*from* flask *import* Flask, request, jsonify

*from* flask\_cors *import* CORS

*import* os

*import* fitz *# PyMuPDF for PDF processing*

*from* werkzeug.utils *import* secure\_filename

*from* dotenv *import* load\_dotenv

*import* time

*from* watchdog.observers *import* Observer

*from* watchdog.events *import* FileSystemEventHandler

*from* langchain\_openai *import* ChatOpenAI, OpenAIEmbeddings

*from* langchain.schema *import* SystemMessage, HumanMessage

*from* pinecone *import* Pinecone

*from* langchain\_pinecone *import* Pinecone *as* LangchainPinecone

*# Load environment variables*

load\_dotenv()

OPENAI\_API\_KEY2 = os.getenv("OPENAI\_API\_KEY2")

PINECONE\_API\_KEY2 = os.getenv("PINECONE\_API\_KEY2")

*if* not OPENAI\_API\_KEY2 or not PINECONE\_API\_KEY2:

*raise* ValueError("Missing API Keys. Please check your .env file.")

*# Initialize Flask app*

app = Flask(\_\_name\_\_)

CORS(app, *resources*={r"/api/\*": {"origins": "\*"}})

UPLOAD\_FOLDER = 'uploads'

os.makedirs(UPLOAD\_FOLDER, *exist\_ok*=True)

*# Initialize OpenAI Chat Model*

chat = ChatOpenAI(*openai\_api\_key*=OPENAI\_API\_KEY2, *model*='gpt-3.5-turbo')

*# Initialize Pinecone*

pc = Pinecone(*api\_key*=PINECONE\_API\_KEY2)

index\_name = "rag-chatbot-index"

print(pc.list\_indexes().names())

*if* index\_name not in pc.list\_indexes().names():

pc.create\_index(index\_name, *dimension*=1536) *# Set correct dimension for embeddings*

print(f"Index '{index\_name}' created.")

*else*:

print(f"Index '{index\_name}' already exists.")

class FileHandler(FileSystemEventHandler):

def **on\_created**(*self*, *event*):

*if* not *event*.is\_directory:

print(f"New file detected: {*event*.src\_path}")

process\_and\_store\_file(*event*.src\_path)

def **on\_deleted**(*self*, *event*):

*if* not *event*.is\_directory:

print(f"File deleted: {*event*.src\_path}")

filename = os.path.basename(*event*.src\_path)

file\_id = str(hash(filename))[:8] *# Generate same file\_id as in store\_in\_pinecone*

delete\_file\_from\_pinecone(file\_id)

def **process\_and\_store\_file**(*filepath*):

"""Process a single file and store it in Pinecone."""

*try*:

print(f"\n=== Starting to process file: {*filepath*} ===")

*if* not os.path.exists(*filepath*):

print(f"❌ File not found: {*filepath*}")

*return*

*if* not *filepath*.lower().endswith('.pdf'):

print(f"❌ Not a PDF file: {*filepath*}")

*return*

*# Extract text from PDF*

text = process\_pdf(*filepath*)

*if* not text or not text.strip():

print("❌ No text extracted from PDF")

*return*

filename = os.path.basename(*filepath*)

print(f"Extracted text length: {len(text)} characters")

store\_in\_pinecone(text, filename)

print(f"Successfully processed and stored {filename}")

print("=== File processing completed ===\n")

*except* Exception *as* e:

print(f"❌ Error processing file {*filepath*}: {e}")

*raise* e

def **setup\_file\_watcher**():

"""Set up a file watcher for the upload directory."""

event\_handler = FileHandler()

observer = Observer()

observer.schedule(event\_handler, *path*=UPLOAD\_FOLDER, *recursive*=False)

observer.start()

print(f"Started watching directory: {UPLOAD\_FOLDER}")

*return* observer

def **clear\_pinecone**():

"""Clear all vectors from Pinecone."""

*try*:

index = pc.Index(index\_name, *pinecone\_api\_key*=PINECONE\_API\_KEY2)

print("🧹 Clearing all vectors from Pinecone...")

index.delete(*delete\_all*=True)

print("✅ All vectors cleared from Pinecone")

*except* Exception *as* e:

print(f"❌ Error clearing Pinecone: {e}")

*raise* e

def **process\_existing\_files**():

"""Process any existing files in the upload folder during startup."""

