Escrita Sincerta LLM – Guia Técnico e Manifesto Local

Formato padrão: **Contexto → Solução → Exemplo/Código → Checklist**. Linguagem direta, técnica e em português-BR.

1️⃣ Contexto – Objetivo do Projeto

- Executar uma **LLM localmente** (sem depender da nuvem) com **interface web**, **agentes dedicados** e **memória vetorial persistente**.
- Integrar uma **camada de persona e ética** (“Escrita Sincerta”) que define tom, regras e coerência das respostas.
- Garantir **reprodutibilidade** e **facilidade de manutenção**, usando **Docker**, **Makefile** e suporte multiplataforma (Windows/Linux).
- Viabilizar futuras expansões: suporte a múltiplos modelos (Qwen, Llama, Phi), troca de interface (Open WebUI / Text Generation WebUI) e integração de ferramentas (RAG, análise de arquivos, web search opcional).

2️⃣ Solução – Arquitetura Base

Stack Principal

- **Ollama**: runtime local para modelos (CPU/GPU).
- **Open WebUI**: interface simples, com histórico, RAG básico e upload de arquivos.
- **Postgres + pgvector**: base de conhecimento vetorial para contexto e memória.
- **FastAPI (API Orquestradora)**: expõe endpoints para chat, ingestão de dados e controle dos agentes.
- **Traefik (opcional)**: camada de roteamento e autenticação.

Fluxo Operacional

1. Usuário → **Open WebUI** → **API Orquestradora (FastAPI)** → **Ollama** (LLM) → Tools (RAG, leitura de arquivos, etc.).
2. O **Manifesto Sincerta** define o comportamento e os limites éticos da IA via *system prompt*.
3. Logs e histórico gerenciados por **stdout**, **uvicorn** e registros do **Open WebUI**.

Pontos Fortes

- 100% offline e local.
- Suporte a múltiplos modelos e quantizações sem alterar a estrutura.
- Agentes independentes e testáveis (foco em manutenção e escalabilidade).

3️⃣ Estrutura Técnica e Exemplo de Implementação

A estrutura completa do projeto está organizada conforme abaixo:

```
escrita-sincerta-llm/
 ├─ .env.example
 ├─ docker-compose.yml
 ├─ Makefile
 ├─ README.md
 ├─ data/
 │   ├─ docs/              # PDFs, TXTs, MDs para ingestão RAG
 │   └─ vectors/           # volume do pgvector
 ├─ api/
 │   ├─ app.py             # FastAPI + endpoints principais
 │   ├─ agents/
 │   │   ├─ base.py
 │   │   ├─ dev_fullstack.py
 │   │   └─ reflexivo.py
 │   ├─ tools/
 │   │   ├─ files.py
 │   │   ├─ rag.py
 │   │   └─ sysinfo.py
 │   ├─ prompts/
 │   │   ├─ manifesto_sincerta.md
 │   │   ├─ system_base.md
 │   │   └─ styles.json
 │   └─ settings.py
 └─ scripts/
     ├─ pull-models.sh
     ├─ dev.ps1
     └─ dev.sh
```

Cada arquivo cumpre função específica, garantindo modularidade, rastreabilidade e reuso.

4️⃣ Execução e Checklist

1. **Clonar** ou extrair o projeto e copiar `.env.example` → `.env`. Ajustar portas e `OLLAMA_MODELS`.
2. Rodar `docker compose up -d --build`.
3. Acessar **Open WebUI**: http://localhost:3000.
4. Baixar modelos: `make pull` ou direto pela interface.
5. Testar chat com o modelo padrão.
6. Inserir arquivos em `data/docs` → `make ingest` para indexação.
7. Usar agentes disponíveis: `dev_fullstack` ou `reflexivo`.
8. Monitorar com `make logs`.
9. Personalizar o comportamento em `api/prompts/manifesto_sincerta.md`.

5️⃣ Decisões e Trade-offs

- **Ollama x LM Studio** → Ollama é simples e ideal para automação, mas o **LM Studio** facilita testes de múltiplos modelos (útil para alternância dinâmica conforme complexidade do código).
- **Modelos recomendados para alternância**:
- `phi3:3.8b` → leve e rápido para tarefas curtas.
- `qwen2.5:7b` → equilíbrio ideal para desenvolvimento cotidiano.
- `gpt-oss-20b` → raciocínio e builds complexos.
- **Open WebUI** → inicialização rápida; pode ser substituído por Text Generation WebUI para tunagem.
- **pgvector** → persistência confiável para grandes volumes de dados.
- **Embeddings locais** → placeholder até integração de modelos como `bge-m3` ou `gte-small`.

6️⃣ Próximos Passos

1. Conectar embeddings locais (`bge-m3` via Ollama) e implementar similaridade real no `rag.py`.
2. Criar ferramentas adicionais: leitor de planilhas, chunking para PDF/HTML e executor seguro de código.
3. Registrar histórico de conversas no Postgres.
4. Expandir `styles.json` para perfis customizados e estilos de resposta.
5. Adicionar painel de métricas (latência, tokens, modelo ativo).

7️⃣ Solução de Problemas (Troubleshooting)

- **UI sem resposta:** verificar `ollama` → `curl localhost:11434/api/tags`.
- **Modelo falha ao carregar:** revisar logs do Ollama; ajustar quantização (`Q4_K_M` / `Q6_K`).
- **Erro 500 na API:** `docker compose logs -f api`.
- **pgvector não inicializa:** confirmar extensão `vector` habilitada.

8️⃣ Manifesto “Escrita Sincerta” – Versão 0.1

- A verdade está acima de agradar: se não souber, admita e proponha solução.
- Entregue direção, não divagação: resolva antes de pedir novos dados.
- Clareza brutal e empatia genuína.
- Sem promessas vazias, sem floreios, sem redundância.

Prompt base (para uso externo à API):

```
Você é minha chegada. Especialista em IA, Python e full-stack.
Seja direto, em PT-BR, técnico e transparente.
Formato: Contexto → Solução → Exemplo/Código → Checklist.
Se houver ambiguidade, assuma e entregue uma primeira versão funcional.
O código deve estar pronto para produção.
```