SOURCE CODE:

```
from fastapi import FastAPI, UploadFile, File, Body, status
from pydantic import BaseModel
from PyPDF2 import PdfReader
from typing import List
import faiss
import numpy as np
import sqlite3
from datetime import datetime
import os

# CONFIG
EMBEDDING_PROVIDER = "sentence_transformers"
# options: "sentence_transformers", "huggingface", "openai"
# APP
app = FastAPI( title="Document Q&A RAG Service", version="1.0.0")

# VECTOR DB
EMBEDDING_DIM = 384
vector_index = faiss.IndexFlatL2(EMBEDDING_DIM)
stored_chunks: List[dict] = []

# METADATA DB
conn = sqlite3.connect("metadata.db", check_same_thread=False)
cursor = conn.cursor()
cursor.execute(""" CREATE TABLE IF NOT EXISTS documents ( id INTEGER PRIMARY
KEY AUTOINCREMENT, filename TEXT, upload_time TEXT ) """)
conn.commit()
```

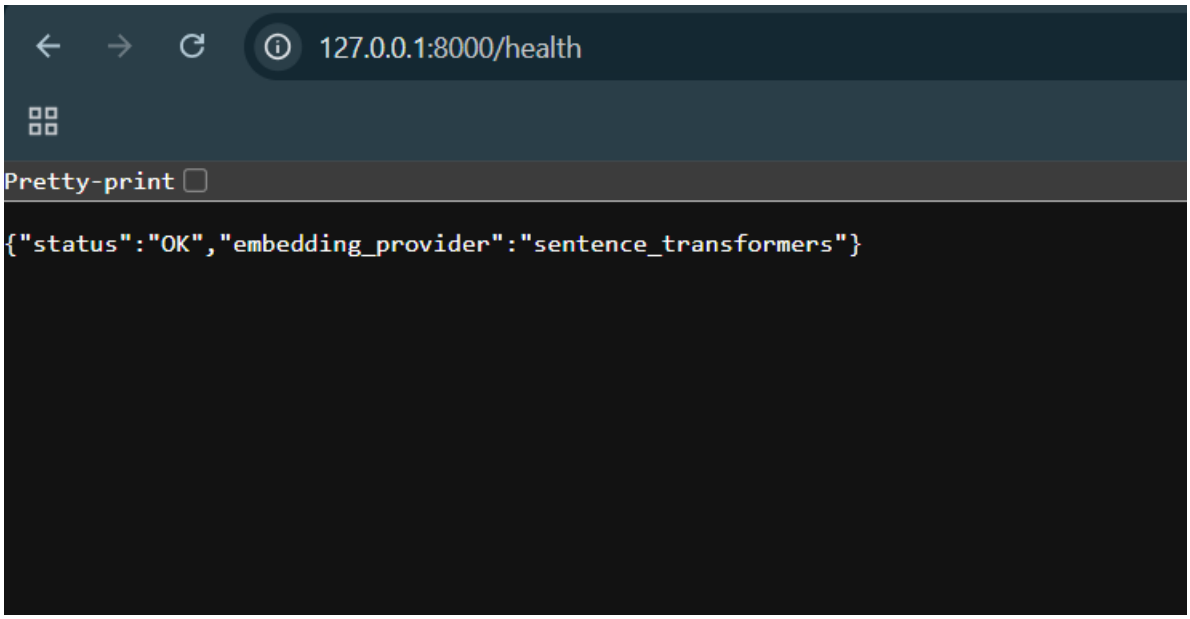
```
# EMBEDDING MODELS
if EMBEDDING_PROVIDER == "sentence_transformers":
    from sentence_transformers import SentenceTransformer
    embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

if EMBEDDING_PROVIDER == "huggingface":
    from transformers import AutoTokenizer, AutoModel
    import torch
    tokenizer = AutoTokenizer.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")
    hf_model = AutoModel.from_pretrained("sentence-transformers/all-MiniLM-L6-v2")

if EMBEDDING_PROVIDER == "openai":
    from openai import OpenAI
    client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

def get_embeddings(texts: List[str]) -> List[List[float]]:
    if EMBEDDING_PROVIDER == "sentence_transformers":

        return embedding_model.encode(texts).tolist()
    if EMBEDDING_PROVIDER == "huggingface":

        inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt")
        with torch.no_grad():
            outputs = hf_model(**inputs)
        return outputs.last_hidden_state.mean(dim=1).tolist()
    if EMBEDDING_PROVIDER == "openai":
        response = client.embeddings.create( model="text-embedding-3-small", input=texts )
        return [item.embedding for item in response.data]
```

```
    raise ValueError("Invalid embedding provider")
# HELPERS
def extract_text(file: UploadFile) -> str:
    if file.filename.endswith(".pdf"):

        reader = PdfReader(file.file)
        return " ".join(page.extract_text() for page in reader.pages)
    return file.file.read().decode("utf-8")

def chunk_text(text: str, size: int = 500) -> List[str]:
    words = text.split()
    return [" ".join(words[i:i + size]) for i in range(0, len(words), size)]

# SCHEMAS
class QueryRequest(BaseModel):
    question: str

class Source(BaseModel):
    filename: str
    content: str

class QueryResponse(BaseModel):
    question: str
    answer: str
    sources: List[Source]

# ENDPOINTS
@app.post("/upload", status_code=status.HTTP_201_CREATED)
async def upload_document(file: UploadFile = File(...)):
```

