Logotipo

El contenido generado por IA puede ser incorrecto.

**UNIVERSIDAD EUROPEA DE MADRID**

**ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO**

**Proyecto de Desarrollo de Web y Apps**

***Página web para gestión de playlists o listas de reproducción***

**GRUPO 08**

**Realizado por: Sergio de Miguel Vallejo y**

**David Novo Rodríguez**

**Profesor: Ana del Valle Corrales Paredes**

**Fecha entrega: 9 de enero de 2026**

**CURSO 2025-2026**

**Índice:**

[Capítulo 1. Introducción 2](#_Toc218693496)

[**1.1 Descripción del Proyecto** 2](#_Toc218693497)

[**1.1 Objetivos** 2](#_Toc218693498)

[**1.2.1 Objetivos generales** 2](#_Toc218693499)

[**1.2.2 Objetivos específicos** 2](#_Toc218693500)

[**1.3 Tecnologías Utilizadas** 3](#_Toc218693501)

[Capítulo 2. Requisitos 4](#_Toc218693502)

[**2.1 Requisitos Funcionales** 4](#_Toc218693503)

[**2.2 Requisitos no Funcionales** 6](#_Toc218693504)

[**2.3 Roles de Usuario** 7](#_Toc218693505)

[Capítulo 3. Diseño del Sistema 8](#_Toc218693506)

[**3.1 Arquitectura General** 8](#_Toc218693507)

[**3.2 Diseño de la Base de Datos** 10](#_Toc218693508)

[**3.3 Estructura del Proyecto** 11](#_Toc218693509)

[Capítulo 4. Desarrollo 13](#_Toc218693510)

[**4.1 Frontend** 13](#_Toc218693511)

[**4.2 Backend** 27](#_Toc218693512)

[Capítulo 5. Instrucciones de Ejecución 59](#_Toc218693513)

[**5.1 Instalación** 59](#_Toc218693514)

[**5.2 Inicialización de la Base de Datos** 59](#_Toc218693515)

[**5.3 Ejecución del Proyecto** 61](#_Toc218693516)

[Capítulo 6. Conclusión 62](#_Toc218693517)

[Capítulo 7. Anexos 63](#_Toc218693518)

[**7.1 Tablas BD** 63](#_Toc218693519)

[**7.2 Información Usada** 65](#_Toc218693520)

# **Capítulo 1. Introducción**

## **1.1 Descripción del Proyecto**

El presente trabajo consiste en el desarrollo de una aplicación web completa y responsive para la gestión de playlists musicales, utilizando exclusivamente música de código abierto. La plataforma se ha diseñado para cubrir tanto el frontend como el backend, aplicando conocimientos fundamentales de desarrollo web sin recurrir a frameworks externos, con el objetivo de consolidar las bases de programación en HTML, CSS, JavaScript, PHP y MySQL.

La aplicación incorpora un sistema de registro e inicio de sesión con dos roles diferenciados:

* **Administrador**, con permisos para gestionar el catálogo musical, crear playlists públicas y moderar contenido.
* **Usuario estándar**, que puede crear, editar y eliminar sus propias playlists, marcar canciones como favoritas, reproducir música y consultar estadísticas personales.

Entre las funcionalidades principales se incluyen:

* **CRUD completo de playlists** (crear, leer, actualizar y eliminar).
* **Reproductor de audio HTML5** con controles básicos y avanzados.
* **Gestión del catálogo musical**, con metadatos detallados de cada canción.
* **Búsqueda y filtrado** por título, artista, género, propósito o popularidad.
* **Interfaz responsive y accesible**, adaptada a móviles, tabletas y ordenadores.
* **Buenas prácticas de seguridad**, como hash de contraseñas, validación de datos, protección contra inyecciones SQL y XSS, y control de acceso por roles.

## **1.1 Objetivos**

### **1.2.1 Objetivos generales**

Desarrollar una aplicación web completa y responsive para la gestión de playlists musicales, que permita registrar usuarios, administrar canciones y reproducir audio utilizando tecnologías web fundamentales (HTML, CSS, JavaScript y PHP) sin el uso de frameworks externos. El proyecto busca consolidar los conocimientos esenciales de desarrollo web, aplicando buenas prácticas de programación, seguridad y diseño responsive en un entorno realista de gestión de contenidos musicales.

### **1.2.2 Objetivos específicos**

1. **Implementar un sistema de registro e inicio de sesión** con validación de datos y roles diferenciados (Administrador y Usuario estándar).
2. **Diseñar y desarrollar el frontend** utilizando HTML, CSS y JavaScript vanilla, garantizando una interfaz intuitiva, responsive y accesible en distintos dispositivos.
3. **Construir el backend** en PHP, con conexión a base de datos y gestión segura de usuarios, playlists y canciones.
4. **Aplicar operaciones CRUD completas** para la gestión de playlists y catálogo musical.
5. **Integrar un reproductor de audio HTML5** con controles básicos y avanzados, incluyendo historial de reproducción y contador automático de escuchas.
6. **Incorporar funcionalidades de búsqueda y filtrado** por título, artista, género, propósito o popularidad.
7. **Garantizar la seguridad de la aplicación**, mediante hashing de contraseñas, consultas parametrizadas y protección contra XSS e inyecciones SQL.
8. **Documentar el proyecto de forma completa**, incluyendo diagramas de base de datos, instrucciones de instalación, estructura de carpetas y capturas de pantalla.
9. **Gestionar el ciclo de desarrollo con control de versiones en GitHub,** fomentando la colaboración y el seguimiento del progreso.

## **1.3 Tecnologías Utilizadas**

Para el desarrollo de la plataforma web de gestión de playlists se utilizaron las siguientes tecnologías y herramientas:

1. Tecnologías de Desarrollo Web:

* **HTML5**: Lenguaje de marcas usado para la estructura y contenido de la interfaz de usuario.
* **CSS3**: Lenguaje de estilo para el diseño visual, estilos responsive y maquetación.
* **JavaScript:** Lenguaje de programación usado para la interactividad, manipulación del DOM y comunicación con el backend mediante peticiones asíncronas.

2. Tecnologías de Backend:

* **PHP**: Lenguaje de programación utilizado para implementar la lógica del servidor, las API y la comunicación con la base de datos.
* **Apache**: Servidor web ejecutado dentro de XAMPP para alojar y procesar las peticiones del backend.

3. Base de Datos:

* **MySQL**: Sistema gestor de bases de datos utilizado para almacenar usuarios, canciones, playlists y estadísticas.

4. Entorno de Desarrollo:

* **XAMPP**: Paquete de servidor local que integra Apache, PHP y MySQL, utilizado para ejecutar la aplicación en entorno de desarrollo.

5. Control de Versiones y Colaboración:

* **Git**: Sistema de control de versiones para gestionar los cambios del proyecto.
* **GitHub**: Plataforma utilizada para alojar el repositorio, documentar el proyecto y hacer seguimiento del desarrollo.

6. Otras Herramientas:

* **Visual Studio Code:** Editor de código utilizado para el desarrollo del proyecto.

# **Capítulo 2. Requisitos**

## **2.1 Requisitos Funcionales**

**1. Registro e Inicio de Sesión**

* El sistema debe permitir registrar usuarios con nombre de usuario único, correo único y contraseña mínima de 8 caracteres.
* Debe validar formato de email y fortaleza de contraseña.
* Debe permitir iniciar sesión con usuario y contraseña.
* Debe mostrar mensajes de error claros ante credenciales incorrectas o usuario existente.
* Debe permitir cerrar sesión de forma segura.

**2. Roles de Usuario**

**Administrador**

* Puede subir nuevas canciones con todos sus metadatos.
* Puede eliminar canciones del catálogo.
* Puede crear playlists públicas.

**Usuario Estándar**

* Puede crear playlists personales (públicas o privadas).
* Puede editar y eliminar sus playlists.
* Puede añadir o eliminar canciones de sus playlists.
* Puede reproducir música del catálogo.
* Puede marcar canciones como favoritas.
* Puede ver playlists públicas de otros usuarios.
* Puede consultar su historial de reproducción.

**3. Gestión de Playlists**

* Crear playlists con nombre, descripción, imagen, visibilidad y autor.
* Mostrar playlists propias y playlists públicas de otros usuarios.
* Editar nombre, descripción, imagen y visibilidad.
* Eliminar playlists con confirmación.
* Añadir canciones a la playlist.
* Eliminar canciones de una playlist.
* Mostrar información automática: número de canciones, duración total, fecha de creación y última modificación.

**4. Gestión del Catálogo Musical**

Cada canción debe incluir:

* Título
* Artista
* Álbum (opcional)
* Duración mm:ss
* Portada
* Género
* Año
* Licencia de uso
* Número de reproducciones (contador automático)

**5. Reproductor de Audio**

* Controles: play, pause, stop, siguiente, anterior.
* Barra de progreso interactiva.
* Control de volumen y mute.
* Mostrar información de la canción actual.
* Incrementar contador de reproducciones automáticamente.
* Registrar historial personal.
* Reproducción continua.

**6. Búsqueda y Filtrado**

* Búsqueda por título, artista o álbum.
* Filtros por género, propósito y canciones más reproducidas.
* Resultados en tiempo real.

**7. Interfaz de Usuario**

* Diseño responsive para móvil, tablet, escritorio y pantallas grandes.
* Menú adaptativo (hamburguesa en móvil).
* Feedback visual inmediato (carga, éxito, error).

**8. Rendimiento**

* Debe soportar al menos 100 canciones sin problemas de carga.

**9. Seguridad**

* Contraseñas hasheadas con bcrypt o Argon2 + salt.
* Consultas parametrizadas para evitar SQL Injection.
* Sanitización para evitar XSS.
* Validación de datos en cliente y servidor.
* Control de acceso por roles.
* Autenticación mediante tokens (JWT o sesiones).

## **2.2 Requisitos no Funcionales**

**1. Usabilidad**

* La interfaz debe ser intuitiva, clara y fácil de navegar para cualquier tipo de usuario.
* El diseño debe ser consistente en todas las páginas (tipografías, colores, espaciados, iconografía).
* Todas las acciones deben proporcionar feedback visual inmediato (carga, éxito, error).
* Los formularios deben incluir validación en cliente y mensajes de error comprensibles.
* El reproductor debe ser accesible y fácil de usar, incluso en pantallas pequeñas.

**2. Responsividad**

* La plataforma debe adaptarse correctamente a:
  + Smartphones (< 768px)
  + Tablets (768px – 1024px)
  + Escritorio (> 1024px)
  + Monitores grandes (> 1440px)
* El menú debe cambiar automáticamente entre modo hamburguesa y modo expandido según el tamaño de pantalla.
* Las imágenes deben escalarse sin perder proporción ni calidad.

**3. Rendimiento**

* La aplicación debe cargar el catálogo de al menos 100 canciones sin ralentizaciones perceptibles.
* Las peticiones al servidor deben ser eficientes y minimizar el tiempo de respuesta.
* El reproductor debe funcionar sin interrupciones ni cortes.
* Las imágenes deben estar optimizadas para reducir tiempos de carga.

**4. Seguridad**

* Las contraseñas deben almacenarse mediante hashing seguro (bcrypt o Argon2) con salt único.
* Todas las consultas a la base de datos deben usar prepared statements para evitar SQL Injection.
* Todo dato introducido por el usuario debe ser sanitizado para prevenir XSS.
* El sistema debe validar datos tanto en cliente como en servidor.
* Debe existir control de acceso estricto según roles (Admin / User).
* La autenticación debe realizarse mediante tokens seguros (JWT o sesiones con expiración).

**5. Mantenibilidad**

* El código debe estar organizado en módulos y carpetas lógicas (frontend, backend, database).
* Debe incluir comentarios significativos en funciones complejas o decisiones importantes.
* La estructura del proyecto debe permitir añadir nuevas funcionalidades sin romper las existentes.
* El código debe seguir convenciones claras de estilo (indentación, nombres de variables, etc.).

**6. Fiabilidad**

* El sistema debe manejar errores de forma controlada sin mostrar información sensible.
* Las operaciones críticas (registro, login, creación de playlists) deben garantizar integridad de datos.
* El reproductor debe continuar funcionando incluso si fallan elementos no esenciales de la interfaz.

**7. Testabilidad**

* El código debe estar estructurado de forma que permita pruebas unitarias y de integración.
* Las funciones clave deben ser independientes y fácilmente testeables.
* Los mensajes de error deben ser claros para facilitar la depuración.

