from fastapi import APIRouter, HTTPException

from fastapi.responses import JSONResponse

from util import getOpenAIClient

from pydantic import BaseModel

import os

from dotenv import load\_dotenv

from opensearchpy import OpenSearch

from rag\_uploadfiles import generate\_embeddings

router = APIRouter()

client = getOpenAIClient()

load\_dotenv()

host = os.environ.get("OPENSEARCH\_HOST")

port = os.environ.get("OPENSEARCH\_PORT")

username = os.environ.get("OPENSEARCH\_USERNAME")

password = os.environ.get("OPENSEARCH\_PASSWORD")

# OpenSearch configuration

OPENSEARCH\_CONFIG = {

    "hosts": [{"host": host, "port": port}],

    "http\_auth": (username, password),

    "http\_compress": True,

    "use\_ssl": True,

    "verify\_certs": False,

    "ssl\_assert\_hostname": False,

    "ssl\_show\_warn": False,

}

INDEX\_NAME = "files"

class ChatRequest(BaseModel):

    query: str

@router.post("/chat")

async def chat(request: ChatRequest):

    query = request.query

    if not query:

        raise HTTPException(status\_code=400, detail="Query cannot be empty")

    # create embedding for the query

    try:

        query\_embedding = generate\_embeddings([query])[

            0

        ]  # Get the first element of the list

        search\_client = OpenSearch(\*\*OPENSEARCH\_CONFIG)

        search\_body = {

            "size": 1000,  # Get more results initially

            "\_source": [

                "name",

                "content",

                "embedding",

            ],  # Only retrieve necessary fields

            "query": {

                "knn": {

                    "embedding": {

                        "vector": query\_embedding,

                        "k": 3,  # Get top 3 results

                    }

                }

            },

        }

        # Search for similar documents based on the query embedding

        response = search\_client.search(index=INDEX\_NAME, body=search\_body)

        # Extract documents and their embeddings

        documents\_string = ""

        # match\_all query returns all documents, so we need to filter based on cosine similarity

        for hit in response["hits"]["hits"]:

            doc = hit["\_source"]

            documents\_string += doc["content"]

            documents\_string += f"\nSource: {doc['name']}\n\n"

        # openai call

        prompt = f"""Answer the question: {query} based only on the following context. At the end also mention source from context:

        context:  {documents\_string}

        """

        # Call OpenAI API

        response = client.chat.completions.create(

            model="gpt-4o",

            temperature=1,  # Higher temperature can result in more creative responses apart from context

            messages=[

                {

                    "role": "system",

                    "content": [

                        {

                            "type": "text",

                            "text": "You are an AI assistant tasked with answering questions using the provided context as the primary source of information.",

                        }

                    ],

                },

                {"role": "user", "content": [{"type": "text", "text": prompt}]},

            ],

        )

        questions\_text = response.choices[0].message.content

        return JSONResponse(content={"response": questions\_text})

    except Exception as e:

        raise HTTPException(

            status\_code=500, detail=f"Failed to create embedding: {str(e)}"

        )