

«Talento Tech»

Front-End JS

Clase 15



Clase 15: API y Procesamiento de Datos en E-commerce

1. Promesas:

- a. ¿Qué es una promesa en JavaScript?
- b. Estados de una promesa: pending, rejected, resolved

2. Manejo de Datos con Fetch y Promesas

- a. Explicación de encadenamiento `.then`

3. Manejo de Errores con Fetch

- a. Introducir la validación de status 400/500 con un solo ejemplo sencillo, no varios casos.

4. Async/Await: Otra Forma de Esperar

- a. Explicar la sintaxis y usar un **único ejemplo** de async/await sobre fetch ya visto.

5. Buenas Prácticas de Accesibilidad y SEO

- a. 4.1. Accesibilidad: etiquetas alt, navegación por teclado, estructura semántica
- b. 4.2. SEO: uso de metaetiquetas, encabezados lógicos y contenido optimizado

6. Guía para el Proyecto Final de E-commerce

- a. 5.1. Estructura y requisitos del proyecto
- b. 5.2. Puntos clave para revisión y entrega

Objetivo de la clase:

En esta clase vas a profundizar el manejo de asincronía en JavaScript a través del uso de promesas, entendiendo sus estados y cómo trabajar con ellas usando `then` y `catch`. También vas a aprender la sintaxis de `async/await` como alternativa para escribir código asíncrono más legible. Además, incorporarás prácticas de accesibilidad y SEO para optimizar la experiencia del usuario y mejorar la visibilidad de tu proyecto en buscadores. Finalmente, te orientarás sobre los requisitos y puntos clave para avanzar con la entrega del proyecto final de e-commerce.

1. Promesas: ¿Qué es una promesa en JavaScript?

Cuando usás `fetch`, te devuelve una promesa: funciona como lo indica la palabra, es un compromiso del programa que tiene tres estados factibles. Pensalo como cuando quedás con tu amiga en cenar el sábado próximo, pero tenés que confirmar si podés más cerca de la fecha. En nuestra programación, esta promesa, como decíamos, puede tener tres estados: **pendiente**, **resuelta** o **rechazada**.

Estados de una promesa:

- **Pendiente (pending)**: La promesa todavía está esperando una respuesta (no sabés si vas a ir a cenar o no).
- **Resuelta (resolved)**: Todo salió bien (confirmaste que vas a cenar).
- **Rechazada (rejected)**: Algo salió mal (tu amiga canceló la cena).

2. Manejo de datos con Fetch y promesas

Imaginate que tu amiga te pasó la lista de compras para la cena, pero vos todavía no la miraste. Eso es lo que pasa cuando usás `fetch`: le pedís datos a una API y la promesa te los devuelve “a futuro”.

Ahora bien, cuando llega esa lista (los datos de la API), tenés que procesarla para ver qué cosas vas a comprar y cómo las vas a acomodar. En nuestro código, hacemos lo mismo:

```
fetch("https://fakestoreapi.com/products")
  .then((response) => response.json())
  .then((data) => {
    const contenedor = document.getElementById("productos-container");
    data.forEach((producto) => {
      contenedor.innerHTML += `
        <div class="card">
          
          <h3>${producto.title}</h3>
          <p>Precio: ${producto.price}</p>
          <button onclick="agregarAlCarrito(${producto.id})">Añadir al
carrito</button>
        </div>
      `;
    });
  })
  .catch((error) => console.error("Error al obtener productos:", error));
```

Explicación:

- Pedimos la lista de productos (como pedirle la lista a tu amiga).
- Esperamos que la promesa se cumpla (lleguen los datos).
- Procesamos esa lista para organizarla en el HTML (mostrando imagen, nombre, precio y botón de compra).
- Si falla, al menos nos enteramos con un `.catch()` (como si tu amiga nos dijera “*me olvidé la lista*”).



3. Manejo de errores con Fetch.

Igual que en la vida, puede pasar que la tienda esté cerrada o que la lista esté incompleta. En programación, los errores también suceden, y hay que estar preparados.