*try*:

files = [f *for* f *in* os.listdir(UPLOAD\_FOLDER) *if* f.endswith('.pdf')]

*if* files:

print(f"Found {len(files)} PDF files in upload folder")

*# Clear Pinecone first*

clear\_pinecone()

*# Process all files*

*for* file *in* files:

filepath = os.path.join(UPLOAD\_FOLDER, file)

print(f"Processing file: {file}")

process\_and\_store\_file(filepath)

*else*:

print("No existing PDF files found in upload folder")

*# Clear Pinecone if no files exist*

clear\_pinecone()

*except* Exception *as* e:

print(f"Error processing existing files: {e}")

*# Flask Routes*

@app.route('/')

def **home**():

*return* jsonify({'message': 'Welcome to the RAG Chatbot API!'})

@app.route('/api/chat', *methods*=['POST'])

def **chat\_endpoint**():

data = request.json

query = data.get('query')

*if* not query:

*return* jsonify({'error': 'No query provided'}), 400

augmented\_query = augment\_prompt(query)

response = generate\_response(augmented\_query)

*if* not response:

*return* jsonify({'error': 'Failed to generate response'}), 500

*return* jsonify({'response': response, 'augmentedQuery': augmented\_query})

*# Helper Functions*

def **process\_pdf**(*filepath*):

"""Extract text from a PDF."""

text = ""

*try*:

print(f"Opening PDF file: {*filepath*}")

*with* fitz.open(*filepath*) *as* doc:

print(f"PDF has {len(doc)} pages")

*for* page\_num, page *in* enumerate(doc):

text += page.get\_text("text") + "\n"

print(f"Processed page {page\_num + 1}")

*if* not text.strip():

*raise* ValueError("Extracted text is empty.")

print(f"Extracted text preview:\n{text[:500]}...") *# Print first 500 characters*

*except* Exception *as* e:

print(f"❌ Error extracting text from PDF: {e}")

*raise* e

*return* text

def **store\_in\_pinecone**(*text*, *filename*):

"""Store extracted text in Pinecone."""

*try*:

print(f"\n=== Starting Pinecone storage for {*filename*} ===")

index = pc.Index(index\_name, *pinecone\_api\_key*=PINECONE\_API\_KEY2)

print(f"Connected to Pinecone index: {index\_name}")

*# Create a shorter unique identifier for the file*

file\_id = str(hash(*filename*))[:8] *# Use first 8 characters of hash as unique ID*

print(f"Generated file\_id: {file\_id} for {*filename*}")

*# Process the current file*

embed\_model = OpenAIEmbeddings(*api\_key*=OPENAI\_API\_KEY2)

text\_chunks = split\_text(*text*)

print(f"Split text into {len(text\_chunks)} chunks")

print("Generating embeddings...")

embeddings = embed\_model.embed\_documents(text\_chunks)

print(f"Generated {len(embeddings)} embeddings")

print(f"✅ Storing {len(text\_chunks)} chunks in Pinecone")

print(f"First chunk preview: {text\_chunks[0][:100]}...")

*# Upload new chunks with fresh embeddings*

batch\_size = 10

*for* i *in* range(0, len(text\_chunks), batch\_size):

batch\_end = min(i + batch\_size, len(text\_chunks))

batch = []

*for* j *in* range(i, batch\_end):

doc\_id = f"{file\_id}-{j}"

metadata = {

'text': text\_chunks[j],

'source': *filename*,

'file\_id': file\_id,

'chunk\_id': j

}

batch.append((doc\_id, embeddings[j], metadata))

index.upsert(*vectors*=batch)

print(f"📝 Stored chunks {i} to {batch\_end-1}")

print(f"=== Pinecone storage completed for file\_id {file\_id} ===\n")

*except* Exception *as* e:

print(f"❌ Error during Pinecone upsert: {e}")

*raise* e

def **delete\_file\_from\_pinecone**(*file\_id*):

"""Delete vectors for a specific file."""

