

A large red circle containing the words "MUSIC MUSE" in white, bold, sans-serif font.

MUSIC
MUSE

Índice

1. Introducción

- Contexto y Justificación
- Objetivos del Proyecto
- Estructura del proyecto

2. Marco Teórico

- Plataformas de Streaming Musical
- Gestión de la Música Digital
- Importancia de la Autonomía para los Artistas

3. Metodología

- Enfoque del Proyecto
- Herramientas y Tecnologías Utilizadas
- Estrategia de Desarrollo

4. Desarrollo del Proyecto

- Diseño del Sistema
- Implementación
 - Backend
 - Frontend
- Pruebas
 - Pruebas Funcionales
 - Pruebas de Usabilidad
 - Pruebas de Seguridad

5. Resultados

- Funcionalidades Implementadas
- Evaluación de la Plataforma
- Retroalimentación de los Usuarios

6. Discusión

- Comparación con Otras Plataformas
- Desafíos Encontrados
- Implicaciones del Proyecto

7. Conclusiones y Trabajo Futuro

- Resumen de Logros
- Recomendaciones para Futuras Mejoras
- Potencial Impacto en la Industria

8. Referencias Bibliográficas

1. INTRODUCCION

En la era digital, la industria de la música ha experimentado una transformación radical, impulsada por el auge de plataformas de streaming como Spotify. Estas plataformas han facilitado el acceso a una vasta cantidad de música para los usuarios y han ofrecido a los artistas una vía para distribuir y monetizar sus obras. Sin embargo, presentan ciertos obstáculos para los músicos emergentes, especialmente en términos de accesibilidad y control sobre la distribución de su música. En este contexto, surge la necesidad de una solución que permita a los artistas gestionar su música de manera más autónoma y directa, sin depender de intermediarios.

MusicMuse es una aplicación web innovadora diseñada para llenar este vacío, ofreciendo una plataforma que combina las funcionalidades de streaming de música con la capacidad de que los artistas puedan crear y gestionar sus perfiles directamente. A diferencia de Spotify, que requiere intermediarios como distribuidoras digitales para que los artistas puedan subir su música, MusicMuse elimina esta barrera, permitiendo a los músicos emergentes subir sus canciones de manera independiente y gestionar su presencia en la plataforma.

El objetivo principal de MusicMuse es facilitar la comunicación y difusión entre artistas emergentes, creando un espacio donde puedan compartir su música y conectar con su audiencia sin las complejidades y costos asociados a las plataformas tradicionales. La aplicación no solo permite a los usuarios escuchar música, sino que también fomenta la interacción y el descubrimiento de nuevos talentos en la industria musical.

Este trabajo de fin de grado se centra en el desarrollo de MusicMuse, desde su concepción inicial hasta su implementación y pruebas finales. A lo largo del documento, se detallará el contexto y la motivación detrás del proyecto, el marco teórico que sustenta su desarrollo, la metodología utilizada y los resultados obtenidos. Además, se discutirá la relevancia de MusicMuse en el panorama actual de la música digital y se propondrán áreas de mejora y desarrollo futuro.

2. MARCO TEÓRICO

En el marco teórico, se realizará una revisión de la literatura relevante sobre plataformas de streaming y gestión de música digital, destacando las ventajas y limitaciones de las soluciones existentes. La metodología abarcará los enfoques y herramientas tecnológicas empleadas, así como las técnicas de desarrollo aplicadas. El desarrollo del proyecto se centrará en el proceso de diseño, implementación y pruebas, detallando las decisiones técnicas y los desafíos enfrentados. Los resultados mostrarán las funcionalidades logradas y las evaluaciones de rendimiento, mientras que la discusión ofrecerá un análisis crítico de estos resultados en relación con los objetivos planteados.

Finalmente, las conclusiones resumirán los principales hallazgos y logros del proyecto, destacando las contribuciones de MusicMuse a la industria musical y señalando posibles direcciones para futuras investigaciones y mejoras en la aplicación.

Esta introducción establece el marco para el desarrollo de MusicMuse, subrayando su relevancia y potencial impacto en el ámbito de la música digital. A medida que avancemos en el documento, se proporcionarán detalles exhaustivos sobre cada aspecto del proyecto, ofreciendo una visión completa y rigurosa del desarrollo de esta innovadora aplicación web.

3. METOLOGÍA

La metodología empleada en el desarrollo de MusicMuse se basó en una combinación de técnicas tradicionales y digitales de gestión de proyectos, las cuales permitieron una planificación estructurada y un seguimiento efectivo del progreso. A continuación, se detallan las herramientas y enfoques utilizados:

Gestión de Tareas con Tarjetas de Cartón

Desde el inicio del proyecto, se utilizó un enfoque físico y visual para la gestión de tareas mediante el uso de tarjetas de cartón. Cada tarjeta representaba una tarea o subtarea específica dentro de la estructura global del proyecto. Este método proporcionó una forma tangible y clara de visualizar el trabajo pendiente y completado.

- Estructura y Organización: Las tarjetas de cartón fueron organizadas en un tablero dividido en columnas que representaban las distintas fases del proyecto, como "Pendiente", "En Progreso" y "Completado". Cada tarjeta incluía una descripción detallada de la tarea, los pasos necesarios para completarla y cualquier recurso adicional necesario.
- Seguimiento del Progreso: A medida que se completaban las tareas, se añadían ticks a las tarjetas correspondientes. Esto no solo facilitó el seguimiento del progreso, sino que también proporcionó una sensación de logro y motivación continua. La visibilidad del avance en el tablero permitió identificar rápidamente las áreas que requerían atención y ajustar las prioridades según fuera necesario.

Herramientas Digitales: Notion

Además de las tarjetas de cartón, se utilizó Notion, una herramienta digital de gestión de proyectos, para complementar la organización y planificación del proyecto. Notion ofreció una plataforma flexible y accesible para la coordinación de tareas y la documentación del progreso.

- Organización Dinámica: En Notion, se creó una base de datos estructurada con todas las tareas y objetivos del proyecto. Cada tarea incluía detalles como la descripción, la

fecha de inicio y finalización prevista, y cualquier comentario o recurso adicional relevante.

- Objetivos y Fechas Clave: Se establecieron objetivos claros y fechas límite para cada tarea, lo cual facilitó la gestión del tiempo y aseguró que el proyecto se mantuviera dentro del cronograma planificado. La capacidad de Notion para integrar calendarios y recordatorios ayudó a mantener el equipo enfocado y alineado con los plazos establecidos.

- Colaboración y Documentación: Notion también se utilizó para documentar las decisiones de diseño, las dificultades encontradas y las soluciones implementadas durante el desarrollo del proyecto. Esta documentación fue esencial para mantener un registro detallado del proceso y facilitar la revisión y reflexión posterior.

Enfoque Iterativo y Flexibilidad

El enfoque metodológico adoptado fue iterativo, permitiendo revisiones y mejoras continuas a lo largo del desarrollo de MusicMuse. Este método se caracterizó por su flexibilidad, adaptándose a los cambios y nuevas necesidades que surgían durante el proyecto.

- Revisiones Periódicas: Se realizaron revisiones periódicas para evaluar el progreso y ajustar las prioridades. Estas revisiones aseguraron que el proyecto se mantuviera alineado con los objetivos generales y permitieron identificar y abordar problemas de manera oportuna.

- Adaptación a Cambios: La combinación de métodos físicos y digitales proporcionó una estructura robusta pero adaptable, permitiendo gestionar eficazmente tanto las tareas planificadas como los desafíos imprevistos.

En resumen, la metodología empleada en MusicMuse combinó técnicas tradicionales de gestión de proyectos con herramientas digitales modernas, proporcionando un enfoque estructurado y flexible. Esta combinación facilitó una organización efectiva, un seguimiento detallado del progreso y una adaptación ágil a los cambios, asegurando el éxito del proyecto en todas sus fases.

4. DESARROLLO DEL PROYECTO:

1.1 Requisitos del Proyecto

- **Requisitos Funcionales:**

- El **usuario no registrado** puede:
 - Visualizar y escuchar a los artistas de la semana (publicación y actualización semanal).
- El **usuario fan** además puede:
 - Escuchar y visualizar a todos los artistas con publicaciones en la aplicación.
 - Usar herramienta de búsqueda
- El **usuario artista** tiene los mismos privilegios que el fan y además puede:
 - Publicar su música.
- El **usuario administrador** puede:
 - Eliminar a cualquier usuario de la base de datos.
 - Editar a los artistas de la semana.

- **Requisitos No Funcionales:**

- El sistema debe ser seguro y proteger los datos de los usuarios.
- La aplicación debe ser rápida y responder en menos de 2 segundos para cualquier acción.
- Debe ser compatible con los navegadores más comunes (Chrome, Firefox, Safari, Edge).
- El tiempo de aprendizaje debe ser inferior a una hora.

1.2 Análisis de Usuarios

- **Identificación de Usuarios:**

- **Usuario no registrado:** Visitantes que pueden explorar contenido limitado.
- **Usuario fan:** Usuarios registrados que pueden acceder a todo el contenido disponible.
- **Usuario artista:** Usuarios registrados que pueden acceder al contenido y publicar su propia música.
- **Usuario administrador:** Usuarios con privilegios especiales para gestionar usuarios y contenido de la App.