**8. Compatibilidad**

* La aplicación debe funcionar correctamente en los navegadores modernos:
  + Chrome
  + Firefox
  + Edge
  + Safari

## **2.3 Roles de Usuario**

En la plataforma existen dos tipos de usuarios con permisos y capacidades diferenciadas: **Administrador (Admin)** y **Usuario Estándar (User)**. Cada rol determina qué acciones puede realizar el usuario dentro del sistema.

**1. Rol Administrador (Admin)**

Los administradores tienen acceso completo a las funciones de gestión del catálogo musical y a herramientas de supervisión de la plataforma. Sus responsabilidades incluyen:

**Gestión del catálogo**

* Añadir nuevas canciones con todos sus metadatos obligatorios y opcionales.
* Eliminar canciones del catálogo.

**Gestión de playlists públicas**

* Crear playlists públicas accesibles para todos los usuarios.

**Acceso general**

* Acceso a todas las funcionalidades disponibles para usuarios estándar.

**2. Rol Usuario Estándar (User)**

Los usuarios estándar pueden gestionar su propia experiencia musical dentro de la plataforma, creando y personalizando playlists personales.

**Gestión de playlists**

* Crear playlists personales (públicas o privadas).
* Editar nombre, descripción, imagen y visibilidad.
* Eliminar sus propias playlists.
* Añadir o eliminar canciones de sus playlists.

**Interacción con el catálogo**

* Reproducir canciones del catálogo.
* Marcar canciones como favoritas.
* Consultar playlists públicas creadas por otros usuarios.

**Historial y actividad**

* Visualizar su historial de reproducción.
* Incrementar automáticamente el contador de reproducciones al escuchar canciones.

# **Capítulo 3. Diseño del Sistema**

## **3.1 Arquitectura General**

La arquitectura general de Musicfy se basa en un modelo **cliente-servidor** dividido en tres capas principales: **Frontend**, **Backend** y **Base de Datos**. Esta estructura permite separar responsabilidades, facilitar el mantenimiento y asegurar la escalabilidad del sistema a medida que se añadan nuevas funcionalidades como autenticación, roles, historial de reproducción o estadísticas avanzadas.

**3.1.1 Capa Cliente (Frontend):**

El frontend está desarrollado con **HTML5, CSS3 y JavaScript vanilla**, sin frameworks, cumpliendo los requisitos del proyecto. Esta capa es responsable de:

* Renderizar todas las vistas de la aplicación (inicio, búsqueda, playlists, favoritos, estadísticas, login, registro…).
* Gestionar la interacción del usuario mediante manipulación del DOM.
* Controlar el reproductor de audio HTML5 (play, pause, siguiente, anterior, barra de progreso, volumen).
* Enviar peticiones al backend mediante fetch() para operaciones como:
  + Crear playlists
  + Añadir/eliminar canciones
  + Marcar favoritos
  + Registrar reproducciones
  + Iniciar sesión o registrarse
* Mostrar feedback visual inmediato (errores, cargas, acciones completadas).
* Adaptarse a móviles, tablets y escritorio mediante diseño responsive.

El frontend actúa como consumidor de la API PHP, sin acceso directo a la base de datos.

**3.1.2 Capa Servidor (Backend en PHP)**

El backend está implementado en **PHP**, organizado en módulos independientes que gestionan cada funcionalidad. Esta capa se encarga de:

**Lógica de negocio**

* CRUD completo de playlists (crear, leer, actualizar, eliminar).
* Gestión del catálogo musical (importación, edición, eliminación).
* Sistema de favoritos por usuario.
* Registro e inicio de sesión (con contraseñas hasheadas).
* Control de roles (Admin / User).
* Historial de reproducción.
* Generación de estadísticas personales y globales.
* Filtrado y búsqueda avanzada de canciones.

**Comunicación con el cliente**

* Responder a peticiones AJAX en formato JSON.
* Renderizar páginas dinámicas mediante PHP cuando sea necesario.

**Seguridad**

* Uso de consultas preparadas para evitar SQL Injection.
* Gestión de sesiones seguras.
* Control de acceso según rol del usuario.

El backend actúa como intermediario entre el cliente y la base de datos, aplicando todas las reglas de negocio.

**3.1.3 Capa de Datos (Base de Datos MySQL)**

La base de datos almacena toda la información persistente del sistema. Está compuesta por tablas diseñadas para cubrir todos los requisitos del proyecto:

* **users** → credenciales, roles, email, nombre
* **songs** → catálogo musical completo con metadatos
* **playlists** → información de cada playlist
* **playlist\_songs** → relación N:N entre playlists y canciones
* **favorites** → canciones marcadas como favoritas por cada usuario
* **history** → historial de reproducción
* **(posibles tablas futuras)** para exportación, estadísticas avanzadas, etc.

La estructura está normalizada y preparada para consultas eficientes, especialmente en búsquedas, estadísticas y gestión de playlists.

**3.1.4 Flujo de Comunicación**

1. El usuario interactúa con la interfaz (por ejemplo, reproduce una canción o crea una playlist).
2. El frontend envía una petición HTTP al backend (GET/POST).
3. El backend procesa la petición, valida permisos y accede a la base de datos.
4. El backend devuelve una respuesta (HTML o JSON).
5. El frontend actualiza la interfaz dinámicamente sin recargar la página.

Este flujo permite una experiencia fluida y moderna.

## **3.2 Diseño de la Base de Datos**

El diseño de la base de datos de Musicfy sigue un modelo **relacional** basado en MySQL, estructurado para soportar todas las funcionalidades del sistema: gestión de usuarios, catálogo musical, playlists, favoritos, historial de reproducción y estadísticas.

La base de datos está normalizada y organizada en tablas independientes pero relacionadas entre sí mediante claves primarias y foráneas, garantizando integridad, consistencia y escalabilidad.

La estructura se ha diseñado para permitir:

* Consultas rápidas sobre canciones, playlists y usuarios.
* Relaciones N:N entre playlists y canciones.
* Registro histórico de reproducciones.
* Gestión de favoritos por usuario.
* Control de roles (admin/user).
* Futuras ampliaciones como estadísticas avanzadas, exportación de playlists o visualización de tendencias.

**Tablas:**

Todas las tablas se muestran detalladas en los [Anexos de BD](#_7.1_Tablas_BD).

**Relaciones principales:**

* **users 1:N playlists**
* **users 1:N history**
* **users N:N favorites (con songs)**
* **playlists N:N songs (playlist\_songs)**
* **songs 1:N history**

Estas relaciones permiten:

* Control de acceso por usuario
* Playlists personalizadas
* Favoritos individuales
* Estadísticas por usuario y globales
* Reproducción continua y registro histórico

**Justificación del diseño:**

La base de datos se ha diseñado para:

* Minimizar redundancia (normalización).
* Facilitar consultas complejas (JOINs eficientes).
* Permitir escalabilidad (más canciones, más usuarios, más playlists).
* Cumplir todos los requisitos del enunciado.
* Integrarse de forma natural con el backend PHP.

## **3.3 Estructura del Proyecto**

La estructura del proyecto Musicfy se ha organizado siguiendo criterios de modularidad, claridad y separación de responsabilidades. Cada carpeta agrupa archivos con funciones relacionadas, facilitando el mantenimiento, la escalabilidad y la comprensión global del sistema. Esta organización permite que el frontend, el backend y la base de datos evolucionen de forma independiente sin generar dependencias innecesarias.

La estructura general del proyecto es la siguiente:

Musicfy/

├── Backend/

│ └── PHP/

│ ├── db.php

│ ├── index.php

│ ├── buscar.php

│ ├── busqueda.php

│ ├── get\_canciones.php

│ ├── get\_playlist.php

│ ├── get\_playlists.php

│ ├── crear\_playlist.php

│ ├── playlist.php

│ ├── like.php

│ ├── increment\_reproducciones.php

│ ├── importar\_jameando.php

│ └── (futuros archivos: login.php, register.php, admin\_panel.php…)

│

├── Frontend/

│ ├── CSS/

│ │ └── style.css

│ │

│ ├── JS/

│ │ ├── player.js

│ │ ├── audio\_line.js

│ │ ├── volume\_line.js

│ │ ├── like.js

│ │ ├── canciones\_crear\_playlist.js

│ │ └── (futuros scripts: login.js, admin.js…)

│ │

│ ├── img/

│ │ ├── logo\_app.png

│ │ ├── icons/

│ │ ├── playlists/

│ │ └── (otras imágenes)

│ │

│ └── HTML/

│ ├── pagina\_inicio.html

│ ├── estadisticas.html

│ ├── crear\_playlist.html

│ └── (futuras vistas: login.html, registro.html, favoritos.html…)

│

├── Database/

│ ├── schema.sql

│ ├── inserts.sql

│ └── er\_diagram.png

│

├── Docs/

│ ├── Documentación.pdf

│ ├── MUSIC\_LICENSES.md

│ └── README.md

│

└── README.md

**Descripción de cada módulo:**

Backend/ 🡪 Contiene todos los archivos de la parte Backend, permitiendo la comunicación de la BD con el Frontend

PHP/ 🡪 Contiene toda la lógica del servidor.

Frontend/ 🡪 Incluye toda la información visible para el usuario.

HTML/ 🡪 Contiene la estructura base de las páginas WEB.

CSS/ 🡪 Contiene las hojas de estilos principales, responsables del diseño responsive, animaciones, layout y estética general.

JS/ 🡪 Incluye todos los scripts de control para la gestión de información y acciones pertinentes.

Img/ 🡪 Incluye las imágenes usadas en los HTML. Dentro de esta, están organizadas por carpetas según su uso.

Database/ 🡪 Contiene el script principal de inicialización y creación de la BD con sus correspondientes tablas.

Docs/ 🡪 Incluye toda la información referida a la Documentación, asi como Memoria Final, mapa BD, …

Esta estructuración del contenido se ha realizado para mantener una clara separación entre el Backend, Frontend y BD, permitiendo trabajar en cada una de las partes del proyecto en paralelo y facilitando asi la colaboración en el proyecto. Además, permite claridad y organización, permitiendo futuras ampliaciones del proyecto.

# **Capítulo 4. Desarrollo**

## **4.1 Frontend**

**Pagina\_inicio.html:**

Este archivo HTML corresponde a la **página principal (Home)** de la aplicación web **Musicfy**. Su función es presentar la plataforma al usuario, explicar brevemente su propósito y ofrecer accesos directos al **registro** y al **inicio de sesión**.

El documento sigue el estándar **HTML5** y está dividido en las secciones principales:  
<head> y <body>.

<!DOCTYPE html>

<html lang="es">

<!DOCTYPE html> indica que se utiliza HTML5.

lang="es" especifica que el contenido está en español, mejorando accesibilidad y SEO.

<head> Contiene información **no visible** para el usuario, pero esencial para el funcionamiento de la página.

<meta charset="UTF-8" />

Permite el uso correcto de caracteres especiales (tildes, ñ, símbolos).

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

Hace que la página sea **responsive**, adaptándose a móviles, tablets y pantallas grandes.

<title>Musicfy</title>

Define el nombre que aparece en la pestaña del navegador.

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

Enlaza la hoja de estilos externa que define el diseño visual de la página.

<link rel="icon" type="image/x-icon" href="../img/logo\_app.png">

Define el icono que se muestra en la pestaña del navegador.

<body> Contiene todo el contenido visible de la página.

<header>

<nav class="navbar">

<img class="logo\_img" src="../img/logo\_app.png" alt="">

<div class="logo">Musicfy</div>

</nav>

</header>

Muestra la **barra superior** de la aplicación.

Incluye:

* El **logo gráfico** de la aplicación.
* El **nombre “Musicfy”**, reforzando la identidad visual.

Se apoya en clases CSS (navbar, logo\_img, logo) para el diseño.

<main class="home-container">

Es el contenedor principal de la página de inicio.

<h1 class="home-title">Bienvenido a Musicfy</h1>

<p class="home-subtitle">Tu plataforma moderna para descubrir, organizar y disfrutar de la música</p>

Presenta la aplicación al usuario.

Transmiten el objetivo y el enfoque del proyecto.

<div class="home-content">

Incluye dos párrafos que explican:

* La filosofía de Musicfy.
* La importancia de la música como experiencia.
* El enfoque moderno, intuitivo y visual de la plataforma.

Se utiliza <b> para destacar el nombre de la aplicación.

