

Presentación del proyecto Moogle!

Arlette Martínez Lemes

Facultad de Matemática y Computación
Universidad de la Habana

C-112

1 Introducción

- ¿Qué es **Moogle!** ?
- Arquitectura del sistema

2 Clases modificadas

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

3 Clases implementadas por mí

- Document.cs
- Operations.cs
- QueryOperations.cs

4 Ejemplos

- TF-IDF

¿Qué es Moogle! ?

Moogle! es una aplicación **totalmente original** cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos.

¿Qué es Moogle! ?

Moogle! es una aplicación **totalmente original** cuyo propósito es buscar inteligentemente un texto en un conjunto de documentos. Es una aplicación web, desarrollada con tecnología .NET Core 6.0, específicamente usando Blazor como **framework** web para la interfaz gráfica, y en el lenguaje C#.

Arquitectura del sistema

Clases modificadas:

- Mooglee.cs

Arquitectura del sistema

Clases modificadas:

- Moogle.cs
- Program.cs

Clases implementadas por mí:

Arquitectura del sistema

Clases modificadas:

- Moogle.cs
- Program.cs
- Index.razor

Clases implementadas por mí:

Arquitectura del sistema

Clases modificadas:

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

Clases implementadas por mí:

Arquitectura del sistema

Clases modificadas:

- Mooglee.cs
- Program.cs
- Index.razor
- Index.razor.css

Clases implementadas por mí:

- Document.cs

Arquitectura del sistema

Clases modificadas:

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

Clases implementadas por mí:

- Document.cs
- Operations.cs

Arquitectura del sistema

Clases modificadas:

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

Clases implementadas por mí:

- Document.cs
- Operations.cs
- QueryOperations.cs

1 Introducción

- ¿Qué es **Moogle!** ?
- Arquitectura del sistema

2 Clases modificadas

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

3 Clases implementadas por mí

- Document.cs
- Operations.cs
- QueryOperations.cs

4 Ejemplos

- TF-IDF

Moogle.cs

Moogle.cs

Principal clase, en la cual se cablean todos los métodos implementados para trabajar la query y asegurar así una respuesta al usuario.

```
1 namespace MoogleEngine;
2
3
4 public static class Moogle
5 {
6     public static SearchResult Query(string query) {
7         // Modifique este método para responder a la búsqueda
8
9         SearchItem[] items = new SearchItem[3] {
10             new SearchItem("Hello World", "Lorem ipsum dolor sit amet", 0.9f),
11             new SearchItem("Hello World", "Lorem ipsum dolor sit amet", 0.5f),
12             new SearchItem("Hello World", "Lorem ipsum dolor sit amet", 0.1f),
13         };
14
15         return new SearchResult(items, query);
16     }
17 }
18
```

Figure: Código que presentaba dicha clase al inicio

Moogle.cs

```
1 namespace MoogleEngine;
2
3 using System.Text.RegularExpressions;
4
5 public static class Moogle
6 {
7
8     public static SearchResult Query(string query, Operations dataBase, Document[] documents)[1]
9
10     //Trabajando la query...
11     (string[], List<string>) queryNormalize = QueryOperations.QueryNormalize(query, documents);
12     double[] queryTF = QueryOperations.QueryTF(queryNormalize, dataBase);
13     double[] cosineSimilarity = QueryOperations.CosineSimilarity(dataBase, QueryOperations.QueryTF_IDF(queryTF, data
14     List<Tuple<double, int>> descendingOrder = QueryOperations.DescendingOrder(cosineSimilarity);
15
16
17     SearchItem[] items = new SearchItem[descendingOrder.Count];
18
19     if(query == string.Empty)
20     {
21         items = new SearchItem[1];
22         items[0] = new SearchItem ("Por favor inserte una consulta","",0.0f);
23     }
24
25
26     string newquery = QueryOperations.SimilarSuggestion(queryNormalize.Item1, dataBase);
27
28     for(int i = 0; i < descendingOrder.Count; i++){
29
30         items[i] = new SearchItem(documents[descendingOrder[i].Item2].Name, QueryOperations.Snippet(queryNormalize.I
31     }
32     return new SearchResult(items, newquery);
33 }
34 }
```

Figure: Código que presenta actualmente

Program.cs

Program.cs

Clase encargada de hacer que sean normalizados y pre-calculados los valores de cada documento que tengamos como base de datos.

