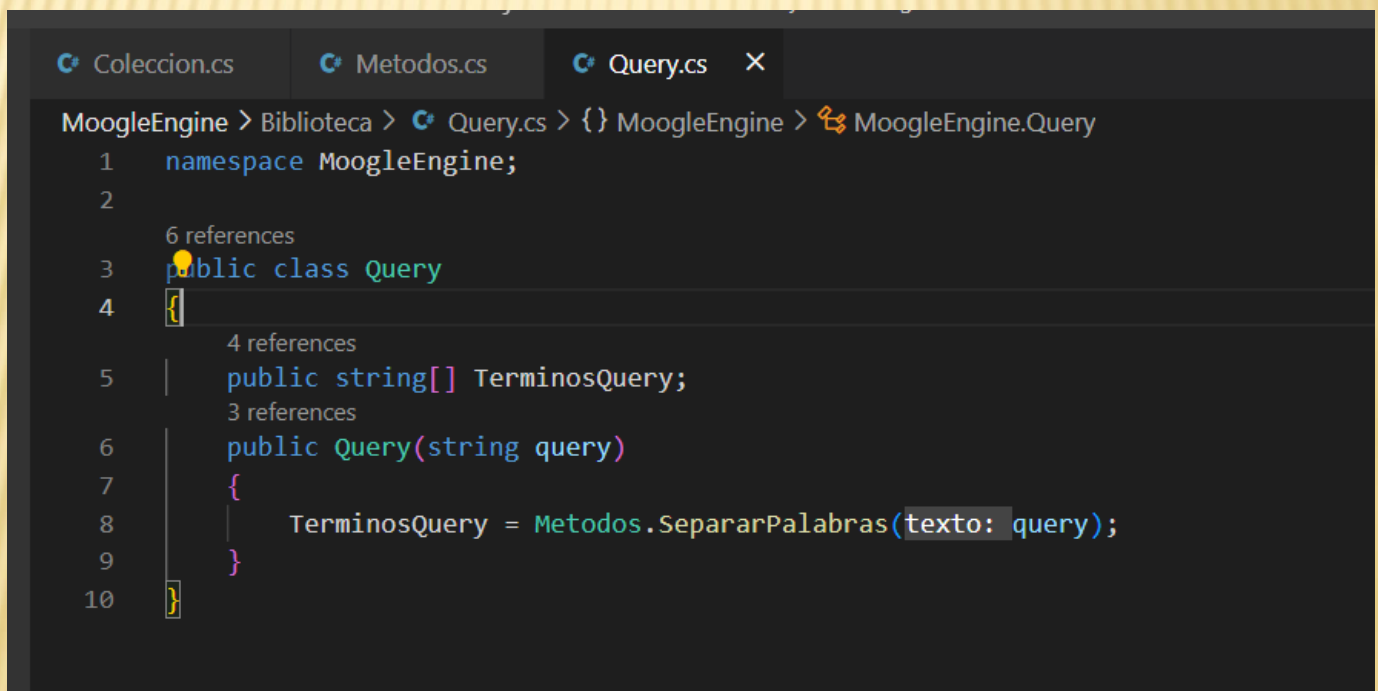

En esta clase además se encuentran los métodos necesarios para la salida de la sugerencia. Si en la cadena introducida por el usuario no aparece al menos una de las palabras en toda la colección entonces aparecerá en la pantalla una sugerencia que será la propia entrada con la palabra que no aparece cambiada por la de menor Distancia de Levenshtein con ella de la colección, para determinar esta distancia se implementa el método DistanciaLevenshtein de la clase Metodos, y los métodos que se encuentran al final de la clase Coleccion determinan la necesidad de sugerencia y la construyen cuando es llamado el método Sugerencia , desde la clase Moogle.

1 reference

```
public string Sugerencia( string query)
{
    //Este Metodo es que retorna la sugerencia final im
    Query objetoQuery = new Query(query: query);
    string[] palabras = objetoQuery.TerminosQuery;
    if(NecesidadSugerencia(query: palabras))
    {
        return ConstructorSugerencia(query: query);
    }
    else
        return "";
}
```

