Word processor – Cyolo home assignment – Guy Paz

Task description\requirements were:

Please write a service in the language you're most comfortable with that consists of 2 endpoints:

- The first endpoint should accept a string of comma-separated words.
- The second endpoint should return stats about the frequency of all the words received in total:
  - Top 5 recurring words
  - Minimum frequency among all words
  - The median frequency

To store in-memory, allow concurrency and that "sending words to the server seems to happen much more than getting the stats"

From questions during development the following was decided:

Service endpoints:

- Service should respond and get requests in Json format.
- Service should be able to handle multiple requests from multiple senders. The storage of information can take an "eventual-consistency" approach. This has the following implications:
  - Word data can be sent to the server, acknowledged but might not be stored immediately.
  - Requests for stats might display stat about data that is a little stale, as some of the incoming data might not have been completely saved yet.
- Both endpoints were developed using Rest API, I considered websockets and gRPC. They were not chosen as I wanted to have a lean service that:
  - does not need to maintain a connection like in websocket, that might affect my server resources
  - support multiple clients
  - is relatively open and does not need client implementation like gRPC.
- All service responses are returned with a request id UUID string, this should allow a programmer/user to track problems in log, and in future versions might enable user to verify its word data was actually added. (see appendix)
- Error responses are returned with a non-verbose, short description. (see appendix)
- Summary endpoint will also return the various stat required in addition to request-id.(see appendix). Also returns timestamp as date might be stale, and client would want to know what was data at certain time
- All endpoint requests must have a header of "token" with value "cyolo" this is sort of a security addition as we might not want anybody to get stat/add words. In real world solution this would have been something else, an HTTPS implementation or\and an OAuth based access.
- Endpoint were developed using a modified CRUD approach of "/words/" endpoint. POST request in order to add words. GET request in order to read summary. (see appendix)

General Service info:

- Service was written in Python using FastAPI mainly for its async features. Service is an “eventual consistency” approach. I thought it was ok to not choose a synced approach, get requests from user, ack with 202, and then do the processing offline in an async matter. This can handle multiple requests with relatively little connection maintenance, and do the necessary work when possible, without “hanging” the client.

Data Storage:

- Data was stored in a python dictionary.
    - Data insertion per word is in o(1) or o(k) per list of k words
    - Assuming these are real words they were not hashed in an attempt to save service space, it is possible to hash the words, if in real world application we would get longer “words” this will require us to save only the real top5 words.
    - Data insertion was locked in order to avoid any problematic changes to the counter of the words.
    - Data retrieval was not locked as we took an “eventual consistency” approach that can get stale data at certain points. This will eventually fix itself.
    - If a word is repeatedly sent to us and is overflowing the python variable, I decided to reset its counter to 1
- On a summary retrieval operation, the data must be sorted by frequency, this leads to a o(nlogn) for this operation.
- Data storage and data service were split in order for us to have a way in the future to move from an im-memory solution to something else, like redis. Data service should not change dramatically.

Testing:

- Unit tests are done on the word service. As for the main rest endpoint I decided to take an e2e approach that adds then gets word summary in order to check the various API mechanism.
- Off course the app can also run locally and verified with following curl commands:
- For adding words:
    - curl --location --request POST 'http://127.0.0.1:8080/words/'
      --header 'token: cyolo'
      --header 'Content-Type: application/json'
      --data-raw
      '{"words":["a","a","plus","a","plus","a","plus","fffff","fffff","b","ddd","b","ddd","b","a","not"]}'

      This will lead to a response like this: {"request_id": "d7e98422-4611-43bf-a33f-e35ecd6127e7"}

- For getting summary:
    - curl --location --request GET 'http://127.0.0.1:8080/words/' --header 'token: cyolo'

this will lead to a response like this: {

```
    "request_id": "4b454c00-9865-4e4e-8210-e59754d82caf",
    "top5": {
        "a": 10,
        "plus": 6,
        "b": 6,
        "fffff": 4,
        "ddd": 4
    },
    "least": 2,
    "median": 6,
    "timestamp": "2023-12-22 14:02:48.460446+00:00"
}
```


If header is not sent -> response: {

```
    "request_id": "b082dcb9-3a32-41fa-a062-38eefc01660b",
    "error": "unauthorized request"
}
```

Appendix – Response\Requests format and API:

1. Word addition
    a. API – POST http://{server-ip}:{port}/words
    b. Request format:
        i. JSON - `{"words":[List of strings]}`
        ii. Headers:
            1. Header name: token
            2. Header value: cyolo
    c. Response format:
        i. JSON - `{"request_id": UUID}`
        ii. Status code 202
2. Word stat summary
    a. API – GET http://{server-ip}:{port}/words
    b. Request format:
        i. Headers:
            1. Header name: token
            2. Header value: cyolo
    c. Response format:
        i. JSON - `{"request_id": UUID, "top5": {up to 5 K, V pairs key string, value number},"least": Number, "median": number, "timestamp": "UTC Format time string"}`
        ii. Status code 200
3. Error responses are returned with the appropriate status code and the following error response format: `{"request_id": UUID, "error": string}`