README.md

API Legis - IA Generativa do Banco de Dados LEGIS

A API Legis faz parte do IAGO e fornece um serviço de perguntas e respostas detalhadas sobre a base de documentos legislativos do TCE-GO. Tem como objetivo auxiliar todos os departamentos do TCE-GO, principalmente os auditores, sobre a legislação vigente, auxiliando a tomada de decisões. Abaixo segue uma visão geral sobre o uso do Legis:

A pergunta realizada a partir do IAGO é repassada para a API Legis adicionando o contexto do usuário (seu perfil) e o histórico de outras perguntas e respostas. Esta tripla é utilizado pelo LEGIS para realizar as consultas no banco de dados local, onde estão os documentos legislativos e para LLMs (large language models) na api da OpenAI. Após a resposta ser gerada ela é devolvida para o IAGO.

Para detalhar todo o funcionamento da API Legis, deste documento tem três seções abaixo:

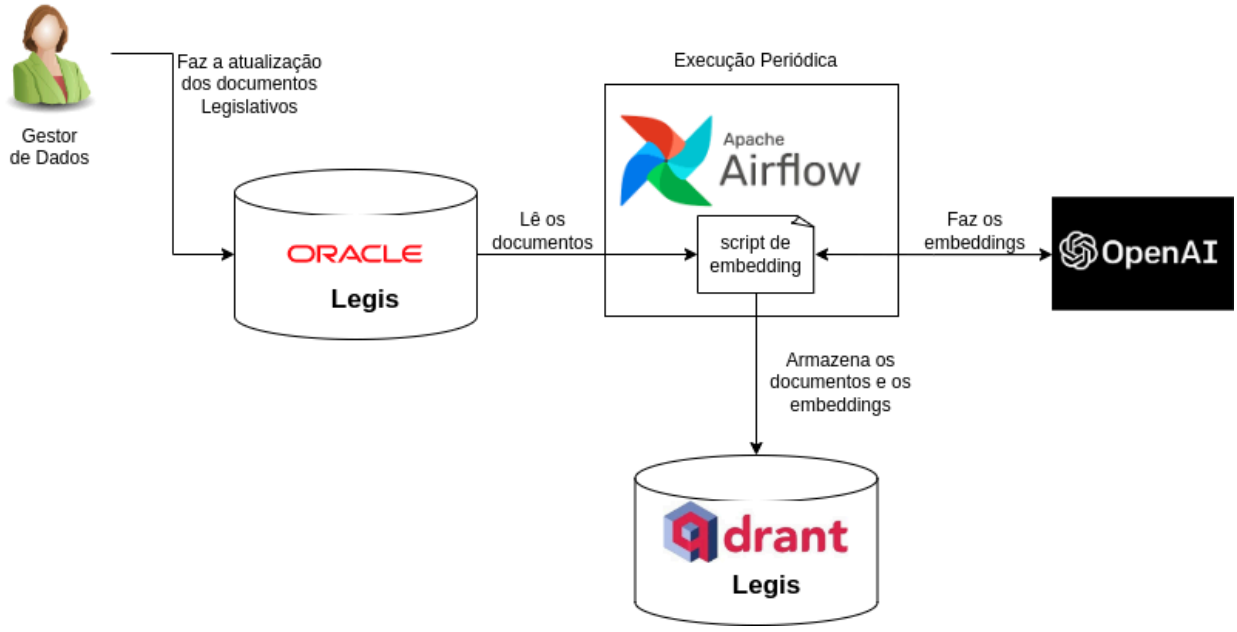
- Detalhamento do pipeline de ingestão de dados
- Detalhamento dos agentes inteligêntes para geração da resposta
- Detalhamento dos componentes de software e bibliotecas utilizadas na API

Pipeline de Dados

O pipeline de dados para a solução API LEgis é uma etapa onde os documentos legislativos são transformados de forma contínua (insert e update) para se adequarem os formatos necessários para que os componentes de AI possam utilizá-los para elaborar as respostas às perguntas. Este pipeline tem duas funções importantes:

1. Ter um fluxo de atualização constante dos documentos legislativos.
2. Ter uma indexação vetorial para que a estratégia RAG (Retrieval-Augmented Generation) possa ser utilizada. A estratégia RAG será descrita em detalhes nos próximos tópicos.

