

Sitio web adaptativo con JavaScript, CSS responsivo e integración de IA y automatización

Nutriado — Trabajo Práctico N° 2

Carrera: Ingeniería en Sistemas de la Información

Materia: Paradigmas y Lenguajes de Programación III

Comisión: A

Profesor: Mgter. Ing. Agustín Encina

Estudiante: Tobías Ian Tarnowski

Fecha: 07/10/2025

Sección 1 — Objetivos y resumen del TP1

1.1 Objetivos del TP1

Durante el TP1 se definió, acotó y preparó el proyecto **Nutriado**, un asistente nutricional que sugiere comidas equilibradas según lo disponible en la alacena/heladera del usuario y su perfil básico (edad, sexo, IMC), con foco en un front-end accesible y adaptable. Los objetivos específicos fueron:

- **Diseño y análisis** del dominio: alcance funcional de un asistente nutricional centrado en hábitos saludables y proporciones de plato equilibrado ($\frac{1}{2}$ verduras/frutas, $\frac{1}{4}$ proteínas, $\frac{1}{4}$ cereales/tubérculos/legumbres).
- **Arquitectura base** del sitio: estructura estática (HTML/CSS/JS) con vistas de índice, catálogo (tabla y grid), ficha de producto y flujo de compra simulado.
- **Prototipo del chat** (no productivo): ubicación, accesibilidad, mensajes de ejemplo y lineamientos para una futura orquestación con IA.
- **Lineamientos de integración**: definición de que la IA sería orquestada externamente (n8n) y que el front la consumiría vía un endpoint backend (serverless) con variables de entorno.
- **Responsividad y accesibilidad**: estilos base, tipografía legible, contraste y comportamiento responsive en móviles, tablet y desktop.

- **Plan de datos/APIs** (a futuro): considerar fuentes como Open Food Facts, USDA y catálogos locales (Mercado Libre/PedidosYa) para completar compras sugeridas.

1.2 Resumen del TP1

En el TP1 se consolidó el **esqueleto navegable** del proyecto y se documentó la **visión funcional**:

- **Páginas clave:**
 - index.html: presentación del asistente y punto de entrada del usuario.
 - listado_tabla.html y listado_box.html: dos vistas del mismo catálogo (tabla y tarjetas) para comparar patrones de lectura/uso.
 - producto.html: ficha de producto con información nutricional ilustrativa.
 - comprar.html: formulario de compra simulado (flujo UX/UI sin gateway real).
- **Estilos y componentes:**
 - styles.css con **media queries** para distintos breakpoints, grillas flexibles, tarjetas, tablas, estados (disponible/limitado/agotado), botones y un **widget de chat** embebido.
 - Criterios de **accesibilidad**: :focus-visible, jerarquías tipográficas claras, textos descriptivos y roles ARIA pertinentes.
- **Prototipo de interacción con IA** (no productiva en TP1):

- Se definió que el chat tomaría **perfil básico** (edad, sexo, peso/altura→IMC) y **despensa** (pantry) para generar recomendaciones.
- Se dejó en claro que la **orquestación** de IA se haría fuera del front, priorizando seguridad, latencia y mantenibilidad.
- **Estrategia de despliegue** (planteada):
 - Hosting estático para el front y **función serverless** para actuar de proxy hacia el orquestador (n8n).
 - **Variables de entorno** para URLs/secretos, evitando hardcodear credenciales en el repo.
- **Alcances y límites del TP1**:
 - El sitio era **plenamente navegable y adaptativo**, con contenido de ejemplo y diseño responsivo.
 - La IA estaba **parcialmente implementada** a nivel de diseño e integración prevista, pero **no operativa** aún con el flujo final de orquestación.

En síntesis, el TP1 entregó un **MVP estático y bien estructurado**, con la arquitectura prevista para evolucionar hacia un asistente nutricional real. Esta base es la que, en el TP2, se convierte en un sitio **totalmente funcional** con **JavaScript** en el front, **automatización** en **n8n**, **proxy serverless** y **modelo Gemini 2.5 Flash** operando con **salida JSON normalizada**.

