Projeto - Portfolio Chatbot

Visão geral

API de **RAG de documento único** para website de portfólio. O pipeline indexa um arquivo fonte (PDF, Markdown ou TXT) com **SentenceTransformers** e **FAISS**, e gera respostas usando **Mistral**. A API é **FastAPI**, com CORS liberado para frontends externos. O desenho privilegia inicialização rápida, **lazy loading** de modelos e deploy simples via **Docker**.

Ferramentas, como foram usadas e para quê

Servidor e API

FastAPI, Uvicorn, Pydantic, CORS

- **Como**: `app.py` define a aplicação, habilita **CORS** para `*`, expõe `GET /health` e `POST /ask`. O endpoint `/ask` recebe `question` (Pydantic), faz **lazy import** de `rag.answer` e retorna `{answer, sources}`.
- **Para quê**: servir a inferência RAG com tipagem, resposta JSON e compatibilidade com frontends (site de portfólio, widget, etc.).

Docker

- **Como**: `Dockerfile` baseado em `python:3.11-slim`, instala `libgomp1` para FAISS, faz o `pip install -r requirements.txt`, copia o código e **pré-baixa** o modelo de embeddings no build para cache. Expõe porta via `PORT` e inicia com `uvicorn`.
- **Para quê**: empacotamento leve e previsível, pronto para Cloud Run ou serviços equivalentes.

RAG “na mão” (sem LangChain)

SentenceTransformers (embeddings)

- **Como**: modelo configurável via `.env`, padrão `paraphrase-multilingual-mpnet-base-v2` em `src/config.py`. No `ingest.py`, carrega o modelo e transforma **chunks** em vetores. No `rag.py`, usa singleton com **lazy loading**.
- **Para quê**: representar semanticamente texto de pergunta e trechos do documento.

FAISS (vector store e busca)

- **Como**: `ingest.py` cria `IndexFlatIP`, normaliza L2, adiciona vetores e salva `data/index/faiss.index`. Também grava `meta.json` com `{source, text}` de cada chunk. O `rag.py` carrega `faiss.index` e `meta.json`, normaliza o embedding da pergunta e faz `search(top_k)`.
- **Para quê**: recuperação semântica rápida e local, sem serviços externos.

Pré-processamento e ingestão

- **Como**: `ingest.py` lê **um** arquivo em `data/source/` (PDF via **pypdf** ou `.md`/`.txt`), aplica split simples em **chunks** configuráveis (`CHUNK_SIZE`, `CHUNK_OVERLAP`), gera embeddings e cria o índice FAISS e `meta.json`.
- **Para quê**: transformar o documento do portfólio em base consultável pelo retriever.

LLM via Mistral (SDK v1)

- **Como**: serviço próprio em `src/llm/mistral_service.py` encapsula o cliente **Mistral**, lê `MISTRAL_API_KEY` e `LLM_MODEL` do ambiente e implementa `generate_response(system, prompt, temperature, max_tokens)`. Em caso de erro, retorna resposta **mock** controlada.
- **Para quê**: gerar a resposta final condicionada pelo contexto recuperado, com controle de temperatura e sistema.

Orquestração da resposta

Módulo `rag.py`

- **Como**: funções internas fazem:

1. **Garantir carregamento** de embeddings, índice e serviço de LLM (lazy singletons).

2. **Buscar contexto** no FAISS com `TOP_K`.

3. **Montar prompts**: um *system prompt* curto sobre persona “Portfolio Bot”, tom profissional e objetivo; um *user prompt* que inclui pergunta e blocos de contexto, com instruções para citar fontes e evitar “over-emoji”.

4. **Chamar o LLM** e **construir citações** com as origens distintas dos trechos retornados.

- **Para quê**: pipeline RAG explícito, legível e sem dependência de frameworks.

Configuração

dotenv e `src/config.py`

- **Como**: `load_dotenv()` no início do projeto. Parâmetros em variáveis de ambiente: `MISTRAL_API_KEY`, `LLM_MODEL`, `EMBEDDING_MODEL`, `TOP_K`, `CHUNK_SIZE`, `CHUNK_OVERLAP`, `TEMPERATURE`.

- **Para quê**: ajustar comportamento em dev e prod sem alterar código.

Fluxo de ponta a ponta

1. **Ingestão**: colocar o **CV** ou `about_me.md` em `data/source/` e rodar `python ingest.py`. Isso gera `data/index/faiss.index` e `data/index/meta.json`.

2. **Pergunta**: `POST /ask` recebe `{"question": "..."};` o servidor carrega embeddings, índice e LLM apenas na primeira chamada.

3. **Recuperação**: calcula embedding da pergunta, normaliza, consulta FAISS e seleciona os `TOP_K` trechos mais relevantes.

4. **Geração**: monta o *system prompt* e o *user prompt* com a pergunta e o contexto; chama o Mistral e recebe o texto.

5. **Resposta**: retorna `answer` e `sources` (nomes do(s) arquivo(s) indexado(s)).

Finalidade de cada componente, em contexto

- **SentenceTransformers**: traduzir texto para vetores, permitindo busca semântica multilingue no portfólio.
- **FAISS**: indexação e busca de alta performance, sem custo de infra adicional.
- **pypdf**: extrair texto de PDFs de forma simples e robusta.
- **Mistral SDK**: geração de linguagem com controle fino de modelo e temperatura.
- **FastAPI + Uvicorn**: expor a funcionalidade como serviço HTTP enxuto, com CORS liberado para integração no site.
- **dotenv/config**: segurança mínima e flexibilidade de parâmetros, inclusive para testes e deploy.
- **Lazy loading**: reduzir **cold start** e evitar falhas de boot quando o índice ainda não existe.

Stack em poucas linhas

- **Linguagem**: Python 3.11
- **API**: FastAPI, Uvicorn, Pydantic, CORS
- **RAG**: SentenceTransformers, FAISS, Numpy
- **Leitura de dados**: pypdf, Markdown/TXT
- **LLM**: Mistral (SDK v1), modelo configurável (`mistral-large-latest` por padrão)
- **Config**: python-dotenv, `src/config.py`
- **Empacotamento**: Docker (imagem slim, cache do modelo no build)

Observações técnicas úteis

- O projeto é **framework-agnostic** para RAG: não usa LangChain, o que reduz dependências e dá controle explícito do pipeline.
- O índice atual é **de um único documento**; a estrutura (`meta.json`) facilita a extensão para múltiplas fontes, se necessário.
- Os testes em `tests/` estão vazios, servindo de esqueleto para futura cobertura.
- O **prompt** é simples e voltado a respostas curtas e objetivas, adequado a um widget de portfólio.
- Como boa prática de produção: considerar **logs estruturados**, métricas de latência, limites de tokens por chamada e roteiros de fallback quando o índice não exis