Segue abaixo uma imagem descrevendo o fluxo do pipeline de dados:



O pipeline tem como ponto mais importante o script utilizando a ferramenta AirFlow do TCE, que executa a etapa de ETL (Extract-Transform-Load) dos documentos legislativos. Esta etapa compreende as seguintes operações:

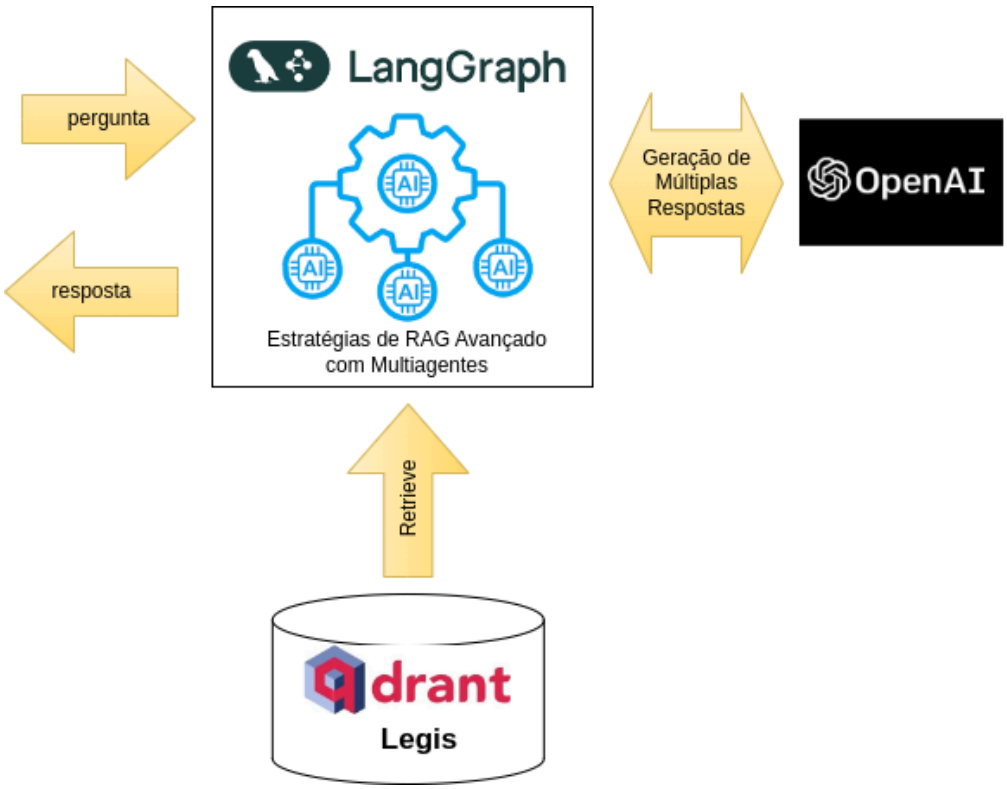
- Ler os documentos que forma inseridos na base de dados oracle. **Obs.: neste momento não estamos assumindo que haverá alterações nos documentos somente inserção de novos.**
- Fazer o particionamentos destes documentos em chuncks (pedaços menores dos documentos). Cada pedaço também contém metadados associados a estes documentos como: número do documento, tipo da normativa, data de publicação, código do setor, nomde do setor, status da vigência. **Obs: o texto legislativo particionado também sofre um tratamento para remover as codificações html.**
- Tanto o chuncks quanto os metadados são enviados para a API da OpenAI para realizar o procedimento de embedding que é a criação um vetor de floats,

mepeando a sintaxe e a semântica do documento. **Obs: o modelo de embedding da OpenAI utilizado atualmente é o text-embedding-ada-002-v2.**

- Os chuncks, os metadados e os embeddings são armazenados no banco de dados Qdrant. **Obs: este banco permite consultar dados a partir dos embeddings e a partir dos filtros dos metadados.**

Este pipeline, da forma como descrito acima, é executada periodiamente (1 vez ao dia) mantendo o banco de dados Qdrant atualizado. Estas funcionalidades de busca do Qdrant favorecem a utilização de da estratégia de RAG, pois, a partir da pergunta enviada para API Legis é possível criar um embedding (vetor) desta pergunta e procurar os documentos no Qdrant que semanticamente podem fornecer a resposta (busca de similaridade). Para isso são utilizadas métricas de busca de similaridade entre os vetores. **Obs: estamos utilizando a métrica de Similaridade de Cossenos, padrão do Qdrant.**

A estratégia de RAG Avaçado é ilustrada na imagem abaixo:



A pergunta enviada para a API Legis é enviada para um sistema de multiagentes de IA que escolhe a melhor estratégia de RAG. A estratégia de RAG tem duas etapas básicas:

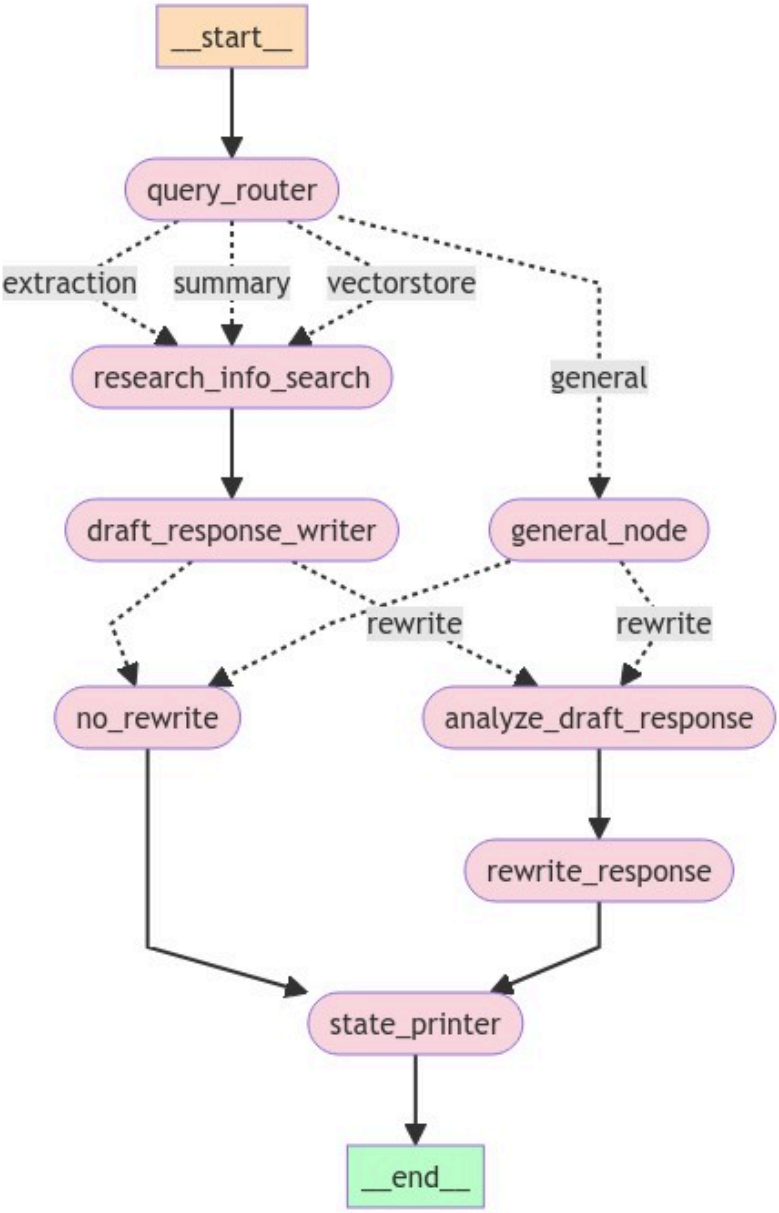
1. Recuperar dados de um banco vetorial a partir da pergunta (Retriever)
2. Gerar uma resposta baseada nos dados recuperados e na pergunta (Generate)

Com o RAG Avançado e os multiagentes podemos, além de recuperar e gerar a resposta, validar e checar, refinar ou refazer a reposta, inclusive obtendo novos dados para complementá-la. Este sistema multiagente é detalhado no próximo tópico.

Agentes Inteligentes e Geração da Resposta

A estratégia utilizada atualmente para construir uma solução para API Legis é a de agentes inteligentes de IA Generativa. As tecnologias utilizadas para construção destes agentes são o Langchain e o LangGraph.

A utilização de agentes (sistema multiagentes) com o LangGraph é bem parecido com uma pipeline, onde um fluxo de informações é construído em etapas descritas (implementadas e configuradas) através de um grafo de fluxo de trabalho. O caminhamento neste grafo tem etapas de tomada de decisão que permitem navegar nos agentes que trarão a melhor estratégia para gerar a resposta correta. Abaixo segue o grafo multiagentes implementado:



Segue uma explicação dos agentes do grafo:

- **query_router**: escolhe a melhor estratégia de RAG analisando o tipo de pergunta. Atualmente temos quatro estratégias mapeadas e duas implementadas. Ainda

faltam ser implementadas: **extration** e **summary**. Elas estão apontando para a estratégia **vectorstore**.