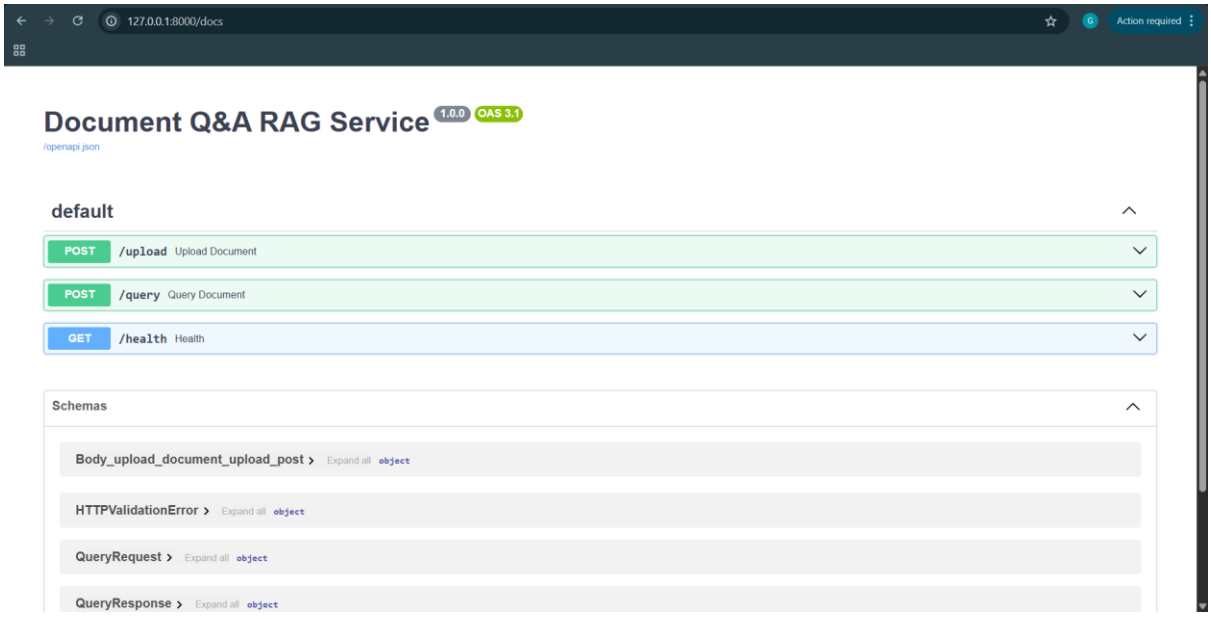
```
    text = extract_text(file)
    chunks = chunk_text(text)
    embeddings = get_embeddings(chunks)
    vector_index.add(np.array(embeddings))
    for chunk in chunks:
        stored_chunks.append({
            "text": chunk,
            "filename": file.filename
        })
    cursor.execute(
        "INSERT INTO documents (filename, upload_time) VALUES (?, ?)",
        (file.filename, datetime.now().isoformat())
    )
    conn.commit()
    return {
        "message": "Document uploaded successfully",
        "filename": file.filename,
        "chunks_stored": len(chunks)
    }

@app.post("/query", response_model=QueryResponse)
async def query_document(payload: QueryRequest = Body(...)):
    question_embedding = get_embeddings([payload.question])[0]
    _, indices = vector_index.search(np.array([question_embedding]), k=3)
    sources = []
    context = ""
    for i in indices[0]:
        chunk = stored_chunks[i]
        context += chunk["text"] + " "
```

