MANUAL TÉCNICO DEL REPOSITORIO

POS DEKU RAMEN

Este documento explica **cómo funciona el repositorio completo del POS Deku Ramen**, su arquitectura, flujos, estados y cómo aplicar parches sin romper el sistema.

Está pensado para que **cualquier IA o programador humano** pueda retomar el proyecto exactamente donde se dejó.

---

1. VISIÓN GENERAL

El proyecto es un **POS web local-first** para restaurante (ramen), compuesto por:

1. **POS de Mesero** (toma de pedidos)
2. **Kitchen Display** (pantalla de cocina en tiempo real)
3. **Backend Node.js** (API + WebSocket + persistencia)

Características clave: - Funciona en **LAN sin internet** - Sin frameworks pesados - Estado en tiempo real (WebSocket) - Pensado para producción real - Extensible a analítica, impresión y sincronización futura

---

2. ESTRUCTURA DEL REPOSITORIO

```
POS-Ramen-Beta-2/
│
├── backend/
│   ├── server.js        # Backend principal (API + WS + lógica de estados)
│   └── data/            # Persistencia local (JSON / futuro SQLite)
│
├── waiter-app/          # POS de mesero
│   ├── index.html
│   ├── styles.css
│   └── app.js
│
├── kitchen-display/     # Pantalla de cocina
│   ├── index.html
│   ├── styles.css
│   └── kitchen.js
│
├── assets/
│   ├── brand/
```

```
│   │   └── logo.png     # Logo único del restaurante
│   └── menu/
│       ├── ramen_*.png
│       ├── extra_*.png
│       └── spicy_1.png / spicy_2.png / spicy_3.png
│
└── package.json
```

---

3. BACKEND (backend/server.js)

Rol

Es el **cerebro del sistema**.

- Sirve el frontend (POS y cocina)
- Expone la API REST
- Maneja WebSockets
- Guarda órdenes
- Controla estados y transiciones

Endpoints principales

```
GET  /api/menu      -> Menú completo
GET  /api/orders    -> Todas las órdenes
POST /api/orders    -> Crear nueva orden
PATCH /api/orders/:id -> Cambiar status de orden
```

Estados de una orden

```
pending     -> recién creada
preparing   -> en preparación
ready       -> lista para servir
delivered   -> entregada (histórico)
```

⚠️**Nunca se borran órdenes**, solo cambian de estado.

---

4. LÓGICA CLAVE: AUTO-MOVE A DELIVERED

Cuando una orden pasa a `ready`:

- El backend inicia un **timer de 3 minutos**

- Si en ese tiempo **no cambia de estado**, automáticamente:

```
ready → delivered
```

Reglas: - El timer vive en backend (no frontend) - Si la orden vuelve a `preparing`, el timer se cancela - Si vuelve a `ready`, se reinicia el timer

Esto evita comandas olvidadas en cocina.

---

5. WEBSOCKET (TIEMPO REAL)

El backend emite eventos:

```
order:new
order:updated
```

Tanto el POS como la cocina escuchan estos eventos para: - Actualizar UI - Evitar refresh manual

---

6. POS DE MESERO (waiter-app)

URL

```
http://localhost:3000/
```

Función

- Seleccionar productos
- Configurar tamaño, picante, extras
- Enviar pedido a cocina

Detalles clave

- UI **touch-first**
- Logo pequeño arriba a la izquierda
- Nunca muestra órdenes entregadas

---

7. KITCHEN DISPLAY (kitchen-display)

URL

```
http://localhost:3000/kitchen/
```

Función

- Ver órdenes activas
- Flujo visual rápido
- Minimizar lectura de texto

Qué se muestra

- Imagen del platillo
- Texto resumido
- Iconos de picante (1–3 chiles)
- Iconos de extras

Botones por orden

- EN PREPARACIÓN
- LISTO
- ENTREGADA (manual)

Importante

- **NO se muestran órdenes** `delivered`
- Delivered = histórico

---

8. ASSETS (IMÁGENES)

Reglas

- Todas las imágenes viven en `/assets`
- Se sirven directamente desde el backend
- No requieren abrir puertos extra

Picante

```
spicy_1.png  -> 1 chile
spicy_2.png  -> 2 chiles
spicy_3.png  -> 3 chiles
```

Las imágenes ya contienen la cantidad correcta (no se repiten dinámicamente).

---

9. PERSISTENCIA ACTUAL

Actualmente: - Persistencia simple en archivos dentro de `backend/data`

Planeado: - Migrar a **SQLite** - Tabla `orders` - Tabla `order_items` - Base para analítica semanal

---

10. ANÁLISIS Y REPORTES (FUTURO)

Diseño previsto: - Todas las órdenes delivered quedan guardadas - Consulta semanal (domingo): - Total de órdenes - Platillos más vendidos - Extras más usados - Horas pico

Nada se elimina.

---

11. LAN / OFFLINE-FIRST

El sistema: - Funciona 100% en red local - No depende de internet

Futuro: - Cola local - Sync cuando vuelva conexión

---

12. IMPRESIÓN DE COMANDAS (FUTURO)

Flujo recomendado: - Al crear orden: - Generar versión texto / ESC-POS - Enviar a impresora térmica

Compatible con: - USB - Red

---

13. CÓMO APLICAR PARCHES (REGLAS DE ORO)

1. **Nunca reescribir todo el archivo**
2. Cambiar solo lo necesario
3. Mantener endpoints
4. No romper WebSocket
5. No tocar assets sin motivo

Siempre usar prompts tipo:

> “Aplica un parche mínimo, no refactorices todo”

---

14. ESTADO ACTUAL DEL PROYECTO

✔️ POS funcional ✔️ Kitchen Display funcional ✔️ Estados claros ✔️ Auto-move a delivered ✔️ Logo integrado ✔️ UX validado en uso real

🟡 Pendiente: - SQLite - Vista de historial - Impresión - Analítica - Deploy en Contabo

---

15. FILOSOFÍA DEL PROYECTO

- Menos código, más claridad
- Todo visible
- Todo auditable
- Pensado para negocio real

Este POS **ya es producción-ready**.

FIN DEL MANUAL