Objetivo do projeto

O projeto tem o intuito apenas de demonstrar o passo a passo de uma simples aplicação RAG (Retrieval-Augmented Generation). São utilizados os componentes dos frameworks LangChain, langGraph e alguns componentes integrados como Google GenAI para os passos de vetorização e geração do output, assim como também o Chroma client para indexação dos embeddings localmente (no caso deste projeto) e posteriormente o passo de recuperação. Também é possível utilizar ainda em conjunto outros frameworks para observabilidade da aplicação, como por exemplo o LangSmith, onde é possível monitorar, realizar debugging e etc. Existem alguns exemplos de como configurar essa etapa, caso tenha interesse, veja a documentação oficial. É possível encontrar todas as informações sobre todos os passos sobre RAG entre outras funcionalidades diretamente na documentação oficial do LangChain. Quanto à documentação oficial do LangGraph, também contém todas as informações necessárias.

O RAG é uma aplicação no modelo de perguntas e respostas, onde é realizada uma busca nos nossos próprios arquivos, ao final, tanto a pergunta quanto o conteúdo recuperado (resposta) são passados ao modelo de LLM escolhido para gerar um output mais amigável ao usuário.

Configuração do Ambiente

Pré-requisitos:

langchain-google-genai ou langchain_openai entre outros modelos que desejar.
langchain-chroma dentre vários outros clientes disponíveis.
langchain-community 0.3+
langchain-text-splitters 0.3+
langchain-core 0.3+
langchain 0.3+
langsmith 0.4+
langgraph 0.5+
python-dotenv 1.1+

Pré-requisitos nativos:

os
typing_extensions

Configuração da chave de API

Para prosseguir com o projeto é necessário utilizar um LLM que fará o processo de vetorização, indexação e também participará do processo de gerar o output para o usuário, por tanto, será necessária utilizar uma chave de API de algum modelo de LLM ou utilizar um modelo local. Para definir a chave de API do modelo LLM escolhido em uma variável de ambiente.

Estrutura da Aplicação

A aplicação segue cinco passos:

O primeiro passo é uma função que define toda a lógica para o processamento dos arquivos que serão futuramente vetorizados e indexados em um diretório local. Primeiro, os arquivos são carregados do diretório onde se encontram, logo depois são contidos em um único objeto como uma unidade de texto. Ao final, essa unidade de texto é dividida em várias partes menores para que sejam futuramente transformadas em embeddings.

O segundo passo define todo o processo de vetorização e indexação dos arquivos previamente carregados e divididos no primeiro passo. Nessa etapa é utilizado um modelo de llm para transformar cada pedaço de texto em embedding, um processo de atribuição de um valor numérico para cada pedaço de texto. Posteriormente, ocorre a indexação desses embeddings, ou seja, o armazenamento em banco de dados local (Nesse caso, foi utilizado o Chroma).

O terceiro passo define o processo de busca com base nas perguntas feitas pelo usuário. É realizada uma recuperação dos arquivos indexados no diretório escolhido no passo 2, e então esse resultado, chamado de contexto é passado para o quarto passo para gerar a resposta final ao usuário através de um modelo de linguagem.

O quarto passo é onde definimos o prompt que será passada ao llm, onde estará tanto a pergunta do usuário quanto o contexto (o conteúdo recuperado dos nossos arquivos) para que seja gerado o output.

O quinto e último passo é onde agrupamos todos os passos anteriores da nossa aplicação com o LangGraph. Nesta etapa, definimos a sequência de qual passo será realizado primero e onde finalizar.