AI Full Stack Developer, Take Home Assessment

💼 Use Case

Many teams in fintech and research need a way to quickly understand documents without manually reviewing them. In this assessment, you'll build a lightweight chat app where users can upload PDFs or Excel files and then ask natural language questions about the contents.
This simulates how a full stack AI system can help teams interact with unstructured documents via a conversational interface.

📥 Introduction

You are tasked with building a chat-based document assistant. The user uploads a document (PDF or Excel), your system extracts and embeds the content, stores it in a vector database, and allows the user to ask questions via a chat interface.

<u>You'll implement:</u>

1. A frontend for uploading files and chatting
2. A backend that parses documents, stores embeddings, and handles AI-powered query responses
3. A vector database (e.g., Pinecone, Weaviate, FAISS) to store and retrieve document chunks

⏱️ Time Frame

- Estimated build time: 1 to 3 hours of focused work
- Submission deadline: Within 24 hours after receiving the assessment

✅ Your Mission

Build a basic end-to-end app with the following flow:

1. User uploads a PDF or Excel file
2. Your backend parses, chunks, and embeds the content
3. Chunks are stored in a vector DB
4. User can ask questions via a chat interface
5. Your system retrieves relevant chunks and generates an answer using an LLM

💬 Example User Queries

- "Summarize this document."
- "What was the total revenue mentioned?"
- "List all the key dates from the Excel."
- "Compare loan amounts across clients."
- "What are the most frequent terms in this contract?"

💡 *Tip: Include a sample PDF or Excel file with test data in your repo.*

AI Full Stack Developer, ASSESSMENT - Page 2

🧱 System Requirements

1. Frontend

- File upload UI for PDF and Excel (XLSX) documents
- Chat interface that allows the user to type questions and receive answers

Use a modern web framework:

- React (preferred)
- TypeScript (preferred)
- You may use UI libraries like Tailwind, shadcn/ui, or component packages

Optional UX features:

- Display source snippets or highlighted context
- Show loading states, validation, and error messages

2. Backend

API endpoints:

- POST /upload: Receives and processes documents
- POST /query: Accepts user question and returns AI-generated answer
- Use document parsing tools like PyMuPDF, pdfplumber, or pandas for Excel
- Use an embedding model (e.g., OpenAI, Hugging Face, etc.) to embed content
- Implement a RAG pipeline: retrieve relevant chunks from vector DB → send to LLM with context

3. Vector Database

- Store embedded text chunks with metadata (filename, page, etc.)
- Use Pinecone, Weaviate.
- Perform similarity search to fetch context based on user queries

📦 Submission Requirements

1. GitHub Repository

- Clear project structure (e.g., /frontend, /backend, /data)
- A README.md that includes:
- Setup and run instructions (frontend + backend)
- Sample queries and responses
- Your tech stack + architecture explanation
- Any trade-offs or shortcuts taken due to time limits

2. Loom Video (5–7 min)

- Demo your app end-to-end:
- Upload a file
- Ask the questions in the chat
- Explain your architecture and how the backend, AI, and frontend interact

AI Full Stack Developer, ASSESSMENT - Page 3

🧪 What We're Evaluating

Area	What We're Looking For:
AI Integration	RAG pipeline, embeddings, context retrieval
API Design	Clean, modular, testable backend endpoints
Frontend UX	Functional, clean chat and upload interface
Code Quality	Clear, readable, well-structured code
Creativity	Any extras like source highlighting, memory, etc.
Problem Solving	Graceful handling of edge cases, file types, and API failures
Documentation	Clear setup and reasoning behind your choices
Copilot Use	Mention where you used AI copilots and how they helped

Good luck – we're excited to see how you bring this to life!