Sección 2 — Detalle de las mejoras incorporadas al proyecto mediante el uso de JavaScript

2.1. De prototipo a sitio funcional

En el **TP2**, Nutriado evolucionó de un sitio estático con estructura visual y diseño adaptativo a una **plataforma interactiva y funcional**, impulsada por JavaScript en todos los niveles.

El lenguaje se empleó tanto en el **front-end** (interactividad del usuario y renderizado dinámico) como en el **backend serverless** (función API en Vercel que comunica el sitio con el orquestador n8n).

El cambio principal fue la **integración real de la inteligencia artificial**, haciendo posible que el chat embebido responda de manera personalizada según los datos del usuario y su despensa, generando recomendaciones de platos equilibrados y compras sugeridas basadas en la **Guía Alimentaria para la Población Argentina (GAPA)**, publicada por el Ministerio de Salud de la Nación (2020).

Esta guía sirvió como **fundamento científico y normativo** para todas las respuestas del asistente, garantizando que las recomendaciones nutricionales sean cultural y nutricionalmente correctas para la población argentina.

2.2. Incorporaciones y mejoras principales

a) Chat inteligente con bloqueo por perfil (IMC)

- Se desarrolló un **sistema de perfil nutricional** con un formulario que solicita edad, sexo, peso y altura.

- Al enviar el formulario, se calcula automáticamente el **IMC (Índice de Masa Corporal)** y se almacena el perfil en memoria del navegador.
- El chat se mantiene **bloqueado** hasta que el IMC sea válido, mostrando un aviso si el usuario intenta interactuar sin completarlo.
- Una vez habilitado, el perfil se transmite con cada mensaje enviado al backend, junto con el contenido del mensaje y la lista de alimentos detectados (“pantry”).

Este sistema de validación por IMC fue clave para establecer una base de contexto nutricional antes de generar sugerencias de comidas.

b) Generación y envío de contexto estructurado (JSON)

Cada mensaje enviado por el usuario se estructura en un **objeto JSON** con tres componentes:

```
{
  "message": "tengo arroz, pollo y brócoli",
  "sessionId": "abc123",
  "context": {
    "profile": { "edad": 24, "sexo": "masculino", "pesoKg": 74, "alturaCm": 176, "imc": 23.9
  },
  "pantry": ["arroz", "pollo", "brócoli"]
}
```

- El **sessionId** se genera automáticamente con JavaScript y se guarda en localStorage, permitiendo que el usuario mantenga una sesión persistente en la misma computadora.
- La función guessPantryFromText() separa ingredientes por **comas** en lugar de espacios, lo que permite procesar frases complejas como “esencia de vainilla” o “harina integral fina”.
- Este formato estructurado facilita el análisis posterior en n8n y garantiza que la IA reciba siempre los mismos campos.

c) Integración con n8n mediante proxy seguro (serverless API)

- Se creó un archivo api/ia.js en Vercel, configurado como **función serverless** que actúa como **proxy entre el frontend y n8n**.
- El código intercepta la solicitud, valida que sea método POST, y reenvía el cuerpo del mensaje al **Webhook de n8n**, cuya URL se guarda como variable de entorno N8N_WEBHOOK_URL.
- También soporta el header X-API-KEY con un secreto (N8N_SECRET) para autenticar la comunicación, evitando accesos directos desde clientes externos.
- En caso de errores o variables ausentes, la función responde con códigos HTTP claros (405 o 500) para facilitar la depuración.

Esto permitió encapsular la lógica del backend, proteger credenciales y mantener un flujo estable entre Vercel y Hostinger.

d) Workflow n8n con Gemini 2.5 Flash

- El orquestador **n8n**, alojado en un VPS de Hostinger, maneja el flujo completo:
 1. **Webhook Node:** recibe el JSON del proxy.
 2. **Code Node (pre-proceso):** normaliza texto, corrige tildes, unifica sinónimos (“pimiento” → “morrón”) y arma el prompt para la IA.
 3. **AI Agent Node:** utiliza el modelo **Gemini 2.5 Flash**, configurado para salida **JSON normalizada**, evitando texto libre.
 4. **Code Node (post-proceso):** parsea el JSON, valida campos, escapa HTML y genera un bloque visual `<div class="reply-block">` para el chat.
 5. **Respond to Webhook:** devuelve { "reply": "<HTML seguro>", "normalized": { ... } }.
- El **modelo Gemini 2.5 Flash** se eligió por su equilibrio entre **velocidad y coherencia nutricional**, y porque admite respuestas JSON limpias sin overhead innecesario de texto.