<div class="home-features">

Contiene dos tarjetas interactivas que actúan como accesos principales.

<a href="registro.html">

<div class="feature-card">

<h3>🎧 Descubre Música</h3>

<p>...</p>

</div>

</a>

Enlace a la página de registro.

Invita a nuevos usuarios a comenzar a usar la plataforma.

<a href="login.html">

<div class="feature-card">

<h3>👤 Inicia Sesion</h3>

<p>...</p>

</div>

</a>

Enlace a la página de inicio de sesión.

Dirigido a usuarios ya registrados.

<footer>

<p>© 2025-2026 PlayListApp - Proyecto académico</p>

</footer>

Muestra información legal y académica.

**Player.js:**

El archivo **player.js** se encarga de gestionar toda la **lógica del reproductor de música** de la aplicación Musicfy. Controla la reproducción de canciones, la navegación entre pistas, la actualización de la interfaz del reproductor y el registro de reproducciones en el backend.

const audioPlayer = document.querySelector('.audio-player');

const playerCover = document.querySelector('.player-cover');

const playerTitle = document.querySelector('.player-info h4');

const playerArtist = document.querySelector('.player-info p');

const playBtn = document.querySelector('.player-play');

const playIcon = document.querySelector('.player-play-icon');

const nextBtn = document.querySelector('.player-next');

const prevBtn = document.querySelector('.player-prev');

Estas líneas seleccionan los elementos HTML del reproductor:

* Contenedor del reproductor
* Imagen de la portada
* Título y artista de la canción
* Botones de play/pausa, siguiente y anterior

Se usan clases CSS para mantener una separación clara entre estructura (HTML) y comportamiento (JavaScript).

const playerAudio = new Audio();

Crea un objeto Audio nativo de JavaScript.

Permite reproducir archivos de audio sin necesidad de un <audio> visible en el HTML.

let currentIndex = -1;

Guarda la posición de la canción actual dentro de la lista songList.

El valor -1 indica que no hay ninguna canción cargada inicialmente.

document.querySelectorAll('.song-card-large').forEach((card, index) => {

card.addEventListener('click', () => {

...

});

});

Detecta clics sobre las tarjetas de canciones.

Cada tarjeta corresponde a una canción y tiene asociado un índice.

Al hacer clic:

* Se actualiza currentIndex
* Se obtienen los datos de la canción:
  + Portada
  + Título
  + Artista
  + URL del audio (mediante data-audio)
* Se actualiza la interfaz del reproductor
* Se carga y reproduce la canción automáticamente

playerAudio.addEventListener('canplay', () => {

playerAudio.play();

audioPlayer.classList.add('active');

playIcon.src = "pause\_circle\_icon.png";

}, { once: true });

Se usa el evento canplay para garantizar que el audio esté listo.

{ once: true } evita múltiples ejecuciones del evento.

playBtn.addEventListener('click', () => {

if (playerAudio.paused) {

playerAudio.play();

...

} else {

playerAudio.pause();

...

}

});

Alterna entre reproducir y pausar la canción actual.

Cambia dinámicamente el icono del botón según el estado del reproductor.

playerAudio.addEventListener('ended', () => {

...

});

Al terminar una canción:

1. Se incrementa el contador de reproducciones en el backend (solo cuando la canción se escucha completa).
2. Se avanza automáticamente a la siguiente canción usando reproducción circular.

currentIndex = (currentIndex + 1) % songList.length;

Permite volver a la primera canción al llegar al final de la lista.

nextBtn.addEventListener('click', () => {

...

});

Avanza a la siguiente canción.

No realiza ninguna acción si no hay canción activa.

prevBtn.addEventListener('click', () => {

...

});

Retrocede a la canción anterior.

Usa aritmética modular para evitar índices inválidos.

function loadSong(index) {

...

}

Esta función centraliza la lógica de carga de canciones:

Funciones principales:

* Obtiene la canción desde songList
* Actualiza portada, título y artista
* Carga el audio correspondiente
* Inicia la reproducción automáticamente
* playerArtist.textContent = song.artista + " • " + (song.album ?? "Sin álbum");
* Usa el operador nullish coalescing (??) para mostrar un valor por defecto si no hay álbum.

IMPORTANTE:

Esta función no incrementa el contador de reproducciones, ya que eso solo ocurre cuando la canción termina.

fetch("../../Backend/PHP/increment\_reproducciones.php", {

method: "POST",

headers: { "Content-Type": "application/x-www-form-urlencoded" },

body: "id=" + encodeURIComponent(songList[currentIndex].id)

});

Envía una petición POST al servidor

Incrementa el número de reproducciones de la canción

Se ejecuta únicamente al finalizar la reproducción completa

**Audio\_line.js:**

El archivo **audio\_line.js** se encarga de gestionar la **barra de progreso del reproductor de audio**, mostrando el tiempo actual de reproducción, la duración total de la canción y permitiendo al usuario avanzar o retroceder manualmente dentro de la pista.

Este archivo complementa la lógica principal del reproductor definida en [player.js](#player_js).

const progressBar = document.getElementById('progress-bar');

const currentTimeLabel = document.getElementById('current-time');

const totalTimeLabel = document.getElementById('total-time');

Se seleccionan los elementos HTML encargados de:

* Mostrar la barra de progreso
* Indicar el tiempo actual de reproducción
* Indicar la duración total de la canción

Estos elementos permiten una interacción visual clara con el estado del audio.

playerAudio.addEventListener('loadedmetadata', () => {

...

});

El evento “loadedmetadata” se dispara cuando el navegador ha cargado la información del archivo de audio.

Se obtiene la duración total de la canción:

const duration = playerAudio.duration;

Se configura el valor máximo de la barra de progreso:

progressBar.max = Math.floor(duration);

Se muestra la duración total formateada en minutos y segundos:

totalTimeLabel.textContent = formatTime(duration);

playerAudio.addEventListener('timeupdate', () => {

...

});

Este evento se ejecuta de forma continua mientras el audio se reproduce.

Acciones realizadas:

* Se actualiza la posición de la barra de progreso según el tiempo actual
* Se muestra el tiempo transcurrido en formato legible

progressBar.value = Math.floor(playerAudio.currentTime);

currentTimeLabel.textContent = formatTime(playerAudio.currentTime);

Esto permite al usuario conocer en todo momento el avance de la canción.

progressBar.addEventListener('input', () => {

playerAudio.currentTime = progressBar.value;

});

Permite que el usuario avance o retroceda la canción moviendo la barra.

El valor seleccionado se asigna directamente al tiempo de reproducción del audio.

function formatTime(seconds) {

...

}

Convierte un valor numérico en segundos a un formato más legible mm:ss.

Funcionamiento:

* Calcula los minutos
* Calcula los segundos
* Añade un cero delante cuando los segundos son menores a 10

Este archivo depende del objeto global playerAudio, definido en [player.js](#player_js).

Permite mantener una separación clara entre:

* Lógica del reproductor (player.js)
* Interfaz de tiempo y progreso (audio\_line.js)

**volume\_line.js:**

El archivo **volume\_line.js** gestiona el **control de volumen del reproductor de música** de la aplicación Musicfy. Permite al usuario ajustar el nivel de sonido de forma interactiva mediante una barra deslizante (slider).

Este módulo trabaja en conjunto con [player.js](#player_js), utilizando el objeto de audio compartido playerAudio.

const volumeBar = document.getElementById('volume-bar');

Se selecciona el elemento HTML que representa la barra de volumen.

volumeBar.addEventListener('input', () => {

...

});

El evento input se dispara cada vez que el usuario mueve el control deslizante.

Esto permite una respuesta inmediata y fluida.

const volumeValue = volumeBar.value / 100;

El valor del slider va de 0 a 100.

El objeto Audio de JavaScript requiere un valor entre 0.0 y 1.0.

Se realiza la conversión dividiendo entre 100.

playerAudio.volume = volumeValue;

Asigna el nuevo nivel de volumen al reproductor.

El cambio se aplica de forma instantánea sobre la canción en reproducción.

volume\_line.js depende del objeto global playerAudio.

Mantiene la lógica de volumen separada del resto de controles:

* Reproducción (player.js)
* Progreso (audio\_line.js)
* Volumen (volume\_line.js)

**like.js:**

El archivo **like.js** gestiona la funcionalidad de **marcar y desmarcar canciones como favoritas** dentro de la aplicación Musicfy. Permite al usuario interactuar con un botón de “like” y sincroniza el estado visual con la base de datos mediante comunicación con el backend.

document.querySelectorAll('.like-bt').forEach(btn => {

Se seleccionan todos los botones que permiten marcar una canción como favorita.

Cada botón corresponde a una canción diferente y contiene información asociada mediante atributos data-\*.

btn.addEventListener('click', () => {

...

});

Se añade un evento click a cada botón.

Permite alternar el estado de “favorito” de la canción.

const img = btn.querySelector('img');

const songId = btn.dataset.song;

const isLiked = img.src.includes("like-red.png");

Se obtiene la imagen del botón para cambiar su icono.

Se extrae el identificador de la canción (songId) desde el atributo data-song.

Se determina si la canción ya está marcada como favorita comprobando la imagen actual.

img.src = isLiked

? "../../Frontend/img/icons/like.png"

: "../../Frontend/img/icons/like-red.png";

Cambia dinámicamente el icono del botón:

* rojo → canción favorita
* normal → no favorita

Mejorando la experiencia del usuario con feedback visual inmediato.

const action = isLiked ? "remove" : "add";

Define la acción a realizar en la base de datos:

* "add" → añadir a favoritos
* "remove" → eliminar de favoritos

fetch("../../Backend/PHP/like.php", {

method: "POST",

headers: { "Content-Type": "application/x-www-form-urlencoded" },

body: `song\_id=${songId}&action=${action}`

});

Se envía una petición POST al servidor.

Se pasan como parámetros:

* ID de la canción
* Acción a realizar (añadir o eliminar)

.then(res => res.text())

.then(msg => console.log("Favorito:", msg))

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

Se procesa la respuesta del servidor.

Se muestran mensajes en consola para depuración.

Se capturan posibles errores de comunicación.

Este archivo se integra con:

* El sistema de usuarios (sesión activa)
* La base de datos de canciones y favoritos

Mantiene la lógica de favoritos separada del reproductor de audio.

**Crear\_playlist.html:**

Este archivo HTML implementa la **interfaz para crear una nueva playlist** en la plataforma **Musicfy**. Permite al usuario ingresar la información de la playlist, subir una portada, definir su visibilidad y seleccionar canciones de su biblioteca.

<head>

Define el charset UTF-8 y meta viewport para responsive design.

Título de la página: Musicfy - Crear Playlist.

Incluye:

* Hoja de estilos style.css para toda la aplicación.
* Icono de la app en la pestaña del navegador.

<body>

Contiene los bloques principales:

* Navbar
* Sidebar
* Contenido principal (main)
* Footer

Scripts al final del body:

* Función para mostrar/ocultar el sidebar en móviles
* Archivo canciones\_crear\_playlist.js que gestiona la carga dinámica de canciones.

Navbar:

Contiene:

* Botón menú móvil (menu-toggle) para mostrar/ocultar el sidebar.
* Logo de la app con enlace al inicio.
* Nombre de la app animado con estilo CSS.
* Formulario de búsqueda para canciones, artistas o álbumes.
* Botón de login para iniciar sesión.

Funcionalidad:

* El formulario envía la búsqueda al backend (busqueda.php).
* El botón de menú activa la clase active en el sidebar para móviles.

Sidebar:

Menú fijo con enlaces a:

* Inicio
* Mi Biblioteca
* Favoritos
* Estadísticas

Íconos representativos para cada sección.

Main:

El contenido principal está dentro de un <section> con clase playlist-editor, que contiene el formulario formPlaylist.

Campos del formulario:

1. Título de la playlist

* Input tipo text
* Máximo 128 caracteres
* Obligatorio (required)

1. Descripción

* textarea con placeholder
* Máximo 1000 caracteres
* No obligatorio

1. Imagen de portada

* Input tipo file
* Acepta únicamente imágenes (image/\*)

1. Visibilidad

* Selector select con dos opciones:
  + Privada
  + Pública
* Obligatorio

1. Selección de canciones

* select múltiple (multiple) para elegir varias canciones de la biblioteca.
* Tamaño visible de 6 canciones.
* Nota indicando cómo seleccionar varias canciones (CTRL o SHIFT).

