El catálogo v.2

La versión original de vuestro catálogo funcionaba perfectamente, pero lo hacía de forma síncrona; es decir, la validación del ID único y la adición del producto eran instantáneas, como si todo estuviera guardado en la memoria de vuestro navegador. Esto, en el desarrollo profesional, **no es real**.

Ahora vais a introducir la Asincronía, y he aquí por qué esta modificación es vital para vosotros:

1. Justifica la Asincronía: La Realidad de la Red

Cuando trabajáis con bases de datos o servicios web (APIs), la validación de un dato —como verificar si un ID de producto ya existe— no es instantánea. Tenéis que "preguntar" al servidor y "esperar" su respuesta. Esta espera es el corazón de la asincronía. Al crear un módulo simulador de API que usa Promesas y `setTimeout`, estáis aprendiendo a:

- Identificar la necesidad de usar `await` para pausar vuestro código hasta que la respuesta de red (simulada) regrese.
- Entender que el código no se puede bloquear; debe seguir siendo interactivo mientras espera la respuesta del servidor.

2. Pensad en el Usuario (Mejora de UX)

Cuando la red es lenta, los usuarios necesitan saber que algo está sucediendo. Con esta modificación, la aplicación os obliga a implementar estados de carga que son cruciales para la Experiencia de Usuario (UX):

- Cuando pulsáis "Guardar", debéis deshabilitar el botón y mostrar un mensaje de "Guardando..." para evitar clics duplicados y dar *feedback*.
- Cuando iniciáis el proceso de "Eliminar", debéis cambiar el estilo de la tarjeta (hacerla opaca) para indicar que el proceso está pendiente.
- Vosotros aprendéis a controlar este flujo usando los bloques `.then()`, `.catch()` y `.finally()` para saber exactamente cuándo restaurar la interfaz.

3. Diferenciad el Manejo de Errores

Hasta ahora, vuestros errores eran simples: ¿es el precio un número? Con la asincronía, vuestros errores se complican:

- Debéis seguir gestionando los errores síncronos (ej. el precio no tiene el formato correcto) antes de llamar a la API.
- Pero ahora, debéis añadir un manejo de errores asíncrono (ej. el servidor reporta un ID duplicado).
- Esto os enseña a usar el `try/catch` y el `.catch()` de las Promesas para gestionar fallos que no están bajo vuestro control inmediato (errores de servidor).

Práctica: "El Catálogo Conectado" (API Simulada)

Objetivo

Modificar la práctica del "Catálogo de productos" para simular que los datos se guardan en un servidor remoto. El alumno deberá sustituir la lógica síncrona de creación y borrado por llamadas asíncronas gestionadas con Promesas.

Contexto: El Cambio de Requisitos

El cliente está contento con el catálogo, pero ahora quiere que los productos se guarden en una base de datos central. Como no tenemos un backend real, te piden crear un módulo simulador de API (`api.js`) que imite los tiempos de espera de la red y posibles errores del servidor.

Instrucciones de Adaptación

Tarea 1: Crear la "API Falsa" (El Módulo Asíncrono)

En lugar de tener un array global `productos` en tu código principal, crea un objeto o clase `API` con los siguientes métodos. Todos deben devolver una Promesa:

1. `API.guardarProducto(producto)`:
    - Debe usar `setTimeout` para simular un retraso de 2 segundos.
    - Validación Asíncrona: Si ya existe un producto con ese `id` en la "base de datos" (array interno de la API), la promesa debe hacer `reject("Error: El ID ya existe")`.
    - Si no existe, lo guarda y hace `resolve("Producto guardado")`.
2. `API.borrarProducto(id)`:
    - Simula un retraso de 1.5 segundos.
    - Si el producto no se puede borrar (simula un error aleatorio del 10%), hace `reject`. Si no, `resolve`.

Tarea 2: Adaptar el Botón "Añadir Producto"

Modifica el evento del formulario para que no cree la tarjeta inmediatamente.

- Feedback visual: Al pulsar "Guardar", deshabilita el botón y cambia el texto a "Guardando...".
- Llama a `API.guardarProducto(datos)`.

- `.then()`: Si tiene éxito, crea la tarjeta en el grid, limpia el formulario y muestra un mensaje de éxito.
- `.catch()`: Si falla (ej. ID duplicado), reactiva el botón, muestra el error en rojo junto al campo ID y no cierres el formulario.
- `.finally()`: Restaura el botón a su estado original ("Añadir producto").

Tarea 3: Borrado Asíncrono

Adapta la funcionalidad del menú contextual (botón derecho).

- Al pulsar "Eliminar", la tarjeta debe volverse semitransparente (indicando proceso).
- Llama a `API.borrarProducto(id)`.
- Usa `async / await` para gestionar esta llamada.
- Solo cuando la promesa se resuelva, elimina el nodo del DOM. Si falla, restaura la opacidad y muestra un `alert`.

Tarea 4: Carga de Imágenes (Promesas en cadena)

Mejora el campo de imagen. Antes de permitir enviar el formulario, crea una función `validarImagen(url)` que devuelva una Promesa.

- La promesa debe intentar cargar la imagen (puedes crear un objeto `Image` en memoria).
- Si la imagen carga (`onload`), resuelve.
- Si la imagen falla (`onerror`), rechaza.
- Usa esto para impedir que se guarden productos con URLs rotas.