Technical Assessment: PDF Chat Application with LLM Integration

Introduction

Welcome! This technical assessment is designed to evaluate your ability to work with modern AI tools, specifically Large Language Model (LLM) APIs, and integrate them into a practical application. The goal is to build a web application that allows users to "chat" with a provided PDF document.

This task will touch upon skills including:

- Interacting with external APIs (LLMs, Embedding Models)
- Text processing and data extraction (PDFs)
- Implementing core AI concepts like text chunking and vector embeddings
- Using vector databases for efficient information retrieval
- Implementing the Retrieval-Augmented Generation (RAG) pattern
- Basic web application development and data persistence

Provided Materials

1. **PDF Document:** 2025 State of Marketing Operations_Public_March 2025.pdf (Attached)
2. **This Document:** Containing the assessment instructions.

The Task

Your primary goal is to create a functional web application where a user can ask questions in natural language and receive answers generated by an LLM, based *solely* on the content of the provided PDF document. The application should also save the user's questions (prompts), and the LLM answers.

Core Requirements

1. **PDF Processing:** The application must be able to ingest the provided PDF document and extract its textual content. This might happen on startup or through a dedicated setup command.

2. **Text Chunking:** Implement a strategy to break down the extracted text into smaller, meaningful chunks suitable for processing by embedding models and fitting within LLM context windows. Common strategies include fixed-size chunks with overlap or recursive character splitting. You should be prepared to briefly justify your chosen method in the README.

3. **Vector Embeddings:** For each text chunk, generate a vector embedding using an appropriate embedding model (e.g., via OpenAI's API, Sentence Transformers library, Google's Embedding APIs, etc.).

4. **Vector Database:** Store the text chunks and their corresponding vector embeddings in a vector database. For this assessment, we recommend using a simple, local vector database like **ChromaDB** or **FAISS** for ease of setup. You are free to use other vector databases if you are more comfortable with them, but ensure setup is straightforward or well-documented.

5. **Web Application:** Build a web interface using a web framework of your choice, the only requirement is to use **TypeScript**. The interface should provide:
    - An input area for the user to type their question.
    - A display area to show the answer generated by the LLM.
    - Create threads to organize interactions.
    - A sidebar with threads. A thread is a collection of interactions, like in ChatGPT.
    - Threads can have new interactions.

6. **Retrieval-Augmented Generation (RAG):** This is the core logic. When a user submits a question:
    - Generate an embedding for the user's question using the same embedding model as used for the document chunks.
    - Use this query embedding to perform a similarity search against the vectors in your vector database. Retrieve the top N most relevant text chunks from the PDF.
    - Construct a prompt for the LLM that includes both the user's original question and the retrieved text chunks as context.
    - Send this augmented prompt to an LLM API (e.g., OpenAI's GPT models, Google's Gemini, Anthropic's Claude).
    - Display the LLM's response to the user.

7. **LLM Constraint:** Crucially, the prompt sent to the LLM must instruct it to answer the user's question **based only on the provided context (the retrieved PDF chunks)**. The goal is to chat *with the document*, not the LLM's general knowledge.

8. **Save Prompts:** Persist the original questions (prompts) entered by the user. A simple approach using a database of your choice is sufficient for this assessment.

Technology & API Access

- **LLM & Embedding Models:** You can use APIs from providers like OpenAI, Google (Gemini), Anthropic, or others. Please use your own free-tier or development API keys for this assessment. Alternatively, you can explore using locally-run models (e.g., via Ollama) and open-source embedding libraries (like Sentence Transformers) if you prefer. Clearly document which models/APIs you used. For interacting with AI models, we

recommend the ai SDK from Vercel.

- **Vector Database:** As mentioned, ChromaDB or FAISS are recommended for simplicity but you are free to use one of your choice.

- **Web Framework & Language:** You have flexibility here. Choose tools you are comfortable with. We recommend Next.js with Server Actions and API routing. You are free to create a Node.js backend or use Next.js server capabilities.

- **PDF Library:** Use any suitable library for extracting text from the PDF. We recommend looking into LangChain TypeScript SDK.

Deliverables

Please provide the following:

1. **Source Code:** A link to a Git repository (e.g., GitHub, GitLab) containing the complete source code for your application.

2. **README File:** A README.md file in the root of the repository that includes:
    - **Technology Stack:** List the main libraries, frameworks, LLM/embedding models, and vector database used.

    - **Setup Instructions:** Clear steps on how to set up the project environment (e.g. package.json for Node.js/Next), including how to configure any necessary API keys (e.g., using environment variables).

    - **Running Instructions:** How to start the application, including any initial data processing or indexing steps (like processing the PDF and populating the vector DB).

    - **Chunking Strategy:** A brief explanation of the text chunking strategy you implemented and why you chose it.

Evaluation Criteria

We will evaluate your submission based on:

- **Functionality:** Does the application work as described? Does it accurately answer questions based *only* on the PDF content? Are prompts saved?
- **UI/UX:** The app should be considered as a high end product thus having a smooth, fluid and great user experience. Think about loading states, feedback, animations and transitions

- **Correctness of Concepts:** Is the RAG pipeline (chunking, embedding, storing, retrieving, prompting) implemented correctly?
- **Code Quality:** Is the code well-structured, readable, and reasonably maintainable?
- **Documentation:** Are the setup and run instructions in the README clear and effective?

Questions?

If you have any questions or need clarification on any part of this assessment, please don't hesitate to reach out to Said and Jonathan.

We look forward to seeing your solution. Good luck!