1. Botón de envío

* Guardar Playlist

<footer>

<p>© 2025-2026 PlayListApp - Proyecto académico</p>

</footer>

Refuerza la autoría y contexto académico del proyecto.

function toggleSidebar() {

const sidebar = document.querySelector('.sidebar');

sidebar.classList.toggle('active');

}

Permite abrir y cerrar el sidebar en dispositivos móviles.

canciones\_crear\_playlist.js

Script externo que carga dinámicamente las canciones disponibles en la selección múltiple.

Mejora la interacción del usuario y mantiene el contenido actualizado sin recargar la página.

**Canciones\_crear\_playlist.js:**

Este script **gestiona la interacción dinámica de la página de creación de playlists**, cargando las canciones desde la base de datos y enviando el formulario para crear la playlist sin recargar la página.

document.addEventListener("DOMContentLoaded", function () {

fetch("../../Backend/PHP/get\_canciones.php")

.then(response => response.json())

.then(data => { ... });

});

Se ejecuta cuando el DOM está completamente cargado.

Realiza un fetch al archivo PHP get\_canciones.php en el backend.

Se espera un JSON con la lista de canciones disponibles.

Proceso interno:

1. Verifica que la respuesta sea un array.
2. Selecciona el elemento <select> con id "canciones".
3. Por cada canción recibida:

* Crea un <option> con:
  + value = id de la canción
  + textContent = título de la canción
* Añade la opción al <select> para permitir selección múltiple.
* Permite que el usuario pueda elegir varias canciones al crear su playlist.

document.getElementById("formPlaylist").addEventListener("submit", function (e) { ... });

Captura el evento submit del formulario formPlaylist.

Evita la recarga de la página (e.preventDefault()).

Crea un objeto FormData con todos los campos del formulario:

* Título
* Descripción
* Imagen
* Visibilidad
* Canciones seleccionadas

Envío al backend:

Se hace un fetch POST a crear\_playlist.php.

Se envía formData directamente como cuerpo de la petición.

El servidor responde en JSON con un estado (status) y mensaje (message).

Manejo de la respuesta:

Si status === "success":

* Redirige al usuario a la biblioteca de playlists (library.php).

Si hay un error:

* Muestra un alert con el mensaje del servidor o un aviso genérico.

En caso de error en la conexión o ejecución del fetch:

* Consola registra el error y muestra un alert al usuario.

**Style.css:**

El archivo **style.css** define el **diseño visual completo** de la aplicación web **Musicfy**. Su objetivo es proporcionar una interfaz moderna, coherente y responsive, manteniendo una experiencia de usuario atractiva tanto en escritorio como en dispositivos móviles.

El diseño se basa en una estética **oscura**, con colores vibrantes, gradientes animados y una estructura modular.

body {

margin: 0;

font-family: 'Segoe UI', sans-serif;

background-color: #000000f0;

color: white;

}

Se elimina el margen por defecto del navegador.

Se define una tipografía moderna y legible.

Se establece un tema oscuro como base de la aplicación.

.navbar {

position: fixed;

top: 0;

...

}

La barra de navegación permanece fija en la parte superior.

Contiene el logo, barra de búsqueda y botones principales.

Se usa z-index para asegurar que esté siempre visible.

.logo {

background: linear-gradient(...);

animation: gradientMove 8s ease infinite;

}

El texto del logo utiliza un gradiente animado.

Reforzando la identidad visual de Musicfy.

La barra de búsqueda usa flex para adaptarse al espacio disponible.

Los filtros se muestran como botones redondeados, manteniendo coherencia visual.

.sidebar {

position: fixed;

...

}

Menú lateral fijo bajo la barra de navegación.

Permite navegación rápida por secciones.

Incluye scroll vertical si el contenido es extenso.

main {

margin-left: 220px;

padding: 2rem;

}

Se deja espacio para la sidebar.

El contenido principal tiene fondo semitransparente y padding amplio para mejor lectura.

Tarjetas de canciones:

.song-card y .song-card-large definen dos tipos de visualización:

* Compacta
* Detallada (lista principal)

Características:

* Uso de grid y flex para alinear elementos.
* Bordes redondeados y colores oscuros.
* Interacción mediante cursor: pointer.

.audio-player.active {

position: fixed;

bottom: 0;

...

}

El reproductor aparece fijado en la parte inferior.

Solo se muestra cuando hay una canción activa.

Incluye:

* Portada
* Información de la canción
* Controles
* Barra de progreso
* Control de volumen

.like-bt {

background-color: transparent;

border: none;

}

Botón sin fondo ni bordes.

Se integra visualmente en la lista de canciones.

Bloque específico para la edición y creación de playlists:

* Diseño con gradientes, sombras y animaciones.
* Campos de formulario estilizados.
* Botón de envío con efecto hover y transición suave.

.home-container { ... }

Contenedor principal con fondo animado.

Títulos con gradientes y animaciones.

Tarjetas de funcionalidades con efecto de escala al pasar el ratón.

Playlists y biblioteca:

Diseño modular con tarjetas de playlists.

Grid adaptable al tamaño de pantalla.

Botones de edición y eliminación claramente diferenciados por color.

Cerrar sesión:

Información del usuario visible en la navbar.

Botón de logout con color rojo para fácil identificación.

Login y registro:

Formularios centrados verticalmente.

Inputs oscuros y botones contrastados.

Incluye:

* Interruptor “Recordarme”
* Control de visibilidad de contraseña

@keyframes fadeIn { ... }

@keyframes gradientMove { ... }

fadeIn: animación de entrada suave.

gradientMove: animación de desplazamiento de gradientes.

Aporta dinamismo sin afectar al rendimiento.

@media (max-width: 768px) { ... }

Adaptaciones para móviles:

* Navbar más compacta
* Sidebar ocultable
* Reproductor reorganizado en columna
* Simplificación de tarjetas de canciones
* Mejor aprovechamiento del espacio vertical

## **4.2 Backend**

**db.php:**

El archivo **db.php** es el encargado de **establecer la conexión a la base de datos MySQL** utilizando **PDO (PHP Data Objects)**.  
Se utiliza como archivo central que puede ser **incluido en otros scripts PHP** para reutilizar la conexión de forma segura y consistente.

$host = "localhost";

$db = "playlists\_db";

$user = "root";

$pass = "";

* $host: Servidor donde se aloja la base de datos (local).
* $db: Nombre de la base de datos.
* $user: Usuario de MySQL.
* $pass: Contraseña del usuario.

Estas variables permiten cambiar fácilmente el entorno (local, producción, etc.) sin modificar el resto del código

$pdo = new PDO(

"mysql:host=$host;dbname=$db;charset=utf8",

$user,

$pass,

[PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION]

);

Se crea una instancia del objeto PDO.

Se especifica:

* Tipo de base de datos: mysql
* Host y base de datos
* Codificación utf8 para soportar caracteres especiales.

Se activa el modo:

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION

Esto hace que los errores SQL se lancen como excepciones, facilitando la depuración y el control de errores.

try {

...

} catch (PDOException $e) {

die("Error BD: " . $e->getMessage());

}

Si la conexión falla:

* Se captura la excepción PDOException.
* Se detiene la ejecución del script con un mensaje de error claro.

Evita que la aplicación continúe sin una conexión válida a la base de datos.

require\_once "db.php";

Esto permite acceder directamente al objeto $pdo para:

* Consultas SQL
* Inserciones
* Actualizaciones
* Eliminaciones

**index.php:**

El archivo **index.php** representa la **página principal de la aplicación Musicfy**.  
Su función es **mostrar contenido personalizado al usuario autenticado**, incluyendo playlists, canciones populares y un reproductor de audio integrado.

session\_start();

if (!isset($\_SESSION["user\_id"])) {

header("Location: ../../Frontend/HTML/LoginScreen.html");

exit;

}

Se inicia la sesión con session\_start().

Se comprueba si el usuario está autenticado mediante $\_SESSION["user\_id"].

Si no lo está:

* Se redirige al formulario de login.
* Se detiene la ejecución del script.

Esto protege la página principal y evita accesos no autorizados.

require\_once(\_\_DIR\_\_ . '/songs.php');

$data = require(\_\_DIR\_\_ . '/get\_playlists.php');

$playlist = $data['playlists'];

songs.php: obtiene el listado de canciones (Top Canciones).

get\_playlists.php: devuelve las playlists disponibles.

Se almacenan los resultados para su uso en la vista.

$mapa\_generos = [];

foreach ($playlist as $p) {

$nombre = strtolower(trim($p['nombre']));

$mapa\_generos[$nombre] = $p['id'];

}

Se crea un array asociativo:

genero => id\_playlist

Permite enlazar botones de géneros (Pop, Rock, Metal, etc.) con su playlist correspondiente.

Facilita una navegación rápida por estilos musicales.

$playlists = array\_slice($playlists, 0, 4);

Se muestran solo las 4 primeras playlists.

Mejora el rendimiento y la estética de la portada.

$stmtFav = $pdo->prepare("SELECT song\_id FROM favorites WHERE user\_id = ?");

$stmtFav->execute([$\_SESSION["user\_id"]]);

$favoritas = $stmtFav->fetchAll(PDO::FETCH\_COLUMN);

Consulta a la base de datos:

* Tabla favorites
* Filtrada por el usuario actual

Se obtiene un array con los IDs de canciones favoritas.

Se usa para:

* Mostrar el icono de favoritos activo o inactivo.

Navbar características principales:

* Logo de la aplicación.
* Barra de búsqueda.
* Acceso a login o información del usuario.
* Botón de cerrar sesión si el usuario está autenticado.

<?= htmlspecialchars($\_SESSION["usuario"]) ?>

Contiene:

* Protección contra XSS
* Información personalizada

Sidebar:

Incluye accesos a:

* Inicio
* Mi Biblioteca
* Favoritos
* Estadísticas

Está pensada para:

* Escritorio (siempre visible)
* Móvil (toggle con botón ☰)

<a href="playlist.php?id=<?= $mapa\_generos['pop'] ?? '#' ?>">

Cada botón redirige a una playlist específica según el género.

Uso de ?? '#' para evitar errores si no existe el género.

<?php foreach ($playlists as $p): ?>

Se recorren las playlists.

Se muestran:

* Imagen
* Nombre
* Descripción

Cada tarjeta es un enlace a playlist.php.

<?php foreach($canciones as $c): ?>

Cada canción muestra:

* Portada
* Título
* Reproducciones
* Artista y álbum
* Duración
* Botón de favorito

El icono del “like” cambia dinámicamente según si la canción es favorita del usuario.

<div class="audio-player">

Incluye:

* Portada de la canción
* Título y artista
* Controles:
  + Anterior
  + Play / Pause
  + Siguiente
* Barra de progreso
* Control de volumen

Se controla mediante JavaScript:

* [player.js](#player_js)
* [audio\_line.js](#audioline_js)
* [volume\_line.js](#volumenline_js)

<script>

const songList = <?= json\_encode($canciones, JSON\_UNESCAPED\_UNICODE) ?>;

</script>

Se pasa el array de canciones de PHP a JavaScript.

Permite:

* Reproducción dinámica
* Navegación entre canciones
* Control del reproductor

<script src="../../Frontend/JS/player.js"></script>

<script src="../../Frontend/JS/like.js"></script>

<script src="../../Frontend/JS/audio\_line.js"></script>

<script src="../../Frontend/JS/volume\_line.js"></script>

Cada uno se encarga de:

* Reproductor
* Favoritos
* Barra de progreso
* Volumen

**songs.php:**

El archivo **songs.php** es un **módulo de obtención de datos** encargado de recuperar desde la base de datos:

* Las **canciones más relevantes**
* Las **playlists más recientes**

Estos datos son utilizados principalmente en la **página principal ([index.php](#index_php))**.

require\_once \_\_DIR\_\_ . "/db.php";

Se incluye el archivo [db.php](#db_php).

Este archivo:

* Establece la conexión con MySQL usando PDO.
* Proporciona la variable $pdo, que se utiliza para ejecutar consultas.

require\_once evita inclusiones duplicadas.

$stmt = $pdo->query("SELECT \* FROM songs LIMIT 10");

$canciones = $stmt->fetchAll(PDO::FETCH\_ASSOC);

Selecciona 10 canciones de la tabla songs.

No aplica filtros, por lo que:

* Se muestran las primeras canciones según el orden natural de la tabla.

Los resultados se almacenan en $canciones.

