The Moogle! search engine

What problem does this software solve?

Given a query, return the most relevant files from the Content folder.

Requirements

Dotnet 6.0

Portability

It 'should' work on Windows. (not tested)

Quick guide

1. Start the server

   make dev

2. Enter http://localhost:5000/
3. Write something in the search bar
4. Click on 'Buscar'
5. ???
6. Profit

FAQ

Taking too long

The first query takes 10s to 1min (depending on your hardware). The following queries will be faster.

Can I put X file in Content?? Will Moogle! load it??

Moogle! only supports .cs, .txt, and .py files. But you can add whatever extension you want to the code manually editing the following line.

```
cs static public string[] PATTERNS = new string[]
```

In MoogleServer/Moogle.cs

Can I add new files after the first query?

Yes. It will load after <5 minutes

Why after 5 minutes?

It would be too slow to check every time.

It just shows a black screen after I try to serch for [word]!

The word is not in the files.

Some weird "json" files just appeared out of nowhere

Those are cache files. After the first query Moogle! saves all the data it requires in those files to speed up searches. Feel free to delete them.

What is tf-idf and why does Moogle! use it?

One of the most naive ways to get a criteria to sort relevant files given a set of words is to see how many times those words are repeated in a text. However, common words like "the" or "is" will have inflated scores and they are not really relevant for our needs. A smart way to filter those words is to get how many times it's found in all files and divide the number of times the word is found in every file by this value. That's exactly what tf-idf is. With this value we can have a more sensible crteria to sort the results of a query.

How does it work?

Startup (during first query)

1. Moogle! searches all valid files inside the Content folder
2. After all the filenames are stored, Moogle! counts the words on each file and assigns that count to their corresponding file
3. Calculates the tf-idf for each word and stores only the top 5 files with the highest tf-idf score for that word
4. Dumps the cache.
5. Parses the query and get the candidates (files)-- 5 per word of the query.
6. Calculate the total score (sum of the tf-idf of the file for each matching word of the query) and find a highlight of the text to show to the user
7. Sort the items and hand them to the front-end