```python
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
```

```python
@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)}"
```

)