Fase 1-De libreta a pantalla: nuestra primera web de la biblioteca musical

Objetivos

* Muestra un título principal y un párrafo de presentación.
* Tiene un formulario con un input de texto y un botón.
* Cuando el usuario pulsa el botón, aparece un mensaje personalizado con su nombre.
* Estéticamente debe tener estilo básico: fondo claro, texto legible, botones visibles.
* Si el usuario no escribe nada y pulsa el botón, debe aparecer un mensaje de error suave

Resultado

Campo/s faltante/s:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Todos los campos correctos:

Interfaz de usuario gráfica, Aplicación, Word

El contenido generado por IA puede ser incorrecto.

Fase 2 – Tu primer sistema digital de registro

Objetivos

* Reusa el formulario anterior.
* Validar que no falte nada importante.
* Crear un objeto con esos datos.
* Guardarlo dentro de un array que representa tu base de datos temporal.
* Mostrar las canciones registradas en una lista en pantalla, con su información ordenada.

Resultado

En mi caso, se pueden agregar portadas, y si no se agregan, se añade una por defecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Preguntas

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

Esto tiene grandes ventajas utilitarias, la principal es que no se tiene que cambiar el contenido en el código manualmente. Si se quieren hacer modificaciones se modifica o genera el archivo de nuevo.

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

El textContent es un método de JS que permite interactuar con el texto de una etiqueta.

En mi caso, lo he usado para añadir el texto interior del mensaje. Primero lo identifico con un ID:

<div *id*="msg" *class*="alert d-none position-fixed top-0 end-0 mt-3 me-3 text- start shadow" *role*="alert">

      <i *id*="alert-icon" *class*="me-2"></i>

<p *id*="alert-msg" *class*="m-0"></p>

</div>

Y luego le añado contenido

const alertMsg = document.getElementById("alert-msg");

alertMsg.textContent = `🎵 "${titulo}" de ${artista} añadida correctamente!`;

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

Esta función es muy importante, ya que impide que se reinicie la página web al pulsar el botón de “submit”. Esto es vital para poder tratar la información a la hora de comprobar el contenido.

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

Compruebo si el valor de las variables existe. Si no existe, se le añade el contenido correspondiente al error. Si error tiene contenido, se añade la alerta.

const titulo = document.getElementById('titulo').value.trim();

const artista = document.getElementById('artista').value.trim();

const genero = document.getElementById('genero').value.trim();

let error = '';

if (!titulo) error = "El título es requerido";

else if (!artista) error = "El artista es requerido";

else if (!genero) error = "Debes seleccionar un género";

if (error) {

    alertIcon.className = "fas fa-exclamation-circle text-danger";

    alertMsg.textContent = error;

    msg.className = "alert alert-danger position-fixed top-0 end-0 mt-3 me-3 text-start shadow";

    msg.classList.remove("d-none");

    setTimeout(() => msg.classList.add("d-none"), 3000);

    return false;

}

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

Este es el formulario. Como se observa, si lo mando vacío, sale el mensaje de error correspondiente.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

En cambio, si lo envío con contenido, se puede ver que se registra.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

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

##### Encuentro las siguientes diferencias:

##### Fiabilidad: En un formulario digital puedes controlar que el nombre esté completo, sin caracteres inválidos y sin errores de ortografía mediante validaciones automáticas. En papel, esto depende 100% del usuario

* Velocidad: Se puede llegar a escribir más rápido digitalmente si se usan funciones de autocompletado, sugerencias, etc. Además, la modificación del texto ya escrito es algo que no se encuentra en el formato analógico.
* Legibilidad: Uno digital asegura que todo se encuentre escrito con el mismo formato de letra. Mientras que, en el papel, no se garantiza que tenga una consistencia.