Program.cs

```
1  using Microsoft.AspNetCore.Components;  
2  using Microsoft.AspNetCore.Components.Web;  
3  
4  var builder = WebApplication.CreateBuilder(args);  
5  
6  // Add services to the container.  
7  builder.Services.AddRazorPages();  
8  builder.Services.AddServerSideBlazor();  
9  
10 var app = builder.Build();  
11  
12 // Configure the HTTP request pipeline.  
13 if (!app.Environment.IsDevelopment())  
14 {  
15     app.UseExceptionHandler("/Error");  
16 }  
17  
18  
19 app.UseStaticFiles();  
20  
21 app.UseRouting();  
22  
23 app.MapBlazorHub();  
24 app.MapFallbackToPage("/_Host");  
25  
26 app.Run();  
27
```

Figure: Código que presentaba dicha clase al inicio

Program.cs

```
1 using Microsoft.AspNetCore.Components;  
2 using Microsoft.AspNetCore.Components.Web;  
3 using MoogLeEngine;  
4  
5 public class Program {  
6  
7     public static Document[] documents;  
8     public static Operations dataBase;  
9  
10    public static void Main(string[] args){  
11  
12        documents = QueryOperations.LoadFiles();  
13        dataBase = new Operations(documents);  
14  
15        var builder = WebApplication.CreateBuilder(args);  
16  
17        // Add services to the container.  
18        builder.Services.AddRazorPages();  
19        builder.Services.AddServerSideBlazor();  
20  
21        var app = builder.Build();  
22  
23        // Configure the HTTP request pipeline.  
24        if (!app.Environment.IsDevelopment())  
25        {  
26            app.UseExceptionHandler("/Error");  
27        }  
28  
29  
30        app.UseStaticFiles();  
31  
32        app.UseRouting();  
33  
34        app.MapBlazorHub();
```

Modificaciones
aplicadas

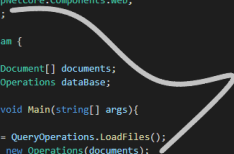


Figure: Código que presenta actualmente

Index.razor

Index.razor

Clase en la que se desarrollan los comando que formarán la interfaz gráfica del proyecto.

Index.razor

```
3 <PageTitle>Moogles!</PageTitle>
4
5 <h1>Moogles!</h1>
6
7 <input class="p-1 my-4" @bind="query" placeholder="Introduzca su búsqueda">
8
9 <button type="default" class="btn btn-primary" @onclick="RunQuery">Buscar</button>
10
11 @if (!string.IsNullOrEmpty(result.Suggestion)) {
12     <div class="suggestion">
13         <p>¿Quizá decir <strong><a href="#">@result.Suggestion</a></strong>?</p>
14     </div>
15 }
16
17 <ul class="results">
18     @foreach (var item in result.Items()) {
19         <li>
20             <div class="item">
21                 <p class="title">@item.Title</p>
22                 <p>... @item.Snippet ...</p>
23             </div>
24         </li>
25     }
26 </ul>
27
28 @code {
29     private string query = "";
30     private SearchResult result = new SearchResult();
31
32     private void RunQuery() {
33         result = Moogles.Query(query);
34     }
35 }
```

