Search Engine Task

# Problem Statement

A group of web pages has been classified by associating a list of keywords, given in decreasing order of relevance, with each page (i.e., the order of keywords is from the most specific keyword to the least specific). For example, on the TopGear website, a page on reviews of Ford cars may have the keywords: Ford, Car, Review in that order; the most relevant keyword is Ford.

Queries also include a list of keywords from most to least relevant. For example, in a query consisting of the keyword Ford followed by the keyword Car, Ford is more important than Car.

In this problem, you are to determine the top five (or fewer) pages that match each of an arbitrary number of queries. To assess the strength of the relationship between a query and a web page, assume the keywords for each page and each query are assigned integer weights, in descending order, starting with N, where N is the maximum number of keywords allowed for a web page and query. (For the purposes of this exercise, you can set N to 10)

The strength of the relationship is the sum of the products of the weights associated with each keyword that appears both in the web page list and the query list.

For example, assume the following web pages and keyword lists:

Page 1: Ford, Car, Review

Page 2: Toyota, Car

Page 3: Car, Ford

For N equal 10, a query with keywords Ford and Car in that order yields the following strength ratings.

Page 1: ((10\*10) + (9\*9)) = 181

Page 2: (9\*9) = 81

Page 3: ((10\*9) + (9\*10)) = 180.

Similarly, a query with keywords Ford and Review yields the following strength ratings.

Page 1: ((10\*10) + (9\*8)) = 172

Page 2: 0

Page 3: (10\*9) = 90

Q Ford Car

P1

P3

P2

# Task

Please note that for this task you will be split in pairs and will be working with a teammate.

### Primary Goals:

1. Write an HTTP REST API that accepts a list of not more than 10 keywords and a page title
   1. {title = “figo”, keywords=”ford, review”}
2. Write an HTTP REST API that accepts a query and returns a list of pages ordered in increasing order of relevance. (If two pages have the same order, prioritize the older page)

### Additional Constraints:

1. The page data should survive an application restart. (The application upon a restart should be aware of what pages it has already received)
2. No data can be saved in a raw text file. Any data that must be saved, should be saved in MongoDB
3. There should be just one repository per team with your code. (You may have as many branches as you would like)
4. The search is case insensitive.

You are free to save the data in Mongo DB in whatever schema makes the most sense to you. How you organize the data is entirely up to you.

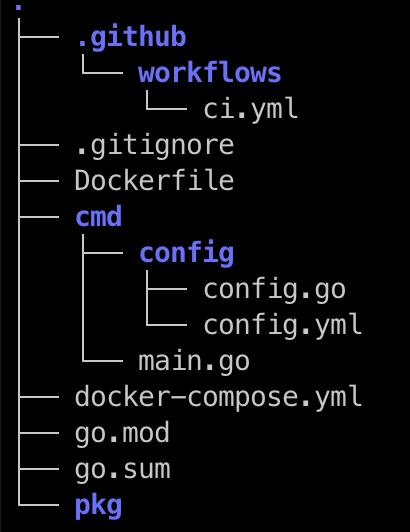
The structure of the API including the METHOD, the endpoint or the data needed is also entirely up to you. However, versioning it would be a wise move. (v1/<>)

The task will require you to distribute work to each other. You will need to make sure to use a single repository. Any remote branch merges into master MUST require a PR and a review from your teammate.

The Queries are evaluated on the previously saved set of Pages.

### Optional Tasks:

1. Create a Makefile with support for build and test. (<https://makefiletutorial.com/>)
2. Add PR Checks for failing tests. (https://docs.github.com/en/actions/using-workflows)
3. Generate a coverage report for your codebase. (<https://go.dev/blog/cover>).
   1. Justify why certain lines could not be covered
   2. You are free to use other tools like codecov into your workflow (https://docs.codecov.com/docs)
4. Utilise an interface for your DB Client. Try and justify why this is a good change in the long term
5. Write a docker-compose that will start your application and MongoDB. Make sure your application is accessible via an external port
6. Match the following project structure:



* 1. cmd/main.go is the entry point to your program
  2. pkg holds all other code organised into directories based on need. (Like /db. for your database driver code, /middleware for any middleware you may add, /service for any external service you require)

Mongo -> Host, user, pass, port

DBCON:

HOST: localhost

Additional References:  
<https://github.com/vektra/mockery>

<https://gin-gonic.com/docs/introduction/>

<https://github.com/spf13/viper>

<https://www.mongodb.com/docs/drivers/go/current/>

<https://github.com/sirupsen/logrus>

Swagger -> OpenAPI specification

<https://www.mongodb.com/docs/manual/core/index-case-insensitive/#case-insensitive-indexes>

# Example CLI Output:

# 