**Documentação da Fase 3: Lógica de Consulta e Cache**

**Visão Geral da Fase 3**

A Fase 3 do projeto foi dedicada a implementar a capacidade de **consultar a base de conhecimento RAG** (o ChromaDB preenchido na Fase 2) e a otimizar essas consultas através de um mecanismo de **cache**. O objetivo era permitir que o sistema respondesse a perguntas utilizando os dados específicos dos tickets.

**As principais etapas executadas foram:**

1. **Carregamento da Base de Conhecimento:** Conectar-se ao ChromaDB existente no disco.
2. **Configuração do Retriever:** Utilizar o LangChain para criar um componente que busca os chunks de documentos mais relevantes no ChromaDB com base em uma pergunta.
3. **Integração com LLM:** Combinar os chunks recuperados com o LLM (Ollama) para gerar respostas contextuais.
4. **Implementação de Cache:** Adicionar uma camada de cache para armazenar perguntas e suas respostas, garantindo retornos instantâneos para consultas repetidas.

**Desafios Enfrentados e Soluções**

A implementação da Fase 3 transcorreu **sem grandes desafios ou erros diretos**, uma vez que os pré-requisitos das fases anteriores (Ollama funcional, ChromaDB preenchido) estavam estabelecidos. O script de consulta foi capaz de se conectar, buscar e gerar respostas com sucesso desde a primeira execução.

A principal atenção foi dada à correta inicialização do OllamaEmbeddings e do ChromaDB dentro do script de consulta, utilizando o mesmo modelo de embedding da fase de ingestão (nomic-embed-text) e apontando para o diretório de persistência correto.

**Código Final da Fase 3 (consulta\_dados.py)**

Este é o script consulta\_dados.py finalizado da Fase 3, que permite consultar seu sistema RAG e utiliza o cache.

Python

# consulta\_dados.py

import os

import json # Adicionado para formatar a saída do cache se necessário

from langchain\_community.embeddings import OllamaEmbeddings

from langchain\_community.vectorstores import Chroma

from langchain.chains import RetrievalQA

from langchain\_community.llms import Ollama as Ollama\_LLM

# --- CONFIGURAÇÃO ---

# O mesmo diretório de persistência usado na ingestão

PERSIST\_DIRECTORY = "./chroma\_db"

EMBEDDING\_MODEL = "nomic-embed-text" # O mesmo modelo de embedding usado na ingestão

LLM\_MODEL = "mistral" # O modelo LLM que você está rodando no Ollama

# Cache de consultas (para esta demonstração, um dicionário simples em memória)

query\_cache = {}

def get\_rag\_chain():

"""

Inicializa o modelo de embedding, carrega o ChromaDB e configura o retriever e a cadeia de RAG.

"""

print(f"Inicializando modelo de embedding '{EMBEDDING\_MODEL}' para consulta...")

try:

embeddings\_model = OllamaEmbeddings(model=EMBEDDING\_MODEL)

except Exception as e:

print(f"ERRO: Falha ao inicializar o modelo de embedding Ollama: {e}")

print("Certifique-se de que o Ollama Server está rodando e o modelo 'nomic-embed-text' foi baixado.")

return None, None

print(f"Carregando banco de dados ChromaDB de '{PERSIST\_DIRECTORY}'...")

if not os.path.exists(PERSIST\_DIRECTORY):

print(f"ERRO: O diretório de persistência do ChromaDB '{PERSIST\_DIRECTORY}' não foi encontrado.")

print("Certifique-se de que a Fase 2 (ingestão de dados) foi concluída com sucesso.")

return None, None

try:

# Carrega o banco de dados ChromaDB existente

vectorstore = Chroma(

persist\_directory=PERSIST\_DIRECTORY,

embedding\_function=embeddings\_model # É crucial usar a mesma função de embedding!