Figure: Código que presentaba dicha clase al inicio

Index.razor

```
3 <PageTitle>Moogle!</PageTitle>
4
5 <h1>Moogle!</h1>
6
7 <input class="p-1" @bind="query" placeholder="Introduzca su búsqueda">
8
9 <button type="default" class="btn btn-primary" @onclick="RunQuery"> ⚡ Buscar</button>
10
11 @if (!string.IsNullOrEmpty(result.Suggestion)) {
12     <div class="suggestion">
13         <p><B>¿Quisiste decir </B><strong><a href="#"><I>@result.Suggestion</I></a></strong><B> ?</B></p>
14     </div>
15 }
16
17 <ul class="results">
18     @foreach (var item in result.Items()) {
19         <li>
20             <div class="item">
21                 <p class="title"><center>@item.Title</center></p>
22                 <p>... @item.Snippet ...</p>
23                 <p class="score"><bottom><center>@item.Score</center></bottom></p>
24             </div>
25         </li>
26     }
27 </ul>
28
29 @code {
30     private string query = "";
31     private SearchResult result = new SearchResult();
32
33     private void RunQuery() {
34         result = Moogle.Query(query, Program.dataBase, Program.documents);
35     }
36 }
```

Elementos
modificados

Figure: Código que presenta actualmente

Index.razor.css

Index.razor.css

Clase en la cual se desenvuelve el diseño que tendrá la interfaz gráfica aplicando colores, fuentes de letras, tamaños y formas a las distintas partes que la conforman.

```
1  .results {  
2      text-align: left;  
3  }  
4  
5  .item .title {  
6      font-size: large;  
7      font-weight: bold;  
8  }  
9
```

Figure: Código que presentaba dicha clase al inicio

Index.razor.css

```
1 p {  
2   color: #050a0a;  
3 }  
4 ul {  
5   list-style: none;  
6 }  
7 .results {  
8   margin: 15px;  
9   margin-top: 6px;  
10  text-align: left;  
11 }  
12 .item .title {  
13   font-size: italic;  
14   font-weight: bold;  
15 }  
16 .title {  
17   background-color: #bd0a0a;  
18   color: #ada29e;  
19 }  
20 .score {  
21   color: #ada29e;  
22 }  
23 .item {  
24   margin-top: 20px;  
25   padding-left: 5px;  
26   padding-right: 5px;  
27   background-color: #bd0a0a;  
28   border: 5px solid rgb(66, 4, 4);  
29   border-radius: 30px;  
30   font-family: Book Antiqua;  
31 }  
32 .moogLeName {  
33   color: #000000fa;  
34 }
```

Figure: Código que presenta actualmente

1 Introducción

- ¿Qué es **Moogle!** ?
- Arquitectura del sistema

2 Clases modificadas

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

3 Clases implementadas por mí

- Document.cs
- Operations.cs
- QueryOperations.cs

4 Ejemplos

- TF-IDF

Document.cs

Document.cs

Esta clase es implementada con el propósito de almacenar en 3 propiedades que esta contiene, el nombre, la dirección y el contenido ya normalizado de cada documento *txt* encontrado en la que llamaremos nuestra base de datos.

```
namespace MoogEngine;  
  
public class Document{  
    public string Name {get; private set;}  
    public string Path {get; private set;}  
    public string[] Words {get; private set;}  
  
    public Document(string Name, string Path, string[] Words){  
        this.Name = Name;  
        this.Path = Path;  
        this.Words = Words;  
    }  
}
```

Figure: Código de la clase Document

Operations.cs

Operations.cs

Objeto utilizado para realizar el pre-procesamiento de los datos encontrados en nuestra carpeta "**Content**", de forma que no sea necesario el procesamiento de todos estos datos cada vez que el usuario introduce una consulta.

Este objeto utilizado en el proyecto bajo el nombre de "**dataBase**", contiene un "*vocabulary*" de extrema importancia para su correcto funcionamiento, ya que este diccionario se encarga de relacionar cada palabra que encontremos en nuestro corpus de documentos con la columna que le corresponderá en la matriz de TF-IDF.

