**1. Backend with Flask:**

python

Copy code

from flask import Flask, request, jsonify

import time

import pinecone

from langchain.embeddings.openai import OpenAIEmbeddings

from langchain.vectorstores import Pinecone

from flask\_limiter import Limiter

from flask\_limiter.util import get\_remote\_address

from threading import Thread

import logging

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

logging.basicConfig(level=logging.INFO)

# Rate limiter: max 5 requests per user

limiter = Limiter(

get\_remote\_address,

app=app,

default\_limits=["5 per minute"]

)

# Initialize Pinecone and OpenAI Embeddings

pinecone.init(api\_key="your\_pinecone\_api\_key", environment="gcp-starter")

index\_name = "tk-policy"

if index\_name in pinecone.list\_indexes():

pinecone.delete\_index(index\_name)

pinecone.create\_index(name=index\_name, dimension=1536)

index = pinecone.Index(index\_name)

embed = OpenAIEmbeddings(openai\_api\_key="your\_openai\_api\_key")

vectorstore = Pinecone(index, embed.embed\_query, text\_field="text")

# Background process to scrape news articles

def background\_scraper():

while True:

# Dummy scraper

logging.info("Scraping news articles...")

time.sleep(3600) # Scrape every hour

scraping\_thread = Thread(target=background\_scraper)

scraping\_thread.start()

# Health endpoint

@app.route('/health', methods=['GET'])

def health():

return jsonify({"status": "API is active!"}), 200

# Search endpoint with caching and rate limiting

@app.route('/search', methods=['POST'])

@limiter.limit("5/minute", override\_defaults=False)

def search():

start\_time = time.time()

data = request.json

user\_id = data.get("user\_id")

text = data.get("text", "")

top\_k = data.get("top\_k", 5)

threshold = data.get("threshold", 0.5)

# Tokenize the input text, embed it, and query Pinecone

query\_embedding = embed.embed\_query(text)

search\_results = index.query(queries=[query\_embedding], top\_k=top\_k, filter={'score': {'$gte': threshold}})

inference\_time = time.time() - start\_time

logging.info(f"Request handled in {inference\_time} seconds")

return jsonify({"results": search\_results, "inference\_time": inference\_time}), 200

# Dockerfile - Include this in your Docker setup

# Dockerize the app with Flask and Pinecone

"""

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]

"""

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

app.run(host='0.0.0.0', port=5000)

**2. Dockerization (Dockerfile):**

dockerfile

Copy code

# Dockerfile to build the Flask app

FROM python:3.9-slim

WORKDIR /app

# Install dependencies

COPY requirements.txt requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

# Expose port 5000 for Flask

EXPOSE 5000

CMD ["python", "app.py"]

**3. Caching with Redis (Optional):**

python

Copy code

from redis import Redis

import pickle

cache = Redis(host='localhost', port=6379, db=0)

@app.route('/search', methods=['POST'])

@limiter.limit("5/minute", override\_defaults=False)

def search():

user\_id = request.json.get('user\_id')

cache\_key = f"{user\_id}\_{request.json.get('text')}"

# Check if cached response exists

cached\_result = cache.get(cache\_key)

if cached\_result:

return pickle.loads(cached\_result), 200

# Regular processing...

query\_embedding = embed.embed\_query(text)

search\_results = index.query(queries=[query\_embedding], top\_k=top\_k, filter={'score': {'$gte': threshold}})

# Cache the result

cache.setex(cache\_key, 3600, pickle.dumps(search\_results)) # Cache for 1 hour

return jsonify(search\_results), 200

**4. Rate Limiting:**

python

Copy code

@app.route('/search', methods=['POST'])

@limiter.limit("5/minute", override\_defaults=False)

def search():

user\_id = request.json.get('user\_id')

# Increment user API call frequency in DB

user\_record = db.get\_user(user\_id)

if user\_record and user\_record.api\_call\_count >= 5:

return jsonify({"error": "Too many requests"}), 429

# Normal processing

**5. Logging:**

python

Copy code

import logging

logging.basicConfig(level=logging.INFO)

@app.route('/search', methods=['POST'])

def search():

logging.info(f"Received search request from user: {request.json.get('user\_id')}")

# Process request and return response