AI Python Engineer Challenge

Welcome to our Deel Python AI Engineer challenge. Thanks so much for taking the time, it helps us to get a picture of the way you work and tackle a certain problem.

As discussed on the intro call, from the moment when you've received that email, you must complete the task and send us the results back within **48 hours (unless agreed otherwise)**. The assignment should take no more than 4 hours of your time.

Please show all your work (including your code), assumptions, and design choices.

You can use ONLY PYTHON for this task and you are welcome to use the framework that you feel most comfortable with.

Good luck!

Business context

Deel handles millions of payments every month, ensuring that customers receive their money in a fast and accurate manner. Deel uses the description field on the payment transactions to inform payment handling and automation.

Problem

Recently, Deel has noticed changes in the way the description field is populated. They have asked you to improve the automation on this dataset using text processing techniques.

Relevant files:

transactions

users

Task

Overview

Create solutions to the following tasks, you are free to use any publicly available Python library. Your code will be evaluated on its efficiency, choice of tools, respect of coding best practices, and of course **the methodology used**.

Please include a README.md file explaining how to run your code.

For tasks 1 and 2, please include a brief discussion of the solution and its possible limitations (in the README/pdf/keynote or otherwise).

Task 1

Implement a simple Python API with an endpoint which takes a transaction id as input (str), and returns the users which could be considered to match the transaction. This endpoint will be used to match users with their payments.

As with any real-world problem, you should try to account for edge cases such as typos, and check both inexact and exact matches. You may assume the input name will consist of latin alphabet characters.

The response should be returned in the following format, with users sorted in order of relevance. You should also include the total number of returned matches as a separate key, and a metric of ‘how well each transaction matches”.

```
{
        users: [{id, match_metric}, {id, match_metric}, ...],
        total_number_of_matches: int
}
```

Task 2

Extend this API with another endpoint, which takes a string as input, and returns all transactions which have "similar" descriptions to this string in a semantical way (meaning, you should use some sort of language model embeddings).

The response should be returned in the following format, with transactions sorted in order of relevance. You should also include the total number of input tokens used to create the embeddings for the tokenizer you use.

```
{
        transactions: [{id, embedding}, {id, embedding}, ...],
        total_number_of_tokens_used: int
}
```

Task 3

Given additional resources, suggest (in the README/pdf/keynote or otherwise), how you might take this proof of concept to production. Include any changes or improvements you might make.