Operations.cs

El modelo de espacio vectorial (**TF-IDF**) permite representar los documentos con una estructura para poder realizar tareas de filtrado, recuperación, indexación y calcular aquellas características o términos que aparecen en un documento.

$$\text{TF-IDF}_{(n,d)} = \text{TF}_{(n,d)} \times \text{IDF}_{(n)}$$

The diagram illustrates the TF-IDF formula. It shows the equation $\text{TF-IDF}_{(n,d)} = \text{TF}_{(n,d)} \times \text{IDF}_{(n)}$. Below each term, there is a bracket and a box containing its definition:

- TF-IDF_(n,d)**: A red bracket connects it to a red box containing the text "Peso de un término (n) en un documento (d)".
- TF_(n,d)**: A blue bracket connects it to a blue box containing the text "Frecuencia de aparición de un término (n) en un documento (d)".
- IDF_(n)**: A yellow bracket connects it to a yellow box containing the text "Factor IDF de un término (n)".

Figure: Ponderación de la fórmula de TF-IDF

Operations.cs

A través de las fórmulas:

$$TF = \frac{n_d}{D} \quad (1)$$

Donde n_d representa el número de veces que se repite una palabra en un documento y D el número total de palabras que contiene ese documento.

$$IDF = \log\left(\frac{1 + N}{n_p}\right) \quad (2)$$

Donde N representa el número total de documentos y n_p el número de documentos que contienen la palabra en cuestion.

QueryOperations.cs

QueryOperations.cs

Clase en la cual se desarrollan la mayoría de los métodos utilizados por el programa. De ellos los más notables son:

QueryOperations.cs

QueryOperations.cs

Clase en la cual se desarrollan la mayoría de los métodos utilizados por el programa. De ellos los más notables son:

Symbols:

Método que ayuda a determinar si el símbolo \$ se encuentra en una palabra y en caso de ser así, agrupa esta en una lista que será utilizada más adelante.

El símbolo puede ser colocado en cualquier parte de la palabra, dándole mayor importancia a los documentos que la contengan, sólo el usuario debe asegurarse de que la palabra esté correctamente escrita.

QueryOperations.cs



Figure: Búsqueda con \$

QueryOperations.cs



Figure: Búsqueda sin \$

QueryOperations.cs

CosineSimilarity:

$$\text{CosineSimilarity} = \frac{A \cdot B}{\| \text{Vert}A \| \cdot \| \text{Vert}B \|} \quad (3)$$

El objetivo de esta fórmula es calcular la similitud entre la pregunta (que se convertiría en el vector A , expresado en función de la aparición de los n términos en la expresión de búsqueda) y los m vectores de documentos almacenados, tomando B momentaneamente como ellos.

De tal forma que si el resultado es igual a 0, significa que los vectores son totalmente diferentes; y en caso de que sea igual a 1, son completamente iguales.

QueryOperations.cs

Snippet:

```
snippet = $"{document1.Substring(Math.Max(0, index - 80), Math.Min(160, (document1.Length - 2)-index))}";
```

Figure: Función que se encarga de buscar un Snippet

Con esta simple función, que apartir de utilizar operaciones simples con son la - y la comparación de valores máximos y mínimos, extrae de un documento un fragmento de este donde casi siempre aparece la palabra en cuestion.

QueryOperations.cs

LevenshteinDistance:

El algoritmo de Levenshtein utilizado para calcular la distancia entre dos cadenas de caracteres, utilizando la mínima cantidad de operaciones que incluyen la inserción, eliminación o sustitución de un caracter.

Hace que su implementación usando **DP**(Programación Dinámica) de forma recursiva, conlleve crear una matriz dp para almacenar los resultados de los subproblemas ya resueltos, y evitar el recálculo de estos. Esta matriz tiene como dimensiones $[longitud_de_la_cadena1 + 1, longitud_de_la_cadena2 + 1]$.

1 Introducción

- ¿Qué es **Moogle!** ?
- Arquitectura del sistema

2 Clases modificadas

- Moogle.cs
- Program.cs
- Index.razor
- Index.razor.css

3 Clases implementadas por mí

- Document.cs
- Operations.cs
- QueryOperations.cs

4 Ejemplos

- TF-IDF

TF-IDF

Teniendo las siguientes oraciones:

- El río Danubio pasa por Viena, su color es azul.

