Technical Documentation

1) Project overview & architecture

Goal: unify two financial sources (QuickBooks, Rootfi), normalize to a monthly fact table, expose analysis APIs, and support **natural-language** querying with **LLM fallback**.

Layers

- **Parsers** (app/parsers):
    - quickbooks.py flattens columnar P&L; classifies accounts (revenue/cogs/expense).
    - rootfi.py walks nested line_items into atomic facts.
- **Repositories** (app/repositories): SQL helpers for facts/metrics, queries for trends and expense.
- **Services** (app/services):
    - ingestion.py (payload ingest + startup auto-ingest)
    - analytics.py (z-score anomalies)
    - nlq.py (rule-based intents + LLM fallback, conversation & trace logging)
- **Routers** (app/routers): ingest, metrics, analytics, nlq, obs, health
- **DB** (app/db): SQLite + WAL, schema bootstrap
- **Domain** (app/domain): Request and Response Schema
- **Observability** (app/obs): JSON logs, Prometheus, trace store
- **Eval** (app/eval): NLQ evaluation
- **Config** (app/config.py): env-driven settings

Schema (key tables)

- facts(period_start, period_end, month_key, source, account, category, kind, amount)
- metrics(period_end, source, revenue, cogs, gross_profit, expenses, net_profit)
- conversations, messages (NLQ context)
- ai_traces(ts, conversation_id, question, answer, model, tokens_prompt, tokens_completion, latency_ms, tool_calls)

Conflict policy: When both sources provide a period, we **prefer Rootfi** for net_profit. Other categories sum by source and month.

2) Technology stack & rationale

FastAPI, SQLite (WAL), Pydantic, Prometheus and **Docker**

3) Setup & installation (Docker-first)

- **Clone → fill .env → build → up**:

  ```
  docker compose build
  docker compose up -d
  ```

- **Logs (service name api)**: docker compose logs --no-color --tail=200 api

4) AI/ML approach & model choices

Design: "Deterministic first, generative second."

- **Rule-based intents** cover high-value queries with transparent logic:
  - Q1/Q2 profit, yearly revenue trends, top expense increase, quarter comparisons.
- **LLM fallback** runs when a query doesn't match intents:
  - Uses **system prompt** to keep answers concise and numeric.
  - **Model forcing** via X-Model: gpt-4o or gpt-4o-mini.
  - **Evaluation**: the app/eval/eval_run.py script runs a small battery, storing CSV + JSONL (answers + traces) for review.

Why gpt-4o-mini + gpt-4o?
Mini is fast/cost-efficient for default; 4o provides stronger reasoning and writing quality when forced or in A/B runs.

5) Known issues & limitations

- **Parser assumptions**: atypical QuickBooks/Rootfi layouts may need tweaks.
- **Partial reconciliation**: we don't resolve deep conflicts across sources (simple preference + sums).
- **LLM grounding**: the LLM doesn't execute SQL; it summarizes numbers provided by deterministic tools.
- **Security**: no auth in this MVP; put behind your gateway and add auth before production.
- **Scaling**: SQLite is fine for single-tenant or pilot. For multi-tenant or heavy write, use Postgres.

6) Ops notes

- **Metrics & logs**: scrape /metrics; send container stdout to your logging stack; traces are queryable via /api/v1/obs/*.
- **Model A/B**: MODEL_VARIANTS controls allowed models; you can force per-request with X-Model, or allow random A/B (service chooses among variants when no header is sent).