Vonage API Interview Task

Brief

Your task is to create a service exposing a RESTful interface for creating, maintaining, and searching a document index. The documents to be indexed are stored in a local directory. The service should support indexing and querying multiple documents. The search functionality should support search for a single word or a sentence. The search should return a score representing how much of the search query was contained in the indexed document set.

Please limit the amount of time you spend on this task to three hours; don't expect to finish everything, but be prepared to explain what you had to leave out due to the lack of time and how you would approach the implementation of the incomplete functionality if given more time.

We expect 'production-quality' code, so write as if this system is going into production 'next week'; if you don't finish everything it should be in a state where someone else could pick up the task without 'on-boarding'.

Production-quality code is more important than complete functionality, take some time to think about what we're looking for; it's NOT your knowledge of ranking algorithms!

Feel free to use any supported version of Java, but newer features might come in handy.

Requirements

- Use the Spring Initializr provided here (you can choose between Java and Kotlin): https://start.spring.io/#!type=gradle-project-kotlin&language=kotlin&platformVersion=3.1.2&packaging=jar&jvmVersion=17&groupId=com.vonage.api&artifactId=interview&name=interview&description=Interview%20project%20for%20Vonage%20API&packageName=com.vonage.api.interview&dependencies=web
- You MUST NOT:
    - Upload your solution to any publicly accessible websites
    - Use third-party dependencies that have not already been included in the Spring Intializr
- The API should be RESTful
- You should be able to submit a local directory URI to the service to create an in-memory index of the files in the directory
- You should then be able to query the index for a given sentence to retrieve a summary that shows the degree of 'match' for that sentence across all files represented by that index

- The search should take the words provided in the request and return a list of the top 10 (max) matching filenames in rank order, giving the rank score against each match
- Assume ‘proper’ formatting of the files and parameters; there's no need to cover every single edge case
- Write up any caveats and things you didn't have time to complete in a README.md file

Ranking

- The rank score must be 100% if a file contains all the words
- It must be 0% if it contains none of the words
- It should be between 0 and 100 if it contains only some of the words, but the exact ranking formula is up to you to choose and implement - simple is fine!

Things to consider

- What constitutes a word
- What constitutes two words being equal (and matching)
- Data structure design: the in-memory representation to search against
- Ranking score design: start with something basic, then iterate as time allows
- Non-functional requirements and operational maturity - logging, error handling, startup/stop, service health, index lifecycle

Deliverables

- **Quality, well-structured, and well-tested** code to implement the requirements
- A README.md containing:
  - Instructions on how to build and run your code
  - How long you spent on the task
  - Any caveats, what you might have done differently if you started again
  - Any explanations on how you would approach functionality that you didn’t have time to complete

PLEASE TAKE NOTE OF ALL THE REQUIREMENTS BEFORE YOU BEGIN!