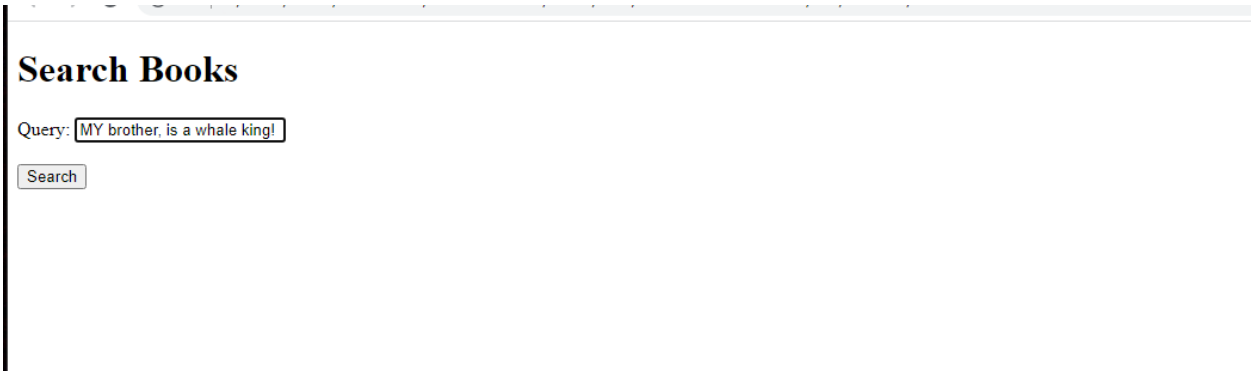
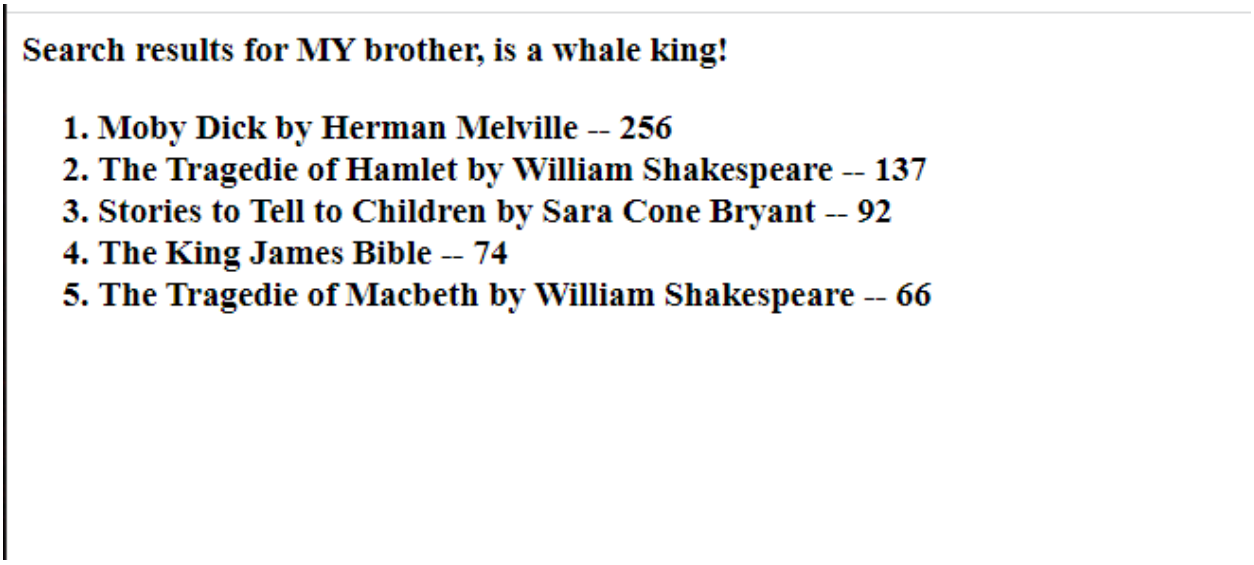
CPSC 5330 -- Spring 2023
Lab 5 (Week 6): End-to-End Application in AWS

Text Search Application

You will write a simple search application that will use TF-IDF to compute the "relevance" of a document to a search query, and display the most relevant documents. Relevancy is defined below.