)

print(f"ChromaDB carregado. Contém {vectorstore.\_collection.count()} documentos.")

except Exception as e:

print(f"ERRO: Falha ao carregar o ChromaDB: {e}")

print("Verifique a integridade da pasta 'chroma\_db'.")

return None, None

# Configura o retriever (busca por similaridade)

# search\_kwargs controla quantos chunks serão retornados (k=N)

retriever = vectorstore.as\_retriever(search\_kwargs={"k": 3})

print(f"Inicializando LLM '{LLM\_MODEL}' com Ollama para Chain de QA...")

try:

llm = Ollama\_LLM(model=LLM\_MODEL)

except Exception as e:

print(f"ERRO: Falha ao inicializar o LLM Ollama: {e}")

print("Certifique-se de que o Ollama Server está rodando e o modelo '{LLM\_MODEL}' foi baixado.")

return None, None

# Configura a cadeia de RAG (Retrieval-Augmented Generation)

# Esta cadeia pega a pergunta, busca chunks relevantes e usa o LLM para gerar uma resposta.

qa\_chain = RetrievalQA.from\_chain\_type(

llm=llm,

chain\_type="stuff", # "stuff" significa que ele coloca todos os chunks no prompt do LLM

retriever=retriever,

return\_source\_documents=True # Para ver de onde a informação veio

)

print("Chain de RAG configurada com sucesso!")

return qa\_chain, retriever

def query\_knowledge\_base(qa\_chain: RetrievalQA, retriever, query: str) -> dict:

"""

Consulta a base de conhecimento e retorna a resposta e as fontes.

Implementa um cache simples em memória.

Args:

qa\_chain (RetrievalQA): A cadeia de RAG configurada.

retriever: O retriever para buscar documentos.

query (str): A pergunta do usuário.

Returns:

dict: Um dicionário contendo a resposta ('result') e os documentos fonte ('source\_documents').

"""

# Verifica o cache primeiro

if query in query\_cache:

print(f"Resultado recuperado do cache para a query: '{query}'")

return query\_cache[query]

print(f"\nProcessando nova query: '{query}'")

# Primeiro, buscar os documentos relevantes (para mostrar ao usuário e entender o RAG)

print("Buscando documentos relevantes na base de conhecimento...")

relevant\_docs = retriever.get\_relevant\_documents(query)

if not relevant\_docs:

print("Nenhum documento relevante encontrado para a consulta. O LLM responderá com conhecimento geral.")

else:

print(f"Encontrados {len(relevant\_docs)} documentos relevantes:")

for i, doc in enumerate(relevant\_docs):

print(f"--- Documento {i+1} (ID: {doc.metadata.get('id', 'N/A')}, Fonte: {doc.metadata.get('source', 'N/A')}) ---")

print(doc.page\_content[:200] + "...")

print("-" \* 50)

# Agora, use a cadeia de RAG para obter a resposta do LLM

print("Gerando resposta com o LLM e os documentos relevantes...")

try:

# A cadeia QA fará a busca e a geração da resposta combinada

response = qa\_chain.invoke({"query": query})

# Armazena no cache antes de retornar

query\_cache[query] = response

return response

except Exception as e:

print(f"ERRO ao gerar resposta com LLM: {e}")

print("Verifique se o LLM está carregado corretamente no Ollama e se há recursos suficientes.")

return {"result": "Desculpe, não foi possível gerar uma resposta no momento.", "source\_documents": []}

if \_\_name\_\_ == "\_\_main\_\_":

qa\_chain, retriever = get\_rag\_chain()

if qa\_chain and retriever:

print("\n--- Sistema de Consulta RAG Pronto ---")

print("Digite sua pergunta ou 'sair' para encerrar.")

while True:

user\_query = input("\nSua pergunta: ")

if user\_query.lower() == 'sair':

print("Encerrando o sistema de consulta.")

break

response = query\_knowledge\_base(qa\_chain, retriever, user\_query)

print("\n--- RESPOSTA ---")

print(response['result'])

if response.get('source\_documents'):

print("\n--- FONTES CONSULTADAS ---")

for i, doc in enumerate(response['source\_documents']):

print(f"[{i+1}] ID: {doc.metadata.get('id', 'N/A')}, Fonte: {doc.metadata.get('source', 'N/A')}")

print("------------------")

else:

print("\nFalha ao inicializar o sistema RAG. Por favor, corrija os erros acima.")