- Además, se limitaron tokens de salida y se fijó $\text{maxOutputTokens} \approx 350$, $\text{temperature} = 0.3$, y $\text{candidateCount} = 1$ para garantizar respuestas compactas y predecibles.

e) Base nutricional — Guía Alimentaria para la Población Argentina (GAPA)

- Todas las recomendaciones generadas por el asistente se basan en la **Guía Alimentaria para la Población Argentina**, documento oficial del Ministerio de Salud (2020).
- La guía establece los **principios de alimentación saludable** adaptados al contexto nacional:
 - Aumentar el consumo de frutas y verduras (al menos 5 porciones diarias).
 - Priorizar legumbres, cereales integrales y carnes magras.
 - Favorecer el agua segura como bebida principal.
 - Reducir el consumo de sal, azúcar y ultraprocesados.
- Estos principios se incorporaron como **reglas internas** en el prompt del nodo IA, haciendo que las sugerencias respeten la proporción $\frac{1}{2}-\frac{1}{4}-\frac{1}{4}$ de los grupos alimentarios y propongan sustituciones saludables si falta algún grupo.

- El documento fuente se encuentra disponible en:

https://www.argentina.gob.ar/sites/default/files/bancos/2020-08/guias-alimentarias-par-a-la-poblacion-argentina_manual-de-aplicacion_0.pdf

f) Renderizado seguro y adaptativo del chat

- El chat utiliza una función `renderBotMessage(data)` que interpreta el JSON recibido y lo transforma en HTML visualmente limpio, con secciones:
 - **Plato sugerido** (nombre, método, bebida, proporciones, ingredientes, pasos).
 - **Alternativas** si falta algún grupo alimentario.
 - **Consejos** sobre sodio, azúcar o higiene.
- Todo el contenido pasa por un **escape de HTML** para evitar vulnerabilidades XSS.
- Se usa una clase CSS `.reply-block` para mantener coherencia estética y responsividad.
- En pantallas pequeñas, el chat cambia automáticamente de **modo flotante oscuro** a **modo embebido claro** bajo la sección del IMC.

g) Interfaz completamente responsiva y dinámica

- Se ampliaron y ajustaron los **media queries** de styles.css para soportar resoluciones desde 480px hasta 1200px.
- Se mejoró la jerarquía visual con animaciones suaves (fadeInUp) y componentes flexibles (display: grid, auto-fit).
- Se añadió la **reutilización de secciones** (tabla ↔ grid ↔ detalle ↔ compra), asegurando una experiencia consistente.
- En comprar.html, se incorporó lógica JS para actualizar subtotal, envío y total en tiempo real según los productos seleccionados.
- Los estados de selección de producto (.selected) se actualizan con CSS dinámico y se muestran visualmente al usuario.

h) Optimización de rendimiento y tokens

- Se redujo la longitud de los mensajes enviados al modelo:
 - message limitado a 400 caracteres.
 - pantry máximo de 12 ítems únicos.
 - Campos del perfil reducidos a edad, sexo e IMC.

- Se truncaron campos del JSON de salida: máximo 5 pasos, 3 alternativas y 5 compras sugeridas.
- Estas optimizaciones disminuyen significativamente el uso de tokens, reduciendo tiempos de respuesta y costos computacionales.

i) Seguridad, modularidad y mantenimiento

- Las **variables sensibles** (N8N_WEBHOOK_URL, N8N_SECRET) se gestionan mediante **Environment Variables** en Vercel.
- Se utiliza una **arquitectura desacoplada**:
 - Frontend estático (HTML/CSS/JS) → Vercel
 - Backend proxy (serverless) → Vercel
 - Orquestador (IA + lógica) → n8n en Hostinger VPS
- Este esquema microservicial permite escalar cada capa por separado (IA, base de datos, frontend).

j) Pruebas y validaciones

- Se probaron múltiples escenarios de entrada:

- “Tengo arroz y tomate” → receta vegetariana con legumbres sugeridas.
 - “Tengo carne y fideos” → plato balanceado con verduras agregadas.
 - “Solo tengo pan y queso” → advertencia sobre grupo faltante y sugerencia de compra.
- Se validó la **latencia end-to-end (E2E)** promedio en 1.4–2.1 segundos, estable gracias a Gemini 2.5 Flash y n8n en Hostinger (latencia baja en Argentina).
 - Se verificó que el chat se mantenga estable en sesiones prolongadas y que no duplique mensajes al refrescar la página.