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

Estos son los siguientes pasos del proceso tradicional que he logrado identificar, junto con sus respectivos ejemplos:

* Entrada de datos manual (escribir) ➜ Entrada por teclado digital:

<input *type*="text" *id*="titulo" *name*="titulo" *class*="form-control" *required*>

* Validación manual (revisar visualmente) ➜ Validación automática

if (!titulo) error = "El título es requerido";

* Almacenamiento físico (libreta, documento) ➜ Almacenamiento digital (arrayList o Json)

songsArray.push(newSong);

##### ¿Cómo afecta la digitalización a la accesibilidad de la información?

Creo que mejora muchísimo la accesibilidad. Esto se refleja en que mejoraría mucho la función de la biblioteca musical: Puedes consultar, modificar o borrar datos desde cualquier dispositivo con conexión, sin necesidad de buscar entre carpetas o papeles. También puedes hacer copias de seguridad o compartir los datos con otras personas en segundos.

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

He añadido diversas funciones visuales que ayudan al usuario a encontrarse más cómodo. Por ejemplo, el mensaje de error o de éxito al introducir los campos no es invasivo y desaparece a los segundos.

Otra función visual es el efecto de zoom en la portada de la canción al hacer hover.

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

Todos los fragmentos de código relacionados con un formulario, con validarlo o con gestionar la información que se introduce, podrían ser fácilmente reusados.

Fase 3 - Buscando canciones reales con una API

Objetivos:

* Introducir el nombre de una Canción o Artista en un campo de texto.
* Al pulsar un botón, realizar una petición HTTP usando fetch() a la API de Deezer.
* Si la canción existe, mostrar en pantalla los datos.
* Si no existe o hay un error, mostrar un mensaje claro y amigable.

Fragmento de código explicado:

let searchArray = [];

const songsArray = [];

// Inicia los listeners al cargar el contenido

function initEventListeners() {

    const checkBtn = document.getElementById("check-song");

    if (checkBtn) {

        checkBtn.addEventListener("click", function (event) {

            event.preventDefault();

            if (!validateForm('check')) return;

            checkSongJSONP();

        });

    }

    const addBtn = document.getElementById("add-song");

    if (addBtn) {

        addBtn.addEventListener("click", function (event) {

            event.preventDefault();

            if (!validateForm('add')) return;

            addSong();

        });

    }

}

// Cambia el contenido del body y reinicia los listeners

function toggleBody(element) {

    const url = element.getAttribute("data-url");

    fetch(url)

        .then(response => response.text())

        .then(html => {

            const contenido = html;

            document.getElementById("main").innerHTML = contenido;

            initEventListeners();

        })

        .catch(error => console.error("Error al cargar el contenido:", error));

}

// Obtiene los datos del formulario

function getFormData() {

    return {

        titulo: document.getElementById('titulo').value.trim(),

        artista: document.getElementById('artista').value.trim(),

        puntuacion: document.querySelector('input[name="puntuacion"]:checked')?.value,

        portada: document.getElementById('portada')?.dataset.remote || null

    };

}

// Valida los datos del formulario según el modo (check/add)

function validateForm(mode) {

    const { titulo, artista, puntuacion, portada } = getFormData();

    let error = '';

    if (mode === 'add') {

        if (!titulo) error = "El título es requerido";

        else if (!artista) error = "El artista es requerido";

        else if (!puntuacion) error = "Debes seleccionar una puntuación";

        else if (!portada) error = "No hay imagen remota de portada disponible";

    } else if (mode === 'check') {

        if (!titulo && !artista) error = "Debes completar al menos un campo para buscar";

    }

    if (error) {

        showMessage('error', error);

        return false;

    }

    if (mode === 'add') {

        showMessage('success', `🎵 "${titulo}" de ${artista} añadida correctamente!`);

    }

    return true;

}

// Muestra un mensaje en pantalla (éxito o error)

function showMessage(type, text) {

    const msg = document.getElementById("msg");

    const alertIcon = document.getElementById("alert-icon");

    const alertMsg = document.getElementById("alert-msg");

    alertIcon.className = type === 'success'

        ? "fas fa-check-circle text-success"

        : "fas fa-exclamation-circle text-danger";

    alertMsg.textContent = text;

    msg.className = `alert alert-${type === 'success' ? 'success' : 'danger'} position-fixed top-0 end-0 mt-3 me-3 text-start shadow`;

    msg.classList.remove("d-none");

    setTimeout(() => msg.classList.add("d-none"), 3000);