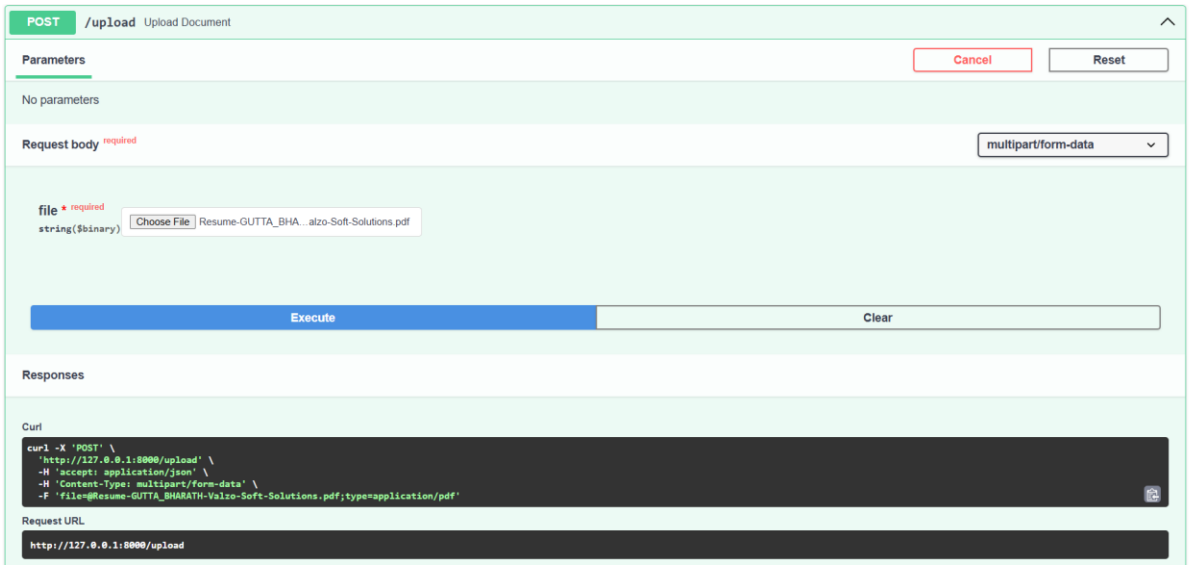
```
            sources.append(
                Source(
                    filename=chunk["filename"],
                    content=chunk["text"]
                )
            )
        answer = f"Answer based on document context:\n{context}"
        return QueryResponse(
            question=payload.question,
            answer=answer,
            sources=sources
        )

@app.get("/health")
def health():
    return {
        "status": "OK",
        "embedding_provider": EMBEDDING_PROVIDER
    }
```