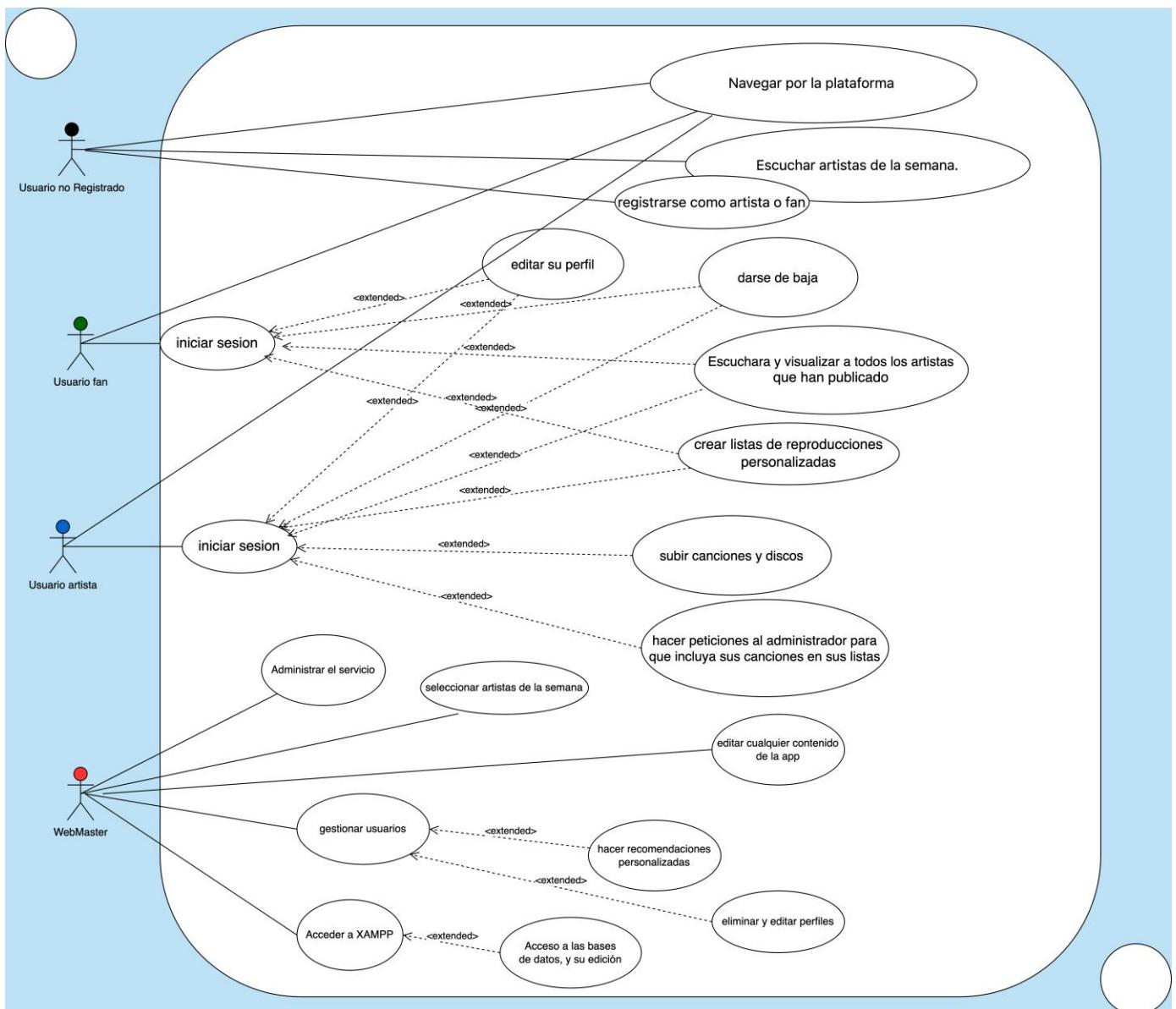
- **Necesidades y Expectativas:**

- **Usuario no registrado:** Quiere una forma rápida y sencilla de explorar la música de los artistas de la semana.

- **Usuario fan:** Busca acceso completo a toda la música publicada en la aplicación y conocer artistas emergentes.
- **Usuario artista:** Necesita una plataforma para publicar y promocionar su música y conocer otros artistas similares a su música.
- **Usuario administrador:** Requiere herramientas eficientes para gestionar usuarios y contenido.

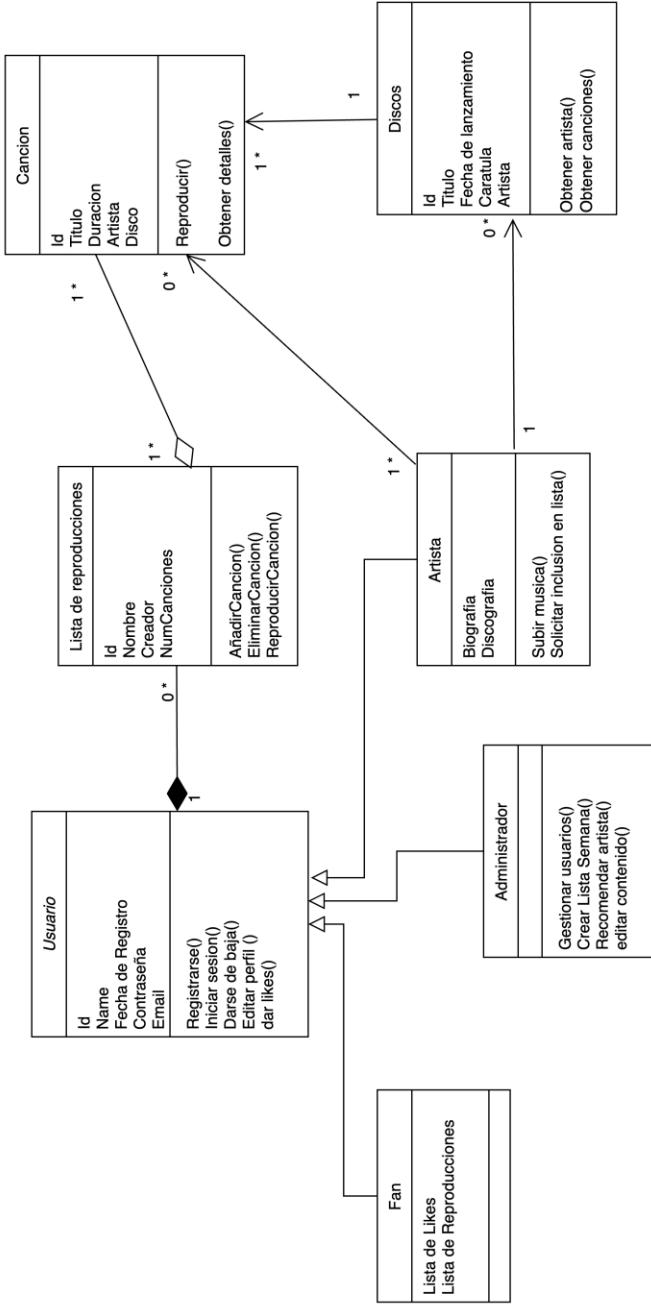
1.3 Diseño Conceptual

- **Diagramas de Casos de Uso:**



- **Diagrama de Clases:**

*La relación entre el usuario y las listas de reproducción es de composición, ya que si el usuario deja de salir su lista de reproducciones desaparece
 *La relación entre la lista de reproducciones y las canciones es de agregación ya que éstas existen de forma independiente, no están ligadas a las listas



Diseño Visual de la Interfaz

En el desarrollo de MusicMuse, se optó por un enfoque de diseño visual minimalista que refleja la usabilidad y estética contemporánea de plataformas líderes como Spotify. Sin embargo, para diferenciar nuestra interfaz y fortalecer la identidad visual de MusicMuse, se implementó una paleta de colores centrada en tonos oscuros y grises, con el rojo como color distintivo de la marca. Este color rojo no solo se utiliza para resaltar elementos interactivos y de importancia en la interfaz, sino que también sirve para invocar una sensación de energía y pasión por la música, un sentimiento que queremos transmitir a nuestros usuarios.

Elección del Esquema de Colores

El uso de colores oscuros proporciona una base neutra que mejora la legibilidad de los textos y elementos gráficos, esencial para la experiencia del usuario durante sesiones prolongadas de navegación o reproducción de música. El contraste del rojo sobre estos fondos oscuros no solo captura la atención de manera efectiva, sino que también guía intuitivamente a los usuarios hacia las funcionalidades más importantes, como la reproducción de música, descubrimiento de nuevos artistas, y la navegación entre diferentes secciones de la aplicación.

Inspiración y Objetivos del Diseño

La inspiración detrás de este diseño se arraiga en la necesidad de ofrecer una experiencia de usuario fluida y centrada en el contenido, donde la música toma el protagonismo sin distracciones visuales innecesarias. El minimalismo en el diseño ayuda a reducir la carga cognitiva de los usuarios, permitiendo una navegación intuitiva y accesible para todos. Además, la elección del rojo como marca de la casa busca no solo diferenciar a MusicMuse de otras plataformas similares, sino también crear una identidad visual memorable que los usuarios puedan asociar con una experiencia musical única y personalizada.

Implementación Técnica

Técnicamente, el diseño se implementó mediante el uso de CSS3 y frameworks como Bootstrap, que facilitaron la creación de una interfaz responsive y adaptable a diferentes dispositivos y tamaños de pantalla.

Implementación

La implementación del proyecto Music Muse se llevó a cabo utilizando un conjunto de tecnologías web tanto en el frontend como en el backend. A continuación, se describen

los aspectos más destacados de esta fase, incluyendo la estructura del código, las herramientas empleadas y algunos fragmentos de código relevantes.

Tecnologías Utilizadas

- **Frontend:**

- **HTML5:** Para la estructura de las páginas web.
- **CSS3:** Para el diseño y la presentación visual de la aplicación.
- **JavaScript:** Para la interactividad del sitio.
- **Bootstrap:** Para el diseño responsive y componentes predefinidos.

- **Backend:**

- **Apache:** Como servidor web.
- **PHP:** Para la lógica del servidor y la gestión de sesiones.
- **MySQL:** Como base de datos para almacenar información de usuarios y canciones.
- **phpMyAdmin:** Para la administración de la base de datos.

- **IDE:**

- **Visual Studio Code:** Como entorno de desarrollo integrado.