- **research_info_search**: executa a estratégia RAG para o **vectorstore**. Para isso cria três perguntas baseadas na pergunta original, recupera os documentos e gera respostas para cada uma das perguntas. Os documentos são recuperados do Qdrant e a geração das respostas são feitas utilizando a API da OpenAi.
- **general_node**: excuta a estratégia **general**. Nesta estratégia não há necessidade de consultar o Qdrant para obeter documentos, somente utilizar o conhecimento paramétrico para responder a pergunta.
- **draft_response_writer**: nesta etapa da estratégia **vectorstore**, a pergunta original, as perguntas geradas, os documentos recuperados para cada pergunta e as repostas dadas, são refinadas para gerar uma resposta de rascunho (draft), que possivelmente será a resposta final.
- **decide_rewrite**: tanto a estratégia **general** quanto a estratégia **vectorstore** passam por uma etapa de validação da **resposta draft**. Esta tomada de decisão verifica se a resposta é satisfatória ou se ela precisa ser reescrita.
- **no_rewrite**: certifica que a resposta de rascunho é a resposta final e encaminha para a saída.
- **analyse_draft_response**: uma vez que a resposta precisa ser reescrita este agente analisa a resposta atual e cria um feedback de como a resposta pode ser corrigida.
- **rewrite_response**: baseado no feedback da etapa anterior, este agente reescreve a resposta final, fazendo as correções necessárias.
- **state_printer**: é um nó que faz o log das respostas para posterior análise. Ele encaminha a resposta para ser devolvida ao solicitante.

Todos os agentes que precisam tomar dicisões ou gerar respostas, drafts e feedbacks precisam utiliar o LLM como gerador de texto. O modelo LLM utilizado em todos eles é o gpt-4o da API da OpenAI.

Componentes de Software e Bibliotecas

Folders:

- **LEGIS_API/base** -> Código do Legis - CEIA
  - **databases** -> datasets estáticos para uso no desenvolvimento local
  - **imgs** -> imagens para frontend e para a documentação
  - **agents** -> código referentes aos agentes langgraph
    - **app_agents** -> constroi os nós e o grafo de dependências
    - **graph_state** -> descreve as informações que são trafegadas no pipeline do grafo
    - **loader_csv** -> carrega o csv com os dados legis (ambiente de dev)

        - **loader_pdf** -> carrega os pdfs com os dados legis (ambiente de dev)
        - **nodes** -> os nós que implementam a estratégia de cada agente
        - **cond_edges** -> os arestas do grafo que implementam as tomadas de decisão da estratégia de agentes
    - **routers** -> códigos referentes ao router (**deprecated**)
        - **app_routers** -> código fonte do router
        - **loader_legins** -> leitura dos dados legis
        - **sqlquery** -> prompts e configurações da rota que envolve o SQL
        - **vanna** -> teste para uso do vanna
    - **services** -> códigos para configuração de serviços
        - **interface** -> código para aplicação frontend
        - **logger_config** -> configuração da lib logging
        - **server** -> gerenciamento de rotas
    - **utils** -> códigos de utilidade do projeto
        - **config** -> inicialização das config
        - **params** -> inicialização params (Legis)
    - **app** -> definição do endpoint Rest da API Legis
    - **legis.conf** -> arquivo de configuração que utiliza as variáveis de ambiente para: chaves de api, modelos, estratégias, diretórios, etc...
    - **requirements.txt** -> as bibliotecas necessárias para que a solução funcione (dependências).

Como iniciar a API no ambiente de desenvolvimento

Instalação de dependências

Instalar o python 3.9.

Instalar as dependências:

```
cd LEGIS_API/base
pip install -r requirements.txt
```

Utilizando o ChromaDB

Copie para o diretório "LEGIS_API/base/databases", o diretório "db" que está no google drive. Estes são os dados já vetorizados, possibilitando pular esta etapa durante a inicialização. Se precisar alterar este diretório, altere o arquivo "LEGIS_API/base/legis.conf" na seção **agents.index_folder**.

Utilização do Qdrant

Copie para o diretório "LEGIS_API/base/databases", o diretório "local_qdrant" que está no google drive. Estes são os dados já vetorizados, possibilitando pular esta etapa durante a inicialização. Se precisar alterar este diretório, altere o arquivo "LEGIS_API/base/legis.conf" na seção **qdrant.path**.

Iniciando os serviços

Para iniciar o projeto, abra dois terminais e execute os seguintes comandos:

No primeiro terminal, execute:

```
cd LEGIS_API/base
python app.py
```

No segundo terminal, execute:

```
cd LEGIS_API/base
streamlit run services/interface.py
```