LeoCozinaSys - Documentación Técnica y Funcional

Versión: 1.0.1

Autor: Leo Zelvys (Chef & IT Manager)

Fecha: 14 de Diciembre de 2025

1. Visión y Alcance

1.1. Resumen Ejecutivo

LeoCozinaSys es una plataforma de gestión gastronómica diseñada para cerrar la brecha entre la **verdad operativa** (costos, mermas, logística interna) y la **propuesta comercial** (experiencia del cliente, precio final). El sistema prioriza la agilidad de carga (UX Keyboard-First) y utiliza inteligencia de datos para automatizar el enriquecimiento de fichas técnicas.

1.2. Problema a Resolver

Los sistemas actuales o son puramente contables (complejos, lentos) o puramente visuales (menús digitales sin costeo). No existe una herramienta que permita diseñar un plato técnicamente y generar *automáticamente* una propuesta comercial limpia, sin exponer la estructura de costos interna, y que maneje la volatilidad de precios argentina.

1.3. Alcance del Proyecto (Scope)

El sistema articula la gestión operativa con la comercial a través de flujos diferenciados, generando distintos artefactos de salida según la unidad de negocio (ej: Restaurante, Catering, Eventos):

1. **Módulo de Diseño (Interno - Operativo):**
    - Gestión de Materia Prima (Precios versionados, Estacionalidad, Nutrición).
    - Ingeniería de Recetas (Costeo en tiempo real, definición de márgenes).
    - Uso exclusivo del staff/gerencia.
2. **Módulo de Propuesta (Externo - Comercial):**
    - Generación de vistas y documentos para el cliente (Menú, Propuesta de Catering, Presupuesto de Evento).
    - Ocultamiento total de datos sensibles (costos, proveedores).
    - Visualización de "Semáforo Nutricional" y datos de valor.

2. Requerimientos Funcionales

2.1. Módulo de Materia Prima (Internal)

- **RF-001 - Carga Inteligente:** El sistema debe permitir ingresar ingredientes por nombre y sugerir autocompletado de datos (precio mercado, unidad, merma) mediante scraping/IA simulada.
- **RF-001.b - Alta en Contexto (Quick Add):** Si un ingrediente ingresado en el buscador no existe, el sistema debe permitir su creación inmediata ("On-the-fly") sin obligar al usuario a abandonar el flujo de carga de la receta. Se debe capturar el nombre y asignar valores predeterminados temporalmente para no interrumpir la UX.
- **RF-002 - Versionado de Precios:** Cada actualización de precio debe generar una nueva versión del ingrediente, manteniendo el histórico para no romper recetas antiguas.
- **RF-003 - Atributos Extendidos:** Soporte para campos de sustitutos, complejidad, estacionalidad y datos normativos (Sodio, Grasas, Calorías).

2.2. Ingeniería de Recetas (Internal)

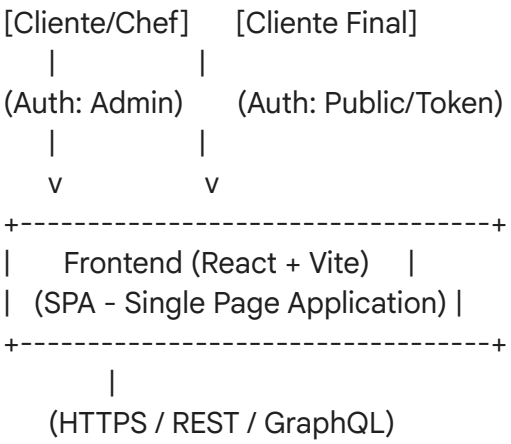
- **RF-004 - Doble UX de Armado:** Soporte para "Drag & Drop" (visual) y "Grid Mode" (carga rápida por código/teclado).
- **RF-005 - Autocompletado Semántico:** Capacidad de sugerir la estructura base de una receta a partir del nombre del plato (ej: "Ravioles" -> sugiere harina, huevo, relleno).
- **RF-006 - Pricing Dinámico:** Cálculo automático de Costo Total + Margen Configurable = Precio Sugerido.

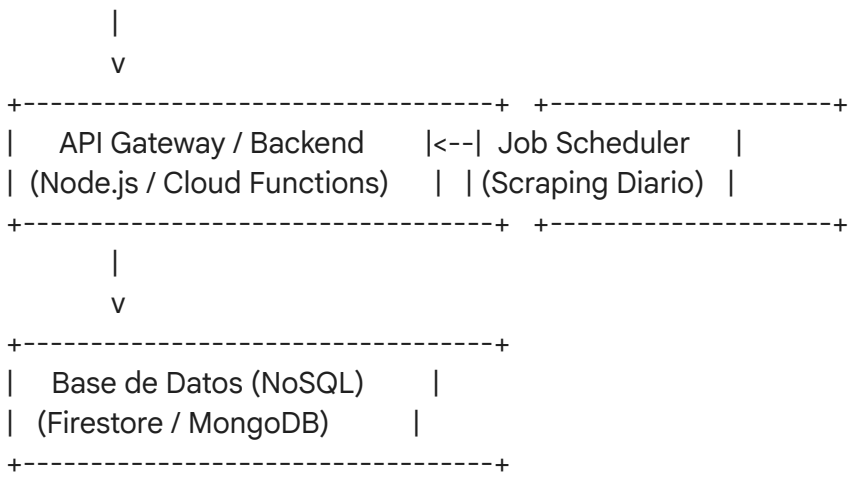
2.3. Generador de Propuestas (External)

- **RF-007 - Snapshot de Datos:** Al generar una propuesta, los precios y descripciones deben "congelarse" (snapshot) para garantizar la validez de la oferta por X días, independientemente de la inflación interna.
- **RF-008 - Sanitización de Datos:** La vista de propuesta NO DEBE enviar al frontend ningún dato relacionado con costos unitarios, márgenes o proveedores.