Here is sample input/output:





You will use two data sources – ideally these would be in a public bucket and could be read directly. But since that has been problematic, you will take the files from the Github repo and upload them to S3. The two sources are:

- Computed TF-IDF values, the result of the batch indexing phase you implemented in previous labs. Note, you will not be running your TF-IDF batch script, you will use the output. The TFIDF output files are in the folder `tfidf` in the repository, and you will upload them to an S3 bucket

that you create. Your bucket must have a unique name, but please make clear in the name that it contains your tfidf values.

- A mapping from document ID to document title. This mapping is stored in the file `document-titles.txt` in the repo. You will upload it to S3.

You will complete the following steps:

1. Upload the two data sets (the tfidf folder and the document-titles file) to S3.
2. Import those two S3 data sources to create two DynamoDB tables, `tfidf` and `doctitle` to hold the data. You will need to think about the correct partition and sort keys for your use cases.
3. Write Python code to implement the relevance search formula. More explanation below.
4. Create a Lambda function that incorporates your search code so it accepts a query string as input and produces HTML as output.
5. Create a file `search.html` that calls your Lambda function and renders the HTML – you will be given a template, but you will need to adjust the URL (which points to your own lambda function), and you are free to make it prettier!

Computing the Relevance Value

At the core of your application is the definition of *relevance*, which is defined in terms of a document ID and a query. (The rest of your code just computes relevance for all "candidate documents," sorts, selects the most relevant, and formats the results.)

Take these steps:

- Preprocess the query -- that is, convert the string to a list of *terms*. Since it is important to use the same preprocessing algorithm as when the document was indexed, the code to preprocess the query ("termify") is supplied for you.
- Find a list of candidate documents. Once you have the query terms, you find all documents that contain any of the query terms. You will calculate relevance only for those documents, since by definition if a document contains none of the query terms, it has 0 relevance to the query.
- For each of those documents, the relevance is just the sum of the TF-IDF values for each term in the query, normalized for the length of the query terms. If Q is a set of terms, then relevance is defined as follows:

$$relevance(doc, Q) = \left\lfloor \sum_{term \in Q} \frac{tfidf(doc, term)}{|Q|} \right\rfloor$$

In this equation, the *tfidf(doc,term)* value is retrieved from DynamoDB, and the brackets just mean truncate to an integer.

The code is partially supplied to you in the file `search.py`

Summary

1. Decide on the proper keys for your two DynamoDB tables, and import `tfidf` and `doctitles` to those tables
2. Working on the Python code, starting in Cloud Shell
    a. Write interfaces that allow you to do three things
        i. Get all the documents associated with a term
        ii. Get the TF-IDF value for a document ID and a term
        iii. Get a document's name from its ID
    b. Now you have all the code you need to compute relevance and format the results. The main phases are
        i. Termify the query
        ii. Get all relevant doc-ids
        iii. Compute relevance for each of those doc-ids
        iv. Sort by relevance, choose the top 5, and retrieve the document name for the top 5
    c. Be sure to test the Python code thoroughly before you move to the next step – debugging in Lambda is much harder than doing it in Cloud Shell!
3. Create a Lambda function just as we did in the lab. Incorporate your Python code into the Lambda function. (You can write a test at this point to verify that your Lambda function is working properly.)
4. Add formatting code that takes your relevance output and generates nice HTML
5. Take the `search.html` file and substitute the External URL of your Lambda function.

Part 0, Style Points (2 points)

(These are easy points to get, just by following instructions and handing in good clean work!)

Did the solution run out of the box? Was the code nicely structured?

To Hand In

A Zip file containing (only) these files

- The file `search.html` that calls your application and renders the search result. (Be sure your application is actually running! Swap files with a classmate to check.)
- A retrospective report in a file `retrospective.pdf` – a reflection on the assignment, with the following components
    - Your name
    - How much time you spent on the assignment
    - If parts of the assignments are not fully working, which parts and what the problem(s) are
    - Were there aspects of the assignment that were particularly challenging? Particularly confusing?

- What were the main learning take-aways from this lab – that is, did it introduce particular concepts or techniques that might help you as an analyst or engineer in the future?