FoodFlow AI — Complete Interview Study Guide (File-by-File)

This is a practical, interview-focused walkthrough of the codebase.

---

0) 90-second project pitch (say this first)

FoodFlow AI is a full-stack food-waste reduction platform built around a synthetic retail supply-chain dataset. It combines: - **Demand forecasting** (XGBoost, optional Prophet ensemble), - **Operational decisioning** (3-tier waste cascade: retailer → food bank → compost), - **Logistics optimization** (OR-Tools VRP with greedy fallback), - **Carbon accounting** (deterministic emissions/ savings calculations), - **User surfaces** (Streamlit dashboard/chatbot/agentic app + FastAPI backend), - **Project MCP server** for structured context/tool access.

Core value proposition: reduce waste, recover cost, and quantify CO_2 impact.

---

1) End-to-end workflows

A. Boot workflow (`python run.py`)

1. Seed DuckDB with synthetic data (`data/seed_database.py`)
2. Train demand model (`models/demand_forecaster.py`)
3. Generate cascade actions (`models/waste_cascade.py`)
4. Optimize routes (`models/route_optimizer.py`)
5. Launch FastAPI (`api/main.py`) + Streamlit multipage app (`app.py`)

B. Streamlit runtime workflow (`app.py` + `pages/*`)

1. Ensures `data/foodflow.duckdb` exists
2. Creates per-page DB copies (`foodflow_dashboard.duckdb` , etc.)
3. Routes to page modules:
4. `pages/1_dashboard.py`
5. `pages/2_chatbot.py`
6. `pages/3_agentic.py`

C. Chatbot workflow (`pages/2_chatbot.py`)

1. Build KB text from DB (`utils/knowledge_base.py`)
2. Send user prompt to Mistral with tool definitions
3. If tool-call requested:
4. `query_database` (SQL)
5. `get_themed_analysis` (mapped SQL)
6. `query_project_mcp` (via `mcporter` to MCP server)
7. Return final natural-language answer + optional chart

D. Agentic workflow (`pages/3_agentic.py`)

Mode-driven analysis: executive summary, deep-dive, client report, live SQL, recommendations, Metabase actions, report generation.

E. API workflow (`api/main.py`)

REST endpoints call helpers/models and return JSON for analytics, forecasts, cascade, routes, carbon, and health.

2) File-by-file explanation

Below covers every meaningful source/config/docs file in the repo.

Root-level docs/config

`README.md`

- **Purpose:** Main setup + architecture + uv-first + Mistral + Metabase + MCP instructions.
- **Workflow role:** Onboarding and local runbook.
- **Interview note:** Mentions modernized setup (env-based secrets, `mistral-large-2512`, local Metabase Docker, project MCP).

`DATA_MODEL_METHODOLOGY.md`

- **Purpose:** Conceptual explanation of data design, model choices, and formulas.
- **Workflow role:** Narrative/documentation for evaluators.
- **Interview note:** Use this to explain *why* each component exists, not just *what*.

`PROJECT_WORKFLOW.md`

- **Purpose:** Architecture and sequence documentation (with mermaid flows).
- **Workflow role:** High-level process map.
- **Interview note:** Useful for “walk me through system flow” questions.

`pyproject.toml`

- **Purpose:** uv project definition + dependencies.
- **Workflow role:** Primary dependency source (`uv sync`).

- **Interview note:** Python >=3.12, includes `fastmcp`, `mistralai`, `duckdb`, `ortools`, `xgboost`, etc.

`requirements.txt`

- **Purpose:** pip-style dependency list.
- **Workflow role:** Compatibility/fallback install path.

`.env.example`

- **Purpose:** Env variable template.
- **Workflow role:** Secret/config centralization.
- **Key vars:** `MISTRAL_*`, `METABASE_*`, `FOODFLOW_MCP_*`.