3. Arquitectura del Sistema

3.1. Diagrama de Alto Nivel





3.2. Estrategia de Separación (Security)

- **Colección internal_master:** Acceso estricto read/write solo para admins. Contiene costos, proveedores y lógica cruda.
- **Colección public_proposals:** Acceso read-only público (o por link único). Contiene solo el JSON sanitizado (Nombre, Descripción, Precio Final, Nutrición). **Nunca** se consulta la colección master desde la vista pública.

4. Modelo de Dominio

Entidad: Ingrediente

```
{
  "id": "uuid",
  "nombre": "string",
  "versiones_precio": [
    { "fecha": "ISO8601", "monto": 1200, "proveedor": "string" }
  ],
  "datos_normativos": { "sodio": "int", "grasas": "int", "kcal": "int" },
  "estacionalidad": ["Verano", "Primavera"],
  "sustitutos_ids": ["uuid_ref"]
}
```

Entidad: Receta (Master)

```
{
  "id": "uuid",
  "nombre": "string",
```

```
  "componentes": [
    { "ingrediente_id": "uuid", "cantidad": 1.5, "costo_al_momento": 1200 }
  ],
  "margen_objetivo": 300, // Porcentaje
  "costo_total": 4500,
  "precio_venta": 18000,
  "estado": "Borrador | Activo | Archivado"
}
```

Entidad: Propuesta (Snapshot)

```
{
  "id": "uuid_publico",
  "tipo": "CATERING | RESTAURANT | EVENTO",
  "cliente": "string",
  "fecha_vencimiento": "ISO8601",
  "items": [
    {
      "nombre": "Lomo al Malbec",
      "descripcion_comercial": "...",
      "precio_final": 18000,
      "nutri_score": { "sodio": "bajo" } // Datos calculados y "aplanados"
    }
  ]
}
```

5. Análisis Técnico

5.1. Stack Tecnológico

- **Frontend:** React 18, Tailwind CSS (estilos rápidos), Lucide React (iconografía).
- **State Management:** React Context o Zustand (para manejar el carrito de ingredientes y recetas).
- **Persistencia:** Firestore (Google Firebase). Ideal por su estructura flexible de documentos JSON para recetas complejas.
- **Inteligencia/Scraping:** Edge Functions que consultan APIs externas (ej: Mercado Central API o modelos de lenguaje ligeros) bajo demanda.

5.2. UX/UI Performance

- **Optimización de Teclado:** Implementación de tabIndex lógico y listeners de onKeyDown

(Enter para guardar) en todos los inputs de carga masiva.
- **Debounce:** En la búsqueda de ingredientes para no saturar la API de sugerencias.

6. Contrato de APIs (Simplificado)

Recursos Privados (Requiere Token Admin)

- GET /api/v1/ingredients/search?q={term}
  - Retorna sugerencias locales + resultados de scraping externo.
- POST /api/v1/recipes
  - Crea una receta. Calcula costos del lado del servidor (doble validación).
- POST /api/v1/proposals/generate
  - Input: Lista de IDs de recetas.
  - Output: Crea un documento en public_proposals con datos congelados.

Recursos Públicos

- GET /api/v1/proposals/{proposal_id}
  - Retorna el JSON sanitizado para renderizar la vista del cliente.

7. Decisiones Clave de Arquitectura (ADRs)

ADR-001: Uso de NoSQL sobre Relacional

- **Contexto:** Las recetas pueden tener estructuras variadas (ingredientes simples, sub-recetas, opcionales).
- **Decisión:** Usar base de datos orientada a documentos (Firestore).
- **Consecuencia:** Mayor flexibilidad en el esquema de datos, pero requiere cuidado en la integridad referencial (ej: si borro un ingrediente, qué pasa con la receta). Solución: Soft-delete.

ADR-002: Patrón Snapshot para Propuestas

- **Contexto:** La inflación cambia los costos semanalmente. Si actualizo el costo del "Lomo", no quiero que cambie el precio de una propuesta enviada ayer.
- **Decisión:** Al crear una "Propuesta", se copian los valores (Deep Copy) a un nuevo documento independiente.
- **Consecuencia:** Duplicidad de datos (storage), pero garantiza integridad comercial y evita conflictos con el cliente.

ADR-003: IA como Asistente, no como Dueño

- **Contexto:** La carga de datos es tediosa.
- **Decisión:** La IA sugiere/autocompleta, pero el usuario (Chef/Manager) siempre debe confirmar con Enter o Click.
- **Consecuencia:** El humano mantiene el control de calidad; la IA solo acelera la digitación.