Bases de Datos MySQL

Para implementar la base de datos de mi proyecto MusicMuse, diseñé y desarrollé un esquema de base de datos utilizando MySQL. Este esquema está compuesto por varias tablas diseñadas para manejar diferentes aspectos de la aplicación, como el registro de

usuarios, la información de los artistas, los discos y las canciones, y las listas de reproducción. A continuación, describo el proceso y la lógica detrás de la creación de estas tablas:

Tablas Principales

A continuación, creé varias tablas para almacenar la información necesaria para el funcionamiento de MusicMuse:

1. **Usuarios:** Esta tabla almacena información sobre los usuarios, incluyendo su nombre de usuario, correo electrónico, contraseña (almacenada como hash por seguridad), rol en la aplicación, y un token de recuperación de contraseña en caso de que olviden su acceso.

```
CREATE TABLE Usuarios (
    UsuarioID INT AUTO_INCREMENT PRIMARY KEY,
    NombreUsuario VARCHAR(255) NOT NULL,
    Correo VARCHAR(255) NOT NULL,
    ContraseñaHash VARCHAR (255) NOT NULL,
    Rol VARCHAR (50),
    FechaCreacionPerfil DATE,
    TokenRecuperacionContraseña VARCHAR (255)
);
```

2. **Artistas:** Relacionada con la tabla Usuarios mediante UsuarioID, esta tabla almacena información específica sobre los artistas, como su nombre artístico y descripción.

```
CREATE TABLE Artistas (
    ArtistaID INT AUTO_INCREMENT PRIMARY KEY,
    UsuarioID INT,
    Nombre VARCHAR(255) NOT NULL,
    Descripcion TEXT,
    Pais VARCHAR(100),
    FOREIGN KEY (UsuarioID) REFERENCES Usuarios(UsuarioID)
);
```

3. **Discos:** Contiene detalles sobre los discos lanzados por los artistas, incluyendo el título, género, fecha de lanzamiento, y otros créditos.

```

CREATE TABLE Discos (
    DiscoID INT AUTO_INCREMENT PRIMARY KEY,
    ArtistaID INT,
    Titulo VARCHAR(255) NOT NULL,
    Genero VARCHAR(255) NOT NULL,
    FechaLanzamiento DATE,
    Creditos VARCHAR(255),
    InformacionDisco VARCHAR(255),
    ImagenPortada VARCHAR(255),
    FOREIGN KEY (ArtistaID) REFERENCES Artistas(ArtistaID)
);

```

4. **Canciones:** Detalla las canciones individuales de cada disco, incluyendo el título y duración.

```

CREATE TABLE Canciones (
    CancionID INT AUTO_INCREMENT PRIMARY KEY,
    DiscoID INT,
    Titulo VARCHAR(255) NOT NULL,
    Duracion TIME,
    NumeroTrack INT,
    FOREIGN KEY (DiscoID) REFERENCES Discos(DiscoID)
);

```

5. **Listas de Reproducción y Canciones en Listas:** Estas tablas gestionan las listas de reproducción creadas por los usuarios y las canciones que contienen, respectivamente.

```

CREATE TABLE ListasReproduccion (
    ListaID INT AUTO_INCREMENT PRIMARY KEY,
    UsuarioID INT,
    NombreLista VARCHAR(255) NOT NULL,
    FechaCreacion DATE NOT NULL,
    Publica BOOLEAN DEFAULT TRUE,
    FOREIGN KEY (UsuarioID) REFERENCES Usuarios(UsuarioID)
);

CREATE TABLE Lista_Canciones (
    ListaID INT,
    CancionID INT,
    Orden INT,
    PRIMARY KEY (ListaID, CancionID),
    FOREIGN KEY (ListaID) REFERENCES ListasReproduccion(ListaID),
    FOREIGN KEY (CancionID) REFERENCES Canciones(CancionID)
)

```

Estructura del Proyecto

El proyecto sigue una estructura MVC (Modelo-Vista-Controlador) básica, aunque con algunas simplificaciones:

- **Modelo:** Representado por la base de datos MySQL.
- **Vista:** Páginas HTML y componentes Bootstrap.
- **Controlador:** Scripts PHP que manejan la lógica de negocio y las interacciones con la base de datos.

Gestión de Sesiones

Para la gestión de sesiones y autenticación de usuarios, se utiliza la variable de sesión `$_SESSION`. A continuación se muestra un fragmento de código que gestiona el inicio de sesión y el cierre de sesión:

```
session_start(); // Inicia la sesión

// Verifica si el parámetro de logout está presente en la URL
if (isset($_GET['logout']) && $_GET['logout'] == 1) {
    // Elimina las variables de sesión y destruye la sesión
    unset($_SESSION['usuario']);
    unset($_SESSION['canciones']);
    session_destroy();
    header("Location: Index.php");
    exit;
}

// Obtiene el nombre de usuario de la sesión
$nombreUsu = $_SESSION['usuario'];
```

Conexión a la Base de Datos

Para la conexión a la base de datos se utiliza MySQL, con la configuración básica y la selección de UTF-8 para asegurar la correcta codificación de caracteres:

```

$host = 'localhost';
$usuario = 'root';
$nombre_base_datos = 'BaseMusicas';

// Crear una única conexión
$conn = new mysqli($host, $usuario, "", $nombre_base_datos);

if ($conn->connect_error) {
    die("Conexión fallida: " . $conn->connect_error);
}

mysqli_set_charset($conn, "utf8"); ↓

```

Consultas a la Base de Datos

Para obtener el rol del usuario y otras consultas, se utilizan declaraciones preparadas para evitar inyecciones SQL:

```

$sql = "SELECT Rol
        |   FROM Usuarios
        | WHERE NombreUsuario = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("s", $nombreUsu);
$stmt->execute();
$result = $stmt->get_result();

if ($result->num_rows > 0) {
    // Obtener el rol del usuario
    $row = $result->fetch_assoc();
    if ($row['Rol'] == "artista") {
        $_SESSION['rol'] = 0;
    } else {
        $_SESSION['rol'] = 1;
    }
}

```

Si el rol es "artista", asignas el valor 0 a `$_SESSION['rol']`.

Si el rol no es "artista", asignas el valor 1 a `$_SESSION['rol']`.

Este mecanismo te permite controlar el acceso y las funcionalidades disponibles para diferentes tipos de usuarios en tu aplicación, basándote en si son artistas o no (por ejemplo, permitir a los artistas subir música pero no a los fans). Esta diferenciación es crucial para saber los permisos que tendrá cada usuario.

En este fragmento del código se muestra como se vera el navegador según el tipo de usuario que sea. Si la condición `$_SESSION['rol']` es falsa, entrara en el perfil de artista y si es verdadera en el de fan:

```
<nav>
<ul>
    <?php if (isset($_SESSION['usuario'])&& !$_SESSION['rol'] ) : ?>
        <li class="nav-item">
            <?= "Hola ".htmlspecialchars($_SESSION['usuario']) .!">
            <ul class="dropdown-content">
                <li><a href="FormularioAñadirMusica.php">Subir Música</a></li>
                <li><a href="miEspacio.php">Mi espacio</a></li>
                <li><a href="EditarPerfil.php">Editar perfil</a></li>
                <li><a href="?logout=1">Cerrar Sesión</a></li>

                <!-- Agrega más enlaces aquí si necesitas -->
            </ul>
        </li>
    <?php elseif (isset($_SESSION['usuario'])&& $_SESSION['rol'] ) : ?>
        <li class="nav-item">
            <?= "Hola ".htmlspecialchars($_SESSION['usuario']) .!">
            <ul class="dropdown-content">
                <li><a href="EditarPerfil.php">Editar perfil</a></li>
                <li><a href="?logout=1">Cerrar Sesión</a></li>

                <!-- Agrega más enlaces aquí si necesitas -->
            </ul>
        </li>
    <?php else: ?>
        <li class="nav-item" id="openVentana1">
            Registrarse
        </li>
        <a href="login.php">
            <li class="nav-item">
                Iniciar Sesión
            </li>
            </a>
        <?php endif; ?>
    </ul>
</nav>
```

- **Usuarios Artistas:** Si el usuario ha iniciado sesión y su rol es artista (`$_SESSION['rol']` es 0), se muestra un saludo y un menú desplegable con enlaces para subir música, acceder a su espacio personal, editar su perfil y cerrar sesión.

- **Usuarios No Artistas:** Si el usuario ha iniciado sesión pero no es artista (`$_SESSION['rol']` es 1), se muestra un saludo y un menú con opciones para editar su perfil y cerrar sesión.
- **Usuarios No Registrados:** Si no hay una sesión de usuario activa, se muestran opciones para registrarse o iniciar sesión, facilitando el acceso a nuevas cuentas o usuarios existentes para ingresar a la plataforma.

Aquí se muestra otro ejemplo:

```

</div>
<div class="caja2">
<div class="tuMusica">
<?php if (isset($_SESSION['usuario'])&& !$_SESSION['rol'] ) : ?>
<div class="iconos">
<a href="miEspacio.php"><i class="bi bi-cassette-fill"></i></a>
</div>
</div>
<div class="tuMusica" >
<div class="iconos">
<a href="EditarPerfil.php" > <i class="bi bi-gear-wide"></i></a>
</div>
</div>
<div class="tuMusica" >
<div class="iconos">
<a href="FormularioAñadirMusica.php"><i class="bi bi-cloud-plus"></i></a>
</div>

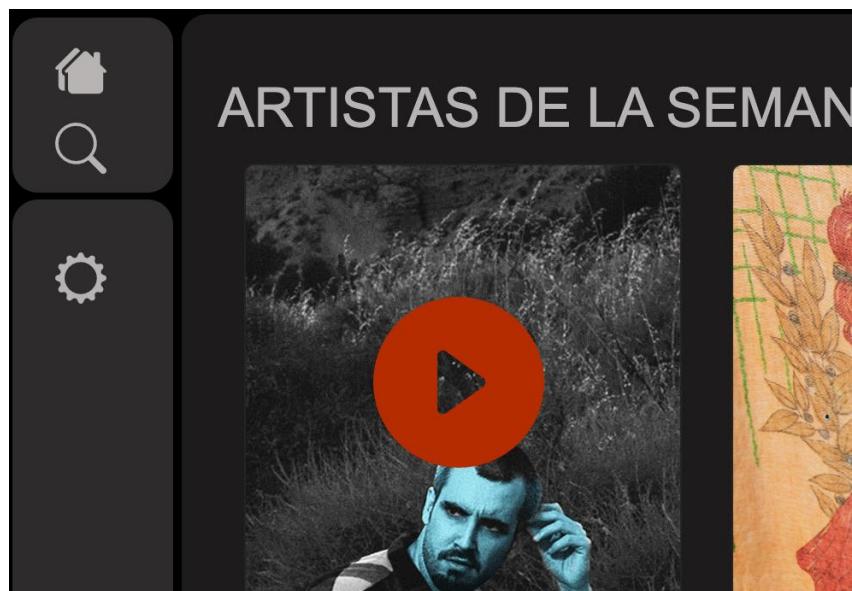
<?php endif; ?>
<!--<p>Tu Música</p> -->
</div>
<div >
<div>
<?php if (isset($_SESSION['usuario'])&& $_SESSION['rol'] ) : ?>
<div class="iconos">
<a href="EditarPerfil.php" > <i class="bi bi-gear-wide"></i></a>
</div>
</div>
<?php endif; ?>
<!--<p>Tu Música</p> -->
</div>