`run.py`

- **Purpose:** One-command orchestrator.
- **Key functions:** `print_banner`, `step`, `main`.
- **Workflow role:** Runs full prep pipeline and starts API + Streamlit.
- **Interview note:** Good demo script for hackathon judges.

`app.py`

- **Purpose:** Unified Streamlit multipage entry.
- **Workflow role:** Actual UI shell in current architecture.
- **Key behavior:** Calls `copy_db_for_page` for each page to avoid DuckDB lock issues.

`main.py`

- **Purpose:** Placeholder “hello” entry.
- **Workflow role:** Not part of production flow.

`uv.lock`

- **Purpose:** Resolved dependency lockfile for reproducibility.

API layer

`api/__init__.py`

- Empty package marker.

`api/main.py`

- **Purpose:** FastAPI backend.
- **Endpoint groups:**
- `/api/overview`, `/api/daily-waste-trend`
- `/api/products`, `/api/stores`
- `/api/forecast*`
- `/api/cascade*`
- `/api/routes*`
- `/api/carbon*`
- `/api/analytics/*`
- `/api/inventory/critical`, `/api/health`
- **Workflow role:** Programmatic interface for dashboards/integrations.
- **Interview note:** Nice separation between UI and service API.

Database layer

`database/__init__.py`

- Empty package marker.

`database/db.py`

- **Purpose:** DuckDB schema + connection manager + per-page DB copy strategy.
- **Key functions:**
- `copy_db_for_page`, `set_page_db`
- `get_db`, `query_df`, `query_one`, `query_scalar`
- `init_database`, `reset_database`
- **Tables created:** `products`, `stores`, `suppliers`, `supplier_products`, `weather`, `events`, `sales`, `forecasts`, `waste_cascade_actions`, `routes`, `carbon_impact`, `inventory`.
- **Workflow role:** Foundational persistence for all modules.
- **Interview note:** Great talking point: solved DuckDB lock contention pragmatically by page-level copies.

Data engineering scripts

`data/__init__.py`

- Empty package marker.

`data/seed_database.py`

- **Purpose:** Generates realistic synthetic data for 2025.
- **Key functions:**
- `set_random_seed`

- `generate_weather_data`
- `generate_events`
- `generate_sales_data`
- `seed_database`
- **Workflow role:** Creates all core data used by models/UI/API.
- **Interview note:** Data generation includes seasonality, weather, events, store capacity effects, perishability, and waste economics.

`data/export_csv.py`

- **Purpose:** Exports raw + analytics CSVs + Kaggle metadata.
- **Key functions:** `export_raw_tables`, `export_analytics`, `write_kaggle_metadata`, `main`.
- **Workflow role:** Dataset publishing/BI handoff.

`data/export_random_csv_bundle.py`

- **Purpose:** One-shot random seed → CSV bundle (raw + analytics + manifest).
- **Workflow role:** Reproducible synthetic dataset packaging.
- **Interview note:** Nice for portfolio/public dataset publication.

`data/load_duckdb.py`

- **Purpose:** Loads exported CSV folders into DuckDB `raw` / `analytics` schemas.
- **Workflow role:** Rehydration pipeline from CSV into analytical DB.

`data/load_csv_to_duckdb.py`

- **Purpose:** Flexible bundle loader with path resolution, sanitized table names, overwrite support.
- **Workflow role:** Alternative ingestion utility with simple KPI view creation.

Model / optimization layer

`models/__init__.py`

- Empty package marker.

`models/demand_forecaster.py`

- **Purpose:** Forecast engine (XGBoost primary, Prophet optional ensemble).
- **Class:** `DemandForecaster`
- **Key methods:**
- `_create_features` (temporal, cyclical, lag, rolling, trend)
- `train`
- `predict`
- `save_forecasts`
- `batch_forecast`
- `get_surplus_alerts`
- **Workflow role:** Drives forecasting and surplus detection.
- **Interview note:** Strong feature engineering + temporal split + confidence interval approximation.

