NutriCoPilot – Technical Architecture & Implementation Documentation

AI Automation Engineering Team

January 5, 2026

Contents

1 Project Overview

NutriCoPilot is an AI-native web application designed to bridge the cognitive gap between complex food regulation labels and human understanding. Unlike traditional OCR tools that only extract text, NutriCoPilot functions as a **Reasoning Engine** that:

- Infers user intent,
- Analyzes ingredient synergy,
- Highlights health trade-offs in real time.

Core Value Proposition: Transitioning from *Data Retrieval* to *Decision Support*.

2 Technical Architecture & Stack

2.1 Frontend Stack

- Framework: React 19 (ES Modules)
- Language: TypeScript (Strict AI schema typing)
- Styling: Tailwind CSS
- Visualization: Recharts
- Build System: No-bundler setup via esm.sh

2.2 AI Engine

- Provider: Google Gemini API
- Model: `gemini-flash-latest`
- Mode: Multimodal (Text + Image)
- Output: Schema-enforced JSON

3 AI Implementation Strategy (geminiService.ts)

3.1 Strict Schema Validation

We define a strict `responseSchema` ensuring predictable structured JSON.

- **intent**: User persona inference.
- **verdict**: Enum (Excellent, Good, Avoid).
- **keyInsights**: Positive / Negative / Neutral indicators.

3.2 Prompt Engineering

The system instruction defines the AI persona:

> "You are a Consumer Health Co-Pilot. Infer intent, analyze synergy, explicitly identify trade-offs."

This prevents encyclopedic output and enforces reasoning behavior.

3.3 Multimodal Handling

The function `analyzeIngredients()` supports:

- Raw text ingredient strings.
- Base64 encoded images.

OCR and reasoning occur in a single pass.

4 Component Architecture

4.1 App.tsx

- Controls application state lifecycle.
- Manages global layout and reset flows.

4.2 InputSection.tsx

- Drag-and-drop support.
- Tab-based switching between Image/Text.
- Preloaded mock ingredient datasets.

4.3 AnalysisResultView.tsx

- CSS Bento-Grid dashboard.
- Dynamic theming based on AI verdict.
- Scrollable insight panels.

4.4 LoadingView.tsx

Implements progressive wait messages to reduce perceived latency.

5 UI/UX Design Philosophy

5.1 Glassmorphism

Uses backdrop blur and translucent panels to create HUD-style experience.

5.2 Typography

- Font: Plus Jakarta Sans

5.3 Information Hierarchy

- Health Score Gauge
- Verdict Badge
- Trade-offs and Uncertainty Panels

6 Data Flow Lifecycle

1. User uploads image or text.
2. UI encodes image to Base64.
3. Gemini API invoked with schema and persona.
4. JSON validated against TypeScript interface.
5. Dashboard renders structured output.

7 Future Roadmap

- Progressive Web App (Offline Support).
- Personalized Health Profiles.
- Scan History and Result Caching.