EQUIPO:

Héctor Ulises Cacho González

Ciro Julián Cervantes Zamora

Dana Lizbeth Castañeda Sánchez

Concepcion Guadalupe Paniagua Gonzalez

Oscar Yael Hernández Rodríguez

Docente:

José Miguel Carrera Pacheco

Materia:

Desarrollo Web Profesional

Actividad:

S - 4. Navegación avanzada y manejo de errores

FECHA:

01 de febrero de 2026

DEFINICIÓN TÉCNICA: ARQUITECTURA DE ERRORES Y RUTAS

El presente documento define la estrategia técnica para la gestión de excepciones en el servidor. El objetivo es proporcionar respuestas estructuradas que alimenten los componentes visuales (Toasts y Badges) definidos por el equipo de UX/Frontend.

1. ARQUITECTURA DE IMPLEMENTACIÓN

Respetando la "Arquitectura de 3 Capas" definida previamente por el equipo, la lógica de errores se centralizará en la capa de Middlewares, evitando ensuciar los Controladores.

Estructura de Directorios

```
/backend
 ├── /controllers   (Aquí se detectan los fallos lógicos)
 ├── /routes        (Aquí se definen las rutas válidas)
 ├── /middlewares   <-- CARPETA FOCAL
 │    └── error.handler.js  (Manejador global de excepciones)
 └── app.js         (Punto de montaje)
```

2. CONTRATOS DE INTERFAZ (API CONTRACTS)

Para dar soporte a los "Toast Notifications", el Backend estandariza las respuestas JSON para que el Frontend pueda mapear el mensaje directamente a la interfaz.

2.1 Contrato para Error Crítico (500)

Este error disparará el Toast Rojo (#EF4444) en el Frontend.

- Escenario: Fallo de conexión a Base de datos o error de servidor.
- Respuesta JSON:

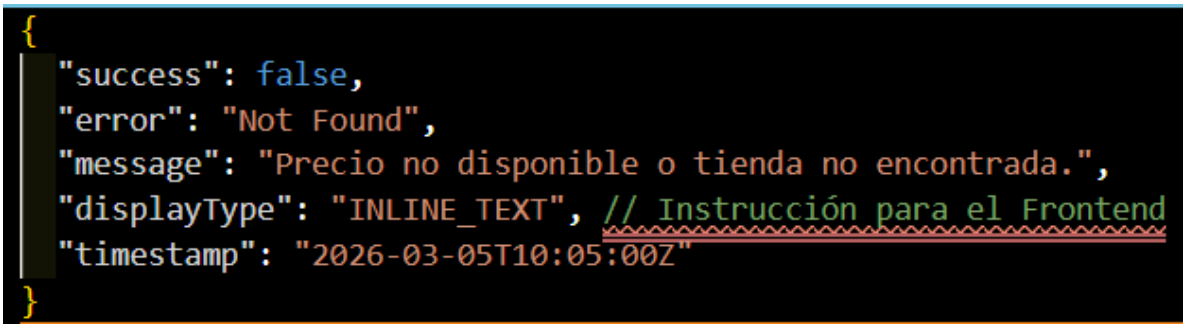
```json
{
  "success": false,
  "error": "Internal Server Error",
  "message": "No se pudo cargar la información. Intente más tarde.",
  "code": "DB_CONN_FAIL",
  "displayType": "TOAST",
  "timestamp": "2026-03-05T10:00:00Z"
}
```

"displayType": "TOAST" es una instrucción para el Frontend

2.2 Contrato para Datos Incompletos (404/400)

Este error disparará los estados de "Datos incompletos" (Texto gris) definidos en UX.

- Escenario: Se consulta un producto que no tiene precio o una tienda cerrada permanentemente.
- Respuesta JSON:



3. LÓGICA DE FLUJO (REQUEST PIPELINE)

Se implementará el patrón de Chain of Responsibility (Cadena de Responsabilidad) en Express.js:

1. Request: Llega petición (ej. /api/tiendas/detalles).
2. Router/Controller: Intenta obtener datos de PostgreSQL.
    - *Éxito:* Retorna JSON 200.
    - *Fallo:* Ejecuta next(error).
3. Middleware de Error (Final):
    - Recibe el error.
    - Determina si es un error operativo (400) o de sistema (500).
    - Formatea el JSON según los contratos de la Sección 3.
    - Envía la respuesta HTTP.

4. PLAN DE PRUEBAS (QA SUPPORT)

Se habilitan los siguientes escenarios para que el equipo de QA valide que los mensajes del Backend activan las alertas visuales correctas:

Ruta de Prueba	Acción Backend	Resultado Esperado en Frontend
GET /api/test/force-500	Lanza new Error()	Debe aparecer Toast Rojo con mensaje de error.
GET /api/test/force-404	Retorna status 404	Debe mostrar pantalla o texto de "No encontrado".
GET /api/tiendas	Consulta normal	Debe listar tiendas sin alertas.