`models/waste_cascade.py`

- **Purpose:** Rule-based 3-tier redistribution planner.
- **Class:** `WasteCascadeOptimizer`
- **Key methods:** `load_data`, `identify_surplus`, `optimize_cascade`, `save_actions`, `get_sankey_data`.
- **Workflow role:** Converts surplus into operational actions with carbon/cost estimates.
- **Interview note:** Explainability-first heuristic; easy to operationalize in hackathon context.

`models/route_optimizer.py`

- **Purpose:** Route planning for cascade actions.
- **Class:** `RouteOptimizer`
- **Key methods:**
- `optimize_routes`
- `_solve_vrp_ortools`
- `_solve_greedy` fallback
- `save_routes`
- `get_route_map_data`
- **Workflow role:** Logistics optimization after cascade action creation.
- **Interview note:** Resilient design: OR-Tools when available, graceful fallback otherwise.

`models/carbon_calculator.py`

- **Purpose:** Deterministic carbon accounting and equivalency conversions.
- **Key functions:**
- `calculate_food_saved_carbon`
- `calculate_redistribution_carbon`
- `calculate_composting_carbon`
- `calculate_transport_emissions`
- `get_carbon_summary`, `get_equivalencies`
- **Workflow role:** KPI backbone for sustainability reporting.

Shared utility layer

`utils/__init__.py`

- Empty package marker.

`utils/helpers.py`

- **Purpose:** Shared DB data loaders + formatting + helpers.
- **Key functions:** `get_sales_dataframe` , `get_stores_dataframe` , `get_inventory_dataframe` , `haversine_distance` , formatting helpers, aggregate summaries.
- **Workflow role:** Reused by models/API/UI.

`utils/knowledge_base.py`

- **Purpose:** Builds structured knowledge context for LLM features.
- **Key parts:**
- Defensive query wrappers (`_safe_query_df` , `_safe_scalar`)
- 13+ section extractors (`get_platform_overview` , `get_monthly_trends` , etc.)
- `build_knowledge_base` (dict)
- `build_knowledge_text` (LLM-friendly long text)
- `run_custom_query` (theme-based SQL mapper)
- **Workflow role:** Central intelligence context for chatbot and agentic UI.

Active Streamlit pages (current app)

`pages/1_dashboard.py`

- **Purpose:** Main analytics dashboard page.
- **Workflow role:** BI-style operational monitoring and actions.
- **Sections (functions):**
- `page_overview`
- `page_forecast`
- `page_cascade`
- `page_routes`

- `page_carbon`
- `page_analytics`
- **Key behavior:** Calls `set_page_db("dashboard")` to use isolated DB copy.

`pages/2_chatbot.py`

- **Purpose:** Mistral chatbot with tool-calling + MCP hook.
- **Workflow role:** Conversational analytics surface.
- **Key functions:**
- `call_project_mcp` (mcporter stdio)
- `execute_tool_call`
- `get_chat_response`
- `render_chart`
- **Tools available to model:**
- `query_database`
- `get_themed_analysis`
- `query_project_mcp`

`pages/3_agentic.py`

- **Purpose:** Multi-mode AI analyst page.
- **Workflow role:** Deeper analysis and report generation UI.
- **Modes include:** executive summary, deep-dive, client report, live SQL, recommendations, Metabase actions, Word report workflow.
- **Key behavior:** env-driven Mistral + Metabase config; loads KB; uses direct Metabase REST API calls.

Legacy standalone apps (still present, mostly superseded)

`dashboard/app.py`

- Standalone single-page dashboard app (older entrypoint).
- Similar logic to `pages/1_dashboard.py`.

`chatbot/app.py`

- Standalone chatbot app (older entrypoint).
- Similar to `pages/2_chatbot.py` but without project MCP hook.

`agentic/app.py`

- Standalone agentic app (older entrypoint).
- Similar to `pages/3_agentic.py` but not the unified-pages route.

