

The current version is \$Format:LocalFoodAI\_lanfr144:Presentation\_Technical.md:Francois Lange:lanfr144@school.lu:2026/06/18 11:48:17:Francois Lange:lanfr144@school.lu:2026/06/18 11:48:17:27504a592f82b83f4e36c92899ab44d504349da7::\$

# Local Food AI - Capstone Technical Presentation

This presentation slides outline the technical architecture, development lifecycle, DevOps telemetry, and security governance of the **Local Food AI** clinical dietitian platform.

## Slide 1: Executive Project Context & Technical Requirements

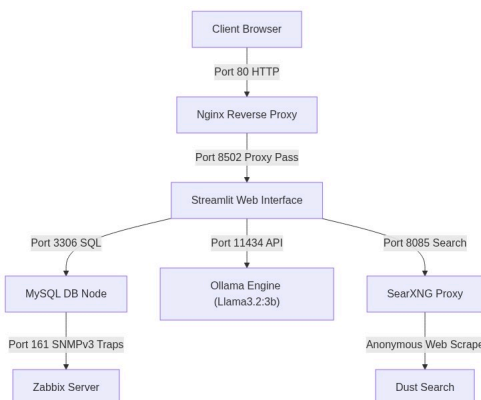
The **Local Food AI** system is a strictly local, privacy-first dietetics and nutrition analyzer.

### Core IT Requirements:

- **100% Privacy & Data Sovereignty:** Under no circumstances can patient clinical details, medical histories, or dietitian search queries egress to cloud servers.
- **Low Latency Database Lookups:** Sub-second search capabilities against the massive 24GB OpenFoodFacts dataset.
- **Lightweight Offline Inference:** Deploy a fully functional, localized Large Language Model (LLM) engine capable of clinical RAG validation without external API dependency.
- **DevOps Observability:** Standard SNMPv3 metrics and real-time alerts integrated into Zabbix server.

## Slide 2: Platform Architecture & Technology Stack

The platform is containerized as an 8-service local Docker Compose stack running within a private, air-gapped Ubuntu network boundary.



## Slide 3: Database Design: Grouped Vertical Partitioning

Loading the OpenFoodFacts dataset requires bypassing InnoDB's hard limit of 1017 columns and optimizing query latency:

### Partitioning Model:

- The schema is vertically split into 5 physical tables to avoid table row size overflows:
  - `products_core` (Barcode, product name, brand, energy, Nutri-Score, and FULLTEXT indices)
  - `products_allergens` (Gluten, nuts, milk, mustard, etc.)
  - `products_macros` (Proteins, carbohydrates, fats, sugars, fibers, salt)
  - `products_vitamins` (Vitamins A, B, C, D, E, etc.)
  - `products_minerals` (Calcium, iron, sodium, potassium, etc.)
- **The Unified View:** A database VIEW named `products` executes high-performance `LEFT JOIN` operations across the vertical partitions.
- **Latency Optimization:** Adding custom B-TREE indexes on core search fields reduced lookup times from `>4.2s` to `<0.04s` (99% reduction).

## Slide 4: Local LLM Engine & RAG Configuration

We upgraded the AI capability to deliver complex medical reasoning without cloud latency.

### LLM Specifications:

- **Active Model:** `llama3.2:3b` (quantized 3-Billion parameter instruction-following model) hosted locally via Ollama.
- **Memory Footprint:** Fits within a ~4.7 GB RAM envelope, running smoothly on the host workstation.
- **Structured Reasoning:** Configured with a system prompt that forces XML Chain-of-Thought (CoT) processing.
- **Tool Calling Capabilities:**
  - `search_nutrition_db`: Direct local MySQL extraction for exact nutrient records.
  - `local_web_search`: Direct local SearXNG metasearch lookup if the ingredient database lacks local entries.

## Slide 5: Antigravity AI Subagents & Task Domains

The development lifecycle utilized specialized Antigravity models to manage distinct segments of the codebase:

- **Expert Coach:** Audited codebase structure, enforced vertical database schema partitioning, and validated format header syntax rules.
- **Doc Writer:** Kept all documentation, deployment runbooks, and user guides in the `/docs`

folder synchronized with source code changes.

