Relatório Técnico — Plataforma LLM-SQL para ERP (versão inicial)

Objetivo do documento
Consolidar, de forma clara e apresentável à gerência, as tecnologias, a arquitetura, os padrões e o status atual do projeto que permite consultar o banco de dados do ERP via linguagem natural (LLM → SQL) com respostas rápidas e seguras.

1) Visão Geral (Executive Summary)
- Problema: Consultas ao ERP (MySQL 5.5) dependem de SQL manual, dificultando o acesso rápido a informações por usuários não técnicos.
- Solução: API local que recebe perguntas em linguagem natural, transforma em SQL com auxílio de um modelo de linguagem (LLM) e retorna os resultados.
- Benefícios: Agilidade de decisão, autosserviço de informações, redução de demanda ao time de TI, padronização e auditabilidade.
- Status: API funcional em ambiente local (Windows), documentação interativa via Swagger, integração LLM local (Ollama + modelo 7B) em uso; módulos de segurança, multi-tenant e canais (WhatsApp) em fase de planejamento/validação.

2) Escopo Atual do Projeto
- Perguntas de negócio → Geração de SQL → Execução read-only no MySQL 5.5 → Retorno tabular/JSON.
- Cobertura inicial: consultas de vendas, títulos financeiros e cadastros (expansível via dicionário de dados).
- Ambiente de desenvolvimento: Windows (máquina local), porta 8080.

3) Arquitetura (alto nível)
Usuário (web/Swagger) / Futuro WhatsApp
■
▼
FastAPI (Uvicorn) ■■■ Valida entrada, orquestra etapas
■
■■■ Módulo LLM (Ollama + Gemma 7B Instruct)
■ ■■■ Prompt + Contexto (dicionário de dados, exemplos)
■
■■■ Camada SQL (conexão read-only ao MySQL 5.5)
■ ■■■ Execução segura (whitelist/parametrização)
■
■■■ Resposta JSON / CSV (Swagger)

- RAG leve: dicionário de dados (JSON) embutido no prompt para orientar a geração de SQL.
- Multi-tenant (planejado): isolamento por empresa com credenciais/rotas separados.

4) Componentes e Tecnologias

4.1 Backend/API
- Linguagem: Python 3.x
- Framework: FastAPI (performance + tipagem + OpenAPI automático)
- Servidor ASGI: Uvicorn (modo desenvolvimento; produção com reverse proxy planejado)
- Documentação interativa: Swagger UI em /docs e OpenAPI em /openapi.json
- Bibliotecas de suporte:
- pydantic (validação), requests (HTTP externo/LLM), python-dotenv (segredos), logging (auditoria), typing/dataclasses (tipagem), utilitários variados.

4.2 Runtime de IA (LLM)
- Ollama (local) — gerencia e serve modelos via HTTP.
- Modelo atual (em uso): Gemma 7B Instruct (quantizado) — bom custo/latência em CPU.

- Modelos em teste: Llama 3 8B Instruct, Mistral 7B Instruct (para comparar qualidade/velocidade).
- Alternativas (planejado):
- vLLM (servidor performático) para modelos maiores,
- GPT■OSS■20B ou provedores externos quando necessário (trade-off custo/latência/privacidade).

4.3 Banco de Dados
- SGBD: MySQL 5.5 (legado, ERP)
- Acesso: conector Python (ex.: mysql-connector-python ou PyMySQL)
- Padrão de uso: read-only, timeouts, LIMIT padrão e pool de conexões (planejado) para estabilidade.
- Ferramentas: HeidiSQL (inspeção), scripts de extração de schema para o dicionário de dados.

4.4 Camada de Conhecimento (RAG leve)
- Dicionário de dados: JSON com tabelas, colunas, chaves e descrições de negócio (mantido no repositório).
- Uso: incluído no prompt para guiar a geração SQL e reduzir alucinação.
- Evoluções (planejado): índice vetorial (FAISS/Chroma) para escalabilidade do contexto; exemplos canônicos de SQL por área (few-shot).

4.5 Agente/Orquestração
- Script: llm_agent.py (gera prompt, chama LLM via HTTP, pós-processa SQL, chama banco e formata resposta).
- Pontos de atenção: tratamento de exceções, timeouts, retry/backoff, circuit breaker (planejado), testes automatizados.

4.6 Interfaces do Usuário
- Atual: Swagger UI (/docs) para explorar e testar a API.
- Futuro: Canal WhatsApp via Meta WhatsApp Cloud API (ou Twilio/360dialog): webhook FastAPI, verificação de sender, controle de sessão e rate limiting.