TF-IDF

Teniendo las siguientes oraciones:

- El río Danubio pasa por Viena, su color es azul.
- El caudal de un río asciende en Invierno.

TF-IDF

Teniendo las siguientes oraciones:

- El río Danubio pasa por Viena, su color es azul.
- El caudal de un río asciende en Invierno.
- El río Rhin y el río Danubio tienen mucho caudal.

TF-IDF

Teniendo las siguientes oraciones:

- El río Danubio pasa por Viena, su color es azul.
- El caudal de un río asciende en Invierno.
- El río Rhin y el río Danubio tienen mucho caudal.
- Si un río es navegable, es porque tiene mucho caudal.

A continuación, obviando las palabras más comunes (que normalmente son eliminadas en este mismo proceso), será mostrado el proceso al que es sometida toda la información usando las fórmulas ya presentadas.

TF-IDF

Tabla 3 Matriz de frecuencia de términos de documentos normalizada

	rio	danubio	viena	color	azul	caudal	invierno	rhin	navegable
D1	0.2	0.2	0.2	0.2	0.2	0	0	0	0
D2	0.33	0	0	0	0	0.33	0.33	0	0
D3	0.4	0.2	0	0	0	0.2	0	0.2	0
D4	0.33	0	0	0	0	0.33	0	0	0.33

Figure: Calculando el TF

TF-IDF

- $\text{Idf}(\text{rio}) = \log(4/4) = \log(1) = 0$ (en 4 documentos rio aparece 4 veces)
 - $\text{Idf}(\text{danubio}) = \log(4/2) = \log(2) = 0.301$ (en 4 documentos danubio aparece 2)
 - $\text{Idf}(\text{viena}) = \log(4/1) = \log(4) = 0.602$
 - $\text{Idf}(\text{color}) = \log(4/1) = \log(4) = 0.602$
 - $\text{Idf}(\text{azul}) = \log(4/1) = \log(4) = 0.602$
 - $\text{Idf}(\text{caudal}) = \log(4/3) = \log(1.33) = 0.124$
 - $\text{Idf}(\text{invierno}) = \log(4/1) = \log(4) = 0.602$
 - $\text{Idf}(\text{rhin}) = \log(4/1) = \log(4) = 0.602$
 - $\text{Idf}(\text{navegable}) = \log(4/1) = \log(4) = 0.602$
- Multiplicando los factores idf por los valores tf de la tabla anterior se obtiene por ejemplo:
- $d_j^i = \text{TF}(t_i, d_j) \times \text{IDF}(t_i) = D1 \text{TF}(\text{danubio}, D1) \times \text{IDF}(\text{danubio}) = 0.2 \times 0.301 = 0.0602$

Tabla 3 Matriz de términos TF-IDF normalizada

	rio	danubio	viena	color	azul	caudal	invierno	rhin	navegable
D1	0	0.0602	0.1204	0.1204	0.1204	0	0	0	0
D2	0	0	0	0	0	0.04122	0.19867	0	0
D3	0	0.0602	0	0	0	0.02498	0	0.1204	0
D4	0	0	0	0	0	0.04122	0	0	0.19867

Figure: Obteniendo la matriz TF-IDF

TF-IDF

Tabla 3 Matriz de términos TF-IDF normalizada incluyendo similitud

	rio	danubio	Viena	color	azul	caudal	invierno	rhin	navegable	Similitud q x Dn
D1	0	0.0602	0.1204	0.1204	0.1204	0	0	0	0	0.46225016
D2	0	0	0	0	0	0.04092	0.19866	0	0	0
D3	0	0.0602	0	0	0	0.0248	0	0.1204	0	0.73283928
D4	0	0	0	0	0	0.04092	0	0	0.19866	0

Figure: Calculando la Similitud de cosenos