Assignment: Backend + Gen AI Engineering Challenge

Objective:

Build a backend service in Python that integrates a Generative AI model to solve a text-processing problem (e.g., summarization, transformation, Q&A), with proper infra setup including API design, background job processing, and caching. The goal is to test your backend design, AI integration, and infrastructure knowledge.

Steps:

1. **Problem Design:**
    - Define a use case (e.g., text summarizer, question-answer bot, tone rewriter, etc.) using a generative AI model.
2. **API Implementation:**
    - Build clear RESTful API endpoints using a Python framework - FastAPI
    - Ensure endpoints validate input and return appropriate responses.
3. **Gen AI Integration:**
    - Integrate a Gen AI model (e.g., OpenAI API, HuggingFace model).
    - Prompt the model dynamically and return relevant output.
4. **Async Job Queue:**
    - Offload AI-related tasks to a background worker using Celery/RQ.
    - Provide endpoints to submit a task and retrieve its status/result asynchronously.
5. **Caching Layer:**
    - Implement a cache to store repeat request results using Redis or in-memory cache.
    - Explain your caching logic in the README.
6. **Containerization:**
    - Include a Dockerfile to containerize the entire service.
    - Optionally include a docker-compose setup if you're using Redis/queue workers.
7. **Deployment:**
    - Deploy the service using any free-tier or preferred hosting platform (e.g., Vercel for frontend, Render/Fly.io for backend).
8. **Submission:**
    - Submit a GitHub/GitLab repository link with the following:
        - Working codebase
        - README with setup instructions, usage guide, API reference, architecture explanation, and any assumptions
        - Example input/output pairs
        - Docker setup and usage instructions

Evaluation (Total: 30 Marks)

- Code Quality: 5 Marks
- Timely Submission: 10 Marks
- Functional Application & Deployment: 10 Marks
- Clarity of Documentation: 5 Marks

Notes:

- Pick a Gen AI use case that excites you.
- Design the system like it could serve 1000s of requests per day.
- Focus on real-world scalability, usability, and clean system design.

Good luck and happy building!