|  |  |
| --- | --- |
| Full Name : | **Duta Kukuh Pribadi** |
| Position : | **Artificial Intellegence Engineer** |
| Start Date : |  |

## 

## **Enhanced Quiz Test for AI Developer: Gemini/OpenAI API Integration**

## **Task**

**Create a Python-based application that:**

* Leverages the Gemini API to generate text based on user prompts.
* Implements asynchronous API calls for improved performance.
* Allows for fine-tuning of API parameters (e.g., temperature, max\_tokens) based on user preferences or specific use cases.
* Integrates the API functionality into a Flask or FastAPI backend.
* Containerizes the application using Docker.

### **Code Structure**

**Python**



import openai

import asyncio

from fastapi import FastAPI, Request, HTTPException

from aiohttp import ClientSession

# Replace with your actual API key

openai.api\_key = "YOUR\_API\_KEY"

app = FastAPI()

async def generate\_text\_async(prompt, api\_params):

# Implement asynchronous API call using aiohttp

# ...

@app.post("/generate")

async def generate\_text\_endpoint(request: Request):

# Parse request body for prompt and API parameters

# Call generate\_text\_async

# Handle errors and return response

# ...

### **Questions**

1. **Asynchronous API Calls:**
   * Implement the generate\_text\_async function using asyncio and aiohttp.
   * Explain the benefits of using asynchronous calls in this context.
2. **Parameter Tuning:**
   * Allow users to specify API parameters (e.g., temperature, max\_tokens) through the API endpoint.
   * Provide default values and validation for parameters.
   * Explain how different parameters affect the generated text.
3. **Backend Integration:**
   * Integrate the generate\_text\_async function into the Flask or FastAPI app.
   * Create a REST API endpoint to handle user requests.
   * Implement error handling and proper response formatting.
4. **Dockerization:**
   * Create a Dockerfile to build the application image.
   * Define necessary dependencies and environment variables.
   * Explain the benefits of using Docker for deployment.
5. **Code Optimization:**
   * Identify potential performance bottlenecks and suggest optimization techniques.
   * Consider using asynchronous libraries for other computationally intensive tasks.

### **Evaluation Criteria**

* Correctness of the code
* Efficiency of the code and API usage
* Code readability and maintainability
* Error handling and robustness
* Depth of understanding of asynchronous programming, backend development, and Docker
* Ability to explain the rationale behind design choices

### **Additional Challenges**

* Consider implementing a caching mechanism for API responses.
* Explore techniques for handling API rate limits and retries.
* Implement security measures to protect API keys and user data.

**Evaluation Criteria:**

**Focus on these aspects while building the application:**

* Correctness: Ensure the code functions as intended and integrates APIs successfully.
* Efficiency: Optimize API usage and implement caching when possible.
* Readability: Use clear variable names, comments, and proper formatting.
* Error Handling: Implement robust error handling to provide informative messages to users.
* Understanding: Demonstrate understanding of asynchronous programming, backend development, and Docker.
* Design Choices: Explain your design decisions for different parts of the application.

## 

## **Tick Tock!**

## You've got a *one-day* showdown with this quiz! As soon as it lands in your inbox, the clock starts ticking. Don't let those brain cells gather dust – crack open those books and dive in!

## And remember, when you're ready to conquer the world with your knowledge, bundle up your answer code in a snazzy zip file and send it soaring through cyberspace!