Take Home Assignment

This assignment consists of the following 1 Optional and 4 mandatory parts

1. Coding Assessment – Python

Create a FastAPI application which will have the following endpoints:

a) **Create User** – Returns a UUID for the user and an autogenerated JSON Web Token for that user. Input parameter for this endpoint will only be the username. This endpoint should return the following:
{
"username" : <username given as input to the endpoint>,
"user_id": <UUID for the user>,
"jwt": <JSON Web token to authenticate the user for using other endpoints>
}

b) **Word Count:** Once you authenticate with the JWT received from Create User endpoint, you will now upload a text file. This endpoint will then return the word count for the text file in a JSON format. For example,
{
"file_id": <autogenerated-id-of-the-file>,
"word_1": count of word_1 in the text file (integer),
"word_2": count of word_2 in the text file (integer),
…… and so on

}
Note: You will have to clean the text file to remove punctuation, set all words to lower case, do stemming/lemmatization, etc. The steps for cleaning up of the text files is up to you to decide.

c) **Get User stats:** You will have to use the JWT to access this endpoint. This will return the count of total files uploaded by the user and the total words in all the files combined. For example, if user 1 uploaded file_1 with total 100 words and file_2 with total of 32 words, this endpoint should return:
{
"total_files_uploaded" : 2,
"total_words_counted": 132
}

d) **Get File Level stats:** You will have to use the JWT to access this endpoint. This will return the count of word count by file_id uploaded by the user. The endpoint should be of the format /get_count/<file_id>. The response should be the word count of a previously uploaded file in step b.
{
"word_1": count of word_1 in the text file (integer),
"word_2": count of word_2 in the text file (integer),
}

In case the file_id is invalid, please send an appropriate response accordingly.

Please feel free to use any format for the endpoints and responses. These guidelines are just for reference. Also, you are free to use any NLP techniques for cleaning the text files.

References:

- https://fastapi.tiangolo.com/tutorial/
- https://realpython.com/fastapi-python-web-apis/
- https://pyjwt.readthedocs.io/en/latest/
- https://christophergs.com/tutorials/ultimate-fastapi-tutorial-pt-10-auth-jwt/
- https://fastapi.tiangolo.com/tutorial/response-model/
- https://progressivecoder.com/a-guide-to-fastapi-request-body-using-pydantic-basemodel/

2. Database

You will need to set up a DB to store the data for the FAST API application. You can use SQLite or Postgres or any other DB of your choice. The database should store the following information

- User name, User ID (UUID generated by Create User Endpoint)
- Files uploaded by a user and word count for each file

You are free to design the Database schema however you see fit. If you feel that you require additional tables, please free to create them. Please note that you will have to explain the schema design in the subsequent interview rounds.

References:

- https://fastapi.tiangolo.com/tutorial/sql-databases/
- https://www.fastapitutorial.com/blog/database-connection-fastapi/
- https://towardsdatascience.com/fastapi-cloud-database-loading-with-python-1f531f1d438a

3. Docker

You will need to create a DockerFile to load your FastAPI application and setup the Database. Please add a ReadMe file with steps to replicate the setup.

References:

- https://fastapi.tiangolo.com/deployment/docker/
- https://www.jeffastor.com/blog/pairing-a-postgresql-db-with-your-dockerized-fastapi-app

4. CI/CD Setup

Create a Github repository for your application which should contain the following:

1. Code for your FastAPI application
2. DockerFile, along with a ReadMe on how to start the application
3. Tests for your applications (you can use pytest or any other test library of your choice)
4. Setup Github Actions to run the tests after every commit.
5. (Optional) Add Github pre-commit actions for linters, formatters, etc.

References:

- https://fastapi.tiangolo.com/tutorial/testing/
- https://www.fastapitutorial.com/blog/unit-testing-in-fastapi/
- https://medium.com/fastapi-tutorials/testing-fastapi-endpoints-f7e78f09b7b6
- https://thiagolopessilva.medium.com/running-unit-testing-on-github-action-using-pytest-61653d993c9c
- https://blog.dennisokeeffe.com/blog/2021-08-08-pytest-with-github-actions

5. User Analytics (Optional)

Please note that this part is optional to this assignment. In this part you have to report the user analytics for the files uploaded to the API. Some examples include,

- Adding an endpoint that creates various charts/statistics on the user files and allows to download the information as PDF files.
- Setup up a Dashboard to visualize user information (use any technology of your choice).

Feel free to add anything to the existing project to better visualize this information. The above examples are just for reference.

Evaluation Criteria

1. Code is clean with relevant variable names, function names, comments etc.
2. The ReadMe file is self-explanatory, and the Dockerfile runs without any issues.
3. Tests and Github actions work properly
4. You can explain the design decisions for your application

Note:

1. Please feel free to use any library of your choice.
2. You are encouraged to add Python Linters/formatter like black, isort, etc. to the GitHub actions as well, but it is not mandatory.
3. Please revert to us in case you have any questions about the assignment. You are free to design the application however you see fit, as long it fulfills the basic requirements.