

Práctica: De libreta a pantalla: nuestra primera web de la granja Pokémon

1. Introducción

Esta práctica consistió en el desarrollo de una aplicación web que permite gestionar digitalmente una granja Pokémon. El objetivo fue aplicar conocimientos de HTML, CSS y JavaScript, incluyendo el uso de formularios, validación, localStorage, y conexión con una API externa (PokéAPI). La solución creada permite registrar, visualizar, filtrar y eliminar Pokémon de manera interactiva.

2. Fases del proyecto

Fase 1: Maquetación inicial

Se diseñó una página web sencilla con un formulario de registro de Pokémon y una sección donde se listan los Pokémon guardados. Se utilizaron etiquetas básicas de HTML como `<form>`, `<input>`, `<label>` y ``.

Fase 2: Registro de Pokémon

El formulario permite ingresar nombre, tipo, nivel, habilidad y fecha de llegada. Se añadió validación de campos obligatorios en JavaScript. Al hacer submit, los datos se almacenan en un array y se guardan en localStorage.

Fase 3: Visualización y persistencia

Los Pokémon registrados se muestran en una lista. Al recargar la página, los datos se mantienen gracias a localStorage. También se implementó un botón de eliminación individual para cada Pokémon.

M4 - Lenguaje de Marcas

Fase 4: Integración con PokéAPI

Se usó la PokéAPI para obtener automáticamente la imagen de cada Pokémon registrado según su nombre. Esto se hizo mediante peticiones fetch asíncronas, extrayendo la URL del sprite (sprites.front_default).

Fase 5: Filtro por tipo y habilidad

Se añadieron campos de texto para filtrar la lista de Pokémon en tiempo real por tipo y habilidad. La lista se actualiza según el filtro introducido.

3. Decisiones técnicas

Persistencia: Se utilizó localStorage para almacenar la lista de Pokémon en el navegador.

API externa: Se empleó fetch para consumir la PokéAPI y obtener imágenes reales de los Pokémon.

Modularidad: El código se organizó en funciones separadas: mostrarPokemon, guardarPokemon, filtrarPokemon, obtenerImagenPokemon, etc.

Interacción: Los botones de eliminar están ligados dinámicamente a cada elemento de la lista.

4. Preguntas

1. ¿Qué ventajas tiene mostrar un mensaje dinámico respecto a una presentación estática?

Un mensaje dinámico permite una comunicación en tiempo real con el usuario, adaptándose a sus acciones, errores o decisiones. A diferencia de un contenido estático que siempre es igual, el contenido dinámico mejora la interacción, la personalización y la usabilidad del sistema.

En mi proyecto, mostré mensajes personalizados cuando se registraba un Pokémon correctamente o cuando había un error en el formulario. Esto simula lo

M4 - Lenguaje de Marcas

que ocurre en muchas páginas web modernas, como cuando haces una compra online y ves confirmaciones o errores al instante.

Situación del mundo real: En una granja Pokémon digital, un mensaje dinámico puede avisar al cuidador si ha introducido mal un nombre o si el Pokémon ya existe, lo que mejora la gestión diaria.

2. ¿Qué es el `textContent` en JavaScript y por qué lo usaste para mostrar el mensaje?

`textContent` es una propiedad de los elementos del DOM que permite establecer o recuperar el texto que contiene un elemento HTML. La usé para mostrar mensajes dentro de un `<p>` o `<div>` sin necesidad de recargar la página ni generar elementos nuevos.

Ejemplo en mi código:

```
const mensaje = document.getElementById('mensaje');  
mensaje.textContent = "Pokémon registrado correctamente.";
```

Esto permitió comunicar al usuario que su acción fue exitosa, sin usar alertas molestas ni cargar de nuevo la página.

3. ¿Qué problema evita la función `event.preventDefault()` en los formularios?

Evita que el formulario se envíe de forma predeterminada (recargando la página). Si no la usamos, el navegador reinicia todo al enviar el formulario, lo que hace que se pierdan los datos o el control del proceso.

En mis palabras: Usarla permite que el formulario se maneje con JavaScript y que podamos validar los datos o guardar cosas sin que la web se recargue.

