

THIS PROJECT WAS DONE TO FAMILARIZE ONE WITH AI TOOLS WHEN CODING.

Project name: AI-Assisted Knowledge Chatbot

Working endpoint: localhost://127.0.0.1:8000/chat/ui

Purpose: This project was implemented to provide the user with the ability to ask various questions regarding already existing documents in the database or any new documents uploaded by the user. This application enables the user to understand various documents easily, understand various context behind it, and other related tasks.

Technology Stack:

- Language: Python 3.14
- Framework: FastAPI, REST API
- Cloud AI: Azure OpenAI ServiceEmbeddings (text-embedding-ada-002) + GPT-4o
- Vector DB: FAISSSimilarity search & storage
- Document Processing: PyPDF, python-docxText extraction
- Data Validation: PydanticSchema validation
- Frontend:HTML/CSS/JavaScriptChat interface
- Storage: JSON (local file system)

Application Pipeline:

Phase 1: Document Ingestion

User uploads PDF/DOCX

- Extract text (PyPDF/python-docx)
- Validate governance (approved documents only)
- Chunk into 500-word segments with 100-word overlap
- Save chunks with metadata to JSON files

Phase 2: Indexing

Load all approved chunks

- Send to Azure OpenAI Embeddings API
- Get back 1,536-dimensional vectors
- Store vectors in FAISS index
- Save index to disk for persistence

Phase 3: Query Processing (RAG)

User asks question in chat

- Convert question to embedding (Azure OpenAI)
- Search FAISS for top K similar chunks (semantic search)
- Retrieve relevant text chunks with metadata

- Build prompt: System message + Context + Question + History
- Send to Azure OpenAI GPT-4o
- Get natural language answer
- Return answer with source citations
- Save to conversation history

Phase 4: Conversation Management

Each chat session:

- Maintains message history (user + assistant)
- Tracks selected topic filter
- Preserves context across turns
- Enables follow-up questions with memory

BREAKDOWN OF THE PROJECT

Day one: project setup, azure resource, skeleton code

Result:

The screenshot shows a developer's environment with three main windows:

- Code Editor:** An IDE window titled "schemas.py" showing Python code for a "DocumentMetadata" class.
- Browser:** A web browser window titled "Gemini" showing the application's API documentation. It includes sections for "Al-Powered Knowledge Framework 1.0.0 OAS 3.1" and "Enterprise RAG system using Azure AI Search and OpenAI". Below this, there's a "default" section with two API endpoints: "/ Root" and "/health Health".
- Terminal:** A terminal window showing the command "uvicorn app.main:app --reload" being run, followed by logs indicating the application is running on port 8000 and responding to various requests like "/favicon.ico" and "/openapi.json".

What was implemented:

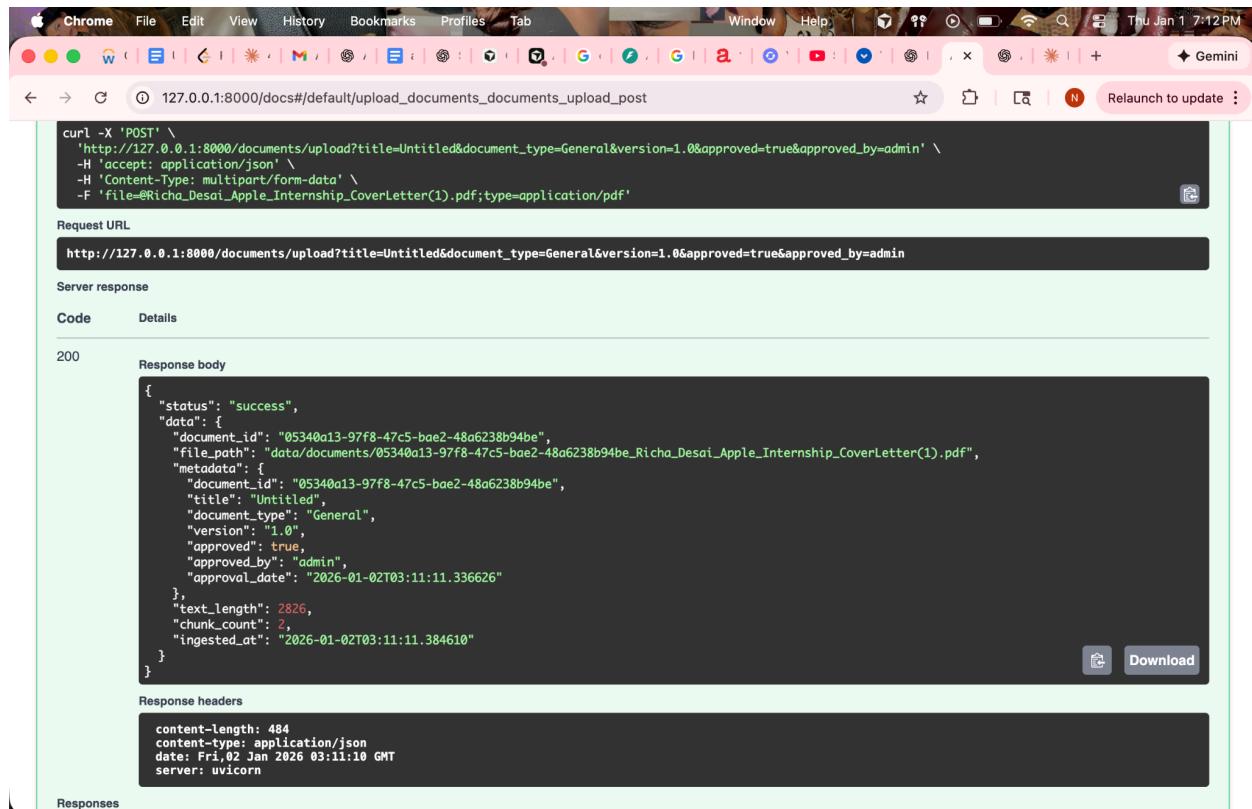
- Clean project structure
- FastAPI backend
- Governance model
- Config-driven architecture
- Cursor + local dev working