*try*:

index = pc.Index(index\_name, *pinecone\_api\_key*=PINECONE\_API\_KEY2)

*# For Serverless/Starter indexes, we need to delete all and re-upload remaining files*

print("🧹 Clearing all vectors...")

index.delete(*delete\_all*=True)

print("✅ All vectors cleared")

*# Re-upload all remaining files except the one being deleted*

files = [f *for* f *in* os.listdir(UPLOAD\_FOLDER) *if* f.endswith('.pdf')]

print(f"Re-uploading {len(files)} remaining files...")

embed\_model = OpenAIEmbeddings(*api\_key*=OPENAI\_API\_KEY2)

*for* f *in* files:

current\_id = str(hash(f))[:8]

*if* current\_id != *file\_id*: *# Skip the file we're deleting*

*try*:

filepath = os.path.join(UPLOAD\_FOLDER, f)

print(f"Processing file: {f}")

*with* fitz.open(filepath) *as* doc:

text = ""

*for* page *in* doc:

text += page.get\_text("text") + "\n"

*if* text.strip():

chunks = split\_text(text)

embeddings = embed\_model.embed\_documents(chunks)

print(f"Re-uploading chunks for {f}")

batch\_size = 10

*for* i *in* range(0, len(chunks), batch\_size):

batch\_end = min(i + batch\_size, len(chunks))

batch = []

*for* j *in* range(i, batch\_end):

doc\_id = f"{current\_id}-{j}"

metadata = {

'text': chunks[j],

'source': f,

'file\_id': current\_id,

'chunk\_id': j

}

batch.append((doc\_id, embeddings[j], metadata))

index.upsert(*vectors*=batch)

*except* Exception *as* e:

print(f"⚠️ Error re-uploading file {f}: {e}")

print("✅ Finished re-uploading remaining files")

*except* Exception *as* e:

print(f"❌ Error during vector deletion: {e}")

*raise* e

def **split\_text**(*text*, *chunk\_size*=500, *overlap*=100):

words = *text*.split()

chunks = [" ".join(words[i:i + *chunk\_size*]) *for* i *in* range(0, len(words), *chunk\_size* - *overlap*)]

*return* chunks

def **augment\_prompt**(*query*):

"""Retrieve relevant context from Pinecone and augment the query."""

*try*:

embed\_model = OpenAIEmbeddings(*model*="text-embedding-ada-002",*api\_key*=OPENAI\_API\_KEY2) *# Consistent model*

index = pc.Index(index\_name,*pinecone\_api\_key*=PINECONE\_API\_KEY2)

vectorstore = LangchainPinecone(index, embed\_model, "text")

print(f"🔍 Searching Pinecone for query: {*query*}") *# Debugging Line*

results = vectorstore.similarity\_search(*query*, *k*=5)

*if* not results:

print("⚠️ No relevant context found in Pinecone.")

*return* f"Query: {*query*}" *# Fallback if no context is found*

*# 🔥 FIX: Use doc.page\_content instead of metadata['text']*

source\_knowledge = "\n".join([doc.page\_content *for* doc *in* results])

print(f"✅ Retrieved {len(results)} relevant chunks from Pinecone") *# Debugging Line*

*return* f"Using the following contexts, answer the query:\n\n{source\_knowledge}\n\nQuery: {*query*}"

*except* Exception *as* e:

print(f"❌ Error retrieving context: {e}")

*return* *query* *# Return original query if retrieval fails*

def **generate\_response**(*augmented\_query*):

*try*:

prompt = HumanMessage(*content*=*augmented\_query*)

res = chat.invoke([SystemMessage(*content*="You are a helpful assistant."), prompt])

*return* res.content

*except* Exception *as* e:

print(f"Error generating response: {e}")

*return* None

*# Main entry point*

*if* \_\_name\_\_ == '\_\_main\_\_':

*# Process existing files first*

process\_existing\_files()

*# Start the file watcher*

observer = setup\_file\_watcher()

*try*:

*# Start the Flask app*

app.run(*debug*=True, *host*='0.0.0.0', *port*=int(os.environ.get("PORT", 5001)))

*finally*:

observer.stop()

observer.join()