Projeto - Securemed

Visão geral

Assistente de anamnese médica com **arquitetura privacy-first** e **zero-persistência**. O fluxo combina **RAG** para gerar perguntas clínicas contextualizadas, coleta respostas do paciente em interface **Gradio**, estrutura um sumário clínico em **JSON** e gera um **PDF** em memória. API em **FastAPI**; LLMs e embeddings via **Vertex AI**; base vetorial **ChromaDB**; deploy pensado para **GCP Cloud Run** (API) e **Hugging Face Spaces** (UI).

Ferramentas, como foram usadas e finalidade

Backend e API

FastAPI, Uvicorn, Gunicorn, Pydantic

- **Como**: `src/securemed_chat/main.py` cria a aplicação e inclui o router (`api/endpoints.py`). Endpoints com `APIRouter` expostos sob `/api`. Pydantic modela payloads e validações. Gunicorn + `uvicorn.workers.UvicornWorker` como comando de produção no `Dockerfile`.
- **Para quê**: expor endpoints performáticos e tipados, com inicialização rápida e suporte a streaming.

Docker (multi-stage)

- **Como**: `Dockerfile` compila wheels no estágio builder e copia só o necessário para a imagem final `python:3.11-slim`. Define `PYTHONPATH=/app/src` e executa Gunicorn com 2 workers.
- **Para quê**: imagem leve e pronta para **Cloud Run**, com cold start e custo menores.

Configuração via ambiente (.env) e `python-dotenv`

- **Como**: `core/config.py` lê variáveis como `SECUREMED_API_KEY`, `GOOGLE_CLOUD_PROJECT`, `CHROMA_HOST`, `COLLECTION_NAME`. O Gradio carrega `.env` para URLs da API.

- **Para quê**: separar segredos e parâmetros de runtime, viabilizando dev e produção com segurança.

Autenticação por API Key

- **Como**: dependência no router exige `SECUREMED_API_KEY` (checado em `core/config.py`).

- **Para quê**: proteger os endpoints sem armazenar dados sensíveis do paciente.

LLM, Embeddings e RAG

Google Vertex AI: `ChatVertexAI` e `VertexAIEmbeddings`

- **Como**: `core/llm.py` implementa *lazy loading* com singletons `get_llm()` e `get_embeddings()` para inicializar modelos só na primeira chamada, usando `PROJECT_ID` e `REGION`.

- **Para quê**: reduzir tempo de inicialização e custo, padronizando acesso aos modelos com logging estruturado.

LangChain (core, community)

- **Como**: `services/rag_service.py` monta *chains* com `Runnable`, `RunnablePassthrough`, `ChatPromptTemplate`, `JsonOutputParser` e encadeia:

    1. **Retriever** com **Chroma**

    2. **Prompt** contextualizado com instruções por idioma

    3. **LLM** do Vertex AI

    4. **Parsers** para streaming de perguntas e para estruturador em JSON do sumário clínico

- **Para quê**: orquestrar RAG, separar prompts por etapa e garantir saída estruturada.

ChromaDB (vector store)

- **Como**: conectado via `chromadb` e `langchain_community.vectorstores.Chroma`, usando host/porta em `core/config.py`. O **retriever usa MMR** (Maximum Marginal Relevance) para diversidade de contexto.

- **Para quê**: recuperar trechos médicos relevantes e diversos, melhorando a qualidade das perguntas e do sumário.

Geração de PDF e i18n

ReportLab

- **Como**: `services/pdf_service.py` cria **PDF em memória** com `io.BytesIO` e `canvas`, renderizando seções a partir do JSON estruturado (anamnese, antecedentes, medicamentos, alergias etc.). Usa `Paragraph` para melhor quebra de texto.

- **Para quê**: exportar um sumário clínico compartilhável sem gravar em disco, alinhado ao princípio de zero-persistência.

Internacionalização (EN/PT)

- **Como**: dicionário de traduções em `pdf_service.py` e instruções multilíngues no Gradio. Prompts e labels adaptados via `lang` em endpoints e UI.