Day two: Document Ingestion + Governance Layer

What the system does:

1. Extracting text from PDFs
2. Chunking text with overlap
3. Storing chunks with metadata
4. Serializing datetime objects properly

RESULT:



```

curl -X 'POST' \
'http://127.0.0.1:8000/documents/upload?title=Untitled&document_type=General&version=1.0&approved=true&approved_by=admin' \
-H 'accept: application/json' \
-H 'Content-type: multipart/form-data' \
-F 'file=@Richa_Desai_Apple_Internship_CoverLetter(1).pdf;type=application/pdf'

```

Request URL

```
http://127.0.0.1:8000/documents/upload?title=Untitled&document_type=General&version=1.0&approved=true&approved_by=admin
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "status": "success", "data": { "document_id": "05340a13-97f8-47c5-bae2-48a6238b94be", "file_path": "data/documents/05340a13-97f8-47c5-bae2-48a6238b94be_Richa_Desai_Apple_Internship_CoverLetter(1).pdf", "metadata": { "document_id": "05340a13-97f8-47c5-bae2-48a6238b94be", "title": "Untitled", "document_type": "General", "version": "1.0", "approved": true, "approved_by": "admin", "approval_date": "2026-01-02T03:11:11.336626" }, "text_length": 2826, "chunk_count": 2, "ingested_at": "2026-01-02T03:11:11.384610" } }</pre> <p>Response headers</p> <pre>content-length: 484 content-type: application/json date: Fri, 02 Jan 2026 03:11:10 GMT server: uvicorn</pre> <p>Responses</p>

DAY 3 and 4: RAG system setup – process

Step 1: Document Upload & Extraction

- One uploads the PDF
- The system extracts the text from the PDF
- It validates the document (checks if it's approved)
- Saves the file to `data/documents/`

Step 2: Chunking

What happens:

Original text: 2,826 characters (entire cover letter) - example file

↓

Chunk 1: First 500 words

Chunk 2: Words 400-end (with 100 word overlap)

Why chunking?

- AI models have token limits
- Smaller chunks = more precise search results
- Overlap ensures context isn't lost at boundaries

Step 3: Embeddings

What happens: Each chunk gets converted into a vector (list of 1,536 numbers):

Text: "Leadership isn't about control"

↓ Azure OpenAI

Vector: [0.023, -0.145, 0.891, ..., 0.234] (1,536 numbers)

Why vectors?

- Similar meanings → similar vectors

Step 4: Vector Storage (FAISS)

What happens: All vectors get stored in a special database (FAISS) that's optimized for fast similarity search.

`data/vector_index/`

```
|--- default.index           ← Binary file with vectors  
|--- default_chunks.json   ← Original text + metadata
```

Why FAISS?

- Super fast at finding "nearest neighbors"
- Can search millions of vectors in milliseconds

Step 5: Semantic Search

What happens when you search:

Query: "leadership experience"
↓
1. Convert query to vector using Azure OpenAI
Query vector: [0.012, -0.234, 0.567, ...]

2. FAISS finds closest chunk vectors
Distance to Chunk 1: 0.471
Distance to Chunk 2: 0.419 ← Closer = more relevant

3. Return the matching chunks with their text

The Complete Flow:

```
PDF Upload  
↓  
Extract Text (2,826 chars)  
↓  
Split into Chunks (2 chunks with overlap)  
↓  
Generate Embeddings (2 vectors of 1,536 dimensions each)  
↓  
Store in FAISS Index  
↓  
Ready for Search!
```

When you search:

Query → Embed → Find Similar Vectors → Return Text

Day 3 result: RAG pipeline was set up for the PDF to be chunked and processed when queries are asked by the user.

- RAG architecture and implementation
- Azure OpenAI integration (Embeddings + Chat)
- Vector databases (FAISS)
- Semantic search vs keyword search
- Document processing pipelines
- REST API development with FastAPI
- Cloud AI services

DAY 4: RAG Query Pipeline

Results achieved:

- Built context retrieval system
- Implemented prompt building with context injection
- Integrated GPT-4o for answer generation
- Created RAG query endpoint (`/query`)
- Added source attribution

Pipeline flow:

User Query

- Embed Query
- Search FAISS
- Get Top 3 Chunks
- Build Prompt (System + Context + Question)
- Send to GPT-4o
- Natural Language Answer

Files Created:

