Upscale AI School Talent Hunt Assessment

Published on: 22nd May, 2025

Submission Deadline: 26th May, 2025

Project Task: Build an API Documentation Q&A Agent

Objective: Develop a functional prototype of an AI agent that can answer natural language questions about a specific API, using its official documentation as the knowledge base. The agent must leverage vector database embeddings to retrieve relevant context from the documentation before generating an answer with a Large Language Model (LLM).

Background: Navigating extensive API documentation can be time-consuming. Developers often need quick answers to specific questions about endpoints, parameters, authentication, or usage examples. This project aims to build an intelligent agent that understands a user's question, finds the most relevant sections in the API documentation using semantic search (via vector embeddings), and then uses an LLM to synthesize a clear and concise answer based only on that retrieved context. This is a practical application of the Retrieval-Augmented Generation (RAG) pattern with an agentic decision-making component (deciding what context is relevant).

Timeframe: 5 Days

Core Requirements:

Documentation Ingestion:

- Select a moderately sized API documentation set (e.g., available as Markdown files, plain text, or a single HTML page you can scrape/convert to text). Suggestion: Stripe API docs (use a subset), Twilio API docs (subset), or a smaller public API.
- Implement a script to load and parse the documentation content.
- Implement text chunking: Break down the documentation into smaller, meaningful segments (e.g., paragraphs, sections, function descriptions).

Embedding and Vector Storage:

- Choose an embedding model (e.g., Sentence Transformers, OpenAI embeddings API).
- Generate vector embeddings for each documentation chunk.
- Choose a vector database (e.g., ChromaDB, FAISS, Pinecone free tier).
- Store the documentation chunks and their corresponding embeddings in the vector database.

Question Answering Core Loop:

- Create an interface (Command Line Interface is sufficient, Streamlit/Gradio bonus) to accept user questions in natural language (e.g., "How do I authenticate my requests?", "What parameters does the create_charge endpoint accept?").
- Embed the user's question using the same embedding model.
- Query the vector database with the question embedding to find the top k most relevant documentation chunks (semantic similarity search).
- Implement the agentic decision step: Use the retrieved chunks as context.

LLM Integration for Synthesis:

- Choose an LLM API (e.g., OpenAI, Anthropic) or a local model (via Ollama/LM Studio).
- Construct a prompt for the LLM that includes:
    - The original user question.
    - The retrieved documentation chunks as context.
    - Clear instructions for the LLM to answer the question based only on the provided context and to indicate if the context is insufficient.
- Send the prompt to the LLM and receive the generated answer.

Output:

- Display the LLM's generated answer to the user.

Suggested Technical Stack (Choose one from each category):

- Language: Python
- Vector DB: ChromaDB (recommended for ease of local setup), FAISS (library), Pinecone (cloud free tier)
- Embedding Model: Sentence Transformers (e.g., all-MiniLM-L6-v2 - free, local), OpenAI text-embedding-ada-002 (API call)
- LLM: OpenAI API (GPT-3.5-Turbo/GPT-4), Anthropic API (Claude), or local models via Ollama (e.g., Llama 3, Mistral)
- UI (Optional): CLI (standard Python input()), Streamlit, Gradio

Key Steps / Workflow:

1. **Setup:** Environment, install libraries, get API keys (if needed).
2. **Data Prep:** Acquire API docs, write script to load/parse/chunk.
3. **Embedding Pipeline:** Write script to embed chunks and load into the chosen vector DB.

4. **Retrieval:** Implement function to take a question, embed it, and query the DB for context.
5. **Generation:** Implement a function to construct the prompt (question + context) and call the LLM.
6. **Integration:** Combine retrieval and generation into a single user-facing loop (CLI or simple UI).
7. **Testing:** Test with various questions – some direct, some more complex. Ensure answers are relevant and derived from the context.

Evaluation Criteria:

- **Functionality:** Does the agent ingest docs, store embeddings, retrieve context, and generate answers?
- **Relevance:** Does the vector search retrieve documentation chunks relevant to the user's question?
- **Groundedness:** Is the LLM's answer clearly based on the retrieved context? Does it avoid hallucination?
- **Clarity:** Is the final answer presented clearly to the user?
- **Code Quality:** Is the code reasonably organized and documented?

Deliverables:

- Source code (e.g., link to a Git repository).
- A requirements.txt file.
- A brief README.md explaining:
    - How to set up the environment and run the agent.
    - The chosen API documentation and any assumptions made.
    - Brief overview of the design choices (vector DB, embedding model, LLM).
- (Optional) A short screen recording or live demo showing the agent in action.

Stretch Goals (If time permits):

- Implement basic conversation history to handle follow-up questions.
- Experiment with different chunking strategies or embedding models and compare results.
- Add source tracking: Indicate which documentation chunk(s) the answer was primarily based on.