Ejemplo en mi código:

```
document.getElementById('pokemonForm').addEventListener('submit', (event) => {  
  event.preventDefault();  
});
```

4. ¿Qué comprobación hiciste para evitar que el usuario dejara el campo vacío?

M4 - Lenguaje de Marcas

Comprobé que los campos no estuvieran vacíos en el html con “required” en los inputs

Código usado:

```
<label>Nombre: <input type="text" id="nombre" required></label><br><br>
```

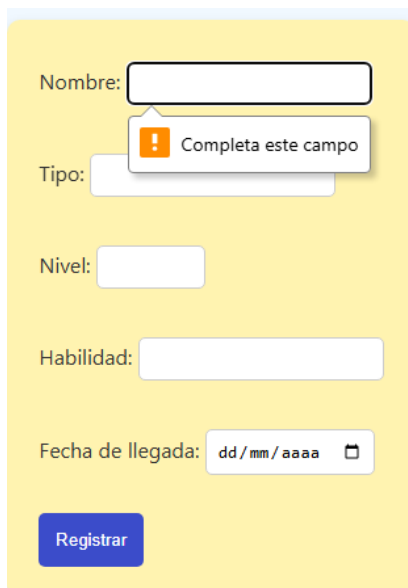
```
<label>Tipo: <input type="text" id="tipo" required></label><br><br>
```

```
<label>Nivel: <input type="number" id="nivel" min="1" max="100" required></label><br><br>
```

```
<label>Habilidad: <input type="text" id="habilidad" required></label><br><br>
```

```
<label>Fecha de llegada: <input type="date" id="fecha" required></label><br><br>
```

5. Muestra una captura del formulario básico y del mensaje personalizado funcionando.



Nombre:

Tipo: ! Completa este campo

Nivel:

Habilidad:

Fecha de llegada: dd/mm/aaaa

6. ¿Qué diferencias encuentras entre escribir un nombre en papel y hacerlo en un formulario digital?

En papel:

- Puede haber errores de ortografía o ilegibilidad.
- El proceso es lento y manual.
- No hay validación inmediata.

En formulario digital:

- Es más rápido y preciso.
- Se puede validar al instante.

M4 - Lenguaje de Marcas

- La información se almacena de forma organizada y reutilizable.

Conclusión: El formulario digital mejora la eficiencia, fiabilidad y acceso a los datos.

7. ¿Qué pasos del proceso tradicional (en papel) se han digitalizado en esta fase?

- Registro manual de Pokémon → Formulario digital interactivo
- Almacenamiento en cuadernos → Guardado automático en localStorage
- Revisión visual en listas de papel → Filtros dinámicos por tipo y habilidad
- Imágenes dibujadas o impresas → Sprites obtenidos automáticamente desde la PokéAPI

8. ¿Cómo afecta la digitalización a la accesibilidad de la información?

Hace que la información esté disponible de forma inmediata, desde cualquier dispositivo con acceso a la aplicación. Evita pérdidas de información por errores humanos o deterioro del papel. También permite automatizar búsquedas y facilitar la toma de decisiones.

9. ¿Crees que este formulario facilitaría el trabajo en una granja Pokémon real?

Sí, porque permite registrar y consultar Pokémon de manera rápida, evitar repeticiones, filtrar por características útiles (tipo o habilidad) y visualizar imágenes para identificación rápida. Reduce el tiempo de gestión y mejora la organización general.

10. ¿Qué función visual o técnica implementaste para mejorar la experiencia del usuario?

- Añadí **imágenes reales** de los Pokémon usando la PokéAPI.
- Mostré **mensajes dinámicos** de confirmación y error.
- El sistema tiene **filtros interactivos** que permiten encontrar Pokémon fácilmente.

M4 - Lenguaje de Marcas

11. Si otra persona quisiera reutilizar tu formulario, ¿qué partes del código necesitaría modificar?

- Cambiar los campos del formulario (nombre, tipo, etc.) si gestionan otra cosa que no sea Pokémon.
- Adaptar la función `obtenerImagenPokemon(nombre)` si usan otra API o base de datos.
- Personalizar los textos mostrados al usuario y los filtros según las necesidades del nuevo contexto.