```
app/
└── rag/
    ├── retriever.py
    └── generator.py
```

Result:

Sample query :

```

curl -X POST
'http://127.0.0.1:8000/query?query=What%20leadership%20experience%20
does%20X%20have?&top_k=3'

{
  (from sample file uploaded
  "status": "success",
  "query": "What leadership experience does X have?",
  "answer": "X has leadership experience as Assistant Vice President
of Growth at the Y at Z, where they lead marketing and engagement
strategies. They has consistently found herself in leadership roles
throughout her journey - on the field, in classrooms, and in every
team they have been part of.",
  "contexts": [
    {
      "chunk_id": "..._chunk_0",
      "text": "...leadership content...",
      "similarity_score": 0.398
    }
  ],
  "metadata": {
    "contexts_used": 3,
    "model": "gpt-4o"
  }
}
```

```

## **DAY 5:** Governance & Enterprise Features

### **File Structure:**

```

app/
├── utils/
│ └── logger.py
logs/
└── rag_system_20260102.log

```

```

{
 "status": "success",
 "stats": { "total_vectors": 4 },
 "governance": { "approved_only": true }
}
```

```

Example Log File:

```
INFO - Loaded 4 approved chunks
WARNING - Skipped 2 chunks from unapproved documents
```

Results Achieved:

- Approved-only indexing
- Full audit trail
- Document type segmentation
- Explainable AI (source tracking)
- Implemented enterprise logging (file + console)
- Added governance controls (approved docs only)
- Built document type filtering
- Created audit trail system
- Enhanced all endpoints with filters

DAY 6: AI Chatbot Interface**File structure:**

```
app/
  └── chat/
    ├── session_manager.py
    └── chatbot.py
  └── models/schemas.py (ChatMessage, ChatSession, ChatRequest,
  ChatResponse)
```

Results achieved after:

- Built interactive web chat UI
- Implemented session management
- Added conversation history
- Created topic selection menu
- Integrated file upload in chat
- Auto-rebuild index after upload
- Real-time chat with GPT-4o

Final Result:

The screenshot shows the AI Knowledge Assistant interface at its initial state. The top bar displays the URL 127.0.0.1:8000/chat/ui. The main area is titled "AI KNOWLEDGE ASSISTANT" with the sub-instruction "Upload documents and ask questions!". A "Choose File" button is present, showing "No file selected". An optional "Document title (optional)" input field and a green "Upload" button are also visible. Below this is a dropdown menu set to "All Topics". A message box contains the AI's greeting: "Hello! I'm your AI Knowledge Assistant.", followed by instructions: "Upload a document above to add it to my knowledge base" and "Ask me anything about the documents I have access to". A placeholder text "Select a topic or just start chatting!" is shown below. A purple button labeled "explain what is the context of neha's resume" is highlighted. The status bar at the bottom indicates "Ai is thinking...". A message input field with "Type your message..." placeholder and a purple "Send" button are at the bottom.

AI KNOWLEDGE ASSISTANT

Upload documents and ask questions!

Choose File No file selected Document title (optional) Upload

All Topics

Hello! I'm your AI Knowledge Assistant.

Upload a document above to add it to my knowledge base

Ask me anything about the documents I have access to

Select a topic or just start chatting!

explain what is the context of neha's resume

Ai is thinking...

Type your message... Send

The screenshot shows the AI Knowledge Assistant interface after a user has entered a message. The AI has responded with a detailed analysis of Neha Thomas's resume. The response is contained within a large message box. It states: "The context of Neha Thomas's resume (Source 1) is to showcase her academic background, technical skills, work experience, projects, publications, and leadership activities. It highlights her qualifications as a Master's student in Computer Science at the University of Southern California, with a strong foundation in Information Technology from her undergraduate studies. The resume emphasizes her expertise in software engineering, AI & ML, web development, and cloud technologies, as well as her experience in internships, research projects, and technical accomplishments. It is tailored to demonstrate her capabilities for roles in software engineering, AI, or related fields." Below this, a small note says "Sources: NehaThomas_Intern_Resume.pdf, Untitled, Untitled". The message input field and "Send" button are at the bottom, and a small preview window of the AI interface is visible on the right side.

The context of Neha Thomas's resume (Source 1) is to showcase her academic background, technical skills, work experience, projects, publications, and leadership activities. It highlights her qualifications as a Master's student in Computer Science at the University of Southern California, with a strong foundation in Information Technology from her undergraduate studies. The resume emphasizes her expertise in software engineering, AI & ML, web development, and cloud technologies, as well as her experience in internships, research projects, and technical accomplishments. It is tailored to demonstrate her capabilities for roles in software engineering, AI, or related fields.

Sources: NehaThomas_Intern_Resume.pdf, Untitled, Untitled

Type your message... Send