**Instructions to Create Index**

Before running the file upload, you need to create an index named "**files**" in OpenSearch to store the documents. Follow these steps to create the index:

1. **Navigate to the Project Directory**:
   * Go to the Project/OpenSearch directory in your project.
2. **Run the Create Index Script**:
   * Execute the Python script \_1\_Opensearch\_CreateIndex.py to create the index. You can run this script using the following command:

python \_1\_Opensearch\_CreateIndex.py

1. **Wait for Acknowledgement**:
   * Once the script runs successfully, you should receive an acknowledgment indicating that the index has been created.
2. **Proceed with File Upload**:
   * After the index is created, you can proceed with uploading files to the "files" index using the /rag/upload endpoint.

Solution

from fastapi import APIRouter, UploadFile, File

from fastapi.responses import JSONResponse

from langchain\_text\_splitters import RecursiveCharacterTextSplitter

import io

from langchain\_community.document\_loaders import PyPDFLoader

from langchain.text\_splitter import RecursiveCharacterTextSplitter

import os

import tempfile

from dotenv import load\_dotenv

from opensearchpy import OpenSearch

from opensearchpy.helpers import bulk

from util import getOpenAIClient

router = APIRouter()

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"

def generate\_embeddings(texts):

    client = getOpenAIClient()  # Get the OpenAI client

    # Generate embeddings using OpenAI API

    response = client.embeddings.create(

        input=texts, dimensions=256, model="text-embedding-3-large"

    )

    # Extract embeddings from the response

    embeddings = [item.embedding for item in response.data]

    return embeddings

def insert\_documents(search\_client, fileChunks, embeddings, filesMetadata):

    documents = []  # List to hold bulk actions

    for i, item in enumerate(filesMetadata):

        # Create an document object

        document = {

            "\_index": INDEX\_NAME,

            "\_id": i,

            "\_source": {

                "name":  os.path.basename(item["source"]),  # Document source name without absolute path

                "content": fileChunks[i],  # Document content

                "embedding": embeddings[i],  # Document embedding

            },

        }

        documents.append(document)  # Add document to the list

    # Perform bulk insert into OpenSearch

    success, \_ = bulk(search\_client, documents)

    print(f"Successfully inserted {success} documents into OpenSearch.")

#Challenge 1: Implement the retrieve\_all\_documents function to retrieve all documents from OpenSearch index files.

def retrieve\_all\_documents(client):

    # Perform the OpenSearch search to get documents

    search\_body = {

        "size": 1000,  # Get more results initially

        "\_source": ["name"],  # Only retrieve necessary fields

        "query": {"match\_all": {}},  # We'll filter based on cosine similarity in Python

    }

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

    return response

@router.get("/rag/files")

async def get\_files():

    try:

        # Connect to OpenSearch

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

        search\_results = retrieve\_all\_documents(search\_client)

        # get hits

        hits = search\_results["hits"]["hits"]

        # To get only distinct files

        sources = []  # List to hold file names

 # Note: For each file uploaded, files index will have multiple records with the same name

        for hit in hits:

            source\_name = hit["\_source"]["name"]

            if source\_name not in sources:

                sources.append(fileName)  # To Remove the absolute path and get only the file name

        return JSONResponse(content={"sources": sources})

    except Exception as e:

        return JSONResponse(content={"error": str(e)}, status\_code=500)

#---------------------------------Route---------------------------------#

@router.post("/rag/upload")

 # Save the text content to OpenSearch index by name files

async def **upload**\_**file**(file: UploadFile = File(...)):

    if not file.filename.endswith(".pdf"):

        return JSONResponse(

            content={"error": "Only PDF files are allowed"}, status\_code=400

        )

    try:

        # Read the uploaded file

        contents = await file.read()

        pdf\_stream = io.BytesIO(contents)

        # Create temporary file with original name in temp directory

        temp\_path = os.path.join(tempfile.gettempdir(), file.filename)

        # Write the PDF content to the temporary file

        with open(temp\_path, "wb") as temp\_file:

            temp\_file.write(pdf\_stream.getvalue())

        try:

            # Use temporary file with PyPDFLoader

            loader = PyPDFLoader(temp\_path)

            text\_content = loader.load()

              # Challenge: Get an object to split the docs into chunks of 3000 characters with 100 character overlap between chunks

# You can use langchain\_text\_splitters.RecursiveCharacterTextSplitter for this

# Split the docs and create embeddings

            text\_splitter = RecursiveCharacterTextSplitter(

                chunk\_size=3000, chunk\_overlap=100

            )

            # Split the text content into chunks

            chunks = text\_splitter.split\_documents(text\_content)

            pages\_content = [] # List to hold the content of the pages for which embeddings will be generated

            pages\_metadata\_content = [] # List to hold the metadata (Page Numbers and Source) of the pages for which embeddings will be generated

            for chunk in chunks:

                pages\_content.append(chunk.page\_content)

                pages\_metadata\_content.append(chunk.metadata)

            # Challenge: Generate embeddings for each page in pages\_content

            embeddings = generate\_embeddings(pages\_content)

            # Connect to OpenSearch

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

            # Insert the documents into OpenSearch

            insert\_documents(search\_client, pages\_content, embeddings, pages\_metadata\_content)

        finally:

            # Clean up temporary file

            os.unlink(temp\_path)

        return JSONResponse(

            content={

                "message": "File uploaded and processed successfully",

                "filename": file.filename,

                "size": len(contents),

                "text\_length": len(text\_content),

            }

        )

    except Exception as e:

        return JSONResponse(content={"error": str(e)}, status\_code=500)