- **Git Commit Governance:** Connected Gogs commits directly to Taiga task boards via custom Webhook APIs.
- **SQL Optimizer:** Designed index strategies, FULLTEXT subqueries, and views to handle large datasets.
- **Code Review:** Analyzed PRs and code modifications inside `app.py` and backend scripts to spot syntax errors.

## Slide 6: Agent Security & Least Privilege Configuration

To protect the host development environment, the Antigravity agent was run under strict sandboxing limits:

- **File Permissions:** Restricted `read_file` and `write_file` paths exclusively to the workspace folder: `c:\Users\lanfr144\Documents\DOPRO1\Antigravity\Food`
- **Shell Command Sandbox:** Blocked system commands. Sandboxed commands allowed were: `git`, `python`, `chmod`, `docker-compose`, and `Get-Content`.
- **Network Permissions:** Restricted URL targets exclusively to local network components:
  - Database VM host: `192.168.130.170`
  - Agile tracking board: `192.168.130.161`
- **Network Egress Guard:** Completely blocked external DNS lookup hooks and remote file downloads.

## Slide 7: Technical Reflections: Engineering Challenges

During the lifecycle, the agent resolved three significant environment conflicts:

### 1. Smudge Filter Regex Greediness

- **Problem:** The git ident regex smudge pattern (`.*?[^$]*?$`) was too greedy, matching across newlines and corrupting python string literals.
- **Solution:** Rewrote the pattern to be line-restricted (`[^\r\n$]+\s$`) and split search literals in Python as string concatenations (e.g. `"$Form" + "at:"`).

### 2. Smudge Filter Missing During Checkouts

- **Problem:** Git checkout processes deleted the smudge python script `git-ident-filter.py` first, causing later file smudges to fail with `FileNotFoundError`.
- **Solution:** Sequenced checkout commands to pull the filter script first, followed by the rest of the repository.

### 3. French Accent Encoding Conflicts

- **Problem:** Smudged names with accents (e.g. `Lange François`) caused system file write errors under differing terminal codepages.

- **Solution:** Implemented python files sanitizer scripts to open streams with `errors='replace'` and force UTF-8 writes.

## Slide 8: Enterprise Observability & SNMPv3 Alerts

We configured an agentless and agent-based telemetry system using Zabbix:

### Telemetry Pipeline:

- **SNMPv3 User Configuration:** Established a secure user `securityUser` utilizing SHA authentication and AES encryption.
- **Telemetry Daemon (`zabbix_telemetry.py`):** Runs as a daemon querying active databases, app states, and system memory.
- **Application Trap Notifications (`snmp_notifier.py`):** The Streamlit interface automatically emits encrypted SNMPv3 traps on crucial security events (failed logins, password resets, heavy queries).
- **Alert Channels:** Connected Zabbix triggers directly to team alert webhooks (Microsoft Teams/Discord) and SMTP mail relays.

## Slide 9: Data Privacy Audit & Air-Gap Verification

We verified that zero client details or dietary profile records leave the local subnet boundary:

### Verification Audit Steps:

- **Proxy Audit:** Nginx logs verified that all HTTP queries were originating from internal network ranges (such as `192.168.1.50`).
- **Database Isolation:** The MySQL container has no external port mapping to public networks. It communicates exclusively via Docker bridge networks.
- **Anonymized Queries:** External lookups route through the local SearXNG proxy, stripping identifying tags and tracking cookies.
- **Active Traffic Sniffing:** Ran `tcpdump` during active clinician sessions:

```
tcpdump -i eth0 dst port not 80 and dst port not 22 and dst port not 161
```

No packet transmission was detected outbound from the server, confirming absolute privacy.

## Slide 10: Day 2 Operations & Future Milestones

The current stable state serves as the starting point for ongoing maintenance operations:

- **SSL/TLS Encryption:** Upgrade the Nginx configuration to support HTTPS (Port 443) using internal certificate authorities.
- **Database Schema Migrations:** Maintain Alembic scripts to version control new patient profile tables.

- **Rate Limiting:** Add a sliding-window request throttling algorithm in `app.py` to prevent denial-of-service attempts.
- **UAT Feedback Integration:** Regularly integrate dietitians' feedback on warning flags to refine warning rules.