Objective:

Develop a Python service that intelligently manages and retrieves information from a collection of domain-specific documents related to upcoming concert tours in 2025-2026. The service will enable users to ingest new documents containing tour plans, schedules, venue details, performer information, and logistical notes. To ensure accuracy and relevance, the service's responses will be strictly grounded in the provided documents rather than relying on general knowledge.

Core Functionalities:

1. Document Ingestion:

Input: The service will receive a user command containing a new concert tour document (plain text).

Processing:

The service must first analyze the document to determine if it falls within the scope of the concert tour domain (e.g., checking for keywords related to concerts, tours, venues). If the document's theme is outside this domain, the service should return an informative message to the user, such as "Sorry, I cannot ingest documents with other themes."
If the document is relevant, the service will generate a concise summary of its content.
The service will then ingest this summary into a Retrieval Augmented Generation (RAG) system for efficient information retrieval.
Finally, the generated summary will be sent back to the user as a confirmation.

Output: The service will return a summary of the ingested document to the user.

Example:
User Query: "Please, add this document to your database: [Some document about 2025–2026 concert tour, including dates, venues, special guest appearances, and logistical notes]."
Output: "Thank you for sharing! Your document has been successfully added to the database. Here is a brief summary of the data from the document: [summary]"

2. Question Answering:

Input: The service will receive a user query seeking specific details about upcoming concert tours.

Processing:

The service will leverage the RAG system to search the ingested documents for information relevant to the user's query.
Based on the retrieved documents, the service will generate a precise and informative answer.

Output: The service will provide an answer that is strictly grounded in the information contained within the ingested documents.

Example:
User Query: "Where is Lady Gaga planning to give concerts during autumn 2025?"
Output: "She plans to go to [list of the cities and other information about the tour]"

(Bonus) UI Implementation: (Optional, but highly encouraged)
A user interface (UI) to facilitate interaction with the bot. A framework like Streamlit is recommended for simplicity.
The UI should include:

- A text input field for user requests (e.g., for adding documents or asking questions).
- A display area to present the bot's responses in a user-friendly format.

(Bonus – Optional) Dual Functionality
Implement an additional mode to enhance the bot's usefulness:
If a user **does not upload a document but provides a musician or band name**, the bot should:

- Perform an online search (e.g., via SerpAPI, Bing API, or another search engine wrapper)
- Retrieve details about the artist's upcoming concerts from publicly available sources
- Generate and return an answer based on this search result.

WARNING: This feature is optional, but implementing it will earn bonus points for creativity and utility. Make sure this feature is clearly separated from the core (RAG-only) functionality.

Turning in the completed task

To turn in the completed test task, create a private GitHub repository with *ProvectusInternship_'Your name and surname in CamelCase'* and give us access.

Be sure to include:
Source Code - All necessary scripts, notebooks, or projects required to run the solution.
README.md file in your repository describing your approach and design choices, detailed setup instructions for the service, and how to run it.

You should've got this document as a part of the email inviting you to solve the test task. Reply to this email and include the URL of your repository.