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
```

3. **Wait for Acknowledgement**:
   - Once the script runs successfully, you should receive an acknowledgment indicating that the index has been created.
4. **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)
```