



Unidad Académica de Economía

Carrera

Licenciatura en Sistemas Computacionales

Grupo y Semestre

Semestre 6

Nombre estudiante

Ricardo Matos Vizcarra

Actividad

Reto IA 7

Maestro

Eligardo Cruz Sánchez

Materia

Programación Distribuida del lado Cliente

UNIVERSIDAD AUTÓNOMA DE NAYARIT

Lista categorizada de hallazgos

1.- Manejo de Errores (Error Handling)

- CRÍTICO: Parseo inseguro de respuesta. En crearProducto, se ejecuta respuesta.json() antes de validar respuesta.ok. Si el servidor devuelve un error 500 con cuerpo HTML o texto plano, el código romperá por un SyntaxError.
- CRÍTICO: Silenciamiento de errores ("Swallowing"). Los bloques catch solo hacen console.error pero no relanzan el error (throw). La función devuelve undefined y quien la invoque creerá que la operación fue exitosa.
- MEJORA: Falta de Timeouts consistentes. Solo listarProductos tiene AbortSignal. Las funciones obtenerProducto y crearProducto pueden dejar la aplicación congelada indefinidamente si la red falla.

2.- Seguridad Basica

- MEJORA: Inyección en URL. Al concatenar \${id} directamente en la URL, se corre el riesgo de romper la petición si el ID contiene caracteres especiales. Falta usar encodeURIComponent(id).
- SUGERENCIA: Protocolo Hardcodeado. La BASE_URL usa http fijo. Debería venir de variables de entorno para asegurar el uso de https en producción.

3.- Mantenibilidad y Arquitectura

- MEJORA: Código Repetitivo (WET). La lógica de fetch, validación de !ok, headers y try/catch se repite en tres lugares. Esto dificulta cambios futuros (como añadir tokens de autenticación).
- SUGERENCIA: Base URL en archivo. La URL está definida en el mismo archivo. Idealmente debería inyectarse desde configuración externa.

4.- Conformidad con el protocolo HTTP

- MEJORA: Headers incompletos. Falta el header Accept: application/json. Sin esto, algunos servidores podrían devolver XML o HTML por defecto, rompiendo el parseo.
- SUGERENCIA: Manejo de estados limitado. Solo se controlan los estados 201, 400 y 404. Faltan manejadores para 401 (Auth), 403 (Forbidden) y 500 (Server Error).

Código corregido:

1. Centralización de la lógica (El patrón "Wrapper")

Qué cambió: En lugar de escribir fetch, headers, try/catch en cada una de las 3 funciones, creé una función única llamada request(endpoint, options).

Qué hace: Esta función actúa como un intermediario o "mayordomo". Todas las peticiones pasan por ella. Ella se encarga de configurar la configuración común (headers, timeouts) y de procesar la respuesta cruda.

Por qué:

- Mantenibilidad (DRY - Don't Repeat Yourself): Si mañana necesitas cambiar la URL base o agregar un token de autenticación para seguridad, solo modificas un lugar (la función request) y automáticamente se arregla en todas las funciones (listar, obtener, crear).

2. Parseo Seguro de JSON (La corrección crítica)

Qué cambió: En tu código original hacías esto: const resultado = await respuesta.json(); (Directamente).

En el nuevo código hago esto:

1. Verifico si existe el header content-type.
2. Verifico si ese header incluye "application/json".
3. Solo entonces ejecuto .json().

Qué hace: Evita que el código intente "forzar" la lectura de JSON cuando el servidor no envió JSON.

Por qué: Es la causa #1 de caídas en aplicaciones frontend. Si el servidor falla (Error 500) a menudo devuelve una página HTML de error ("Nginx Error..."). Si tu código intenta leer eso como JSON, JavaScript lanza un SyntaxError y tu aplicación se rompe antes de que puedas manejar el error.

3. Propagación de Errores (Evitar el "Silencio")

Qué cambió: En los bloques catch, cambié esto:

- Antes: console.error("Error"); (Y la función terminaba devolviendo undefined).
- Ahora: console.error(...); throw error;

Qué hace: "Relanza" el error hacia arriba.

Por qué: Imagina que listarProductos es un cocinero y tu interfaz de usuario (UI) es el mesero.

- En tu código original, si al cocinero se le quemaba la comida (fetch falla), simplemente lo anotaba en su libreta (console.error) pero no le decía nada al mesero. El mesero iba a la mesa con un plato vacío (undefined).
- Con el cambio, el cocinero grita "¡Fuego!" (throw error), y el mesero sabe que no debe servir el plato, sino disculparse con el cliente (mostrar un mensaje de error en pantalla).

4. Sanitización de Entradas

Qué cambió: En obtenerProducto(id), cambió \${id} por \${encodeURIComponent(id)}.

Qué hace: Codifica caracteres especiales. Si el ID fuera computadora/negra, la URL original se rompería (.../productos/computadora/negra parecería una ruta distinta). La nueva versión lo convierte a .../productos/computadora%2Fnegra.

Por qué: Es una medida básica de seguridad y estabilidad. Previene inyecciones involuntarias en la URL y asegura que el servidor reciba exactamente el ID que pretendías enviar, sin importar qué caracteres contenga.

5. Configuración de Headers Explícita

Qué cambió: Agregué el header 'Accept': 'application/json' en todas las peticiones.

Qué hace: Le dice explícitamente al servidor: "Por favor, respóndeme solo con datos JSON".

Por qué: Es un "contrato" de buena educación HTTP. Algunos servidores antiguos o mal configurados pueden devolver XML por defecto si no se lo especificas, lo cual rompería tu aplicación.