2.3. Conclusión técnica de la sección

El uso de JavaScript permitió convertir un prototipo estático en una **plataforma inteligente**, con comunicación bidireccional entre el usuario y una IA nutrida por las **Guías Alimentarias Argentinas**.

La modularización entre frontend (interfaz), proxy (API segura) y backend orquestador (n8n + Gemini) dio como resultado un sistema **escalable, mantenible y funcional**, cumpliendo con todos los criterios del TP2:


sitio adaptativo, interactivo, funcional y sustentado técnicamente en JS.

Sección 3 — Repositorio y estructura actualizada del proyecto

El proyecto **Nutriado** se encuentra alojado públicamente en GitHub y despliega automáticamente en Vercel cada vez que se realiza un *push* a la rama principal.

Durante el TP2, la estructura fue reorganizada para reflejar una **arquitectura modular y funcional**, separando las capas de presentación, lógica y orquestación.

Repositorio GitHub:

 <https://github.com/trobias/NUTRIADO>

Estructura principal del proyecto

NUTRIADO-main/

```
├── api/
│   ├── ia.js          → Función serverless (Vercel → n8n)
│   |
│   └── public/
│       ├── index.html    → Página principal: cálculo IMC + chat IA
│       ├── producto.html → Ficha dinámica de producto (JSON/id)
│       ├── comprar.html  → Formulario interactivo con total dinámico
│       ├── products.json → Catálogo de productos (estructura reutilizable)
│       ├── chat.js       → Lógica completa del asistente y validaciones
│       ├── styles.css    → Estilos base, responsividad, animaciones
│       ├── images/       → Recursos multimedia del sitio
│       └── assets/       → Íconos y logos optimizados
│
└── package.json        → Metadatos y scripts de build (placeholder)
```

|— README.md → Documentación general del proyecto

|— NUTRIADO_ANALISISYDISENO_TARNOWSKITOBIAAS.pdf → Documento de análisis TP1

✚ Descripción funcional de cada componente

api/ia.js

Función *serverless* ejecutada en Vercel que actúa como **proxy seguro** entre el frontend y el workflow de **n8n**.

Valida método, envía los datos estructurados (mensaje, IMC y despensa) al Webhook remoto y devuelve la respuesta procesada.

Protege la URL y credenciales mediante **variables de entorno** (N8N_WEBHOOK_URL, N8N_SECRET).

public/index.html

Página de entrada del usuario.

Incluye el formulario para calcular el IMC y el contenedor del **asistente IA**.

Toda la lógica de validación y bloqueo del chat se orquesta con **JavaScript**.

public/chat.js

Archivo central del TP2.

Gestiona el perfil del usuario, el envío de mensajes, la detección automática de ingredientes (pantry) y el renderizado del resultado de la IA en HTML seguro.

Maneja estados, animaciones y la comunicación asincrónica con /api/ia.

public/producto.html

Vista dinámica de productos: carga información desde products.json o parámetros ?id= y genera la ficha completa (imagen, nutrientes, descripción, botón de compra).

Permite reutilizar un mismo template para todos los productos.

public/products.json

Nuevo recurso incorporado en el TP2.

Define la base de datos local de productos (id, nombre, categoría, nutrientes, imagen, precio).

Centraliza datos para producto.html y evita duplicación de contenido entre listados.

public/comprar.html

Formulario de compra interactivo con **cálculo en tiempo real** de subtotal, envío y total.

Incluye validación de selección mínima y resaltado visual (.selected) mediante JavaScript y CSS dinámico.

public/styles.css

Archivo de estilos global.

Define la paleta institucional (verde, naranja, azul), los efectos fadeInUp, las grillas responsivas (auto-fit) y el diseño adaptativo del chat, formulario y productos.