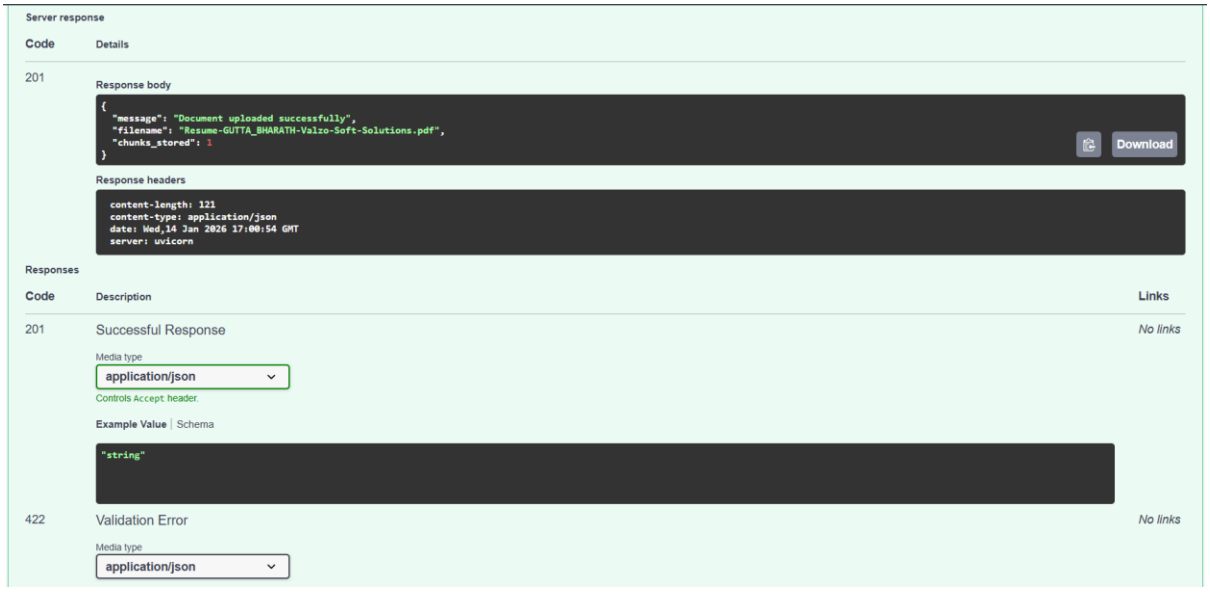
Outputs:



With swagger

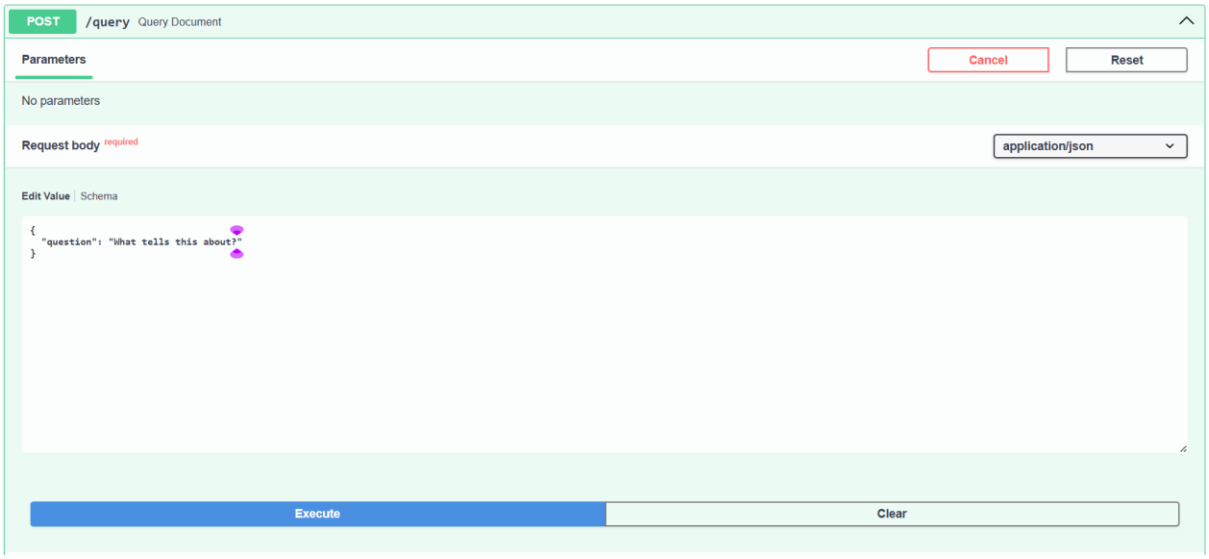


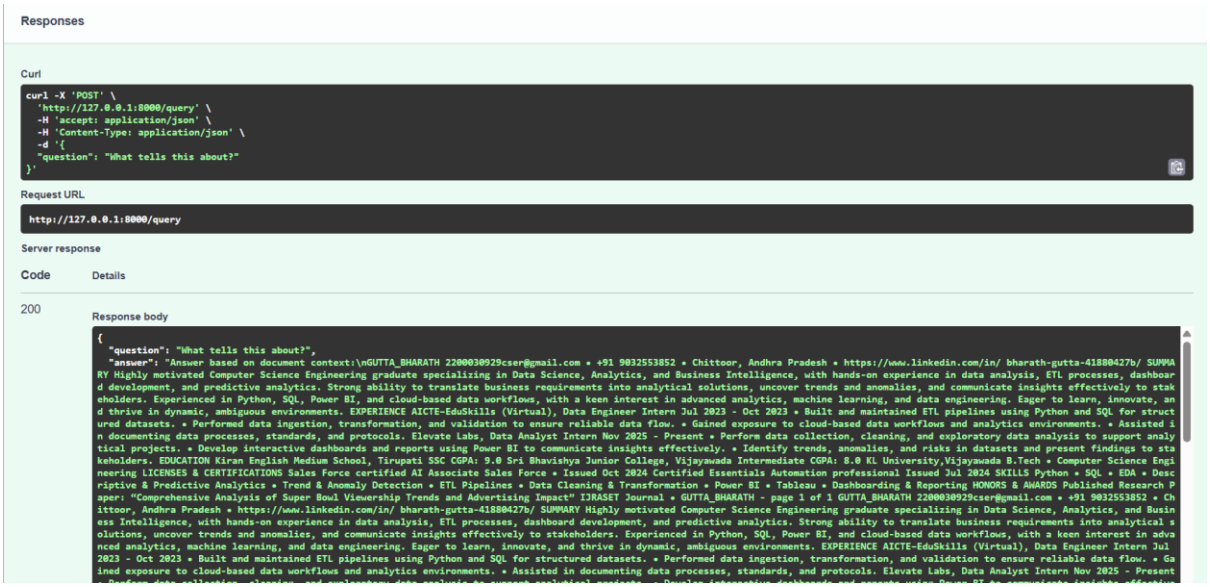




Here 201 means ok(successful).

After uploading the any file in the /upload. Then check the /query.