PDO::FETCH\_ASSOC devuelve un array asociativo:

[

'id' => 1,

'titulo' => 'Nombre',

'artista' => 'Autor',

'audio\_url' => 'ruta.mp3',

...

]

$stmt = $pdo->query("SELECT \* FROM playlists ORDER BY fecha\_creacion DESC LIMIT 5");

$playlists = $stmt->fetchAll(PDO::FETCH\_ASSOC);

Funcionalidad:

* Recupera las 5 playlists más recientes.
* Ordena los resultados por:
  + fecha\_creacion DESC
  + (de la más nueva a la más antigua).

Uso:

* Se muestran en la sección “Top Playlists” de la página principal.
* Permite destacar contenido recién creado por los usuarios.

Este archivo no genera HTML, solo datos:

$canciones: Mostrar canciones en [index.php](#index_php)

$playlists: Mostrar playlists destacadas

**like.php:**

El archivo **like.php** gestiona la funcionalidad de **añadir y eliminar canciones favoritas** de un usuario.  
Se comunica con el frontend mediante **peticiones AJAX (fetch)** y actualiza la base de datos en tiempo real.

require\_once(\_\_DIR\_\_ . '/db.php');

Incluye el archivo [db.php](#db_php) para:

* Establecer la conexión con la base de datos mediante PDO.
* Acceder al objeto $pdo.

session\_start();

$user\_id = $\_SESSION['user\_id'] ?? null;

Se inicia la sesión para identificar al usuario autenticado.

Se obtiene el user\_id desde la sesión.

Si no existe sesión activa, el valor será null.

Esto garantiza que solo usuarios autenticados puedan marcar canciones como favoritas.

$song\_id = $\_POST['song\_id'] ?? null;

$action = $\_POST['action'] ?? null;

Datos recibidos vía POST:

song\_id: ID de la canción

action: Acción a realizar (add o remove)

Estos datos provienen del archivo [like.js](#like_js).

if (!$user\_id || !$song\_id || !$action) {

http\_response\_code(400);

echo "Datos incompletos";

exit;

}

Comprueba que:

* El usuario esté autenticado.
* Exista una canción seleccionada.
* Se indique una acción válida.

Si falta algún dato:

* Se devuelve HTTP 400 (Bad Request).
* Se cancela la ejecución.

if ($action === 'add') {

$stmt = $pdo->prepare(

"INSERT IGNORE INTO favorites (user\_id, song\_id) VALUES (?, ?)"

);

$stmt->execute([$user\_id, $song\_id]);

echo "Añadido";

}

Se usa INSERT IGNORE para:

* Evitar duplicados en la tabla favorites.

Relaciona:

* Usuario (user\_id)
* Canción (song\_id)

Devuelve el mensaje "Añadido" al frontend.

elseif ($action === 'remove') {

$stmt = $pdo->prepare(

"DELETE FROM favorites WHERE user\_id = ? AND song\_id = ?"

);

$stmt->execute([$user\_id, $song\_id]);

echo "Eliminado";

}

Elimina la relación usuario–canción.

Se ejecuta solo si el usuario había marcado previamente la canción.

Devuelve "Eliminado" como confirmación.

else {

http\_response\_code(400);

echo "Acción inválida";

}

Controla valores incorrectos en action.

Evita ejecuciones inesperadas o maliciosas.

**get\_all\_playlists.php:**

El archivo **get\_all\_playlists.php** se encarga de **recuperar todas las playlists almacenadas en la base de datos**, devolviendo la información en una estructura reutilizable para otros archivos PHP del proyecto.

require\_once(\_\_DIR\_\_ . '/db.php');

Incluye el archivo [db.php](#db_php).

Permite acceder al objeto $pdo para ejecutar consultas SQL.

require\_once evita que el archivo se cargue más de una vez.

$sql = "SELECT id, nombre, descripcion, imagen FROM playlists ORDER BY id DESC";

Selecciona los campos principales de la tabla playlists:

* id
* nombre
* descripción
* imagen

Ordena los resultados por id DESC:

* Las playlists más recientes aparecen primero.

Se excluyen campos innecesarios (como fechas o visibilidad) para optimizar el rendimiento.

$stmt = $pdo->query($sql);

Ejecuta la consulta directamente.

Se usa query() porque no hay parámetros dinámicos.

$playlists = $stmt->fetchAll(PDO::FETCH\_ASSOC);

Recupera todas las playlists como un array asociativo.

Cada playlist se representa como:

[

'id' => 1,

'nombre' => 'Rock Clásico',

'descripcion' => 'Grandes éxitos del rock',

'imagen' => 'rock.jpg'

]

return [

'playlists' => $playlists

];

Devuelve los datos como un array estructurado.

Permite incluir este archivo con:

$data = require 'get\_all\_playlists.php';

$playlists = $data['playlists'];

**get\_canciones.php:**

El archivo **get\_canciones.php** proporciona un **endpoint en formato JSON** que devuelve la lista de canciones disponibles en la base de datos. Su principal función es servir datos a peticiones AJAX realizadas desde el frontend.

Este archivo se utiliza, por ejemplo, al **crear una playlist**, donde se necesita mostrar un listado de canciones seleccionables.

require\_once \_\_DIR\_\_ . "/db.php";

Incluye el archivo [db.php](#db_php).

Permite acceder al objeto $pdo para ejecutar consultas SQL mediante PDO.

header("Content-Type: application/json");

Indica que la respuesta del servidor será en formato JSON.

Es obligatorio para que el navegador y JavaScript interpreten correctamente la respuesta.

$sql = "SELECT id, titulo FROM songs ORDER BY titulo ASC";

Se seleccionan únicamente los campos necesarios:

* id
* titulo

Las canciones se ordenan alfabéticamente por título (ASC).

Mejora la experiencia del usuario al mostrar una lista ordenada.

$stmt = $pdo->query($sql);

$canciones = $stmt->fetchAll(PDO::FETCH\_ASSOC);

Se ejecuta la consulta.

Se obtienen todas las canciones como un array asociativo. Ejemplo de salida:

[

{ "id": 1, "titulo": "Bohemian Rhapsody" },

{ "id": 2, "titulo": "Imagine" }

]

echo json\_encode($canciones);

Convierte el array PHP a JSON.

Se envía directamente al frontend.

catch (Exception $e) {

echo json\_encode([

"error" => "Error al obtener canciones",

"detalle" => $e->getMessage()

]);

}

Captura errores de conexión o consulta.

Devuelve un objeto JSON con:

* Un mensaje de error genérico.
* El detalle técnico del error (útil en desarrollo).

**Importar\_jameando.php:**

El archivo **importar\_jameando.php** se encarga de **importar canciones desde la API pública de Jamendo** y almacenarlas en la base de datos del proyecto **Musicfy**.  
Este script se utiliza como herramienta de **carga inicial o actualización de contenido musical**.

require "/db.php";

Incluye el archivo de conexión [db.php](#db_php).

Permite usar el objeto $pdo para ejecutar consultas SQL.

Es imprescindible para guardar los datos importados.

$API\_KEY = "4690e24f";

Clave pública necesaria para autenticar las peticiones a la API de Jamendo.

Identifica la aplicación cliente.

$url = "https://api.jamendo.com/v3.0/tracks?" . http\_build\_query([...]);

Parámetros utilizados:

* client\_id: clave de la API.
* format: respuesta en JSON.
* limit: máximo de 100 canciones.
* include: incluye información musical adicional.
* audioformat: formato MP3.
* license\_cc: canciones con licencias Creative Commons.

Se garantiza que las canciones pueden usarse legalmente en el proyecto.

$json = file\_get\_contents($url);

$data = json\_decode($json, true);

Se obtiene la respuesta de la API.

Se convierte el JSON en un array asociativo de PHP.

if (!isset($data["results"])) {

die("Error obteniendo canciones");

}

Comprueba que la API ha devuelto resultados válidos.

Detiene la ejecución si hay un error.

foreach ($data["results"] as $track) {

Itera sobre cada canción devuelta por la API.

Cada $track representa una canción individual.

$check = $pdo->prepare("SELECT id FROM songs WHERE api\_id = ?");

$check->execute([$track["id"]]);

Comprueba si la canción ya existe en la base de datos.

Se utiliza api\_id como identificador único externo.

if ($check->rowCount() > 0) {

continue;

}

Si la canción ya existe, se omite y se pasa a la siguiente.

$duracion = gmdate("i:s", $track["duration"]);

Convierte la duración de segundos a formato mm:ss.

Facilita su visualización en el frontend.

$stmt = $pdo->prepare("

INSERT INTO songs

(api\_id, titulo, artista, album, duracion, portada, audio\_url, genero, licencia)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

");

Campos insertados:

* api\_id: ID original de Jamendo.
* titulo: nombre de la canción.
* artista: autor o grupo.
* album: nombre del álbum.
* duracion: duración formateada.
* portada: imagen del álbum.
* audio\_url: enlace al archivo de audio.
* genero: género musical.
* licencia: tipo de licencia Creative Commons.

echo "Canciones importadas correctamente";

Confirma que el proceso se ha completado con éxito.

**favoritos.php:**

El archivo **favoritos.php** se encarga de **mostrar al usuario autenticado la lista de canciones que ha marcado como favoritas** dentro de la aplicación **Musicfy**.  
Combina lógica **PHP**, consultas a **MySQL con PDO**, y estructura **HTML** para presentar la información de forma dinámica.

session\_start();

require\_once(\_\_DIR\_\_ . "/db.php");

session\_start() permite acceder a los datos del usuario autenticado.

Se incluye [db.php](#db_php) para establecer la conexión con la base de datos mediante $pdo.

$stmtFav = $pdo->prepare("SELECT song\_id FROM favorites WHERE user\_id = ?");

$stmtFav->execute([$\_SESSION["user\_id"]]);

$fav\_ids = $stmtFav->fetchAll(PDO::FETCH\_COLUMN);

Se consultan los IDs de canciones que el usuario ha marcado como favoritas.

Se usa una consulta preparada para evitar inyecciones SQL.

El resultado es un array simple con los IDs de las canciones.

if (empty($fav\_ids)) {

$canciones\_fav = [];

}

Si el usuario no tiene canciones favoritas:

* Se evita ejecutar una consulta IN () inválida.
* Se inicializa el array vacío para evitar errores posteriores.

$in = implode(",", array\_fill(0, count($fav\_ids), "?"));

$stmtSongs = $pdo->prepare("SELECT \* FROM songs WHERE id IN ($in)");

$stmtSongs->execute($fav\_ids);

$canciones\_fav = $stmtSongs->fetchAll(PDO::FETCH\_ASSOC);

Se construye dinámicamente la cláusula IN (?).

Se recupera toda la información de cada canción:

* título
* artista
* álbum
* duración
* portada
* audio

Se almacenan en un array asociativo.

Navbar Muestra:

* Logo de Musicfy
* Barra de búsqueda
* Nombre del usuario conectado
* Botón de cierre de sesión

<?= htmlspecialchars($\_SESSION["usuario"]) ?>

Se utiliza htmlspecialchars() para prevenir XSS.

Sidebarincluye accesos a:

* Inicio
* Mi Biblioteca
* Favoritos
* Estadísticas

<?php if (empty($canciones\_fav)): ?>

<p>No tienes canciones favoritas todavía.</p>

<?php else: ?>

Si no hay canciones favoritas, se muestra un mensaje informativo.

Si existen, se recorren con un foreach.

Cada canción se muestra con:

* Portada del álbum
* Título
* Artista y álbum
* Duración
* Botón de favorito (corazón rojo)

<div class="song-card-large" data-audio="<?= htmlspecialchars($c['audio\_url']) ?>">

* data-audio se usa para reproducir la canción con el reproductor global.

$iconoLike = "like-red.png";

Todas las canciones en esta vista ya son favoritas.

El icono se muestra siempre en rojo.

Reproductor de audio incluye:

* Portada
* Información de la canción
* Controles:
  + Anterior
  + Play / Pause
  + Siguiente
* Barra de progreso
* Control de volumen

Se apoya en los siguientes scripts:

<script src="../../Frontend/JS/player.js"></script>

<script src="../../Frontend/JS/like.js"></script>

<script src="../../Frontend/JS/audio\_line.js"></script>

<script src="../../Frontend/JS/volume\_line.js"></script>

**Increment\_reproducciones.php:**

Este archivo se encarga de **incrementar en 1 el contador de reproducciones** de una canción cuando el usuario la reproduce en la aplicación Musicfy. Se comunica típicamente con el **reproductor de audio en el frontend** mediante **AJAX/Fetch**.

require\_once(\_\_DIR\_\_ . '/db.php');

Incluye el archivo [db.php](#db_php) para usar la variable $pdo.