Por ejemplo, si la respuesta no viene bien del servidor (códigos 400 o 500), podemos mostrarle un mensaje al usuario para que no se quede esperando:

```
fetch("https://fakestoreapi.com/products")
  .then((response) => {
    if (!response.ok) {
      throw new Error(`Error de servidor: ${response.status}`);
    }
    return response.json();
  })
  .then((data) => {
    // Mostrar productos normalmente
  })
  .catch((error) => {
    console.error("Error al obtener productos:", error);
    alert("Hubo un problema al cargar los productos. Por favor, intentá más tarde.");
  });
```

Explicación:

- Validamos si la respuesta es correcta (`response.ok`), porque a veces el servidor contesta “no encontré nada” (error 404) o “me rompí” (error 500).
-  Si algo sale mal, usamos `throw` para “avisar fuerte” que no podemos seguir.
-  Y con el `.catch()` le damos un mensaje al usuario, así no queda perdido sin explicación.

4. Async/Await: Otra forma de esperar

¿Te acordás de la promesa de cenar con tu amiga? Con `then` le vas mandando mensajitos uno tras otro:

- *¿Podés venir?*
- *¿Traés algo?*
- *¿A qué hora llegás?*

Esos mensajes van encadenados, como pasa con las promesas en JavaScript.

Pero con **async/await**, en vez de andar mandando mensajes sueltos, es como si se sentaran a charlar cara a cara y resuelven todo de una:

- *Confirmamos la cena.*
- *Definimos la hora.*
- *Ya sabemos qué traemos.*

Más directo, más claro, más fácil de leer.

```
async function mostrarProductos() {  
  try {  
    const respuesta = await fetch("https://fakestoreapi.com/products");  
    const data = await respuesta.json();  
    // procesar datos...  
  } catch (error) {  
    console.error("Error al obtener productos:", error);  
  }  
}
```

Explicación:

- `async` indica que la función puede esperar tareas asíncronicas.
- `await` pausa el flujo hasta obtener la respuesta.
- Así evitamos encadenar varios `.then`, y el código queda más limpio.

¿Por qué usar `async/await`?

● Porque hace el código más fácil de leer y entender ya que se parece a un paso a paso escrito en lenguaje natural en lugar de cadenas de `.then` evita anidaciones confusas y permite manejar errores con `try/catch` de forma más clara además la lectura es más lineal y sencilla de mantener

5. Buenas Prácticas de Accesibilidad y SEO

5.1. Accesibilidad en el Proyecto

- **Etiquetas Alt en Imágenes:** Cada imagen debe tener un atributo `alt` que describa el contenido de la imagen, para mejorar la experiencia de usuarios con discapacidades visuales.
- **Navegación por Teclado:** Asegurarse de que los elementos interactivos, como botones y enlaces, puedan ser accedidos con el teclado.
- **Estructura Semántica:** Usar etiquetas como `<header>`, `<main>`, `<section>`, `<footer>` para que el contenido sea fácilmente interpretable.

5.2. Optimización SEO en el Proyecto

- **Metaetiquetas:** Las etiquetas `<meta>` en el `<head>` mejoran la visibilidad en los motores de búsqueda.
- **Encabezados Lógicos:** Usar encabezados (`<h1>`, `<h2>`, `<h3>`) en una estructura jerárquica para que los motores de búsqueda puedan entender la relevancia del contenido.
- **Contenido Optimizado:** Asegurarse de que las descripciones y los nombres de productos sean claros y relevantes para los usuarios.

6. Guía para el Proyecto Final de E-commerce



6.1. Estructura y Requisitos del Proyecto

- **HTML:** Uso de etiquetas semánticas para organizar la página.
- **CSS:** Implementación de un diseño responsivo y atractivo usando Bootstrap y Flexbox.
- **JavaScript:** Integración de una API REST para obtener datos y renderizar productos en el DOM, además de la funcionalidad de un carrito de compras usando `localStorage`.
- **Accesibilidad y SEO:** Implementar prácticas que mejoren la experiencia del usuario y optimicen la página para los motores de búsqueda.

