

# AI Document Interview System

## 1. Project Overview

This project is an AI-powered document interrogation system that allows users to upload large collections of unstructured documents and ask natural language questions about their contents. The system is designed for organizations that have dozens or hundreds of PDFs, DOCX files, and other text-based documents that are not indexed or easily searchable, but which contain valuable institutional knowledge.

Rather than relying on keyword search or manual review, the system uses modern language models combined with vector-based retrieval to extract relevant information, present accurate answers, and cite original source documents. The system is designed to scale beyond traditional token limits while remaining auditable, deterministic, and cost controlled.

The intended use cases include internal policy review, technical documentation lookup, legal and compliance reference, historical document analysis, and general institutional knowledge retrieval.

---

## 2. Project Goals

The primary goals of the project are:

1. Enable users to upload large volumes of unstructured documents and query them using natural language.
2. Provide accurate, source-grounded answers that reference the original documents.
3. Avoid hallucination by constraining the AI to retrieved source material only.
4. Scale to document collections far larger than a single model context window.
5. Maintain a clear audit trail showing which documents were used to generate each answer.
6. Provide a clean foundation for future expansion into multi-user organizations, access controls, and advanced analytics.

Non-goals for the initial version include real-time collaboration, fine-grained role-based permissions, and end-user prompt customization.

---

## 3. High-Level System Design

The system follows a Retrieval-Augmented Generation (RAG) architecture. Documents are ingested, parsed, chunked, embedded, and stored in a vector database. When a user asks a question, the system retrieves the most relevant document chunks and supplies them to a language model as context for answer generation.

All persistence and decision-making happens in the backend. The language model remains stateless and is used strictly as a reasoning and summarization engine over provided evidence.

## Core Components

- Web UI for document upload and querying
  - Backend API server
  - Document ingestion and processing pipeline
  - Vector database for semantic search
  - Relational database for metadata and sessions
  - Object storage for raw documents
  - OpenAI API for embeddings and answer generation
- 

## 4. Technology Stack

### Backend

- Language: Python
- Web Framework: FastAPI
- Async task processing: Background workers or task queue (initially synchronous for MVP)

### Storage

- Relational database: PostgreSQL
- Users
- Documents
- Chunk metadata
- Conversation sessions
- Query logs
- Object storage: S3-compatible storage (AWS S3 or MinIO)
- Original uploaded documents
- Extracted raw text snapshots

### Vector Store

- Qdrant (self-hosted)
- Stores embeddings for document chunks
- Supports fast similarity search
- Enables metadata filtering by user and document

### AI Services

- OpenAI API
  - Embedding model for document and query embeddings
  - Chat completion model for answer generation
-

## 5. Document Ingestion Pipeline

The ingestion pipeline runs when documents are uploaded or updated.

### Ingestion Steps

1. User uploads a document via API or UI.
2. Backend assigns a unique document ID and stores the file in object storage.
3. Text is extracted from the document:
4. PDF: pdfplumber or PyMuPDF
5. DOCX: python-docx
6. OCR fallback for scanned PDFs using Tesseract
7. Extracted text is normalized to remove noise such as repeated headers, footers, and inconsistent whitespace.
8. Text is split into overlapping chunks of approximately 500 to 800 tokens.
9. Each chunk is embedded using an OpenAI embedding model.
10. Embeddings and associated metadata are stored in Qdrant.
11. Document and chunk metadata are stored in PostgreSQL.

The raw extracted text is preserved to allow re-chunking or re-embedding if models change.

---

## 6. Query and Answer Pipeline

The query pipeline runs every time a user asks a question.

### Query Flow

1. User submits a natural language question.
2. Backend creates or resumes a conversation session.
3. The question is embedded using the same embedding model used for documents.
4. The vector database is queried to retrieve the top K most relevant chunks, filtered by user ownership.
5. Retrieved chunks are assembled into a structured prompt along with the user question.
6. The prompt is sent to the OpenAI chat completion API.
7. The model generates an answer constrained to the provided source excerpts.
8. The backend returns the answer along with citations referencing document names and chunk identifiers.

If the retrieved sources do not contain sufficient information, the model is instructed to respond with an explicit "I do not know."

---

## 7. Prompting Strategy

The system uses a fixed system prompt to enforce behavior:

- Answers must be based only on provided sources
- Outside knowledge is disallowed
- Citations are required for factual claims
- Uncertainty must be acknowledged explicitly

User prompts consist of the question followed by labeled source excerpts. Temperature is kept low to prioritize factual accuracy and consistency.

---

## 8. API Surface (MVP)

### Document Management

- POST /documents
- GET /documents
- DELETE /documents/{id}

### Querying

- POST /query
- POST /conversations/{id}/query
- GET /conversations/{id}

Responses include both the generated answer and the source references used.

---

## 9. Security and Access Control (MVP)

- Each user can only query their own documents
- Vector search results are filtered by user ID
- No document sharing between users in MVP

This provides a secure baseline that can later be extended to organization-level permissions and role-based access control.

---

## 10. Observability and Debugging

For every query, the system records:

- User question
- Retrieved document chunks

- Model response
- Timestamps and latency

These logs enable debugging of incorrect answers, relevance tuning, and cost analysis.

---

## 11. Future Extensions

Planned post-MVP enhancements include:

- Organization and team support
  - Fine-grained access controls
  - Hybrid keyword and vector search
  - Auto-generated document summaries
  - UI document browsing and highlighting
  - Streaming responses
  - On-premise and air-gapped deployments
- 

## 12. Summary

This project delivers a practical, scalable system for extracting value from large collections of unstructured documents using AI. By combining deterministic retrieval with constrained language model reasoning, it avoids common pitfalls such as hallucination and token limits while providing accurate, auditable answers. The MVP is intentionally focused and production-oriented, providing a strong foundation for future growth into a full enterprise knowledge platform.