MCP / integration files

`mcp/project_context_server.py`

- **Purpose:** Project-scoped FastMCP server.
- **Server name:** `foodflow-project-context`
- **Tools:**
- `get_knowledge_base_snapshot(max_chars)`
- `query_project_context(question)`
- `run_sql(sql, limit)` (SELECT/WITH only)
- **Workflow role:** Safe/structured project-context access for LLM tool calls.

`config/mcporter.json`

- **Purpose:** Registers MCP server for mcporter.
- **Configured server:** `foodflow_project` via `uv run python ../mcp/project_context_server.py`.

Scripts / ops

`scripts/setup_uv.sh`

- **Purpose:** uv setup/bootstrap + dependency sanity check.
- **Workflow role:** one-command dev environment prep.

`scripts/run_metabase_local.sh`

- **Purpose:** Runs local Metabase in Docker with mounted data/plugins.
- **Workflow role:** BI dashboard integration path.

`scripts/run_project_mcp.sh`

- **Purpose:** Starts project MCP server with uv.
- **Workflow role:** Manual MCP server run for local testing.

---

3) Interview-critical architecture talking points

Strengths to highlight

1. **Full vertical slice**: data gen → ML → optimization → API → UX.
2. **Operational realism**: weather/events/expiry/supplier/store structure.
3. **Fallback-first design**: Prophet optional, OR-Tools optional.

4. **LLM with tools**: not just chat, but SQL + project MCP.
5. **Carbon accountability**: decision impact quantified in CO_2 and equivalencies.

Trade-offs / limitations (say honestly)

1. Data is synthetic, not production telemetry.
2. Forecast confidence intervals are heuristic, not probabilistic modeling.
3. Cascade logic is heuristic (not full min-cost flow optimization).
4. Per-page DuckDB copies reduce lock issues but can diverge state across pages.
5. Some legacy docs/comments still reference older architecture details.

---

4) Common interview questions + short answers

Q: Why DuckDB?

Because this is analytics-heavy and local-first. DuckDB gives fast OLAP queries, simple embedding, and no external DB ops overhead.

Q: Why XGBoost here?

Tabular, non-linear interactions (weather/events/store/product), fast training, strong baseline performance, and explainable feature importances.

Q: Why not deep learning/time-series transformer?

Scope and speed: hackathon timeline, need robust baseline + interpretability + lower ops complexity.

Q: How does waste cascade work?

Surplus detection (inventory vs forecast horizon), then policy-first routing: 1) retailer redistribution, 2) food bank donation, 3) compost for remainder.

Q: What is agentic in this project?

Mode-based AI workflows where LLM can generate structured analyses/reports and trigger data/BI operations, not just plain Q&A.

Q: How do you ensure safe SQL in MCP?

`run_sql` allows only queries starting with `SELECT` or `WITH` and truncates output with `limit`.

---

5) Last-minute prep checklist (for tomorrow)

1. Practice the 90-second pitch.
2. Be ready to draw architecture (data → model → optimization → UI/API).
3. Explain one concrete scenario:
4. forecast detects surplus,
5. cascade reallocates,
6. route optimizer schedules deliveries,
7. carbon module reports impact.
8. Mention one weakness + one improvement roadmap (shows maturity).
9. If asked about productionization:
10. monitoring, drift detection, model registry,
11. event-driven jobs, stronger optimization constraints,
12. multi-user concurrency strategy beyond DB-copy approach.

---

6) Quick command recap

```
# full pipeline + app launch
python run.py

# unified app only
uv run streamlit run app.py --server.port 8501

# api only
uv run uvicorn api.main:app --reload --port 8000

# seed only
uv run python data/seed_database.py

# run project MCP server
./scripts/run_project_mcp.sh

# run local metabase
./scripts/run_metabase_local.sh
```

If you can explain this document clearly, you’ll sound like someone who both built and understood the system end-to-end.