5) Endpoints Principais (API)
- POST /consulta
- Entrada: { "pergunta": "quantos pedidos de venda ontem?", "formato": "tabela|json", "limite": 100 }
- Fluxo: validação → geração SQL pelo LLM (com dicionário) → verificação de segurança (allowlist de tabelas, no DML/DDL) → execução MySQL → retorno.
- Saída (exemplo):
{
"sql": "SELECT COUNT(*) AS qtd FROM mgpve01010 WHERE DT_PVE = CURDATE() - INTERVAL 1 DAY;",
"dados": [{"qtd": 127}],
"latencia_ms": 840
}

6) Segurança e Governança
- Princípios: mínimo privilégio (read-only), isolamento por cliente (multi-tenant), auditabilidade e reprodutibilidade.
- Medidas:
- Allowlist de tabelas/colunas por área; bloqueio de UPDATE/DELETE/INSERT/DDL.
- Parametrização de consultas, LIMIT e timeouts.
- Registro de auditoria: pergunta original, SQL gerado, tempo, usuário/empresa (hash/anônimo quando necessário).
- Segredos em .env (planejado: cofre/vault).
- Criptografia em trânsito (HTTPS com reverse proxy; planejado para prod).
- Controles de acesso: chave de API/JWT (planejado), perfis e cotas.

7) Desempenho e Confiabilidade
- Metas (referência inicial):
- P95 de resposta < 3 s em consultas simples (em CPU local com modelo 7B quantizado).
- Disponibilidade ≥ 99,5% (em ambiente controlado) — dependerá da topologia final.
- Otimizadores:
- Prompt enxuto + few-shot; modelos 7B otimizados; quantização; n_ctx ajustado.
- Caching de perguntas/SQL frequentes (planejado: Redis/SQLite).
- Pool de conexões e índices no MySQL para consultas críticas.

8) Monitoramento, Logs e Qualidade
- Logs: Python logging (níveis INFO/ERROR), correlação por requisição.
- Monitoramento (planejado): métricas de API (Prometheus + Grafana), erros/exceções (Sentry), health checks.
- Qualidade:
- Test set de perguntas com SQL esperado (golden set) e playbooks de regressão.
- Evaluation harness para comparar modelos (acurácia SQL, latência, custo).

9) Implantação e Infra
- Atual (dev): Windows local, Uvicorn (:8080).
- Planejado (prod):
- Docker/Docker Compose; Nginx como reverse proxy (TLS, compressão, rate limit).
- Logs centralizados; rolling updates; backups de configs/índices.
- Multi-tenant: segregação por credenciais/instâncias ou schema dedicado por cliente.

10) Custos (estimativas qualitativas)
- Infra local (CPU) com 7B: custo baixo, latência aceitável; sem custo por token.
- Modelos maiores/externos: custo por uso (tokens/chamadas); melhor qualidade/latência, impacto orçamentário.
- WhatsApp: custo por conversa (Meta) via provedor escolhido.

11) Riscos & Mitigações
- Qualidade do SQL: mitigar com allowlist, exemplos canônicos, revisão de prompts e golden set.
- Latência: modelos 7B quantizados, caching, hardware com mais RAM/GPU se necessário.
- Segurança de dados: read-only, segregação, auditoria, TLS, vault para segredos.
- Legado MySQL 5.5: limites de recursos e recursos SQL antigos → testes + otimização de índices.

12) Roadmap (próximas 4–8 semanas)
1. Endurecer segurança: allowlist por domínio, timeouts, auditoria estruturada.
2. Estabilizar agente: tratamento de erros, retry/backoff, testes de integração.
3. Avaliação de modelos: comparar Gemma 7B × Mistral 7B × Llama 3 8B (qualidade/latência).
4. Pool e caching: conexões MySQL + cache de resultados.
5. WhatsApp PoC: webhook FastAPI, fluxo Q&A, políticas de mensagem.
6. Empacotar: Docker + Nginx + TLS + documentação de deploy.

13) Como usar (Swagger e cURL)
- Acesse: http://localhost:8080/docs → POST /consulta → preencha o body com { "pergunta": "..." } → Execute.
- Exemplo cURL:
curl -X POST http://localhost:8080/consulta -H "Content-Type: application/json" -d '{"pergunta": "quantos pedidos de venda ontem?", "limite": 100}'

14) Itens de Configuração (checklist)
- .env: credenciais MySQL (usuário read-only), URL do Ollama, chaves futuras (JWT, WhatsApp).
- allowlist.json: tabelas/colunas permitidas por domínio de negócio.
- schema_dict.json: dicionário de dados atual.

- prompts/: templates de geração (SQL, validação, reformatação).

15) Anexos (referências internas)
- Repositório: LLM-SQL-skeleton (API + agente + prompts + dicionário).
- Dicionário de dados: blocos para mgpve01010 (Pedidos), mgcta01014 (Financeiro), etc.
- Ambiente local: Windows + Ollama (modelo 7B) + FastAPI/Uvicorn.

Resumo
A plataforma está funcional em ambiente local, com API documentada e LLM local (7B) em uso. Próximos passos focam em segurança, confiabilidade, avaliação de modelos e habilitação de canais (WhatsApp), além do empacotamento para produção (Docker + Nginx + TLS).