Practical Task: Smart Knowledge Assistant API Platform

Definition:

Design and develop a secure, modular, and scalable backend system that allows users to upload knowledge documents, generate a custom knowledge base using vector embeddings, and retrieve contextual answers using LLMs. The system will support asynchronous background processing, secure authentication, request validation, logging and developer-friendly documentation.

Objective:

Build a secure, scalable backend API service that allows users to:

- Upload knowledge documents (PDFs, TXT, or Markdown),
- Create a searchable knowledge base,
- Ask questions to an LLM and get context-rich answers using RAG (retrieval augmented generation)
- Perform background indexing,
- Log API usage and errors,
- Authenticate via JWT + OAuth2,
- Document the API with Swagger,
- Implement unit tests for core functionalities.

Requirements Breakdown:

1. Frameworks

- **Django + DRF / FASTAPI** for API development.
- **Swagger/OpenAPI** for API documentation

2. Authentication

- Implement **JWT-based authentication**
- Also include **OAuth2 (Google)** login for user authentication(Only API no need of UI).

3. Knowledge Base & Retrieval (GenAI)

- Users upload knowledge files (PDF, TXT, MD).
- Parse and chunk the content into vector embeddings using **LangChain + FAISS/ OpenSearch**(not AWS OpenSearch).
- Use **Google Gemini LLM** models to answer user questions from uploaded content.

4. Asynchronous Task Processing for KB creation

- Use **Celery + Redis** to handle:
    - File parsing
    - Embedding creation
    - Indexing into vector DB

5. API Features

- User Registration/Login (JWT & OAuth2)
- Upload file → triggers async embedding job
- Ask question → retrieves relevant context + answer from LLM
- View query history
- Error handling/logging for each step

6. Validations & Error Logging

- Use **custom validators** to ensure supported file formats.
- Implement **custom exception handling middleware**.
- Error Logs (in file with rotating cycle).

7. Bonus (Optional)

- Add **RBAC**: Admin vs. user (e.g., admin sees all questions).
- Add usage limits: e.g., only 20 questions/day for free users.
- Implement **basic guardrails** (e.g., block offensive content in queries/responses).

Submission Guidelines

- A GitHub repository with:
    - Clean, modular code
    - Setup instructions (`README.md`)
    - Sample `.env.example`
- Postman collection or Swagger UI enabled

Note: So, dear candidate, please **do not refer to any ready-made materials such as GitHub repositories or blogs while working on this task.** The objective is to develop a secure, modular, and scalable backend system from scratch, ensuring that you apply your own understanding and skills to meet the outlined requirements. **You may use any LLM app like ChatGPT, Google Gemini, etc., to generate code but manage the code on your own.**

Good luck!