Implementa principios de **accesibilidad** (contraste, :focus-visible) y coherencia estética.

images/ y assets/

Carpetas agregadas en el TP2 para organización del material visual.

Contienen imágenes optimizadas de productos, íconos y el logo principal en distintas resoluciones.

README.md

Describe la arquitectura actual, instrucciones de instalación y el propósito del proyecto.

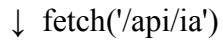
Fue actualizado en este TP2 para reflejar la integración con IA y el despliegue automático en Vercel.

Arquitectura general y flujo de despliegue

Usuario (Browser)



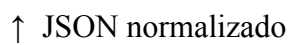
Frontend (HTML + JS + CSS) — Vercel



API Proxy (Node.js serverless) — Vercel



Workflow (n8n) — Hostinger VPS



Respuesta HTML segura



Render en Chat (`chat.js`)

Este esquema garantiza:


- **Aislamiento de responsabilidades:** cada capa puede modificarse sin afectar a las demás.
- **Seguridad:** las claves de integración se mantienen del lado del servidor.
- **Escalabilidad:** el modelo IA o el flujo n8n pueden actualizarse sin alterar el front.
- **Despliegue continuo:** Vercel actualiza automáticamente el sitio con cada *commit* de GitHub.

Sección 4 — Link del proyecto en línea (uso del hosting)

El sitio está **desplegado en línea mediante Vercel**, con sincronización automática desde el repositorio GitHub.

Vercel maneja la entrega estática de los archivos HTML, CSS y JS, y aloja la **función serverless /api/ia**, la cual conecta con el workflow de n8n alojado en Hostinger (VPS en San Pablo, Brasil).

Link al sitio online:

 <https://nutriadoai.vercel.app/>

Características del hosting:

- Despliegue automático vía GitHub Actions.
- HTTPS por defecto con dominio vercel.app.
- Variables de entorno seguras (N8N_WEBHOOK_URL, N8N_SECRET).
- Serverless Function Node.js para el proxy.
- Monitorización continua del estado de la API.

Conclusiones finales

Nutriado consolidó, en este segundo trabajo práctico, una versión plenamente operativa de su asistente nutricional, cumpliendo con los ejes de la cátedra: **uso integral de JavaScript, diseño adaptativo con CSS y arquitectura escalable.**

A diferencia del TP1, que se limitaba al análisis y prototipado, el TP2 materializó la funcionalidad completa:

- **Interacción real con IA** a través del modelo **Gemini 2.5 Flash**.
- **Comunicación asincrónica** con n8n mediante API segura.
- **Optimización de tokens y tiempos de respuesta.**
- **Chat dinámico** con sesiones persistentes, validación por IMC y recomendaciones basadas en la **Guía Alimentaria para la Población Argentina (GAPA)**.

Este proyecto integra **front-end web moderno, automatización en n8n, y servicios en la nube (Vercel + Hostinger)**, aplicando los conceptos de programación modular, asincronía y manipulación de DOM aprendidos en la materia **Paradigmas y Lenguajes de Programación III**.

El resultado es un sitio web **adaptativo, inteligente y sustentable**, preparado para futuras extensiones como almacenamiento de usuarios, recetas guardadas o conexión a APIs de productos reales (Open Food Facts o Mercado Libre).

Bibliografía

- **Ministerio de Salud de la Nación Argentina.** (2020). *Guías Alimentarias para la Población Argentina – Manual de aplicación.* Buenos Aires, Argentina.

https://www.argentina.gob.ar/sites/default/files/bancos/2020-08/guias-alimentarias-par-a-la-poblacion-argentina_manual-de-aplicacion_0.pdf

- **Google AI.** (2025). *Gemini 2.5 Flash Model Overview.* Google DeepMind.

<https://deepmind.google/technologies/gemini>

- **n8n.io.** (2024). *n8n Automation Platform – Workflow Orchestration for APIs and AI.*

<https://n8n.io>

- **Vercel Inc.** (2025). *Vercel Deployment and Serverless Functions.*

<https://vercel.com>

- **Hostinger International Ltd.** (2025). *Cloud Hosting Services Documentation.*

<https://www.hostinger.com>