📄 Take-Home Assignment: GenAI-Powered Document Chatbot

Overview:

We'd like you to develop a simple **web-based chatbot application** that allows users to upload documents (PDF, Word, and CSV files), processes and chunks their content, stores the embeddings in a vector database, and enables users to ask questions about the uploaded content via a chat interface.

This assignment is designed to assess your **full stack development skills**, **GenAI integration capability**, **prompt engineering discipline**, and awareness of challenges like **hallucination control**.

✳️ Requirements:

Front-End:

- Build a clean, responsive React-based interface.
- Provide functionality for users to:
    - Upload PDF, Word (.docx), and CSV files.
    - View a chat interface to interact with a chatbot.
    - See a display of uploaded document titles.
    - Reset uploaded documents and conversation.

Back-End:

- Implement a Node.js and/or Python-based backend.
- Process uploaded documents:
    - Extract text from PDF, DOCX, and CSV files.

    - Chunk text into manageable segments.
    - Compute embeddings using a suitable embedding model (e.g., OpenAI, Hugging Face transformers).
- Store embeddings in a vector database (e.g. **ChromaDB**, **Pinecone**, **Weaviate**, or **FAISS**).

Chatbot:

- Implement a chat interface for querying uploaded documents.
- Retrieve relevant chunks from the vector database based on user queries.
- Use an LLM API to generate responses based on retrieved context.

✳ Prompt Engineering Guidelines:

- Demonstrate techniques to **minimize hallucinations** by constraining the model's answers to retrieved document context.
- Include system prompts that explicitly instruct the model to answer only from provided context.
- Gracefully handle out-of-scope questions.

Example system prompt:

"You are an AI assistant answering user questions based only on the provided context. If the answer isn't in the context, respond with 'I'm sorry, I don't have information about that.' "

✳ Deliverables:

- Public **GitHub repository** containing:
    - Source code for front-end and back-end.

    - Instructions to run the application locally (Docker or manual setup).
    - A `README.md` including:
        - Overview of your approach.
        - Description of your prompt engineering strategy.
        - Any limitations or assumptions.
- Link to a code walkthrough video (max 5 min) and a video of a functional code - mimic operation
- Link to a live demo (Optional).

📊 Bonus (Optional):

- User authentication.
- Support for multiple concurrent documents and conversations.
- Conversation history.
- Deployment-ready configuration (Docker Compose or similar).

📅 Time Expectation:

Estimated **8–12 hours** of focused work. Please submit within **2 days**.

✅ Evaluation Criteria:

- Code clarity, structure, and documentation.
- Proper chunking, embedding, and storage logic.
- Vector search accuracy and relevance.

- Prompt engineering quality and hallucination control.
- UI/UX simplicity and usability.
- Bonus implementations (if any).
- Clear instructions and repository setup.