- **Para quê**: experiência bilíngue consistente na coleta e na saída do relatório.

Frontend e Integração

Gradio

- **Como**: dois frontends (`gradio_app.py` raiz em PT-BR e `gradio/gradio_app.py` com EN/PT e UI mobile-first). Chamadas assíncronas com **httpx** para:

    1. **/initial-questions-stream**: obter perguntas iniciais

    2. **/refine-questions-stream**: aprofundar histórico

3. **/generate-pdf**: baixar o PDF
   Estados de UI controlam as etapas e exibem mensagens de erro.

- **Para quê**: interface simples, hospedável em **Hugging Face Spaces**, sem backends adicionais.

HTTPX e Requests

- **Como**: consumo da API com timeouts e tratamento de exceções (exibição de HTML de erro no Gradio).

- **Para quê**: robustez na comunicação cliente-servidor.

Observabilidade e Resiliência

Logging estruturado

- **Como**: substituição de `print()` por `logging` em `core/llm.py`, `rag_service.py` e tratamento granular de exceções em `api/endpoints.py` e `pdf_service.py`.

- **Para quê**: diagnóstico de produção, rastreabilidade de falhas e análise de latência.

Tratamento de erros e sanitização

- **Como**: sanitização da `chief_complaint`, `HTTPException` com códigos adequados (400, 503, 500) e mensagens amigáveis para UI.

- **Para quê**: proteger o sistema contra entradas malformadas e quedas de serviços externos.

Endpoints principais e fluxo

1. **`POST /api/initial-questions-stream`**
   - Entrada: queixa principal, idade, sexo, gestante, idioma.
   - Ação: RAG busca contexto, gera **perguntas iniciais** por streaming.

2. `POST /api/refine-questions-stream`
    - Entrada: respostas dos sintomas, dados demográficos e idioma.
    - Ação: novo RAG para **perguntas de aprofundamento** (história pregressa, medicamentos, alergias, fatores de risco).
3. `POST /api/generate-pdf`
    - Entrada: respostas consolidadas dos passos anteriores, queixa principal e idioma.
    - Ação: *structuring chain* sintetiza **JSON clínico**; `pdf_service` gera **PDF em memória** e retorna o arquivo.
4. `GET /`
    - Saúde do serviço e instrução para `/docs`.

Arquitetura lógica resumida

- **UI (Gradio)**: coleta dados, chama endpoints, guia o usuário por etapas e baixa o PDF.
- **API (FastAPI)**: orquestra o fluxo, aplica autenticação por API key, valida entradas.
- **RAG Service (LangChain + Chroma)**: recupera contexto médico com **MMR** e gera perguntas e sumário via **Vertex AI**.
- **PDF Service (ReportLab)**: renderiza o sumário clínico em EN/PT **sem gravar em disco**.
- **Config/Segurança**: `.env` e variáveis de ambiente; **zero-persistência** de dados sensíveis.

Stack final em poucas linhas

- **Linguagem**: Python 3.11
- **API**: FastAPI + Gunicorn/Uvicorn
- **LLM/Embeddings**: Google Vertex AI (ChatVertexAI, VertexAIEmbeddings)
- **RAG**: LangChain, ChromaDB (MMR retriever)
- **Frontend**: Gradio (EN/PT), httpx
- **Relatórios**: ReportLab (PDF em memória)
- **Infra**: Docker, GCP Cloud Run (API), VM p/ Chroma, Hugging Face Spaces (UI)
- **Observabilidade**: logging estruturado; tratamento de exceções por endpoint

---

Observações técnicas relevantes

- **Lazy loading** de LLM e embeddings reduz cold start.
- **MMR no retriever** tende a aumentar a diversidade do contexto, melhorando perguntas e reduzindo redundância.
- **Zero-persistência**: não há banco de dados de pacientes; PDFs são gerados em memória e enviados diretamente.
- **i18n completo**: prompts, UI e PDF localizados.
- **Segurança operacional**: `SECUREMED_API_KEY` obrigatório em produção; variables de ambiente para hosts e coleções do Chroma.