Search Engine Challenge:

General Instructions:

- **Please use a github repository to submit your code**
- **Use smaller commits with proper messages showing how you proceeded with the development**
- **There are three tasks in this challenge. Submitted repository should have solutions to all tasks clearly distinguishable.**
- **Task specific expectations and notes are outlined below each task description. Please read them carefully before submitting.**
- **Please do not use libraries for search related functionality, auto complete behaviour or style management**

1. Write a utility to search summaries (~ 60 min)

Unibuddy wants to build a service that allows students to search through coursebooks summaries which would make picking and buying a coursebook, a much better experience for students.

First we need to develop a local version of the system. Our bot scraped a website with book summaries, and stored them in *data.json* in the *summaries* array. The summaries array is a small data example for local development. You should assume that the real service will have ~10^6 summaries.

Your goal is to code a search utility function/class that given a search query, searches the book summaries and returns the *K* most relevant ones. A search engine query is the set of keywords that users will type in order to find a relevant document. You are allowed to use only basic language (python/javascript) functionality.

The api of the search engine should be as follows:

```
Input: The input should be a user query of type string and number of
items to return
           query (string): eg. 'is your problems'
           K (integer): eg. 3

Output: List of K relevant summaries sorted according to order of
```

```
relevance given a query. A summary is a dictionary that follows the
schema: {'summary': string, 'id': integer, “title”: string}
            summaries: eg. [
                  {'summary':'The Book in Three Sentences: Practicing
meditation and mindfulness will make you at least 10 percent
happier....', 'id':0, “title”: “Anything You Want”},
                  {'summary':'The Book in Three Sentences: Finding
something important and meaningful in your life is the most productive
use of...', 'id':48, “title”: “Strangers to Ourselves”},
                  {'summary':'The Book in Three Sentences: Everything in
life is an invention. If you choose to look at your life in a new
way...', 'id':7, “title”: “Resplendent Light”}
            ]
```

Information regarding titles is also available in *data.json*. There is a 1:1 match between the list of summaries and list of titles.

Expectations for the task:

1. Readable, testable, modular and well-commented code
2. Use of proper data structures for making your code reusable, optimal and efficient
3. Optimal and/or accurate search results
4. Unit-tests with required mocks
5. Clarity on challenges you faced in your implementation and thoughts on areas of your code that you would go back to and improve on

NOTE 1: We are not making a server or a frontend component at this stage
NOTE 2: Relevance of match is to be defined by candidate. Can be percentage match and/or number of instances of partial match etc.
NOTE 3: There is a 1:1 match between titles and summary objects, ie. the first title would correspond to the first summary, second title corresponds to the second summary and so on.

2. Write a server to find books (~ 60min)

We need to make that functionality available remotely as a service, so users can find coursebook summaries anywhere in the world.

Use the previously built search engine to offer an API that given a *list* of queries and an integer *K*,
it will return the top K matched books as list, for every query in the list, so the result will be a list of lists.
A *book* object is defined as follows:

```
{ id: "string", author: "string", summary: "string", query: "string",
title: “string”}
```

The information about the book author is provided by another microservice which you can call *https://ie4djxzt8j.execute-api.eu-west-1.amazonaws.com/coding*.
The api accepts POST *application/json* content like *{'book_id: integer}* and returns the book author *{'author': string}*.

The api of the server should be as follows:

```
 Input: A list of queries and number of results to return for each
            queries (list(string)): eg. ["is your problems", "achieve
take book"]
            K (integer): eg. 3
    Output: A list of lists of books.
            books: eg. [
                  [
                        {'summary':'The Book in Three Sentences:
Practicing meditation and mindfulness will make you at least 10 percent
happier....', 'id':0, 'query': "is your problems", "author": "Dan
Harris", “title”: “Anything You Want”},
                        {'summary':'The Book in Three Sentences: Finding
something important and meaningful in your life is the most productive
use of...', 'id':48, 'query': "is your problems", "author": "Mark
Manson", “title”: “Strangers to Ourselves”},
                        {'summary':'The Book in Three Sentences:
Everything in life is an invention. If you choose to look at your life
in a new way...', 'id':7, 'query': "is your problems", "author":
"Rosamund Zander and Benjamin Zander", “title”: “Resplendent Light”}
                  ], [
                        {'summary':'The Book in Three Sentences: The 10X
Rule says that 1) you should set targets for yourself that are 10X
greater than what....', 'id':1, 'query': "achieve take book", "author":
"Grant Cardone", “title”: “The Richest Man in Babylon”},
                        {'summary':'The Book in Three Sentences: Many of
our behaviors are driven by our desire to achieve a particular level of
status relative...', 'id':20, 'query': "achieve take book”, "author":
"Keith Johnstone", “title”: “10 Happier”},
                        {'summary':'The Book in Three Sentences:
Ultimately, profit is the only valid metric for guiding a company, and
there are only three...', 'id':14, 'query': "achieve take book”,
"author": "Hermann Simon", “title”: “What Got You Here Won’t Get You
There”}
```

```
                ]
            ]
```

Expectations for the task:

1. Readable, testable, modular and well-commented code
2. Use of proper data structures for making your code reusable, optimal and efficient
3. Optimal author api usage and/or efficient data manipulation
4. Unit-tests with required mocks
5. Clarity on challenges you faced in your implementation and thoughts on areas of your code that you would go back to and improve on

NOTE 1: Please use search utility made in the earlier task without any further modification.
NOTE 2: *book_id* is the same as *id* in summary objects.
NOTE 3: Author API expects a POST call with content-type JSON (e.g. *requests.post('https://someurl', json={'key':'value'})*)
NOTE 4: Use the framework of your choice (python/javascript) to serve this endpoint.

3. Write a frontend application to find and select books (~ 60 min)

We need to make the search functionality available as a frontend component, so users can find coursebooks using keywords from summaries.

Use the previously built api server for searching books to make a **responsive ReactJS SPA**.

The SPA will have two parts:
1. **Form**: A form containing an asynchronous autocomplete. This autocomplete would take "*input search string*", and "*number of suggestions to display*", as mandatory arguments. You need to **fetch the books from the api server** made in the previous task and **display the titles** of the suggestions in the autocomplete as the user looks for a search string.
When the user **selects a title** from the suggestion, and the **submits the form**, form is cleared and **selected book is appended** in a list below the form.

2. **List**: A list of cards displaying the books selected by the user. Each card should display:
    - Title
    - Author
    - Short Summary (substring of the summary ending in "...")

Preview the sample output here

Expectations for the task:

1. Well defined structure to your front end components and business logic
2. Thorough Unit-tests added with proper mocks
3. Accessibility in styles, responsive design and overall good looking auto-complete behaviour
4. Clarity on challenges you faced in your implementation and thoughts on areas of your code that you would go back to and improve on

NOTE 1: Please do not use any design framework like Material-UI or bootstrap for styling

NOTE 2: Example output is only base level expectation in terms of design. Please improve on that.