CLASE QUERY

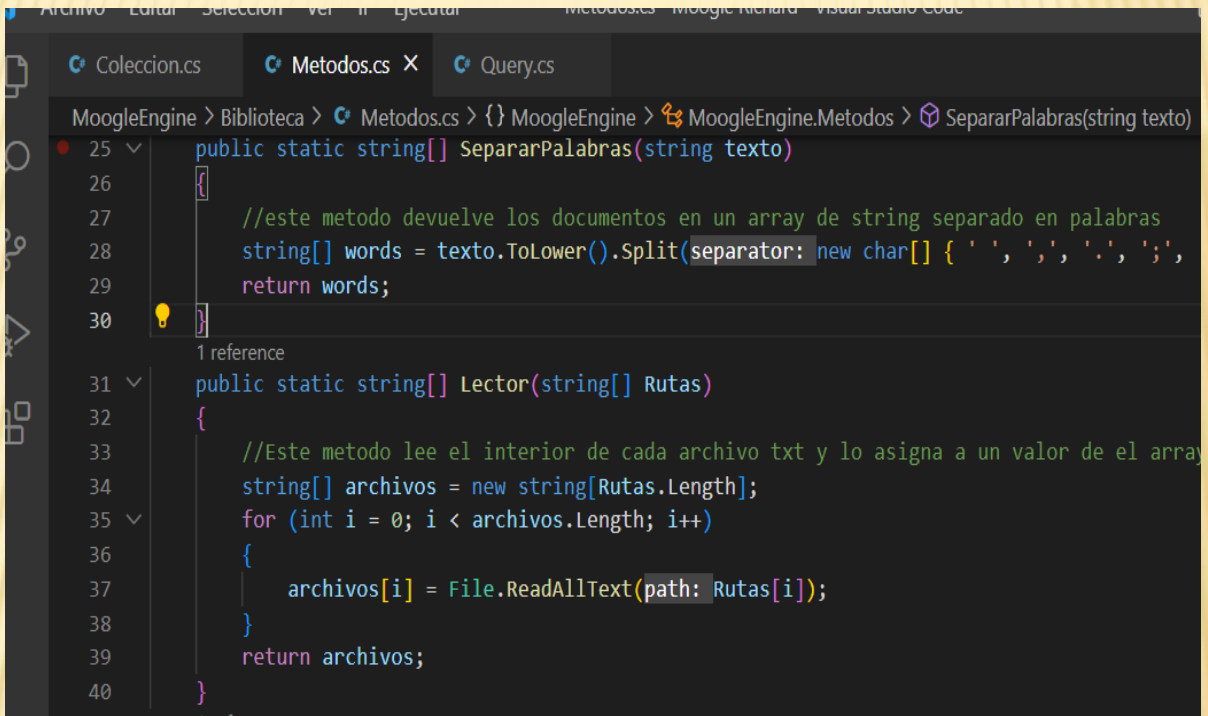
La clase query es la más simple , su única propiedad es un array de string que contiene como términos a las palabras de la query y por tanto para crear un objeto de este tipo es necesario pasarle como argumento un string que es precisamente la entrada del usuario.



```
MooglesEngine > Biblioteca > Query.cs > {} MooglesEngine > MooglesEngine.Query
1 namespace MooglesEngine;
2
3 public class Query
4 {
5     public string[] TerminosQuery;
6     public Query(string query)
7     {
8         TerminosQuery = Metodos.SepararPalabras(texto: query);
9     }
10 }
```

CLASE METODOS

La clase es asistencial, solamente contiene métodos para ser usados en la clase Coleccion, encargados de leer los archivos y asistir a otros métodos.



```
MoogEngine > Biblioteca > Metodos.cs > {} MoogEngine > MoogEngine.Metodos > SepararPalabras(string texto)
25 public static string[] SepararPalabras(string texto)
26 {
27     //este metodo devuelve los documentos en un array de string separado en palabras
28     string[] words = texto.ToLower().Split(separator: new char[] { ' ', ',', '.', ';' },
29     return words;
30 }
1 reference
31 public static string[] Lector(string[] Rutas)
32 {
33     //Este metodo lee el interior de cada archivo txt y lo asigna a un valor de el array
34     string[] archivos = new string[Rutas.Length];
35     for (int i = 0; i < archivos.Length; i++)
36     {
37         archivos[i] = File.ReadAllText(path: Rutas[i]);
38     }
39     return archivos;
40 }
```

```
public static double Norma(double[] a)
{
    //Este Metodo Calcula la norma de cada vector
    double suma = 0;
    for (int i = 0; i < a.Length; i++)
    {
        suma += Math.Pow(x: a[i], y: 2);
    }
    double norma = (Math.Sqrt(d: suma));
    return norma;
}
1 reference
public static double[] MatrizPorVector(double[,] Matriz, double[] vectorQuery)
{
    //Este metodo multiplica una matriz por un vector
    double[] result = new double[Matriz.GetLength(dimension: 0)];
    for (int i = 0; i < Matriz.GetLength(dimension: 0); i++)
    {
        double sum = 0;
        for (int j = 0; j < Matriz.GetLength(dimension: 1); j++)
        {
```

La clase Moogle contiene un método void que inicializa al objeto de tipo Coleccion coleccion, y dentro de su método Query de tipo SearchResult llama a los métodos de la clase Coleccion , ObtenerSearchItems y Sugerencia para devolver el objeto SearchResult.

```
1 namespace MoogleEngine;
2
3
4 2 references
5 public static class Moogle
6 {
7     3 references
8     public static Coleccion coleccion;
9     1 reference
10    public static SearchResult Query(string query)
11    {
12        SearchItem[] searchItem = coleccion.ObtenerSearchItems(query: query);
13        string sugerencia = coleccion.Sugerencia(query: query);
14        return new SearchResult(items: searchItem, suggestion: sugerencia);
15    }
16
17    1 reference
18    public static void Iniciar()
19    {
20        coleccion = new Coleccion();
21    }
22 }
```


La creación de este proyecto ha sido sumamente instructiva Sobre temas como encapsulamiento y Álgebra Lineal y ha contribuido sólidamente a fortalecer habilidades como la investigación y otras menos interactivas pero igualmente importantes como son la organización y la planificación , tan necesarias para la vida profesional.