**“Boogle”**

Imagine you have a successful book search engine where users can search within public domain books for specific words or even phrases. You can type “Elementary, my dear Watson” and the engine happily spits out a list of Sir Arthur Conan Doyle’s Sherlock Holmes series ordered by how many times such phrase appeared in each volume - which turns out to be quite a lot. Not only that but you can also tell the users which page originally contained the terms. Useful indeed.

Your site looks great, users are loving the UX and the backend management interface is efficient and intuitive. There’s just one big, big problem: your search engine response time is, let’s say, less than ideal (i.e. it sucks).

Your original solution based on regular expression queries simply cannot cope with the flood of users howling at the gates to read Sherlock Holmes’ original adventures after a first contact with a recently released movie of that franchise. Your newly acquired advertisers are angry because their conversion rates dropped drastically due to slow page load times and you risk losing your happy - and for the moment tolerant - early adopter user base. What a conundrum!

**Solution**

Being the creative and intuitive developer that you are, you realize you need a solution that can index book pages, search efficiently by words and return an ordered list of matching pages.

For the sake of argument let’s say that you don’t need to persist this indexed data nor the actual page contents (the main database stores the book and page data permanently) and you want to implement an API to index and search this data using the least amount of third party libraries or services.

The response should contain a list of matching pages ordered by a **score**. That score is the number of words within a query that matched a given page. Ex:

* Content: “the quick brown fox jumped over the lazy dog”
* Queries + Scores:
  + “quick but lazy brown dog”: 4
  + “the quick fox”: 3
  + “the fox”: 2
  + “the”: 1

This API should have the following two routes:

**# Endpoint to handle indexing**

**POST /index**

**- Request**

Content-Type: application/json

Accept: application/json

{

“pageId”: 300,

“content”: “Elementary, dear Watson”

}

**- Response**

HTTP/1.1 204 No Content

Content-type: application/json

**# Endpoint to handle searching**

**GET /search?query=Elementary,%20dear%20Watson**

**- Request**

Accept: application/json

**- Response**

HTTP/1.1 200 OK

Content-Type: application/json

{

“matches”: [

{

“pageId”: 300,

“score”: 3

},

{

“pageId”: 12,

“score”: 1

}

]

}

**OBS:**

* Queries should be **case insensitive**
* Disregard punctuation
* Disregard word order
* Disregard frequency - “the the the the” should have a score of “1”, not “4” for matching pages
* Matching pages don’t need to contain all words in the query: a single word would generate a match. Ex: “Red fox” would match “the brown fox” with a score of “1”
* Remember to add tests!