}

// Llama a la API de Deezer usando JSONP

function checkSongJSONP() {

    const { titulo, artista } = getFormData();

    const parts = [];

    if (titulo) parts.push(`track:"${titulo}"`);

    if (artista) parts.push(`artist:"${artista}"`);

    const query = encodeURIComponent(parts.join(' '));

    const script = document.createElement('script');

    script.src = `https://api.deezer.com/search?q=${query}&output=jsonp&callback=deezerCallback`;

    script.onerror = () => showMessage('error', '😹😉 Error al cargar datos de Deezer.');

    script.onload = () => document.body.removeChild(script);

    document.body.appendChild(script);

}

// Callback que recibe datos desde Deezer

function deezerCallback(response) {

    if (response && response.data && response.data.length) {

        searchArray = response.data.slice(0, 10);

        fillResultData();

    } else {

        showMessage('error', 'No se encontró ningún resultado.');

    }

}

// Llena la lista de resultados con las canciones encontradas

function fillResultData(){

    const resultList = document.getElementById("results-list");

    const emptyMsg = document.getElementById("empty-msg");

    resultList.style.display = searchArray.length ? "flex" : "none";

    emptyMsg.style.display = searchArray.length ? "none" : "flex";

    resultList.innerHTML = searchArray.map((song, index) => `

    <li class="d-flex justify-content-between align-items-center search-result">

        <div class="d-flex align-items-center">

            <p class="mb-0 me-5">#${index + 1}</p>

            <div class="search-album-art">

                <img src="https://e-cdns-images.dzcdn.net/images/cover/${song.md5\_image}/500x500-000000-80-0-0.jpg" alt="${song.title}" class="album-cover">

            </div>

            <div class="search-song-info ms-3 mt-2">

                <h5 class="text-primary">${song.title}</h5>

                <p class="mb-1"><strong>Artista:</strong> ${song.artist.name}</p>

            </div>

        </div>

        <i class="fa-solid fa-square-plus fs-4 text-secondary" onclick='fillFormData(${JSON.stringify(song.title)}, ${JSON.stringify(song.artist.name)}, ${JSON.stringify(song.md5\_image)})'></i>

    </li>

    `).join('');

}

// Rellena el formulario con los datos de una canción seleccionada

function fillFormData(titulo, artista, portada) {

    document.getElementById('titulo').value = titulo;

    document.getElementById('artista').value = artista;

    if (typeof portada === 'string') {

        const imgURL = `https://e-cdns-images.dzcdn.net/images/cover/${portada}/500x500-000000-80-0-0.jpg`;

        document.getElementById('preview-image').src = imgURL;

        document.getElementById('preview-container').style.display = 'flex';

        document.getElementById('portada').dataset.remote = portada;

    }

}

// Añade una canción al array de canciones

function addSong() {

    const { titulo, artista, puntuacion, portada } = getFormData();

    const defaultCover = 'img/default-cover.png';

    const newSong = {

        titulo,

        artista,

        puntuacion,

        portada: portada ? `https://e-cdns-images.dzcdn.net/images/cover/${portada}/500x500-000000-80-0-0.jpg` : defaultCover,

        md5\_image: portada

    };

    songsArray.push(newSong);

    resetForm();

    resetResultData();

}

// Reinicia el formulario tras añadir una canción

function resetForm() {

    document.getElementById("formulario").reset();

    document.getElementById("preview-container").style.visibility = 'hidden';

    document.getElementById("preview-image").src = '';

    document.getElementById("portada").dataset.remote = '';

}

function resetResultData() {

    const resultList = document.getElementById("results-list");

    const emptyMsg = document.getElementById("empty-msg");

    resultList.style.display = "none";

    emptyMsg.style.display = "flex";

}

// Muestra la lista de canciones añadidas

function showSongs() {

    const songsList = document.getElementById("songs-list");

    const emptyMsg = document.getElementById("empty-msg");

    songsList.style.display = songsArray.length ? "grid" : "none";

    emptyMsg.style.display = songsArray.length ? "none" : "flex";

    songsList.innerHTML = songsArray.map(song => `

        <div class="song-card bg-light p-3 mb-3 rounded">

            <div class="album-art"><img src="${song.portada}" alt="${song.titulo}" class="album-cover"></div>