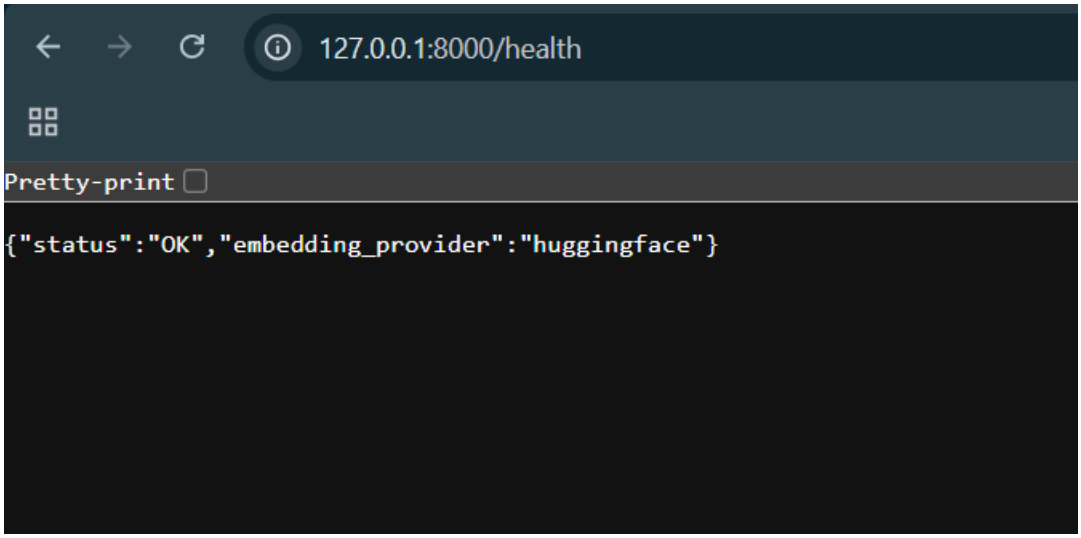


If we interrupt b/w the /upload and /query. It throws logical error…

For HuggingFace

→We have to change embedding_provider to huggingface

Make sure you installed torch. If not use this →python -m pip install torch



OpenAI

Make sure you whether installed →python -m pip install openai

For checking the version

```
PS C:\Users\gutta\Downloads\FastAPI> python -c "import openai; print(openai.__version__)"
2.15.0
PS C:\Users\gutta\Downloads\FastAPI>
```

After

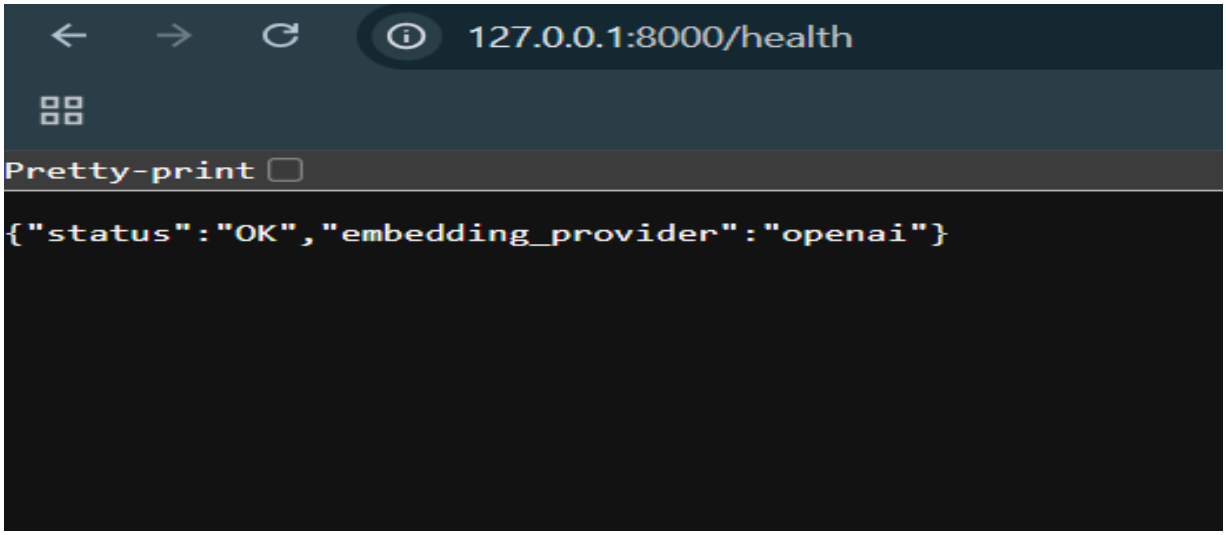
```
PS C:\Users\gutta\Downloads\FastAPI> setx OPENAI_API_KEY "your_openai_api_key_here"

SUCCESS: Specified value was saved.
PS C:\Users\gutta\Downloads\FastAPI>
```

astAPI › main.py

API keys

In Powershell

```
PS C:\Users\gutta> $env:OPENAI_API_KEY="sk-xxxxxxxxxxxxxxxxxxxxxxx"
PS C:\Users\gutta> python -c "import os; print(os.getenv('OPENAI_API_KEY'))"
sk-xxxxxxxxxxxxxxxxxxxxxxx
PS C:\Users\gutta>
```


127.0.0.1:8000/health
Pretty-print
{"status":"OK","embedding_provider":"openai"}