$pdo permite ejecutar consultas seguras usando PDO.

$songId = $\_POST['id'] ?? null;

Se recibe mediante POST el ID de la canción que se ha reproducido.

Si no se recibe, se asigna null.

if (!$songId) {

http\_response\_code(400);

echo "ID de canción no especificado.";

exit;

}

Si $songId es null o vacío:

* Se devuelve un código HTTP 400 (Bad Request)
* Se muestra un mensaje de error.

exit; detiene la ejecución para evitar que se haga la consulta.

$stmt = $pdo->prepare("UPDATE songs SET reproducciones = reproducciones + 1 WHERE id = ?");

$stmt->execute([$songId]);

Consulta preparada para evitar inyecciones SQL.

Incrementa el campo reproducciones de la canción específica en 1.

$songId se pasa como parámetro seguro a la consulta.

echo "OK";

Devuelve un mensaje simple de confirmación.

Esto permite que el frontend sepa que la operación fue exitosa y no requiere recarga de página.

**Get\_playlists.php:**

Este archivo PHP se encarga de **obtener todas las playlists de la base de datos** y devolverlas en un formato que pueda ser utilizado directamente por otros scripts PHP, como [index.php](#index_php).

require\_once(\_\_DIR\_\_ . '/db.php');

Incluye el archivo [db.php](#db_php).

Permite usar la variable $pdo para consultas seguras con PDO.

try {

...

} catch (PDOException $e) {

die("Error al obtener playlists: " . $e->getMessage());

}

Se usa try-catch para capturar errores en la consulta a la base de datos.

Si ocurre un error, el script se detiene con die() mostrando el mensaje de error.

Esto evita que la aplicación falle silenciosamente.

$stmt = $pdo->query("SELECT id, nombre, descripcion, imagen FROM playlists ORDER BY nombre ASC");

$playlists = $stmt->fetchAll(PDO::FETCH\_ASSOC);

SELECT id, nombre, descripcion, imagen FROM playlists → selecciona los campos necesarios.

ORDER BY nombre ASC → ordena alfabéticamente por nombre de la playlist.

fetchAll(PDO::FETCH\_ASSOC) → devuelve un array asociativo, fácil de usar en PHP.

return [

"playlists" => $playlists

];

Devuelve un array asociativo con la clave "playlists".

Permite a otros scripts hacer:

$data = require('get\_playlists.php');

$playlists = $data['playlists'];

Si ocurre un **PDOException**, se detiene el script y se muestra un mensaje claro:

die("Error al obtener playlists: " . $e->getMessage());

Esto asegura que no se intente usar datos inexistentes.

**get\_user\_playlists.php:**

Este archivo PHP se encarga de **obtener todas las playlists creadas por un usuario específico**, usando su user\_id de la sesión.

require\_once \_\_DIR\_\_ . "/db.php";

Incluye el archivo [db.php](#db_php).

Permite usar la variable $pdo para consultas PDO.

$userId = $\_SESSION["user\_id"] ?? null;

Se intenta obtener user\_id de la sesión.

Si no existe, $userId será null.

if (!$userId) {

return ["playlists" => []];

}

Si no hay usuario logueado, devuelve un array vacío.

Esto evita errores al consultar la base de datos sin un usuario válido.

$sql = "SELECT \* FROM playlists WHERE user\_id = :uid";

$stmt = $pdo->prepare($sql);

$stmt->execute(["uid" => $userId]);

:uid → parámetro nombrado para evitar inyección SQL.

prepare + execute → forma segura de pasar valores a la consulta.

Se seleccionan todas las columnas (\*) de la tabla playlists del usuario.

return [

"playlists" => $stmt->fetchAll(PDO::FETCH\_ASSOC)

];

Devuelve un array asociativo con clave "playlists".

Cada elemento del array es una playlist completa (id, nombre, descripción, imagen, fecha, etc.).

Esto permite a otros scripts hacer:

$data = require('get\_user\_playlists.php');

$userPlaylists = $data['playlists'];

**logout.php:**

Este archivo se encarga de **cerrar la sesión del usuario** y devolverlo a la página de inicio de la aplicación.

session\_start();

Inicia o reanuda la sesión activa.

Es obligatorio para poder manipular las variables de sesión existentes.

session\_unset();

Borra todas las variables almacenadas en $\_SESSION.

El usuario deja de estar autenticado a nivel lógico.

session\_destroy();

Elimina completamente la sesión del servidor.

Invalida el identificador de sesión (cookie).

Garantiza que no quede rastro de la sesión activa.

header("Location: ../../Frontend/HTML/pagina\_inicio.html");

exit;

Redirige al usuario a la página de inicio pública.

exit asegura que no se ejecute ningún código adicional tras la redirección.

**buscar.php:**

Este archivo se encarga de **buscar canciones y playlists** en la base de datos según un término proporcionado por el usuario a través de la barra de búsqueda.

require\_once \_\_DIR\_\_ . "/db.php";

Importa la conexión PDO establecida en [db.php](#db_php).

Permite realizar consultas a la base de datos de manera segura.

$termino = $\_GET['q'] ?? '';

$termino = trim($termino);

Obtiene el parámetro q enviado vía GET desde el formulario de búsqueda.

trim() elimina espacios al inicio y fin para evitar coincidencias incorrectas.

Si no se recibe ningún término, se considera vacío ''.

$canciones = [];

$playlists = [];

Se crean arrays vacíos para almacenar los resultados de canciones y playlists.

Esto evita errores si no se encuentra ningún resultado.

if ($termino !== '') {

// Buscar canciones

$sqlCanciones = "SELECT \* FROM songs WHERE titulo LIKE :term OR artista LIKE :term OR album LIKE :term";

$stmt = $pdo->prepare($sqlCanciones);

$stmt->execute(['term' => "%$termino%"]);

$canciones = $stmt->fetchAll(PDO::FETCH\_ASSOC);

// Buscar playlists

$sqlPlaylists = "SELECT \* FROM playlists WHERE nombre LIKE :term OR descripcion LIKE :term";

$stmt = $pdo->prepare($sqlPlaylists);

$stmt->execute(['term' => "%$termino%"]);

$playlists = $stmt->fetchAll(PDO::FETCH\_ASSOC);

}

Condición: Solo se ejecuta si $termino no está vacío.

Canciones:

* Consulta todos los campos relevantes (titulo, artista, album) usando LIKE con comodines %.
* :term es un placeholder seguro para evitar inyección SQL.
* Se almacena el resultado en $canciones.

Playlists:

* Consulta campos nombre y descripcion usando el mismo principio.
* Resultados almacenados en $playlists.

**playlist.php:**

Este archivo muestra la **información de una playlist específica** y las canciones que contiene, además de integrar el **reproductor de música** y los controles de favoritos.

session\_start();

if (!isset($\_SESSION["user\_id"])) {

header("Location: ../../Frontend/HTML/login.html");

exit;

}

Se inicia la sesión.

Se verifica si el usuario está logueado; si no, se redirige al login.

$data = require(\_\_DIR\_\_ . '/get\_playlist.php');

$playlist = $data['playlist'];

$canciones = $data['canciones'];

Incluye get\_playlist.php que obtiene:

* Información de la playlist ($playlist)
* Todas las canciones de la playlist ($canciones)

$stmtFav = $pdo->prepare("SELECT song\_id FROM favorites WHERE user\_id = ?");

$stmtFav->execute([$\_SESSION["user\_id"]]);

$favoritas = $stmtFav->fetchAll(PDO::FETCH\_COLUMN);

Se obtienen los IDs de las canciones que el usuario ha marcado como favoritas.

Esto permite mostrar el icono de corazón rojo en las canciones ya favoritas.

Navbar:

Logo, buscador, botones de login/logout y menú lateral.

Dinámico: muestra nombre de usuario si está logueado.

Sidebar:

Enlaces a:

* Inicio
* Mi Biblioteca
* Favoritos
* Estadísticas

main:

<section class="playlist-visual">

<img src="../../Frontend/img/playlists/<?= htmlspecialchars($playlist['imagen']) ?>" alt="">

<h1><?= htmlspecialchars($playlist['nombre']) ?></h1>

<p><?= htmlspecialchars($playlist['descripcion']) ?></p>

<p><strong>Canciones:</strong> <?= $data['total\_canciones'] ?></p>

<p><strong>Duración total:</strong> <?= $data['duracion\_total'] ?></p>

<p><strong>Última modificación:</strong> <?= $data['ultima\_modificacion'] ?></p>

Muestra portada, nombre, descripción, cantidad de canciones, duración total y última modificación.

Si viene de la biblioteca del usuario (?from=library), permite editar o eliminar la playlist.

Lista de canciones:

<?php foreach($canciones as $c): ?>

<div class="song-card-large" data-audio="<?= htmlspecialchars($c['audio\_url']) ?>" data-id="<?= htmlspecialchars($c['id']) ?>">

<img src="<?= htmlspecialchars($c['portada']) ?>" alt="">

<h3><?= htmlspecialchars($c['titulo']) ?></h3>

<p>Reproducciones: <?= htmlspecialchars($c['reproducciones'] ?? 0) ?></p>

<p><?= htmlspecialchars($c['artista']) ?> • <?= htmlspecialchars($c['album'] ?? 'Sin álbum') ?></p>

<span><?= htmlspecialchars($c['duracion'] ?? '0:00') ?></span>

<button class="like-bt" data-song="<?= $c['id'] ?>">

<img src="../../Frontend/img/icons/<?= in\_array($c['id'], $favoritas) ? 'like-red.png' : 'like.png' ?>" alt="">

</button>

</div>

<?php endforeach; ?>

Muestra todas las canciones de la playlist con:

* Portada
* Título
* Artista y álbum
* Duración
* Botón de “me gusta” según si es favorita

Reproductor:

Sección fija al final con:

* Portada del tema
* Título y artista
* Controles (play, pause, siguiente, anterior)
* Barra de progreso y control de volumen

<script>

const songList = <?= json\_encode($canciones, JSON\_UNESCAPED\_UNICODE) ?>;

</script>

<script src="../../Frontend/JS/player.js"></script>

<script src="../../Frontend/JS/like.js"></script>

<script src="../../Frontend/JS/audio\_line.js"></script>

<script src="../../Frontend/JS/volume\_line.js"></script>

songList: lista de canciones disponible para el reproductor.

Scripts controlan:

* Reproducción
* Likes
* Barra de progreso
* Volumen

**Crear\_playlist.php:**

Este script **recibe los datos de un formulario** para crear una nueva playlist y asociarle canciones. Devuelve respuesta **JSON** para ser consumida por JavaScript.

session\_start();

require\_once \_\_DIR\_\_ . "/db.php";

if (!isset($\_SESSION["user\_id"])) {

echo json\_encode(["status" => "error", "message" => "Debes iniciar sesión."]);

exit;

}

$user\_id = $\_SESSION["user\_id"];

$rol = $\_SESSION["rol"] ?? "user";

Inicia la sesión.

Verifica que el usuario esté logueado.

Obtiene el ID del usuario y su rol (por ejemplo admin o user).

$nombre = $\_POST['nombre'] ?? '';

$descripcion = $\_POST['descripcion'] ?? '';

$visibilidad = $\_POST['visibilidad'] ?? 'privada';

$canciones = $\_POST['canciones'] ?? [];

Obtiene los datos enviados por el formulario:

* Nombre de la playlist
* Descripción
* Visibilidad (privada o publica)
* Canciones seleccionadas (array de IDs)

if ($visibilidad === "publica" && $rol !== "admin") {

echo json\_encode([

"status" => "error",

"message" => "Solo los administradores pueden crear playlists públicas."

]);

exit;

}

Solo los administradores pueden crear playlists públicas.

Los usuarios normales quedan restringidos a privada.

$imagen = null;

if (!empty($\_FILES['imagen']['name'])) {

$imagen = basename($\_FILES['imagen']['name']);

$destinoCarpeta = \_\_DIR\_\_ . "/../../Frontend/img/playlists/";

if (!is\_dir($destinoCarpeta)) {

mkdir($destinoCarpeta, 0777, true);

}

$rutaDestino = $destinoCarpeta . $imagen;

if (!move\_uploaded\_file($\_FILES['imagen']['tmp\_name'], $rutaDestino)) {

$imagen = null;

}

}

Si el usuario sube una imagen, se guarda en la carpeta Frontend/img/playlists/.

