AI-Powered Vault Document System: Case Study

Overview

Build an intelligent document management system where users can upload documents, get AI-powered insights, and chat with their documents using natural language. This case study focuses on creating an AI-first backend solution that enables conversational interaction with document content, automated understanding, and comprehensive monitoring.

Think of it as "ChatGPT for your documents" - users should be able to ask questions about uploaded documents and get contextual, accurate answers with source citations.

Core Objectives

1. Demonstrate ability to build AI-first backend solutions with RAG (Retrieval-Augmented Generation)
2. Show effective use of AI coding tools (e.g., Cursor, Copilot)
3. Create intelligent document processing and conversational interfaces
4. Implement async processing pipelines and monitoring APIs
5. Showcase product thinking and practical system design

Technical Requirements

Backend Components

AI-Powered Document Processing

- Intelligent Upload Handler
- Smart Document Analysis
- Asynchronous Processing Pipeline

Document Management APIs

- Upload & Ingestion
- Document Retrieval
- AI Insights

Document Chat APIs

Core Chat Functionality:

- Start Chat Session
- Ask Question
- Get Chat History
- Multi-turn Context

Metrics API Endpoints

Provide API endpoints that return monitoring data:

- Document Statistics API
- Processing Metrics API

Data Storage

Design appropriate data structures for:

- Document metadata and status
- AI-generated summaries and insights
- Document chat conversations and history
- Chat messages with context and citations
- Processing metrics and history
- System performance data

Evaluation Criteria - Equal Weightage

AI-First Development (33%)

- Effective Use of AI Coding Tools: How well you leveraged Cursor, Copilot, or other AI assistants
- AI Integration Quality: Thoughtful integration of AI for document analysis and insights
- Prompt Engineering: Effective prompts for document summarization and analysis
- AI-Driven Automation: Using AI to reduce manual work and improve user experience
- Innovation: Creative use of AI beyond basic requirements

Product Thinking (33%)

- User Experience: Intuitive APIs and data structures
- Feature Prioritization: Smart choices about what to build first
- Error Handling: Thoughtful handling of edge cases and failures
- Real-World Usability: Practical considerations for actual usage
- Solution Architecture: Well-designed system that solves the problem effectively

Technical Implementation (33%)

- Code Quality: Clean, maintainable, well-organized code
- System Performance: Efficient processing and quick responses
- API Design: Well-structured endpoints and data models
- Asynchronous Processing: Proper use of background tasks
- Documentation: Clear setup instructions and architecture explanation

Technical Stack

Required

- Python (Backend - FastAPI recommended for async support)
- PostgreSQL (Use Supabase or local instance)
- Task Queue (Celery + Redis, RQ, or similar for async processing)
- File Processing (PyPDF2, pdfplumber, python-docx, or similar)
- OpenAI API (GPT-4 + Embeddings API, or Claude - Please ask for API key if needed)
- Storage (Local filesystem or S3-compatible)

Recommended Libraries

- RAG Framework (LangChain or LlamaIndex - highly recommended for document chat)
- Embeddings (OpenAI embeddings API or sentence-transformers)
- AI Coding Tools (Cursor, GitHub Copilot, or other assistants)
- API Documentation (FastAPI auto-docs, Swagger, or Postman)
- Testing Framework (pytest)

Optional

- Frontend (React/Next.js for simple UI - optional but nice to have)
- Streaming (SSE or WebSocket for streaming chat responses)
- Caching (Redis for caching embeddings and responses)

Deliverables

6. Working Backend Application

- Note: Focus on backend/APIs - frontend/dashboard is optional

7. Source Code Repository
8. Documentation

- README.md with:

- AI_USAGE.md (or section in README):

9. Demo (Required)

   - Document chat functionality (asking questions about uploaded docs)

Notes for Candidates

- Backend/API Focus - This is primarily a backend case study. Focus on robust APIs and intelligent processing. Frontend/UI is optional.
- Document Chat is Key - The conversational interface with documents is a core feature. Implement RAG properly.
- Leverage AI Tools Actively - Use Cursor, Copilot, or other AI assistants. Document how they helped.
- Prioritize Working Features - We prefer a fully functional subset over partially implemented features.
- API Usability Matters - Even without UI, make your APIs intuitive and well-documented.
- Think About Scale - How would this handle 1000s of documents? Consider async processing and caching.
- Don't Reinvent the Wheel - Use existing libraries (langchain, llamaindex, etc.) if helpful.
- Document Assumptions - Made a design choice? Explain it in your README.
- Prompt Engineering - Show us your thought process for AI prompts, especially for document chat.

Bonus Points

Advanced AI Features:

- Multi-document chat (ask questions across multiple documents)
- Intelligent follow-up question suggestions
- Summary customization (length, focus areas, tone)
- Automated document categorization and tagging
- Sentiment analysis or key insight extraction
- Comparative analysis between documents

Technical Excellence:

- Simple, intuitive frontend/dashboard (even basic is impressive!)
- Real-time updates (WebSocket for processing status and chat)
- Smart caching strategies (Redis for embeddings, responses)
- Vector database integration (Pinecone, Weaviate, or pgvector)
- Streaming responses for chat (SSE or WebSocket)
- Comprehensive test coverage

Production Readiness:

- Cost tracking and optimization for AI API usage
- Rate limiting and quota management
- Document version tracking
- Batch processing optimizations
- Detailed logging and observability
- Health check and monitoring endpoints