Explicación de los ejercicios 8.5, 8.6, 8.7 y 8.8 orientados a E-commerce

Cesar Andrés Sánchez

17 de noviembre de 2025

8.5: Uso esencial y avanzado de `fetch()`

El ejercicio 8.5 presenta un catálogo de productos que se carga mediante:

- Peticiones **GET** con `fetch()` para obtener productos destacados.
- Peticiones **POST** para añadir un producto al carrito.
- Manejo de errores HTTP mediante comprobación explícita de `response.ok`.
- Renderizado dinámico de tarjetas de producto.

8.5: Flujo general del ejemplo

1. Usuario pulsa “Recargar catálogo”.
2. Se envía un **fetch GET** a la API de productos.
3. Los productos se muestran en una cuadrícula.
4. Al pulsar un producto, se consulta su detalle vía otro **GET**.
5. Se permite añadir al carrito vía **POST JSON**.

Este ejercicio enseña buenas prácticas en la carga de datos asíncronos.

8.5: Ejemplo de petición GET

```
const r = await fetch(API_BASE + "/productos?destacado=true", {
    headers: { "Accept": "application/json" }
});
if (!r.ok) throw new Error("Error HTTP " + r.status);
const productos = await r.json();
```

8.6: Diseño UX para Carrito Asíncrono

Este ejercicio implementa:

- **Spinner** y estados de carga.
- **Bloqueo de botones** mientras se procesa una acción.
- **Optimistic UI**: actualizar la vista antes de que el servidor responda.
- Reversión del estado cuando ocurre un error.
- Mensajes accesibles mediante `aria-live`.

8.6: Objetivo principal

El propósito es mejorar:

- La experiencia del usuario (**UX**).
- La accesibilidad web.
- La interacción suave con un carrito de compras.

Este patrón es estándar en plataformas modernas como Amazon o Zalando.

8.6: Ejemplo de Optimistic UI

```
const copiaAntes = { ...linea };
linea.cantidad++;
renderCarrito();   // Vista optimista

try {
    await apiActualizarLinea(linea);
} catch (err) {
    linea.cantidad = copiaAntes.cantidad; // Reversión
    renderCarrito();
}
```

8.7: Seguridad en Opiniones de Producto

Este ejercicio cubre tres áreas esenciales:

1. **CORS**: Permitir peticiones desde dominios autorizados.
2. **CSRF**: Envío de token CSRF mediante cabecera personalizada.
3. **XSS**: Inserción segura mediante `textContent`.

8.7: Riesgos abordados

- Evita que otros sitios hagan peticiones maliciosas.
- Evita que un usuario envíe opiniones en nombre de otro.
- Impide la inyección de código JavaScript en opiniones.

Mediante estas técnicas, el formulario de opiniones es seguro y robusto.

8.7: Inserción segura en DOM

```
div.textContent = op.texto; // evita XSS
```

8.7: Cabeceras seguras

```
headers: {
  "Content-Type": "application/json",
  "X-CSRF-Token": CSRF_TOKEN,
  "Authorization": AUTH_TOKEN
}
```

8.8: Optimización del rendimiento

Este ejercicio implementa:

- **Debounce** en la barra de búsqueda.
- **Infinite Scroll** con IntersectionObserver.
- **Caché de resultados** para evitar peticiones repetidas.

8.8: ¿Qué problema resuelve?

Los catálogos grandes requieren optimización:

- Evitar saturar la API con muchas búsquedas.
- Cargar solo las páginas necesarias de productos.
- Mejorar tiempos de respuesta usando caché local.

8.8: Debounce en búsqueda

```
const onBuscar = debounce(() => {
    terminoActual = inputBusqueda.value.trim();
    limpiarLista();
    cargarPagina();
}, 400);
```

8.8: Infinite Scroll

```
const observer = new IntersectionObserver(entries => {
  if (entries[0].isIntersecting) cargarPagina();
});
observer.observe(sentinela);
```

Entregable

- Integra 3 de los 4 ejercicios en la práctica.