Si falla la subida, se deja como null.

Se asegura que la carpeta exista (mkdir).

$sql = "INSERT INTO playlists (user\_id, nombre, descripcion, imagen, visibilidad)

VALUES (?, ?, ?, ?, ?)";

$stmt = $pdo->prepare($sql);

$stmt->execute([$user\_id, $nombre, $descripcion, $imagen, $visibilidad]);

$playlist\_id = $pdo->lastInsertId();

Se inserta la nueva playlist en la tabla playlists.

Se obtiene el id generado automáticamente (lastInsertId) para asociar canciones.

if (!empty($canciones)) {

$sql\_cancion = "INSERT INTO playlist\_songs (playlist\_id, song\_id) VALUES (?, ?)";

$stmt\_cancion = $pdo->prepare($sql\_cancion);

foreach ($canciones as $song\_id) {

$stmt\_cancion->execute([$playlist\_id, $song\_id]);

}

}

Si el usuario seleccionó canciones, se insertan en la tabla playlist\_songs.

Cada fila conecta una playlist con una canción (playlist\_id + song\_id).

echo json\_encode(["status" => "success"]);

exit;

Devuelve un JSON indicando éxito.

**eliminar\_playlist.php:**

Este script **elimina una playlist completa** y todas las relaciones con canciones, y luego redirige al usuario a su biblioteca.

require\_once \_\_DIR\_\_ . "/db.php";

Importa [db.php](#db_php) para poder usar $pdo y acceder a la base de datos.

$id = $\_GET['id'] ?? null;

if (!$id) {

die("ID inválido");

}

Obtiene el id de la playlist desde la URL (?id=XX).

Si no se proporciona, termina el script mostrando un mensaje de error.

$sql = "DELETE FROM playlist\_songs WHERE playlist\_id = ?";

$stmt = $pdo->prepare($sql);

$stmt->execute([$id]);

La tabla playlist\_songs almacena la relación entre playlists y canciones.

Antes de eliminar la playlist, borramos todas sus relaciones para evitar errores de integridad referencial.

$sql = "DELETE FROM playlists WHERE id = ?";

$stmt = $pdo->prepare($sql);

$stmt->execute([$id]);

Borra la playlist de la tabla playlists.

header("Location: library.php");

exit;

Una vez eliminada la playlist y sus canciones asociadas, redirige al usuario a su biblioteca.

**library.php:**

Este archivo muestra al usuario **todas las playlists que ha creado** y permite acceder a cada playlist para reproducirla o editarla.

session\_start();

if (!isset($\_SESSION["user\_id"])) {

header("Location: ../../Frontend/HTML/login.html");

exit;

}

Inicia la sesión PHP.

Verifica que el usuario esté logueado.

Si no hay sesión activa, redirige al login.

$data = require(\_\_DIR\_\_ . '/get\_user\_playlists.php');

$playlists = $data['playlists'];

Incluye [get\_user\_playlists.php](#get_user_playlists_php), que devuelve un array con todas las playlists del usuario logueado.

$playlists contendrá la información de cada playlist: id, nombre, descripción, imagen, etc.

<header>

<nav class="navbar">

<button class="menu-toggle" onclick="toggleSidebar()">☰</button>

<a href="index.php" class="logo-link">

<img class="logo\_img" src="../../Frontend/img/logo\_app.png" alt="">

</a>

<div class="logo">Musicfy</div>

<form action="busqueda.php" method="get" class="bloque-busqueda">

<input type="text" name="q" placeholder="Buscar canciones, artistas, álbumes..." class="search-bar" />

<button type="submit" class="search-btn">

<img src="../../Frontend/img/icons/search\_icon.png" alt="icono de lupa de búsqueda">

</button>

</form>

<?php if (isset($\_SESSION["usuario"])): ?>

<div class="user-info">

<span class="username">👤 <?= htmlspecialchars($\_SESSION["usuario"]) ?></span>

<a href="logout.php" class="logout-btn">Cerrar sesión</a>

</div>

<?php else: ?>

<a href="../../Frontend/HTML/login.html" class="login-btn">Iniciar sesión</a>

<?php endif; ?>

</nav>

</header>

Muestra la barra de navegación, buscador y estado del usuario.

Si el usuario está logueado, aparece su nombre y el botón de logout.

<aside class="sidebar">

<ul>

<li><a href="index.php"><img src="../../Frontend/img/icons/home\_icon.png"> Inicio</a></li>

<li><a href="library.php"><img src="../../Frontend/img/icons/library\_music\_icon.png"> Mi Biblioteca</a></li>

<li><a href="favoritos.php"><img src="../../Frontend/img/icons/favorite\_icon.png"> Favoritos</a></li>

<li><a href="../../Frontend/HTML/estadisticas.html"><img src="../../Frontend/img/icons/analytics\_icon.png"> Estadísticas</a></li>

</ul>

</aside>

Menú lateral con enlaces rápidos a secciones importantes de la app.

<main>

<section class="quick-picks">

<h2>Mis Playlists <a href="crear\_playlist\_html.php" class="btn-edit">Crear Playlist</a></h2>

<div class="playlist-list">

<?php foreach ($playlists as $p): ?>

<a href="playlist.php?id=<?= $p['id'] ?>&from=library" class="song-card">

<img src="../../Frontend/img/playlists/<?= htmlspecialchars($p['imagen']) ?>" alt="Portada playlist" />

<div class="song-info">

<h3><?= htmlspecialchars($p['nombre']) ?></h3>

<p><?= htmlspecialchars($p['descripcion']) ?></p>

</div>

</a>

<?php endforeach; ?>

</div>

</section>

</main>

Muestra todas las playlists del usuario.

Cada playlist se muestra como tarjeta con portada, nombre y descripción.

El enlace apunta a playlist.php?id=XX&from=library para poder editar/eliminar si es necesario.

Botón para crear nueva playlist.

<div class="audio-player">

<img src="../../Frontend/img/logo\_app.png" alt="Portada" class="player-cover">

<div class="player-info">

<h4></h4>

<p></p>

</div>

<div class="player-controls">

<button class="player-prev"><img src="../../Frontend/img/icons/previous\_song\_icon.png"></button>

<button class="player-play"><img class="player-play-icon" src="../../Frontend/img/icons/play\_circle\_icon.png"></button>

<button class="player-next"><img src="../../Frontend/img/icons/next\_song\_icon.png"></button>

</div>

<div class="player-progress">

<span id="current-time">0:00</span>

<input type="range" id="progress-bar" min="0" max="100" value="0">

<span id="total-time">0:00</span>

<div class="player-volume">

<input type="range" id="volume-bar" min="0" max="100" value="80">

</div>

</div>

</div>

Controles de reproducción globales.

Se conecta al script [player.js](#player_js) para manejar reproducción, avance y volumen.

<script>

function toggleSidebar() {

const sidebar = document.querySelector('.sidebar');

sidebar.classList.toggle('active');

}

</script>

<script src="../../Frontend/JS/player.js"></script>

<script src="../../Frontend/JS/like.js"></script>

<script src="../../Frontend/JS/audio\_line.js"></script>

<script src="../../Frontend/JS/volume\_line.js"></script>

toggleSidebar() abre/cierra el menú lateral.

Scripts gestionan reproducción, like de canciones y barras de progreso/volumen.

**Editar\_playlist.php:**

Este archivo permite **editar una playlist existente**: cambiar su nombre, descripción, imagen, visibilidad y modificar las canciones asociadas.

require\_once \_\_DIR\_\_ . "/db.php";

$id = $\_GET['id'] ?? null;

if (!$id) {

die("Playlist no encontrada");

}

$sql = "SELECT \* FROM playlists WHERE id = ?";

$stmt = $pdo->prepare($sql);

$stmt->execute([$id]);

$playlist = $stmt->fetch(PDO::FETCH\_ASSOC);

if (!$playlist) {

die("Playlist no encontrada");

}

Se requiere la conexión a la base de datos.

Se obtiene la playlist según el id pasado por GET.

Si no existe, se detiene la ejecución.

$sql = "SELECT s.id, s.titulo, s.artista

FROM songs s

INNER JOIN playlist\_songs ps ON ps.song\_id = s.id

WHERE ps.playlist\_id = ?";

$stmt = $pdo->prepare($sql);

$stmt->execute([$id]);

$canciones\_vinculadas = $stmt->fetchAll(PDO::FETCH\_ASSOC);

$sql = "SELECT id, titulo, artista FROM songs ORDER BY titulo";

$stmt = $pdo->query($sql);

$todas\_canciones = $stmt->fetchAll(PDO::FETCH\_ASSOC);

$ids\_vinculados = array\_column($canciones\_vinculadas, 'id');

canciones\_vinculadas: canciones ya agregadas a la playlist.

todas\_canciones: lista completa de canciones de la base de datos.

ids\_vinculados: array de IDs de canciones ya incluidas, usado para no duplicarlas al añadir nuevas.

El formulario permite:

* Cambiar nombre y descripción de la playlist.
* Mostrar la imagen actual y subir una nueva.
* Cambiar la visibilidad (privada o pública).
* Eliminar canciones mediante checkboxes.
* Agregar canciones mediante un <select> múltiple.

<div class="bloques">

<label>Título de la Playlist:</label>

<input type="text" name="nombre" value="<?= htmlspecialchars($playlist['nombre']) ?>" required>

</div>

Se usan htmlspecialchars() para evitar inyecciones XSS.

Todos los campos están prellenados con los datos actuales.

<div class="bloques">

<label>Canciones en la Playlist: (Seleccionar para eliminar)</label>

<?php foreach ($canciones\_vinculadas as $c): ?>

<label>

<input type="checkbox" name="quitar\_canciones[]" value="<?= $c['id'] ?>">

<?= htmlspecialchars($c['titulo']) ?> - <?= htmlspecialchars($c['artista']) ?>

</label><br>

<?php endforeach; ?>

</div>

Permite seleccionar canciones para quitar de la playlist.

Se enviarán como quitar\_canciones[] a update\_playlist.php.

<div class="bloques">

<label>Añadir canciones:</label>

<select name="agregar\_canciones[]" multiple size="6">

<?php foreach ($todas\_canciones as $c): ?>

<?php if (!in\_array($c['id'], $ids\_vinculados)): ?>

<option value="<?= $c['id'] ?>"><?= htmlspecialchars($c['titulo']) ?> - <?= htmlspecialchars($c['artista']) ?></option>

<?php endif; ?>

<?php endforeach; ?>

</select>

</div>

Solo se muestran las canciones no incluidas en la playlist.

Se permite seleccionar varias canciones manteniendo CTRL o SHIFT.

<button type="submit">Guardar Cambios</button>

Envía los cambios a update\_playlist.php mediante POST.

Navbar: botón de menú, buscador y logo.

Sidebar: enlaces a Inicio, Mi Biblioteca, Favoritos y Estadísticas.

<script>

function toggleSidebar() {

document.querySelector('.sidebar').classList.toggle('active');

}

</script>

Controla la apertura/cierre del menú lateral.

Funcionamiento completo:

1. Usuario accede a editar\_playlist.php?id=XX.
2. Se carga la playlist y sus canciones.
3. Formulario prellenado con datos actuales.
4. El usuario puede:

* Cambiar nombre, descripción, imagen.
* Modificar visibilidad.
* Eliminar o agregar canciones.

1. Al enviar, los datos van a update\_playlist.php para actualizar la base de datos.

**Get\_playlist.php:**

Este archivo **no imprime HTML**, sino que **devuelve un array de datos** que luego usa [playlist.php](#playlist_php).  
Se encarga de:

* Obtener la playlist por ID
* Cargar sus canciones
* Autocompletar playlists “por género”
* Calcular estadísticas (duración total, nº de canciones)

$playlistId = $\_GET['id'] ?? null;

if (!$playlistId) {

die("Playlist no especificada.");

}

Se espera el id por GET

Si no existe → error inmediato

$stmt = $pdo->prepare("SELECT \* FROM playlists WHERE id = ?");

$stmt->execute([$playlistId]);

$playlist = $stmt->fetch(PDO::FETCH\_ASSOC);

if (!$playlist) {

die("Playlist no encontrada.");

}

Se recupera la playlist desde la base de datos

Si el ID no existe → error

$nombre = strtolower(trim($playlist['nombre']));

$generos\_validos = [ "metal", "pop", "rock", "classical", "choral", "otros" ];

Si el nombre de la playlist coincide con un género, la playlist se comporta como dinámica.