6.2. Puntos Clave para Revisión y Entrega

- **Subida del Proyecto:** Debe estar disponible en GitHub Pages o Netlify para facilitar su acceso.
- **Control de versiones:** Mantener un historial de commits detallado para documentar cada avance.
- **Presentación:** El archivo README.md debe incluir una descripción del proyecto, las tecnologías usadas, instrucciones de instalación y cualquier detalle relevante.


Entrega Final del Proyecto - E-Commerce

Curso de Desarrollo Web - Proyecto Integrador

Punto 1: Introducción al Proyecto Final

En esta etapa final, desarrollarás una página web completa que integre todo lo aprendido a lo largo del curso.

- ♦ El proyecto consiste en un sitio web de e-commerce interactivo que:
 - Consume datos de una API REST.
 - Permite añadir productos a un carrito de compras.

 Tecnologías y enfoques a aplicar:

- HTML: estructura semántica.
- CSS: diseño responsivo con Bootstrap y Flexbox.
- JavaScript: renderizado dinámico, carrito, API.
- Accesibilidad y SEO: buenas prácticas.

Punto 2: Puntos Clave para Revisión y Entrega

- Subida en GitHub Pages o Netlify.
- Control de versiones con commits detallados.
- Archivo README.md con descripción, tecnologías, instalación.

Punto 3: Formato de Entrega

Repositorio en GitHub

El repositorio debe ser público e incluir todos los archivos del proyecto.


Hosting del Proyecto

Subirlo a Netlify o GitHub Pages con un enlace funcional.

Entrega en el Campus Virtual

Subir los enlaces en la sección de Pre-Entrega de Proyecto.

Punto 4: Condiciones de Entrega

 A partir de la clase N°15 contás con 7 días corridos para entregar el proyecto.

- • HTML semántico (header, nav, main, etc.).
- • Formulario funcional con Formspree.
- • Archivo README.md con resumen del proyecto.

Punto 5: Estilos y Diseño Responsivo

- • CSS externo: estilos para header, footer y navegación.
- • Google Fonts correctamente aplicadas.
- • Uso de background (color, imagen o gradiente).
- • Productos: cards con Flexbox.
- • Reseñas: uso de Grid.
- • Contacto: Media Queries para responsividad.

Punto 6: Multimedia y Navegación

- • Imágenes, videos o iframe correctamente integrados.
- • Menú de navegación con lista desordenada e ítems internos.
- • Hosting obligatorio con enlace funcional.

Punto 7: Funcionalidad con JavaScript

- • Archivo script.js enlazado.
- • Validación de formularios (campos requeridos, email).
- • Manipulación del DOM para interacciones.
- • Fetch API para consumir productos desde una API REST.
- • Renderizado de productos en tarjetas con imagen, título y precio.

Punto 8: Carrito de Compras Dinámico

- • Agregar productos desde tarjetas.
- • Guardar carrito con localStorage o sessionStorage.
- • Contador dinámico de productos.
- • Lista de productos con cantidad, precio y total.
- • Posibilidad de editar o eliminar productos.

Punto 9: SEO y Accesibilidad

- • Etiquetas alt en imágenes.
- • Navegación accesible con teclado.
- • Uso de metaetiquetas en el head para mejorar el SEO.

✓ Punto 10: Funcionalidad Esperada

- • Página interactiva que permita ver productos, agregar al carrito, editarlo y simular la compra.
- • Formulario de contacto que funcione correctamente.
- • Diseño adaptable a distintos tamaños de pantalla.
- • Carrito persistente con localStorage/sessionStorage.

A stylized wireframe map of Buenos Aires, Argentina, rendered in white lines on a dark blue background. The map shows the city's outline and internal street patterns, with a few small white circles marking specific locations.

Buenos Aires
aprende
Agencia de Habilidades para el Futuro

BA Buenos
Aires
Ciudad