Full-Stack & AI Systems Design Engineer: Take-Home Assignment

Our Philosophy
At our core, we are a company focused on first-principles thinking and solving deeply complex problems. We believe the most impactful engineers are not just coders, but architects who can design robust, scalable, and elegant systems from the ground up.

This assignment is intentionally open-ended. It is designed to move beyond typical CRUD applications and test your ability to think critically about system architecture, data flow, AI integration, and user experience in a real-world context. We want to see how you reason about trade-offs, how you structure a complex project, and what technical choices you make when given the freedom to design the optimal solution.

Project Vision: The "Second Brain" AI Companion
The challenge is to design and build a foundational prototype for a "second brain"—a personal AI companion that can ingest, understand, and reason about a user's information. The goal is to create a system where a user can have a natural language conversation and receive intelligent, human-like responses based on a history of provided documents, audio, and web content.

Imagine a system that has perfect memory of everything you've shown it. You should be able to ask it, "What were the key concerns raised in the project meeting last Tuesday?" or "Summarize the main points from that article I saved about quantum computing," and get a fast, accurate, and synthesized answer.

Detailed Requirements

The assignment is broken into three parts, with the heaviest emphasis placed on Part 1. This assignment is to test your understanding of the different parts under a strict 48 hr timeline. We expect you to use any available resources and/or tools.

Part 1: System Design & Architecture (The Primary Focus)
Before writing a single line of implementation code, you must produce a detailed system design document. This document is the most critical deliverable and should clearly articulate your architectural vision. It must include diagrams, schema definitions, and written explanations covering the following:

1.1. Multi-Modal Data Ingestion Pipeline:
Detail the architecture for ingesting and processing a variety of data types ("modalities").

- Audio: Transcribe spoken words from audio files (e.g., `.mp3`, `.m4a`).
- Documents: Extract text and relevant metadata from files like `.pdf` and `.md`.

- Web Content: Given a URL, scrape and process the text content of a web page.
- Plain Text: Handle raw text inputs or notes.
- Images: Propose a method for storing images and making them searchable via associated text or metadata.

1.2. Information Retrieval & Querying Strategy:
This is the core of the "brain." You must document your chosen approach for finding the most relevant information to answer a user's query. Justify your choice. This strategy could involve:

- Semantic search (using vector embeddings).
- Keyword-based search (like full-text search).
- Graph-based systems (using nodes and relationships).
- A hybrid approach combining multiple techniques.

1.3. Data Indexing & Storage Model:
Describe the full lifecycle of a piece of information.

- How is raw data processed and "chunked" for indexing?
- What indexing technique will you use?
- What does the database schema look like? What metadata is stored alongside the content (e.g., source, type, timestamp)?
- What are the trade-offs of your chosen storage solution (e.g., SQL vs. NoSQL vs. Vector DB) in terms of scalability, cost, and query flexibility?

1.4. Temporal Querying Support:
The architecture must explicitly support time-based queries. Your design document must explain how you associate timestamps with all ingested data and how your retrieval strategy will use this information to answer questions like, *"What did I work on last month?"*

1.5. Scalability and Privacy:
Briefly discuss how your design would scale to handle thousands of documents for a single user. Address privacy by design, explaining the trade-offs between a cloud-hosted solution and a potential local-first approach.

Part 2: Backend Implementation

Based on your design, build the core backend services.

2.1. Data Processing Pipeline:

Implement an asynchronous pipeline that can process at least two modalities (Audio and one other, like Documents or Web Content).

2.2. Intelligent Q&A Service:
Create an API endpoint that takes a user's query. This service should:

1. Execute your designed retrieval strategy to fetch relevant context from the knowledge base.
2. Use an LLM (e.g., via the OpenAI, Anthropic, or Google AI APIs) to synthesize the retrieved context and the user's query into a concise, accurate answer.

Part 3: Frontend Implementation

Develop a minimal but clean user interface to interact with your system.

3.1. Chat Interface:

- A simple web page with a text input for the user to ask questions.

3.2. Responsive Interaction:

- The interface should display the AI's response. For a more human-like feel, consider streaming the response token by token as it's generated by the LLM.

Deliverables

1. System Design Document: A comprehensive PDF document covering all points in Part. Diagrams are highly encouraged.
2. Source Code: A link to a well-documented code repository (e.g., on GitHub).
3. Working Demo: A hosted URL for the final application.
4. Video Walkthrough: A short (5-10 minute) video where you demonstrate the application and, most importantly, walk through your architectural decisions and the trade-offs you made.

Evaluation Criteria

You will be evaluated on the following, in order of importance:

1. Architectural Rigor: The depth, foresight, and clarity of your system design. We are looking for a robust and scalable architecture that is well-justified.
2. Problem-Solving & Justification: The quality of your reasoning behind technical choices and trade-offs. Why did you choose this database? Why this retrieval strategy?
3. Code Quality & Craftsmanship: Clean, efficient, well-documented, and testable code. Your implementation should reflect the elegance of your design.
4. Functional Correctness: The application performs as required, is robust, and handles errors gracefully.

5. User Experience: Though the UI is expected to be simple, the interaction should feel intuitive and responsive.

We are excited to see how you approach this challenge. Good luck