```

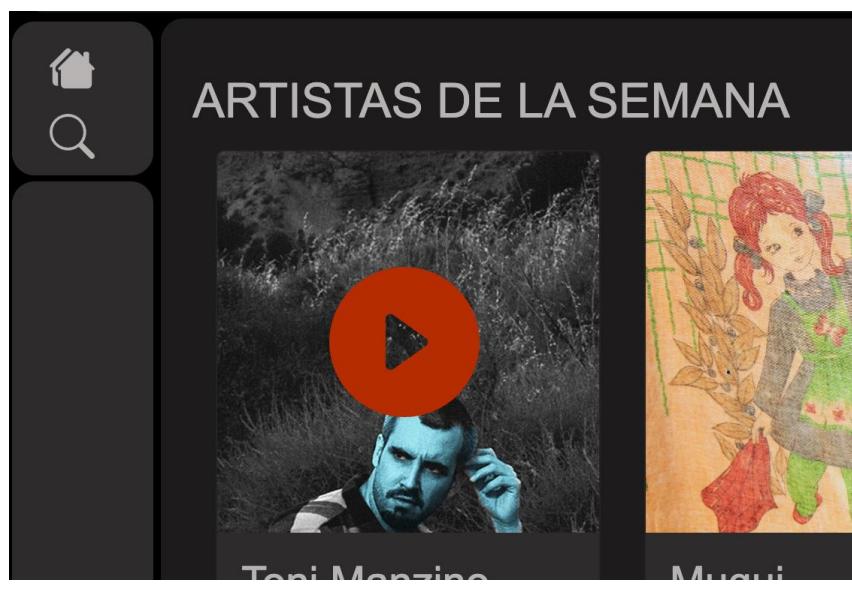
Este será el resultado para usuarios registrados que son Artistas:



Para usuarios Fan:



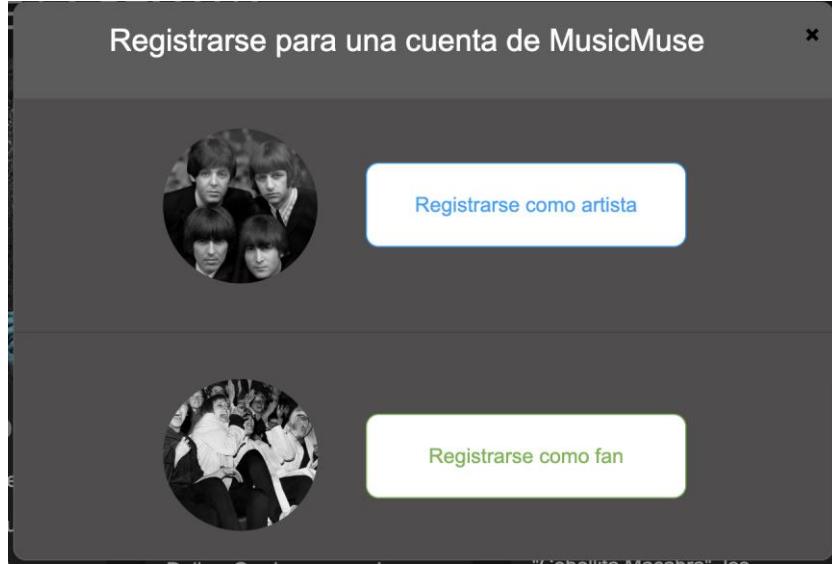
Usuarios no registrados:



Estas opciones equivalen a las mostradas en el Nav.

Artistas pueden editar su perfil, editar su discografía, y subir música. Los fans solo editar su perfil.

REGISTRO CUENTA MUSICMUSE



Para la ventana emergente de registro, creé un modal que permite a los usuarios elegir entre registrarse como fan o como artista. Este modal se implementa utilizando HTML, CSS y JavaScript.

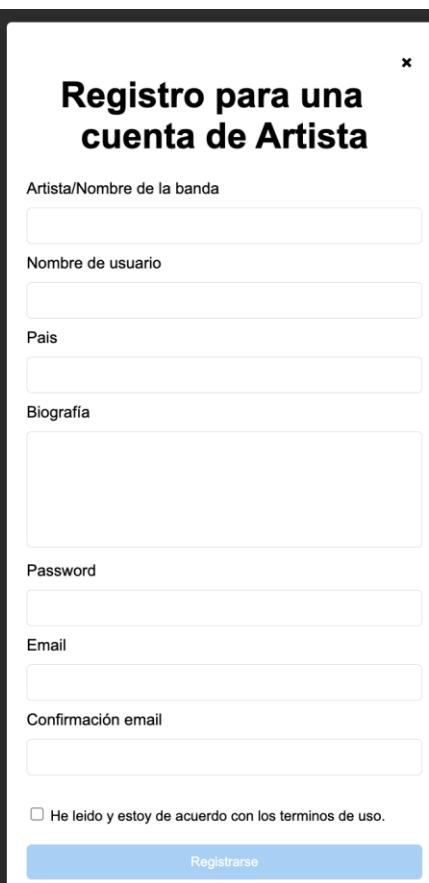
```
<!-- VENTANA CREAR CUENTA -->
<div id="ventana1" class="ventana">
    <!-- Contenido del modal -->
    <div class="contenidoVentana">
        <div class="rotulo">
            <!-- &times; representa una x -->
            <span class="close">&times;
```

Manejo la interactividad del modal mediante JavaScript. Agregué eventos de clic para abrir y cerrar el modal. Al hacer clic en el botón "Registrarse", cambio la propiedad de estilo del modal para hacerlo visible. Igualmente, al hacer clic en el ícono de cierre (×), el modal se oculta.

```
document.addEventListener('DOMContentLoaded', function () {  
    // Para el primer modal  
    var btn1 = document.getElementById('openVentana1');  
    var ventana1 = document.getElementById('ventana1');  
    var span1 = document.querySelector('.close'); // Si solo hay un botón de cerrar con esta clase  
  
    if (btn1 && ventana1 && span1) {  
        btn1.onclick = function() {  
            ventana1.style.display = 'block';  
        }  
  
        span1.onclick = function() {  
            ventana1.style.display = 'none';  
        }  
    }  
}
```

Cuando un usuario selecciona uno de estos botones en el modal, es redirigido a la página correspondiente donde puede completar y enviar su formulario. Cada formulario tiene su propio botón de envío que, una vez activado, procesa la información utilizando métodos POST. Este proceso incluye la validación de los datos ingresados y, posteriormente, el almacenamiento de estos datos en la base de datos de MusicMuse.

Veamos como se gestiona el formulario del Artista que es el que tiene más complejidad



The screenshot shows a registration form titled "Registro para una cuenta de Artista". The form consists of several input fields:

- Artista/Nombre de la banda
- Nombre de usuario
- Pais
- Biografía
- Password
- Email
- Confirmación email

At the bottom of the form is a checkbox labeled "He leído y estoy de acuerdo con los términos de uso." (I have read and agree to the terms of use.) and a blue "Registrarse" (Register) button.

```

<?php
session_start();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $host = 'localhost'; // Cambiar por tu host
    $usuario = 'root'; // Cambiar por tu usuario de la base de datos
    $nombre_base_datos = 'BaseMusicas'; // Cambiar por el nombre de tu base de datos

    // Crear conexión
    $con = mysqli_connect($host, $usuario, '', $nombre_base_datos) or die("Problemas con la conexión a la base de datos");

    // Establecer la codificación de caracteres a UTF-8
    mysqli_set_charset($con, "utf8");

    // Obtener los valores del formulario
    $nombreArtista = $_POST['artistName'];
    $biografia = $_POST['Bio'];
    $pais = $_POST['pais'];
    $contraseña = $_POST['password'];
    $email = $_POST['email'];
    $nombreUsuario = $_POST['nombre'];
    $rol = "artista";
    $fechaAlta = date('Y-m-d'); // Obtiene la fecha actual en formato 'Año-Mes-Día'

    // Codificar la contraseña para guardarla de forma segura
    $hash = password_hash($contraseña, PASSWORD_DEFAULT);

    $sqlUsuario = "INSERT INTO Usuarios (NombreUsuario, Correo, ContraseñaHash, Rol, FechaCreacionPerfil) VALUES (?, ?, ?, ?, ?)";
    $stmtUsuario = $con->prepare($sqlUsuario);
    $stmtUsuario->bind_param("sssss", $nombreUsuario, $email, $hash, $rol, $fechaAlta);

    // Ejecutar la consulta para insertar el usuario
}

```

Comenzamos creando variables para recoger la información enviada a través de cada campo, con la variable `$_POST`.

Cuando un usuario completa un formulario y hace clic en el botón de enviar (submit), los datos del formulario se envían a un servidor. En PHP, estos datos se pueden recoger de manera segura utilizando la superglobal `$_POST`, que almacena la información enviada a través del método POST. Este método es preferido para el envío de formularios, especialmente cuando se maneja información sensible o personal, ya que no expone los datos en la URL del navegador como si hace el método GET.

```

<form action="FormArtista.php" method="post" id="miFormulario" autocomplete="nope">
    <div class="rotulo">
        <a href="Index.php">
            <span class="close">&times;</span>
        </a>
        <h2>Registro para una cuenta de Artista:</h2>
    </div>
    <label for="artistName">Artista/Nombre de la banda</label>
    <input type="text" id="artist-name" name="artistName" required>

    <label for="nombre">Nombre de usuario</label>
    <input type="text" id="username" name="nombre" pattern="[a-zA-Z0-9\-\_]+" required
    oninvalid="this.setCustomValidity('Completa este campo usando únicamente letras, numeros, - y _')"
    oninput="this.setCustomValidity('')" autocomplete="off">

    <label for="pais">País</label>
    <input type="text" id="pais" name="pais" required>

    <label for="Bio">Biografía</label>
    <textarea id="bio" name="Bio" rows="6" cols="45" required></textarea>

    <label for="password">Password</label>
    <input type="password" id="password" name="password" required pattern="^(?=.*[A-Z])(?=.*[a-z])(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$"
    oninvalid="this.setCustomValidity('La contraseña debe contener al menos 8 caracteres, mayúsculas, minúsculas, algún número y algún carácter especial (@$!%*?&)')"
    oninput="this.setCustomValidity('')" autocomplete="off">

    <label for="email">Email</label>
    <input type="email" id="email" name="email" pattern="^([a-zA-Z0-9\-\_\.]+@[a-zA-Z]+\.(com|net|es)$" required
    oninvalid="this.setCustomValidity('Escriba una dirección de correo válida')"
    oninput="this.setCustomValidity('')">

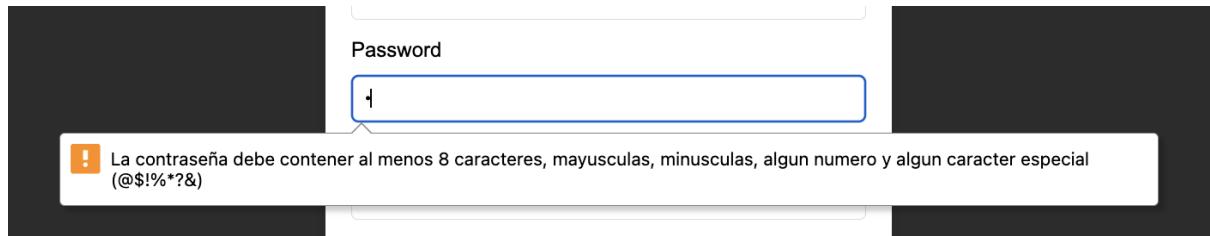
    <label for="confirm-email">Confirmación email</label>
    <input type="email" id="confirm-email" name="confirmEmail" required>
    <span id="email-error" class="error"></span>

    <div class="terms">
        <input type="checkbox" id="terms" name="terms" required>
        <label for="terms">He leído y estoy de acuerdo con los términos de uso.</label>
    </div>

    <input type="submit" value="Registrarse">
    <?php if (!empty($_SESSION['resultadoRegistro'])): ?>
    <span id="resultadoRegistro" class="registro" style="display: none;">
        <?php echo $_SESSION['resultadoRegistro']; ?>
    </span>
    <?php unset($_SESSION['resultadoRegistro']); // Limpia la variable de sesión después de mostrarla ?>
    <?php endif; ?>
</form>

```

Véase que también he utilizado patrones para que cada campo tenga unas características: "`^(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8}$`"



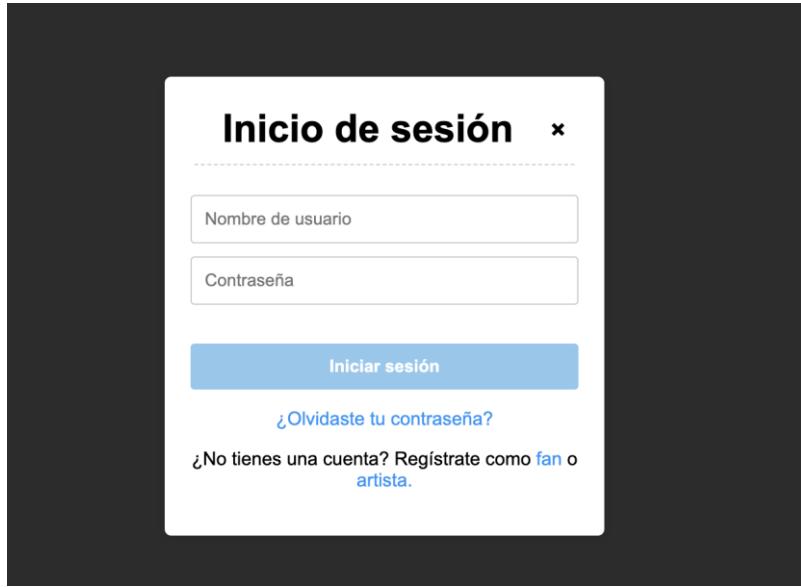
Luego realizo una inserción en la base de datos en la tabla Usuarios con toda la información.

```
$sqlUsuario = "INSERT INTO Usuarios (NombreUsuario, Correo, ContraseñaHash, Rol, FechaCreacionPerfil) VALUES (?, ?, ?, ?, ?)";
$stmtUsuario = $con->prepare($sqlUsuario);
$stmtUsuario->bind_param("sssss", $nombreUsuario, $email, $hash, $rol, $fechaAlta);

// Ejecutar la consulta para insertar el usuario
if ($stmtUsuario->execute()) {
    // Obtener el ID del usuario recién insertado
    $usuarioID = $stmtUsuario->insert_id;

    // Preparar la consulta SQL para insertar el artista
    $sqlArtista = "INSERT INTO Artistas (UsuarioID, Nombre, Descripcion, Pais) VALUES (?, ?, ?, ?)";
    $stmtArtista = $con->prepare($sqlArtista);
    $stmtArtista->bind_param("isss", $usuarioID, $nombreArtista, $biografia, $pais);
```

INICIO DE SESIÓN



Al principio del código login.php creo la variable de sesión `$_SESSION['resultadoRegistro']` que posteriormente usare.

```
<?php
session_start();
$_SESSION['resultadoRegistro'];

?>
```

Luego creo un formulario:

```
<form method="post" action="login.php">
    <input type="text" id="username" name="username" placeholder="Nombre de usuario" required>
    <input type="password" id="password" name="password" placeholder="Contraseña" required>
    <input type="submit" value="Iniciar sesión">
    <?php if (!empty($_SESSION['resultadoRegistro'])): ?>
    <span id="resultadoRegistro" class="registro" style="display: block;">
        <?php echo $_SESSION['resultadoRegistro']; ?>
    </span>
    <?php unset($_SESSION['resultadoRegistro']); // Limpia la variable de sesión después de mostrarla ?>
    <?php endif; ?>
    <p><a href="#">Olvidaste tu contraseña?</a></p>
    <p><a href="#">No tienes una cuenta? Regístrate como <a href="formularioFan.php">fan</a> o <a href="FormArtista.php">artista.</a></p>
</form>
```

Ahora veremos que si la variable esta vacia es porque el nombre de usuario y la contraseña no coinciden con ninguno en la base de datos, y por lo tanto se mostrara un mensaje de error, despues se vaciara la variable de sesion.

```

// Manejo del formulario de inicio de sesi n
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $nom_usuario = $_POST['username'];
    $contraseña_ingerada = $_POST['password'];

    $host = 'localhost'; // Asume localhost como host
    $usuario = 'root'; // Usuario de la base de datos
    $nombre_base_datos = 'BaseMusicas'; // Nombre de la base de datos

    // Crear conexi n
    $con = mysqli_connect($host, $usuario, '', $nombre_base_datos) or die("Problemas con la conexi n a la base de datos");

    // Establecer la codificaci n de caracteres a UTF-8
    mysqli_set_charset($con, "utf8");

    // Consulta para obtener el hash de la contraseña y el UsuarioID
    $sql = "SELECT UsuarioID, ContraseñaHash FROM Usuarios WHERE NombreUsuario = ?";
    $stmt = $con->prepare($sql);
    $stmt->bind_param("s", $nom_usuario);
    $stmt->execute();
    $result = $stmt->get_result();

    // Verifica que el usuario exista
    if ($result->num_rows > 0) {
        $row = $result->fetch_assoc();
        $hash_guardado = $row['ContraseñaHash'];
        $usuario_id = $row['UsuarioID'];

        // Verifica la contraseña
        if (password_verify($contraseña_ingerada, $hash_guardado)) {
            $sqlArtista = "SELECT ArtistaID, Nombre FROM Artistas WHERE UsuarioID = ?";
            $stmtArtista = $con->prepare($sqlArtista);
            $stmtArtista->bind_param("i", $usuario_id);
            $stmtArtista->execute();
            $result = $stmtArtista->get_result();
            if ($row = $result->fetch_assoc()) {
                $_SESSION['IDArtista'] = $row['ArtistaID'];
                $_SESSION['Artista'] = $row['Nombre'];
            }
            $stmtArtista->close();

            // Si la contraseña es correcta, guarda el nombre de usuario y UsuarioID en la sesi n
            $_SESSION['usuario'] = $nom_usuario;
            $_SESSION['IDusuario'] = $usuario_id;
            // Guardar UsuarioID en la sesi n
            header("Location: Index.php");
            exit;
        } else {
            $_SESSION['resultadoRegistro'] = "Contraseña incorrecta.";
        }
    } else {
        $_SESSION['resultadoRegistro'] = "No se encontr  el usuario.";
    }

    // Cierra la declaraci n preparada y la conexi n
    $stmt->close();
    $con->close();
    header("Location: Index.html");
}
?>

```

Aqui vemos como mediante la variable global `$_POST` recogemos la contraseña y el nombre de usuario enviado a traves del formulario. Despues iniciamos una conexión con la base de datos y preparamos una consulta para ver si se devuelve alguna fila que coincida con el nombre introducido, en caso afirmativo, se asigna el `UsuarioID` y la `contraseñahash` en dos variables diferentes.

Se verifica si la contraseña introducida en el formulario coincide con el `contraseñahash` extraída de la base, si coincide se prepara otra consulta. Ahora mediante la variable '`UsuarioID`' vamos a extraer de la tabla artista, el '`artistaID`' y '`Nombre`', que van a ser

almacenados en dos variables de sesión diferentes, para poder acceder a ellas en el futuro. También se asignara a la variable `$_SESSION['usuario']` el nombre de usuario introducido en el formulario y se redirigirá a la página principal , ya logeada.

En caso de que las contraseñas no coincidan se asignara:

```
$_SESSION['resultadoRegistro'] = "No se encontró el usuario.";
```

Que sera mostrada en el formulario.



FORMULARIO AÑADIR MUSICA (libreria getID3)

Importante mencionar que a este formulario como ya hemos visto antes solo podrán acceder los usuarios registrados como artistas.

```
// Verifica si el parámetro de logout está presente en la URL
if (isset($_GET['logout']) && $_GET['logout'] == 1) {
    // Elimina la variable de sesión 'usuario'
    unset($_SESSION['usuario']);
    unset($_SESSION['canciones']);

    // Opcional: Destruye la sesión completa
    session_destroy();

    // Redirige al usuario a la página de inicio o de inicio de sesión
    header("Location: Index.php");
    exit;
}
```

este código es un mecanismo efectivo para manejar el cierre de sesión en aplicaciones web, asegurando que las credenciales y los datos del usuario no permanezcan en la sesión una vez que han decidido cerrar sesión, y redirigiendo al usuario a una página

segura para evitar la exposición de datos sensibles o acciones no autorizadas después de que la sesión ha terminado.

Este formulario ha sido, sin duda, una de las partes más complicadas de gestionar en la aplicación. He dedicado mucho tiempo a preparar y organizar todos los datos correctamente para asegurar su correcto almacenamiento en la base de datos.

He tenido que descargar un librería externa para poder gestionar los archivos de audio.

Se trata de **getID3**.

GetID3 es una biblioteca de PHP que se utiliza para leer metadatos de archivos multimedia. Es capaz de extraer información de una amplia variedad de formatos de archivos, incluyendo audio, video e imágenes. getID3 es muy útil en aplicaciones que necesitan manejar o administrar archivos multimedia, como aplicaciones de gestión de música, editores de video, y sistemas de galería de fotos.

Características principales de getID3:

Extracción de Datos: Puede leer y analizar información como la duración de un archivo de audio o video, tasas de bits, resolución de video, formatos de compresión utilizados, y muchos otros detalles técnicos.

Compatibilidad de Formatos: Soporta una amplia gama de formatos de archivo, como MP3, MP4, WAV, AVI, MOV, y muchos otros. Esto lo hace extremadamente versátil para proyectos que involucran múltiples tipos de medios.

Escritura de Tags: Además de leer metadatos, getID3 puede también escribir (o modificar) información en algunos formatos de archivos, como tags ID3 en archivos MP3, permitiendo la personalización y gestión de información de los archivos.

Análisis de Errores: Puede detectar y reportar si los archivos están corruptos o si contienen datos que no cumplen con las especificaciones de su formato.

Uso sin Base de Datos: getID3 opera directamente sobre los archivos y no requiere una base de datos, lo que simplifica su integración en proyectos y reduce la sobrecarga de mantenimiento de infraestructura adicional.

En el caso de esta aplicación sobre todo lo he utilizado para mostrar la duración de las pistas y otros metadatos importantes en interfaces de usuario.

Inclusión de Bibliotecas y Verificación

Carga de la Biblioteca getID3:

```
require_once 'getID3/vendor/autoload.php';
require_once('getID3/getID3.php');
```

Estas líneas aseguran que todas las dependencias necesarias para utilizar la biblioteca getID3 estén cargadas. autoload.php se utiliza para cargar automáticamente las clases necesarias sin tener que incluirlas manualmente.

Verificación de la Clase getID3:

```
if (!class_exists('getID3')) {
    die('getID3 library is missing!');
}
```

Esta condición verifica si la clase getID3 está disponible. Si no está disponible, el script se detiene y muestra un mensaje de error, lo que evita errores de ejecución posteriores si la biblioteca no está correctamente incluida.

Procesamiento del Formulario

Verificación de Envío del Formulario:

```
if (isset($_POST["submit1"])) {
```

Esta línea verifica si el formulario ha sido enviado. Esto se determina por la presencia de un botón de envío llamado submit1 en los datos del formulario enviado.

Manejo de Archivos

Obtención de Información del Archivo:

```
$nombreArchivo = $_FILES['archivoMusica']['name'];
$rutaTemporal = $_FILES['archivoMusica']['tmp_name'];
```

Aquí, se recupera el nombre del archivo y la ruta temporal del archivo cargado, utilizando la superglobal \$_FILES.

Definición de la Ruta de Destino:

```
$rutaDestino = $directorioDestino . basename($nombreArchivo);
```

Esta línea construye la ruta de destino donde el archivo será guardado permanentemente en el servidor.

Previamente en declarado:

```
$directorioDestino = /discos;
```

De esta manera todos los archivos de audio se guardarán en esta carpeta, con el nombre extraído del archivo.

Análisis del Archivo con getID3

Extracción de Metadatos:

```
$getID3 = new getID3;  
$fileInfo = $getID3->analyze($rutaDestino);  
$duracionSegundos = $fileInfo['playtime_seconds'];
```

Se crea una instancia de getID3 y se analiza el archivo cargado. Se extrae la duración en segundos del archivo de música.

Formateo de la Duración:

```
$minutos = floor($duracionSegundos / 60);  
$segundos = round($duracionSegundos % 60);  
$duracionFormateada = sprintf('%d:%02d', $minutos, $segundos);
```

La duración en segundos se convierte a un formato más legible de minutos y segundos.

Actualización de la Sesión y Redirección

Actualización de Datos en Sesión:

```
$_SESSION['canciones'][$index] = [
    'colaboracion' => $_POST['colaboracion'],
    'nombre' => $_POST['Name'],
    'genero' => $_POST['genero'],
    'archivo' => $nombreArchivo,
    'ruta' => $rutaDestino,
    'reproNom' => $nombreAudio,
    'duracion' => $duracionFormateada
];

```

Los metadatos extraídos y otros datos del formulario se almacenan en la sesión del usuario, lo que permite el acceso y manipulación de estos datos en otras partes de la aplicación. De esta forma almaceno en el array [canciones] toda la información que necesito para cada canción.

La variable:

```
$index = $_POST['indice'];
```

Es muy importante para la identificación la de canción en el array.

Está introducido en un campo oculto del formulario y es manejado a través de JavaScript.

Código HTML:

```
<div class="form-group">
    <label for="Name">Nombre canción</label>
    <input type="text" id="Name" name="Name" >
    <input type="hidden" id="indice" name="indice" value=" " >
</div>
```

Código JavaScript:

```
// Repite el proceso para el segundo modal con los IDs y clases correspondientes
var btn2 = document.getElementById('openVentana2');
var ventana2 = document.getElementById('ventana2');
var indiceInput = document.getElementById('indice'); // Obtiene el input donde se da el índice
var span2 = document.querySelector('.close2'); // Asegúrate de que esta clase exista
```

```
if (btn2 && ventana2 && span2 && indiceInput) {
    btn2.onclick = function() {
        ventana2.style.display = 'block';
        // Establece el valor del input 'indice' al siguiente índice disponible en el array de PHP
        <?php
            // Comprueba si el array 'canciones' existe y calcula el siguiente índice
            if (isset($_SESSION['canciones'])) {
                echo "indiceInput.value = " . count($_SESSION['canciones']) . ";";
            } else {
                echo "indiceInput.value = 0;"; // Si no hay canciones, el índice inicial es 0
            }
        ?>
    }
}
```

En este fragmento de JavaScript se ve como se le asigna a 'indiceInput' el valor del número de elementos del array ['canciones'] en el caso de que este exista, en caso contrario se le asigna el valor de 0, es decir la primera posición del array.

```
<?php
if (isset($_SESSION['canciones']) && count($_SESSION['canciones']) > 0) {
    echo '<table class="tablalistado">';
    echo '<tr><th>Pista</th><th>Canción</th><th>Colaboraciones</th><th>Duración</th><th>Editar</th><th>Borrar</th></tr>';
    $id = 1;
    foreach ($_SESSION['canciones'] as $index => $cancion) {
        echo '<tr>';
        echo '<td>' . htmlspecialchars($id) . '</td>';
        echo '<td>' . htmlspecialchars($cancion['nombre']) . '</td>';
        echo '<td>' . htmlspecialchars($cancion['colaboracion'] ?? 'No disponible') . '</td>';
        echo '<td>' . htmlspecialchars($cancion['duracion']) . '</td>';
        echo '<td><button class="openVentana3" data-index="' . $index . '" data-nombre="" . htmlspecialchars($cancion['nombre']) . "" data-colaboracion="" . htmlspecialchars($cancion['colaboracion'] ?? '') . '>Editar</button></td>';
        echo '<td><form action="eliminarCancion.php" method="post">';
            echo '<input type="hidden" name="cancionIndex" value="' . $index . '"';
            echo '<button type="submit" name="delete">Eliminar</button>';
        echo '</form></td>';
        echo '</tr>';
        $id++;
    }
}

echo '</table>';
```

En este fragmento de código se ve cómo voy creando una tabla con las canciones cargadas que se mostrara de la siguiente manera:

The screenshot shows a user interface for managing songs. At the top, there is a blue button labeled 'Añadir canción' and a note stating '300MB max por canción .wav, .aif o .flac'. Below this is a table with the following data:

Pista	Canción	Colaboraciones	Duración	Editar	Borrar
1	Cancion 1		5:13	<button>Editar</button>	<button>Eliminar</button>
2	cancion2		4:12	<button>Editar</button>	<button>Eliminar</button>

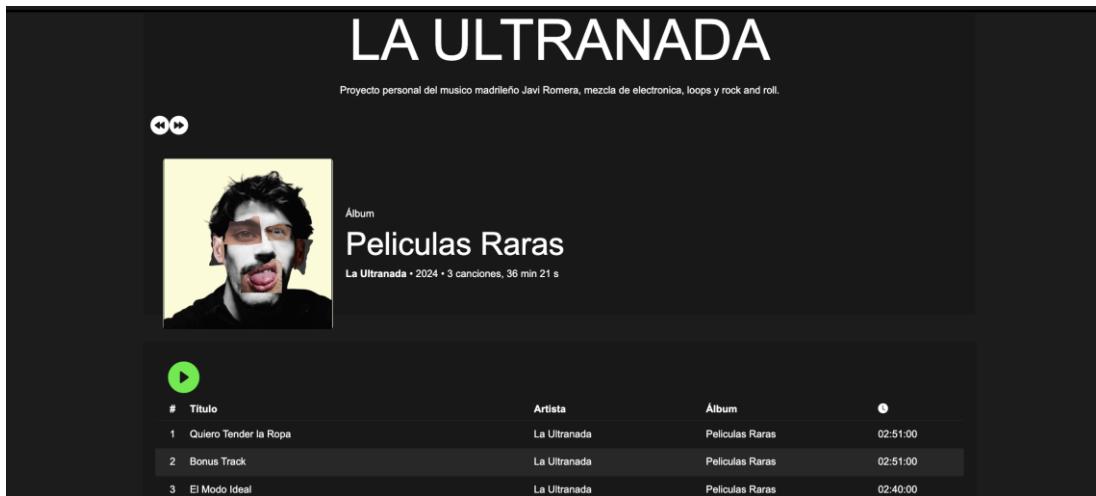
En la tabla también se irán cargando los botones de eliminar y editar que serán perfectamente funcionales para cada canción.

Finalmente, una vez haya subido todas las canciones, podre previsualizar en el reproductor o subirlas a la base de datos, mediante el botón 'Publicar' que me redirigirá a 'CargarBaseDatos.php' donde tendré preparados una serie de querys, que me permitirán cargar todos los datos correctamente en la base de datos.

```
<!-- aqui se va a mostrar el boton publicar en cuanto se cree la variable de sesion [canciones]-->
<?php if(isset($_SESSION['canciones'])): ?>
    <div class="form-actions">
        <a href="CargarBaseDatos.php"><button type="button" >Publicar</button></a>
    </div>
    <div class="form-actions">
        <a href="vistaPrevia.php"><button type="button" id="previsualizar">Previsualizar</button></a>
    </div>
<?php endif; ?>

<!-- En FormularioAladirMusica.php dentro del HTML donde deseas los botones -->
```

Previsualización:



Query Base de datos:

```

// Establecer la codificación de caracteres a UTF-8
mysqli_set_charset($con, "utf8");

// Datos del Álbum y artista
$album = $_SESSION['albumName'];
$fecha = $_SESSION['releaseDate'];
$genero = $_SESSION['Genero'];
$info = $_SESSION['infoDisco'];
$creditos = $_SESSION['Creditos'];
$portada = $_SESSION['caratula'];
$pais = $_SESSION['Etiquetas'];
$usuarioID = $_SESSION['IDusuario']; // ID del usuario en sesión
$artistaID= $_SESSION['IDartista'];// ID del artista en sesión

/* Insertar en la tabla Artista
$sqlArtista = "INSERT INTO Artistas (Nombre, Pais, UsuarioID) VALUES (?, ?, ?)";
$stmtArtista = $con->prepare($sqlArtista);
$stmtArtista->bind_param("ssi", $artista, $pais, $usuarioID);
$stmtArtista->execute();
// $artistaID = $con->insert_id; // Captura el ID generado para el artista
$stmtArtista->close();*/

// Insertar en la tabla Discos
$sqlAlbum = "INSERT INTO Discos (Titulo, Genero, FechaLanzamiento, ImagenPortada, ArtistaID) VALUES (?, ?, ?, ?, ?)";
$stmtAlbum = $con->prepare($sqlAlbum);
$stmtAlbum->bind_param("ssssi", $album, $genero, $fecha, $portada, $artistaID);
$stmtAlbum->execute();
$discoID = $con->insert_id; // Captura el ID generado para el disco
$stmtAlbum->close();

```

Como se muestran todos los artistas en la página principal (solo usuarios logueados).

En la aplicación, todos los artistas se muestran en una sección dedicada de la interfaz de usuario. Utilizando una consulta a la base de datos, se recopilan detalles de cada artista, como el nombre y una breve descripción. Estos datos se organizan en tarjetas individuales que se disponen en una cuadrícula o lista dentro de la página. Cada tarjeta muestra una imagen representativa del artista y algunos detalles básicos, y ofrece un enlace para ver más información o escuchar toda su discografía. Esta presentación no solo facilita la navegación y el acceso rápido a la información del artista, sino que también mejora la experiencia visual del usuario al explorar la variedad de artistas disponibles en la plataforma.

```

if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        // Limitar la descripción a 8 palabras
        $descripcion = $row['descripcion'];
        $palabras = explode(' ', $descripcion);
        if (count($palabras) > 8) {
            $descripcion = implode(' ', array_slice($palabras, 0, 6)) . '...';
        }

        echo '<div class="card" style="width: 18rem; background-color: #2D2C2C">';
        echo '<div style="position: relative;">';
        echo '';
        echo '</div>';
        // Enlace con el ID del artista como parámetro en la URL
        echo '<a href="perfil.php?artista_id='.$row['ArtistaID'].'">';
        echo '<div class="card-body">';
        echo '<h3 style="font-size:20px">' . $row['nombre'] . '</h3>';
        echo '<p class="card-text">' . $descripcion . ' </p>';
        echo '</div>';
        echo '</a>';
        echo '</div>';
    }
} else {
    echo "No hay artistas disponibles.";
}

$conn->close();

```

Creo un bucle while para recorrer todos los artistas de la base de datos, y además que en la descripción solo se puedan incluir 7 palabras seguidas de puntos suspensivos, gracias a la magnífica función de php, array_slice().

FUNCIONAMIENTO REPRODUCTOR

En MusicMuse, el reproductor de audio está diseñado para ser una parte integral de la experiencia del usuario, permitiendo la reproducción inmediata de música directamente en la plataforma. Este reproductor es accesible y fácil de usar, ofreciendo una serie de funcionalidades claves:

Interfaz de Usuario: Se presenta con una interfaz clara y amigable, donde los controles básicos como reproducir, pausar, y saltar canciones son inmediatamente visibles y accesibles. Esto facilita a los usuarios el control total sobre la reproducción de música sin necesidad de navegación complicada.

Barra de Progreso: Los usuarios pueden ver cuánto tiempo ha transcurrido de la canción y cuánto falta, gracias a una barra de progreso. Esta barra también permite a los usuarios saltar a cualquier parte de la canción con solo un clic o toque.

Cambio de Canciones: Con botones para avanzar o retroceder, los usuarios pueden fácilmente cambiar entre canciones de la lista de reproducción actual o del álbum que están escuchando.

Visualización de Información: Mientras se reproduce una canción, se muestra información relevante como el nombre del artista y el título de la canción, lo cual enriquece la experiencia al proporcionar contexto sobre lo que se está escuchando.

Compatibilidad: El reproductor está diseñado para ser compatible con diversos dispositivos y navegadores, asegurando que los usuarios puedan disfrutar de la música sin preocuparse por problemas técnicos relacionados con la compatibilidad.



Aquí muestro el fragmento de código html:

```
<audio></audio>
<p id="cancionplay">artista-canción</p>
<div class="contenedoreIconos">
    <div class="conteinerButtonTime">
        <div>
            <p id="current_time">00:00:00</p>
        </div>
        <div class="buttons">
            <div id="prev" style="font-size: 1.6rem;">
                <i class="bi bi-rewind-circle"></i>
            </div>
            <div id="play" >
                <i class="bi bi-play-circle"></i>
            </div>
            <div id="next" style="font-size: 1.6rem;">
                <i class="bi bi-fast-forward-circle"></i>
            </div>
        </div>
        <div>
            |   <p id="current_audio"></p>
        </div>
    </div>
    <div class="progresConteiner">
        |   <div id="progress"></div>
    </div>
</div>
```

figura1

Código JavaScript:

```

//SECCION PARA MANEJAR REPRODUCTOR
var audio= document.querySelector('audio');
var cancion= document.getElementById('cancionplay');
var prev= document.getElementById('prev');
var play= document.getElementById('play');
var next=document.getElementById('next');
var current_time=document.getElementById('current_time');
var current_audio=document.getElementById('current_audio');
var progres=document.getElementById('progres');
var progresConteiner=document.querySelector('.progresConteiner');
var canciones = <?php echo json_encode($archivosPorDisco); ?>;
var ruta=<?php echo json_encode($rutasPorDisco); ?>;
var audioIndex=0;//con esto podria hacer que el usuario eligiera que cancion comienza

```

Estas son todas las variables que declaro en JavaScript para poder manejar el reproductor correctamente. Principalmente estamos usando getElementById, esto quiero decir, coger cada elemento por medio del id. Utilizo la función json_encode() para convertir datos de PHP en formato JSON y luego se asignan a variables JavaScript.

```

loadAudio(audioIndex)
function loadAudio(index) {
    cancion.textContent = canciones[index]; // Asigna el nombre de la canción actual al texto descriptivo.
    audio.src = `discos/${ruta[index]}`; // Asigna la ruta del archivo de la canción actual al reproductor.
    audioIndex = index; // Actualiza el índice global de audio
    audio.load(); // Carga la nueva fuente de audio.

    // Elimina el event listener anterior para evitar múltiples triggers
    audio.removeEventListener("ended", handleSongEnd);

    // Añade el event listener para cargar la siguiente canción cuando la actual termine
    audio.addEventListener("ended", handleSongEnd);

    audio.addEventListener("loadedmetadata", () => {
        timesong(audio.duration, current_audio); // Actualiza el tiempo de la canción.
    });
}

```

En la siguiente parte del código vemos la función para poder cargar las canciones, y accedemos a la var canciones utilizando un 'index', lo mismo pasa con la var ruta, que hemos asignado el valor convertido previamente, con el uso de la función json_encode().

```

function playSong() {
    var icono = document.querySelector('#play i');
    var icono1 = document.querySelector('#play1 i'); // Asume que hay un i dentro de #play1

    audio.play();

    // Ajusta el ícono para reflejar el estado de "reproduciendo"
    icono.classList.remove('bi-play-circle-fill');
    icono.classList.add('bi-pause-circle-fill');

    icono1.classList.remove('bi-play-circle');
    icono1.classList.add('bi-pause-circle');

    // Agrega la clase 'play' para indicar que está reproduciendo
    document.querySelector('#play').classList.add('play');
    document.querySelector('#play1').classList.add('play');
}

```

Aquí se muestra la función para manejar el play utilizando el logo insertado en el código html, véase en la figura 1. Utilizare una función pauseSong() invirtiendo los valores.

```
function prevSong(){
    audioIndex--
    if(audioIndex < 0){
        audioIndex = canciones.length -1;
    }
    loadAudio( audioIndex);
    playSong();
}

function nextSong(){
    audioIndex ++
    if(audioIndex>canciones.length -1){
        audioIndex =0;
    }
    loadAudio( audioIndex);
    playSong();
}
```

Como se muestra arriba, la función prevsong() es muy sencilla, decrementa el audioindex. La condición usada verifica que si el audioindex es menor que 0 se pase a la última canción del array, cuya posición será la longitud del array menos 1 (canciones.length -1).

Luego llama a la función loadAudio() y playSong()

```
function timesong(audio, element){
    var totalSecond =Math.round(audio);
    var minutes = Math.floor(totalSecond/60);
    var seconds =totalSecond % 60;

    element.textContent ="00:" + minutes.toString().padStart(2,"0") + ":" + seconds.toString().padStart(2,"0");
}
```

Por último, en esta parte del código muestro como manejo el tiempo.

var totalSecond = Math.round(audio);

Esta línea redondea la duración del audio (generalmente obtenida en segundos y posiblemente con fracciones de segundo) al número entero más cercano. Esto asegura que el tiempo se maneje en segundos completos para facilitar los cálculos siguientes.

var minutes = Math.floor(totalSecond / 60);

Calcula cuántos minutos completos hay en el total de segundos redondeados.

Math.floor() se usa para asegurar que solo se tomen los minutos completos, descartando cualquier resto.

var seconds = totalSecond % 60;

Utiliza el operador de módulo para encontrar el número de segundos restantes después de extraer los minutos completos.

```
element.textContent = "00:" + minutes.toString().padStart(2, "0") + ":" +  
seconds.toString().padStart(2, "0");
```

Esta línea actualiza el contenido de texto del elemento HTML proporcionado, mostrando el tiempo en formato hh:mm:ss. Aunque las horas están siempre establecidas como 00 aquí, muestra los minutos y segundos con ceros a la izquierda si es necesario (por ejemplo, 07:05). `padStart(2, "0")` asegura que tanto los minutos como los segundos se muestren siempre con dos dígitos, añadiendo un cero a la izquierda si el número es menor de 10.

CARGA DEL PERFIL DE ARTISTA

Por último, queda mostrar cómo funciona este apartado.