            <div class="song-info mt-2"><h5 class="text-primary">${song.titulo}</h5>

                <p class="mb-1"><strong>Artista:</strong> ${song.artista}</p>

                <p class="mb-0"><strong>Puntuación:</strong><span id="rating${song.titulo}"></span></p>

            </div>

        </div>

    `).join('');

    songsArray.forEach(song => addRating(song.puntuacion, song.titulo));

}

// Añade estrellas según la puntuación

function addRating(rating, title) {

    const id = document.getElementById('rating' + title);

    for (let i = 0; i < rating; i++) id.appendChild(document.createTextNode("★"));

    for (let i = rating; i < 5; i++) id.appendChild(document.createTextNode("☆"));

}

Preguntas:

##### ¿Qué es una API y qué beneficios tiene utilizarla en una aplicación como esta?

Una API es un programa o programas que permiten que diferentes programas o sistemas se comuniquen entre sí.

En el caso de esta aplicación, me ayuda a coger la información de una pase de datos grande, como la de Deezer, y transformarla en contenido legible y usable.

##### ¿Qué diferencias hay entre usar .then() y async/await? ¿Cuál elegiste y por qué?

En mi caso, para llamar a la API, estoy usando JSONP, ya que la API de Deezer no permite peticiones directas desde el navegador por política de CORS. A continuación, el ejemplo explicado:

// Llama a la API de Deezer usando JSONP

function checkSongJSONP() {

    const { titulo, artista } = getFormData(); //Recoge datos del formulario

    const parts = [];

    if (titulo) parts.push(`track:"${titulo}"`);

    if (artista) parts.push(`artist:"${artista}"`);

    const query = encodeURIComponent(parts.join(' ')); //Crea una query para añadir a la busqueda

    const script = document.createElement('script');  //Añade un script al documento

    script.src = `https://api.deezer.com/search?q=${query}&output=jsonp&callback=deezerCallback`; //Le añade como source la dirección de la API

    script.onerror = () => showMessage('error', 'Error al cargar datos de Deezer.');

    script.onload = () => document.body.removeChild(script); //Le añade funciones de onerror y onload para gestionar esos casos

    document.body.appendChild(script); //Añade el script

}

##### ¿Cómo seleccionaste y mostraste los datos que devuelve la API?

Una vez se realiza la petición, una función llamada deezerCallBack, trata la respuesta, introduciendo los 10 primeros resultados en un array.

// Callback que recibe datos desde Deezer

function deezerCallback(response) {

    if (response && response.data && response.data.length) { //Si existe la respuesta

        searchArray = response.data.slice(0, 10); //añade los primeros 10 resultados

        fillResultData(); //Y ejecuta la funcion para llenar los resultados

    } else {

        showMessage('error', 'No se encontró ningún resultado.');

    }

}

##### ¿Qué estrategia utilizaste para gestionar errores si el nombre escrito no era válido?

En la función deezercallback:

function deezerCallback(response) {

    if (response && response.data && response.data.length) {

        searchArray = response.data.slice(0, 10);

        fillResultData();

    } else {

        showMessage('error', 'No se encontró ningún resultado.'); //Ejecuta la función showMessage

    }

}

##### Muestra una captura de una búsqueda exitosa y una de un error gestionado correctamente. Añade una breve explicación.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Si se introduce un nombre y se pulsa en “Check Song”, se busca los resultados y se pueden añadir al formulario con la portada, para posteriormente añadirlos al array temporal con los datos.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Sitio web

El contenido generado por IA puede ser incorrecto.

Si se introduce mal, sale un error.

Fase 4: Sistema completo de digitalización (con filtros y memoria)

Objetivos

* Guardar los datos localmente con localStorage, para que no se pierdan al cerrar el navegador.
* Recuperar esos datos automáticamente al volver a cargar la página.
* Filtrar los Pokémon por tipo o habilidad usando un menú desplegable.
* Dividir tu interfaz en secciones claras: registro, búsqueda, filtro, reflexión.
* Organizar el código por funciones para mejorar la lectura.
* Añadir una sección final que explique la digitalización vivida.

Preguntas

¿Qué función cumple localStorage en tu web? ¿Qué diferencia hay entre setItem y getItem?

localStorage se usa en esta web para guardar de forma persistente las canciones añadidas por el usuario almacenadas en el array songsArray. Esto permite que, aunque se recargue la página o se cierre el navegador, las canciones sigan estando disponibles la próxima vez que se acceda al sitio.

Estas son las diferencias entre setItem y getItem:

* **setItem** guarda un valor en el almacenamiento local con una clave específica.
* **getItem** recupera el valor asociado a una clave del almacenamiento local.

En mi código, uso el setItem en saveLocalStorage:

function saveLocalStorage() {

    localStorage.setItem("cancion", JSON.stringify(songsArray));

}

Y el getItem en toggleBody:

function toggleBody(element) {

    const modalContainer = document.getElementById("modal-container");

    const url = element.getAttribute("data-url");

    fetch(url)

        .then(response => response.text())

        .then(html => {

            document.getElementById("main").innerHTML = html;

            appendVideo();

            if (url === `html/songslist.html`) {

                const datosGuardados = localStorage.getItem("cancion");

                if (datosGuardados) {

                    const array = JSON.parse(datosGuardados);

                    songsArray.length = 0;

                    array.forEach(song => songsArray.push(song));

                    showSongs();

                }

            }

¿Qué significa JSON.stringify() y por qué lo necesitas para guardar datos?

Este método convierte un objeto JavaScript, como un array, a una cadena de texto en formato JSON. En mi código, lo he usado en la función:

function saveLocalStorage() {

    localStorage.setItem("cancion", JSON.stringify(songsArray));

}

¿Cómo creaste y aplicaste el filtro por tipo o habilidad? ¿Qué función utilizaste?

Para aplicar el filtro, he usado una función que he creado llamada fillFilterDataResult().

function fillFilterDataResult() {

    const { titulo, artista, puntuacion } = getFormData('filter'); //Coge la información del formulario de filtro

    const datosGuardados = localStorage.getItem("cancion"); //convierte el localStorage la constante datos guardados

    if (!datosGuardados) return;

    const array = JSON.parse(datosGuardados); //Convierte de JSON a Array la constante

    const filteredArray = array.filter(song => { //Uso la función filter para crear un array

        const matchesTitulo = !titulo || song.titulo.toLowerCase().includes(titulo.toLowerCase());

        const matchesArtista = !artista || song.artista.toLowerCase().includes(artista.toLowerCase());

        const matchesPuntuacion = !puntuacion || song.puntuacion === puntuacion;

        //Se comprueba si cada canción cumple con las condiciones del filtro

        return matchesTitulo && matchesArtista && matchesPuntuacion; //Devuelve true solo si la revision cumple las 3 condiciones. Esto hace que solo las que cumplan el filtro se guarden en el array.

    });

    if (filteredArray.length === 0) { //Si el array es igual a 0, no hay resultados

        showMessage("error", "No se han encontrado resultados");

    } else { //Si el array es mayro, si que hay resultados

        showMessage("success", "Se han encontrado resultados");

        songsArray.length = 0; //Se vacía el array de canciones (el que las muestra)

        filteredArray.forEach(song => songsArray.push(song)); //Y se llena con las canciones del filtro.

    }

    showSongs(); //Posteriormente se llama a que se ejecuten las funciones

}

¿Cómo dividiste tu interfaz en secciones (registro, búsqueda, reflexión)? ¿Qué método usaste?

En mi caso, tengo un Layout con un menú flotante.