Ejemplos:

* Playlist llamada “Rock”
* Playlist llamada “Pop”
* Playlist llamada “Otros”

$stmt = $pdo->prepare("

SELECT id, titulo, artista, album, portada, duracion, audio\_url

FROM songs WHERE genero = ?

");

$stmt->execute([$genero\_buscar]);

$canciones = $stmt->fetchAll(PDO::FETCH\_ASSOC);

Se buscan canciones por género

No depende de playlist\_songs

$stmt = $pdo->prepare("SELECT song\_id FROM playlist\_songs WHERE playlist\_id = ?");

$stmt->execute([$playlistId]);

$ya\_vinculadas = $stmt->fetchAll(PDO::FETCH\_COLUMN);

Obtiene canciones ya asociadas

Evita insertar la misma canción dos veces

foreach ($canciones as $c) {

if (!in\_array($c['id'], $ya\_vinculadas)) {

$sqlInsert->execute([$playlistId, $c['id']]);

}

}

Resultado:

* La playlist se autoactualiza
* Si mañana hay más canciones Rock → aparecen solas

else {

$stmt = $pdo->prepare("

SELECT s.\*

FROM songs s

JOIN playlist\_songs ps ON s.id = ps.song\_id

WHERE ps.playlist\_id = ?

");

$stmt->execute([$playlistId]);

$canciones = $stmt->fetchAll(PDO::FETCH\_ASSOC);

}

Carga solo canciones añadidas manualmente

Usa la tabla intermedia playlist\_songs

foreach ($canciones as $c) {

if (!empty($c['duracion']) && strpos($c['duracion'], ":") !== false) {

list($min, $sec) = explode(":", $c['duracion']);

$total\_segundos += ((int)$min \* 60) + (int)$sec;

}

}

Convierte "mm:ss" → segundos

Suma todas las canciones

**update\_playlist.php:**

Este script procesa el formulario de **editar playlist** y se encarga de:

* Actualizar datos básicos (nombre, descripción, visibilidad)
* Cambiar la imagen (si se sube una nueva)
* Quitar canciones existentes
* Añadir canciones nuevas
* Redirigir a la vista de la playlist

$id = $\_POST['id'] ?? null;

$nombre = $\_POST['nombre'] ?? '';

$descripcion = $\_POST['descripcion'] ?? '';

$visibilidad = $\_POST['visibilidad'] ?? 'privada';

$quitar = $\_POST['quitar\_canciones'] ?? [];

$agregar = $\_POST['agregar\_canciones'] ?? [];

Datos principales de la playlist

Arrays de canciones:

* quitar\_canciones[]
* agregar\_canciones[]

if (!$id) {

die("ID inválido");

}

Evita ejecutar el script sin un ID válido.

if (!empty($\_FILES['imagen']['name'])) {

1. Obtiene el nombre del archivo
2. Define carpeta destino
3. La crea si no existe
4. Mueve el archivo subido

$destinoCarpeta = \_\_DIR\_\_ . "/../../Frontend/img/playlists/";

Si falla la subida, simplemente no cambia la imagen.

DELETE FROM playlist\_songs

WHERE playlist\_id = ? AND song\_id = ?

Se ejecuta una vez por canción marcada

Solo elimina las seleccionadas

INSERT INTO playlist\_songs (playlist\_id, song\_id) VALUES (?, ?)

Se añaden las nuevas canciones seleccionadas

No se comprueba duplicado aquí (pero el selector ya los filtra)

**busqueda.php:**

Este archivo se encarga de:

* Ejecutar la búsqueda (delegándola a [buscar.php](#buscar_php))
* Mostrar **canciones** y **playlists** relacionadas
* Mantener el sistema de favoritos
* Integrarse con el reproductor global

session\_start();

require\_once(\_\_DIR\_\_ . '/buscar.php');

Inicia la sesión (necesaria para favoritos y usuario)

Incluye [buscar.php](#buscar_php), que:

* Lee $\_GET['q']
* Ejecuta las consultas SQL
* Define estas variables:
  + $termino
  + $canciones
  + $playlists

$stmtFav = $pdo->prepare("SELECT song\_id FROM favorites WHERE user\_id = ?");

$stmtFav->execute([$\_SESSION["user\_id"]]);

$favoritas = $stmtFav->fetchAll(PDO::FETCH\_COLUMN);

Recupera los IDs de canciones marcadas como favoritas.

Permite pintar el corazón rojo o blanco.

header incluye:

* Navbar con buscador
* Sidebar con navegación
* Información del usuario logueado
* Logout

<h2>Resultados de búsqueda para "<?= htmlspecialchars($termino) ?>"</h2>

Muestra el término buscado

Protegido con htmlspecialchars → evita XSS

<?php foreach($canciones as $c): ?>

Cada canción muestra:

* Portada
* Título
* Artista y álbum
* Duración
* Botón de favorito
* data-audio para el reproductor

$esFavorita = in\_array($c['id'], $favoritas);

Decide si usar like.png o like-red.png

Funciona con [like.js](#like_js) vía AJAX

<?php foreach ($playlists as $p): ?>

Cada playlist:

* Es un enlace a playlist.php?id=...
* Muestra imagen, nombre y descripción
* Mantiene estilos consistentes

Reproductor incluye:

* Controles de reproducción
* Barra de progreso
* Control de volumen
* Portada y metadatos

Se conecta con:

<script>

const songList = <?= json\_encode($canciones) ?>;

</script>

Permite reproducir **solo las canciones del resultado de búsqueda**  
Reutiliza el mismo [player.js](#player_js)

<script src="../../Frontend/JS/player.js"></script>

<script src="../../Frontend/JS/like.js"></script>

<script src="../../Frontend/JS/audio\_line.js"></script>

<script src="../../Frontend/JS/volume\_line.js"></script>

[player.js](#player_js) → reproducción

[like.js](#like_js) → favoritos AJAX

[audio\_line.js](#audioline_js) → progreso

[volume\_line.js](#volumenline_js) → volumen

**crear\_playlist\_html.php:**

Este archivo es **la interfaz visual** que permite al usuario **crear una nueva playlist**, mientras que toda la lógica real (BD, validaciones, subida de imagen) vive en [crear\_playlist.php](#crear_playlist_php).

session\_start();

Necesario para:

* Mostrar el nombre del usuario
* Permitir creación de playlists

Navbar incluye:

* Logo
* Buscador
* Info del usuario
* Sidebar de navegación

<form id="formPlaylist" enctype="multipart/form-data">

Se envía por AJAX con JS

Permite subir archivos (multipart/form-data)

Nombre:

<input type="text" name="nombre" maxlength="128" required>

Obligatorio

Descripción:

<textarea name="descripcion" maxlength="1000"></textarea>

Opcional

Imagen:  
<input type="file" name="imagen" accept="image/\*">

Restringe a imágenes

Procesada luego en [crear\_playlist.php](#crear_playlist_php)

Visibilidad:

<select name="visibilidad">

<option value="privada">Privada</option>

<option value="publica">Pública</option>

</select>

El rol se valida en backend

Usuario normal → solo privada

Admin → ambas

Canciones:

<select id="canciones" name="canciones[]" multiple>

Se llena dinámicamente con JS

Permite selección múltiple (CTRL / SHIFT)

Conectado a la BD vía [get\_canciones.php](#getcanciones_php)

<script src="../../Frontend/JS/canciones\_crear\_playlist.js"></script>

Este script:

1. Carga canciones desde la BD (fetch)
2. Llena el <select>
3. Envía el formulario por AJAX a:

Backend/PHP/crear\_playlist.php

1. Redirige a [library.php](#library_php) si todo va bien

# **Capítulo 5. Instrucciones de Ejecución**

## **5.1 Instalación**

Jsncjdbcn

Ckmdcskm

## **5.2 Inicialización de la Base de Datos**

Para inicializar la BBDD lo que se deberá hacer es lo siguiente:

* 1. Deberemos tener instalado en nuestro sistema XAMPP Control Panel, en caso de no tenerlo instalado vaya a [instalaciones necesarias](#_5.1_Instalación).
  2. Una vez tenemos XAMPP Control Panel accedemos a él y iniciamos tanto Apache como MySQL. Deberá mostrarse algo tal que así:

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* 1. Una vez realizado esto pasaremos al navegador (no importa el navegador que usemos) y pondremos en la barra de navegación <http://localhost/phpmyadmin/>. (en la imagen aparecen BBDD que ya están creadas pero puede ser que no les aparezca ninguna si es la primera vez que acceden)

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

* 1. Ahora pasaremos a generar la BBDD. Para ello hay dos formas de hacerlo. La primera será importar directamente la BBDD ya que en caso de no estar creada la BBDD el archivo la creará automáticamente. Para ello le daremos a importar y elegiremos el archivo de init\_playlists\_db.sql.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

* 1. Una vez hecho esto bajaremos abajo del todo y le daremos a importar.

Patrón de fondo, Tabla

El contenido generado por IA puede ser incorrecto.

* 1. En caso de que haya funcionado nos debería aparecer una BBDD en el lado izquierdo que se llame playlists\_db y que contenga las tablas favorites, history, playlist\_song, playlists, songs, users. En caso de que no te aparezca, no te preocupes porque hay otra manera de hacerlo. Esta otra manera consiste en crear la BBDD manual y todo lo demás con el archivo. Para ello pinchamos en nueva, ponemos el nombre “playlists\_db” (IMPORTANTE, si no se pone bien el nombre de la BBDD fallará) y le damos a crear.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

* 1. Ahora accedemos a la BBDD que hemos creado y dentro de esta le damos a importar y seguimos los mismos pasos de antes (le damos a elegir archivo, seleccionamos el archivo init\_playlists\_db.sql y le damos a importar).

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

* 1. Ahora sí, si se han seguido bien los pasos nos aparecerán una BBDD en el lado izquierdo que se llame playlists\_db y que contenga las tablas favorites, history, playlist\_song, playlists, songs, users.

Aplicación

El contenido generado por IA puede ser incorrecto.

## **5.3 Ejecución del Proyecto**

Jsncjdbcn

Ckmdcskm

# **Capítulo 6. Conclusión**

El desarrollo de Musicfy ha permitido integrar de forma coherente todos los elementos fundamentales de una aplicación web moderna: una interfaz responsive y dinámica, un backend modular en PHP, una base de datos bien estructurada y un reproductor de audio completamente funcional.

A lo largo del proyecto se han aplicado principios clave de arquitectura cliente-servidor, manipulación del DOM, diseño centrado en el usuario y gestión eficiente de datos, cumpliendo con los objetivos planteados en el enunciado.

El sistema implementa funcionalidades esenciales como la gestión de playlists, la reproducción continua de canciones, la búsqueda avanzada, el incremento automático de reproducciones.

Musicfy demuestra una comprensión sólida de los conceptos fundamentales del desarrollo web sin frameworks, así como una correcta separación de responsabilidades entre frontend, backend y base de datos. El resultado es una plataforma funcional, extensible y preparada para evolucionar hacia un sistema completo de gestión musical.

Este proyecto no solo cumple los requisitos académicos, sino que sienta las bases para futuras mejoras y ampliaciones, consolidando una experiencia de usuario fluida y una arquitectura técnica robusta.

# **Capítulo 7. Anexos**

## **7.1 Tablas BD**

USERS:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

SONGS:

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

PLAYLIST\_SONGS:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

PLAYLISTS:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

HYSTORY:

Una captura de pantalla de una computadora

El contenido generado por IA puede ser incorrecto.

FAVORITES:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

MAPA DE BD:

Diagrama

El contenido generado por IA puede ser incorrecto.

## **7.2 Información Usada**

Para el uso de HTML:

W3Schools: <https://www.w3schools.com/html/default.asp>

Temario dado:



Para el uso de CSS:

W3Schools: <https://www.w3schools.com/css/default.asp>

Temario dado:



Animaciones: <https://amitmerchant.com/animated-gradient-effect-in-css/>

Para el uso de JS:

W3Schools: <https://www.w3schools.com/js/default.asp>

Temario dado:



Para el uso de MySQL:

W3Schools: <https://www.w3schools.com/sql/default.asp>

Para el uso de PHP:

W3Schools: <https://www.w3schools.com/php/default.asp>

Temario dado:

