WeChallenge Review Application

Wongnai has built a large review dataset and food dictionary that everyone can utilize from. We came up with an idea to build a web application that meets the business requirements as follows.

1. We can read a review by specifying review ID through the web application and the correct review must be shown.
2. We can search for reviews by specifying a query. Only the food keywords in **food_dictionary.txt** are searchable. The application returns all matching reviews and also **highlights the keyword** in each review. If no matches are found, the system may return an empty result or show an error.
3. When a review is displayed, users are able to edit its description. The system always remains in a consistent state after processing **concurrent** editing requests on the same review. For example,
    - If two users are updating the same review (PUT request) with different content at the same time, the review description is expected to be updated according to the last processed request.
    - We haven't yet designed the expected behavior when a write conflict occurs. Whoever develops this application may define it on their own.

Your assignment is to build a web application that satisfies all the requirements above.

Datasets

test_file.csv
https://github.com/wongnai/wongnai-corpus/tree/master/review

Column	Type	Description
ReviewID	**Integer**	**ID of the review, unique across the dataset**
Review	**String**	**Review text/description, can span for multiple lines.**

food_dictionary.txt **(Use only first 20,000 rows)**
https://github.com/wongnai/wongnai-corpus/tree/master/search

Column	Type	Description
Keyword	**String**	**Food keyword**

API Specification

Get a review by a specific ID

Method: GET
Path: /reviews/:id
Response: Design your own response body

Search for reviews by a query (Food text)

Method: GET
Path: /reviews?query=:text
Response: Design your own response body, a matching keyword must be highlighted using the <keyword> tag. For example, given a query `fried rice`, the review description must be: `"the review body with <keyword>fried rice</keyword> matched"`

Editing a review

Method: PUT
Path: /reviews/:id
Request Body: Design your own request body
Response: Design your own response body

Technical Requirements

- Your project must be built from scratch. No limitation for technologies to be used. However, we prefer the following languages and frameworks for the backend side.
  - Java with Spring
  - Python with Django
  - Node.js with Express.js
  - Go with stdlib
- Any Frontend technologies and frameworks are allowed. A HTML + CSS + Vanilla JS is also acceptable. There are no scores for beautifulness, so don't spend much time on it, just make it secure and work.
- Use git for your project, commit wisely, make your commit history clean. You can use any branching strategy.
- Write Unit and Integration tests, aim for most coverages, both line and branches. Write tests for server side only.
- The project will be deployed using docker and docker-compose. Write docker and docker-compose so we can check it easily.
- Please maximize system performance as much as you can like you are building a system for the real world where the dataset could be larger than what is provided here. e.g. millions of food names and reviews.

Score Criteria (100 marks)

- All API must be satisfied per API specification. (10 marks)
- System basic functions are working as expected (15 marks)
- Code Readability. (5 marks)
- Project documentation. (e.g. diagram, API spec) (5 marks)
- Data structure & Design pattern. (10 marks)
- Unit test coverage. (20 marks)
- The system always remains in consistent state (20 marks)
- Performance of the system (e.g. execution time). (15 marks)
- The frontend technique is an extra score, again no score for beautifulness. (up to 10 marks)

*Total score maximum is 100 only if you get full marks for all criteria including the frontend technique.

How We Test

- We'll run the following command:

```
docker-compose -f docker-compose.yml up -d
```

- Then, we'll enter the website using this URL ONLY: http://localhost:5555 So, please make sure the server listens to this port and serves the correct index page.
- We'll run load testing on your Search and Get Review endpoints, the two-step combined as one transaction. Your system must satisfy at least 20 TPS steady for 60 seconds
    - Load testing environment: AWS EC2 m5.large
    - Testing Tools: Apache JMeter or similar at 100 concurrent connections
- We'll run load testing on your Editing Review endpoint and check the result against our internal dataset. The edited review must be searchable using a food keyword that exists in the body. Your system must also highlight and show the correct review body.