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Mi Biblioteca Musical</title>

    <script src="script.js" defer></script>

    <link rel="stylesheet" href="style.css">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">

    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css" rel="stylesheet">

    <link href="https://fonts.googleapis.com/css2?family=Orbitron&display=swap" rel="stylesheet">

    <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap" rel="stylesheet">

    <script src="https://cdn.jsdelivr.net/npm/spark-md5@3.0.2/spark-md5.min.js"></script>

</head>

<body>

    <header class="text-white text-center ">

        <ul class="nav justify-content-center">

            <li class="nav-item" id="toggle-modal">

                <a><i class="fa-solid fa-chevron-down"></i></a>

            </li>

            <div id="modal-container">

                <div class="form-container">

                    <p class="text-left mb-0">Pulsa en este botón en las otras páginas para interactuar con ellas.</p>

                </div>

            </div>

        </ul>

        <ul class="nav justify-content-center">

            <li class="nav-item">

                <a class="nav-link active" data-url="html/index.html" onclick="toggleBody(this)"><i

                        class="fa-solid fa-house"></i></a>

            </li>

            <li class="nav-item">

                <a class="nav-link active" data-url="html/register.html" onclick="toggleBody(this)"><i

                        class="fa-solid fa-music"></i></a>

            </li>

            <li class="nav-item">

                <a class="nav-link active" data-url="html/songslist.html" onclick="toggleBody(this)"><i

                        class="fa-solid fa-bookmark"></i></a>

            </li>

        </ul>

    </header>

    <main class="container-fluid pt-3 pb-3" id="main">

        <div class="row justify-content-center gap-5">

            <section class="col-12 col-md-6 mt-3 mt-md-0 rounded shadow p-4" style="max-width: 1000px;">

                <h2>Bienvenido a tu biblioteca musical.</h2>

            </section>

        </div> <video autoplay muted loop id="backgroundVideo">

            <source src="img/background.mp4" type="video/mp4">

            Tu navegador no soporta videos HTML5.

        </video>

    </main>

    <div id="msg" class="alert d-none position-fixed top-0 end-0 mt-3 me-3 text-start shadow" role="alert">

        <i id="alert-icon" class="me-2"></i>

        <p id="alert-msg" class="m-0"></p>

    </div>

</body>

</html>

Cuando se pulsa en un elemento del menú, se activa la función toggleBody, que cambia el contenido del main dependiendo del botón pulsado:

function toggleBody(element) {

    const modalContainer = document.getElementById("modal-container");

    const url = element.getAttribute("data-url"); //Añado el elemento de data-url para hacer el fetch a esa url

    fetch(url)

        .then(response => response.text()) //Convierto la respuesta a texto

        .then(html => {

            document.getElementById("main").innerHTML = html; //Con el contenido, lo añado.

            appendVideo(); //También le añado el video de fondo

            if (url === `html/songslist.html`) { //Si me encuentro en songslist.html

                const datosGuardados = localStorage.getItem("cancion");

                if (datosGuardados) {

                    const array = JSON.parse(datosGuardados);

                    songsArray.length = 0;

                    array.forEach(song => songsArray.push(song));

                    showSongs(); //Muestro las canciones

                }

            }

            appendModal(url); //También le añado el modal de filtrado / registro

            initEventListeners(); //Y le añado los eventListeners correspondientes para cada botón.

        })

        .catch(error => console.error("Error:", error));

}

Muestra una captura de tu filtro funcionando y otra tras recargar la página con los datos restaurados.

Filtro funcionando:

Una captura de pantalla de un celular con texto e imagen

El contenido generado por IA puede ser incorrecto.

Filtro eliminado:

Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

¿Qué ventajas notaste al dividir tu código en funciones? ¿Qué parte fue más fácil de mantener?

El hecho de dividir mi código en funciones me ha ayudado mucho a

Reflexiona sobre la diferencia entre registrar manualmente y tener este sistema digital. ¿Qué has automatizado?

Creo que mejora muchísimo la accesibilidad. Esto se refleja en que mejora mucho la función de la biblioteca musical: puedes consultar, modificar o borrar datos desde cualquier dispositivo con conexión, sin necesidad de buscar entre carpetas o papeles. Además he automatizado varias tareas importantes que antes requerirían hacerlo todo a mano, como:

* Recuperar esos datos automáticamente al volver a cargar la página, lo cual evita tener que volver a escribirlos.
* Filtrar los Pokémon por tipo o habilidad usando un menú desplegable, lo que permite encontrar resultados específicos en segundos sin revisar uno por uno.