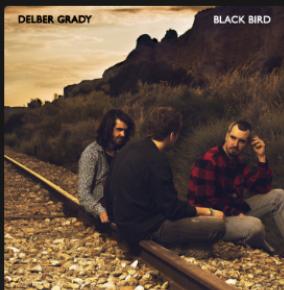
Envío el id de cada artista a la página perfil.php con el método GET, en este caso me facilita mucho la tarea, el id del artista no es una información comprometida por lo que considero apropiado el uso del método GET.

```
// Crear una única conexión  
$conn = new mysqli($host, $usuario, "", $nombre_base_datos);  
if ($conn->connect_error) {  
    die("Conexión fallida: " . $conn->connect_error);  
}  
mysqli_set_charset($conn, "utf8");  
  
if(isset($_GET['artista_id']) && is_numeric($_GET['artista_id'])) {  
    // Recuperar el ID del artista desde la URL  
    $IdRecibido = $_GET['artista_id'];  
  
    $sql = "SELECT Nombre, Descripcion  
           FROM Artistas  
          WHERE ArtistaID = ?";  
  
    $stmt = $conn->prepare($sql);  
    $stmt->bind_param("i", $IdRecibido);  
    $stmt->execute();  
    $result = $stmt->get_result();  
  
    // Verificar si se encontró un artista con el ID dado  
    if ($result->num_rows > 0) {  
        // Obtener el nombre del artista  
        $row = $result->fetch_assoc();  
        $Nombre = $row['Nombre'];  
        $descripcion=$row['Descripcion'];  
  
    } else {  
        echo "No se encontró ningún artista con el ID: $IdRecibido";  
    }  
}
```

Aquí vemos como cargo el id recibido en la variable \$IdRecibido.

DELBER GRADY

Banda de rock progresivo de los 90.



Álbum

Black Bird

Delber Grady • 2015 • 5 canciones, 36 min 21 s



#	Título	Artista	Álbum	Tiempo
1	Black Bird	Delber Grady	Black Bird	09:41:00
2	Maggiepie	Delber Grady	Black Bird	00:34:00
3	Stay	Delber Grady	Black Bird	03:15:00
4	No es Verdad	Delber Grady	Black Bird	02:07:00
5	Goodbye	Delber Grady	Black Bird	03:09:00

Así se mostrará el perfil del artista, con su descripción bajo la cabecera y todo su material disponible para la escucha.

Despliegue y Pruebas

Despliegue

El despliegue de MusicMuse se realizó en un ambiente controlado que replicaba las condiciones de producción para asegurar la estabilidad y funcionamiento óptimo antes de su lanzamiento oficial. Elegimos un servidor basado en Linux debido a su robustez y compatibilidad con las tecnologías empleadas, como PHP y MySQL. Utilizamos Apache como servidor web por su flexibilidad y amplia aceptación en la industria.

Para gestionar la base de datos, optamos por MySQL debido a su escalabilidad y soporte extensivo para grandes volúmenes de datos, lo cual es esencial para una aplicación de streaming de música que maneja grandes cantidades de información de usuario y archivos multimedia. El despliegue final se realizó en un entorno de cloud computing

para garantizar la escalabilidad y la disponibilidad del servicio frente a variaciones en la demanda de acceso de los usuarios.

Pruebas

Pruebas Funcionales

Realizamos pruebas funcionales intensivas para verificar que todas las características de MusicMuse funcionaran como se esperaba. Esto incluyó pruebas de carga de música, gestión de perfiles de usuario, streaming de audio, y la interacción entre usuarios a través de la plataforma. Cada funcionalidad se probó bajo diferentes escenarios y condiciones de uso para garantizar una respuesta adecuada del sistema ante cualquier situación.

Pruebas de Usabilidad

Se llevaron a cabo sesiones de prueba de usabilidad con grupos focales compuestos por potenciales usuarios finales. Estas pruebas ayudaron a identificar problemas de navegación y experiencia de usuario que no fueron evidentes durante la fase de desarrollo. Los resultados obtenidos permitieron realizar ajustes en la interfaz para mejorar la accesibilidad y la intuitividad de la aplicación.

Pruebas de Seguridad

Las pruebas de seguridad fueron prioritarias, dada la naturaleza de la aplicación que maneja datos personales y propiedad intelectual en forma de archivos musicales. Se implementaron pruebas de penetración y se utilizaron herramientas automatizadas para identificar y mitigar vulnerabilidades, como inyecciones SQL, XSS y problemas de autenticación y sesión.

Pruebas de Rendimiento

Para asegurar que MusicMuse pudiera manejar un alto número de solicitudes simultáneas sin degradar la experiencia del usuario, se realizaron pruebas de carga y estrés. Estas pruebas fueron cruciales para entender el comportamiento de la aplicación bajo carga extrema y para optimizar el manejo de recursos del servidor.

Resultados de las Pruebas

Los resultados de las pruebas demostraron que MusicMuse cumple con los requisitos funcionales y de rendimiento establecidos en las fases iniciales del proyecto. Las mejoras realizadas como resultado de las pruebas de usabilidad y las correcciones aplicadas tras las pruebas de seguridad proporcionaron una plataforma robusta y amigable para el

usuario. No obstante, se identificaron áreas de mejora continua, especialmente en la gestión de metadatos de las canciones y la personalización del contenido musical recomendado.

Conclusión del Despliegue y Pruebas

El proceso de despliegue y pruebas de MusicMuse fue fundamental para garantizar que la aplicación no solo fuera funcional, sino también segura, eficiente y agradable de usar. Las lecciones aprendidas de este proceso han establecido una base sólida para futuras actualizaciones y han asegurado que la plataforma pueda evolucionar en respuesta a las necesidades cambiantes de los usuarios y los avances tecnológicos.

- .Logging de usuario
- .Unlogging de usuario
- .Registro de usuario
- .Perfil de usuario – Actualizar datos
- .Perfil de usuario – Baja usuario
- .Mostrar artista
- .Subir musica a la plataforma
- .Añadir fotografía de disco
- .Editar canciones
- .Eliminar canciones
- .Subir disco
- .Mostrar disco subido
- .Quitar disco de la discografía
- .Filtrar artistas en la plataforma
- .Reproducir canciones

6. Discusión

Análisis Crítico de los Resultados Obtenidos

La implementación de MusicMuse ha producido resultados significativos en términos de funcionalidad y respuesta del usuario. La plataforma ha demostrado ser capaz de proporcionar un servicio eficiente de streaming de música, al tiempo que ofrece a los artistas emergentes un control directo sobre la distribución de su música. Sin embargo, a pesar de estos logros, el proyecto ha enfrentado desafíos relacionados con la escalabilidad del sistema y la gestión eficiente de los recursos del servidor, especialmente durante picos de alta demanda.

Comparado con otras plataformas de streaming existentes como Spotify, MusicMuse ofrece una menor barrera de entrada para los artistas, lo que representa una ventaja

competitiva significativa. No obstante, la plataforma todavía carece de algunas características avanzadas, como algoritmos sofisticados de recomendación de música y herramientas de análisis de datos para artistas, que podrían mejorar tanto la experiencia del usuario como la visibilidad del artista dentro de la plataforma.

Comparación con la Literatura Existente

La revisión de la literatura destacó la importancia de la interacción social y la personalización en las plataformas de streaming digital. Estudios recientes sugieren que las características sociales, como compartir listas de reproducción y seguir a otros usuarios, aumentan significativamente la retención de usuarios. Aunque MusicMuse incorpora algunas de estas características, como la capacidad de los artistas para gestionar sus perfiles y comunicarse con los fans, hay un margen considerable para expandir estas funcionalidades y alinearlas más estrechamente con las expectativas modernas de los usuarios de redes sociales y plataformas de contenido.

Reflexión sobre las Implicaciones del Proyecto

El desarrollo de MusicMuse tiene implicaciones significativas para la industria de la música digital, especialmente en términos de democratizar la producción y distribución de música. Al eliminar intermediarios, MusicMuse no solo reduce los costos asociados con la publicación de música, sino que también empodera a los artistas al permitirles un control total sobre su contenido y su relación con los oyentes.

Sin embargo, el proyecto también subraya la necesidad de una infraestructura robusta y escalable, junto con una inversión continua en tecnología para soportar las crecientes demandas de los usuarios y la gestión eficiente de grandes volúmenes de datos. Además, el proyecto debe abordar de manera proactiva cuestiones de derechos de autor y compensación justa para los creadores de contenido para asegurar un modelo sostenible y éticamente responsable.

La discusión sobre los resultados de MusicMuse ilustra un paso positivo hacia la innovación en el ámbito del streaming de música, ofreciendo una alternativa viable que favorece a los artistas independientes. Aunque hay áreas que requieren mejoras y expansión, las bases establecidas prometen una evolución continua que podría alterar significativamente el panorama del streaming musical en el futuro. Esta reflexión nos insta a continuar el desarrollo y la mejora de MusicMuse, considerando las tendencias emergentes y las expectativas de los usuarios en el dinámico mercado de la música digital.

Resumen de los Hallazgos Principales

El desarrollo de MusicMuse ha demostrado ser un paso significativo hacia la facilitación del acceso independiente al mercado de la música digital para artistas emergentes. Los hallazgos principales de este proyecto incluyen la capacidad de MusicMuse para ofrecer una plataforma robusta que combina la funcionalidad de streaming de música con herramientas de gestión de perfiles artísticos sin la necesidad de intermediarios. Este enfoque no solo simplifica el proceso de distribución de música, sino que también fortalece la relación directa entre artistas y fans, potenciando así una mayor interacción y participación.

Logros Alcanzados

MusicMuse ha logrado establecer una infraestructura técnica que soporta la carga y reproducción eficiente de contenido musical, así como la gestión de perfiles de usuario. La plataforma ha sido capaz de atraer a un número inicial de usuarios y artistas, validando el interés y la viabilidad del concepto. Además, la implementación de funcionalidades básicas de interacción social ha recibido una respuesta positiva, lo que indica un camino prometedor hacia el desarrollo de una comunidad más activa y comprometida dentro de la plataforma.

7. CONCLUSIONES Y TRABAJO FUTURO

Áreas de Mejora y Trabajo Futuro

A pesar de los logros obtenidos, existen varias áreas que requieren mejoras para el futuro desarrollo de MusicMuse:

Intercomunicación Entre Usuarios:

Actualmente, la plataforma permite una interacción básica entre usuarios y artistas. Para mejorar esto, planeo introducir una funcionalidad completa de red social que permita a los usuarios y artistas interactuar más dinámicamente. Esto incluirá la capacidad de comentar, compartir y reaccionar a la música y a las publicaciones de los artistas, fomentando así una comunidad más viva y conectada.

Creación de Listas Personalizadas:

Ampliaré la funcionalidad de la plataforma para permitir a los usuarios crear sus propias listas de reproducción personalizadas. Esta característica les permitirá guardar sus canciones y artistas favoritos en listas que pueden ser públicas o privadas, mejorando así la experiencia personalizada en la plataforma.

Mercadotecnia y Venta de Música:

Otro aspecto crucial para el desarrollo futuro es la creación de un apartado de mercadotecnia dentro del perfil de cada artista. Este espacio permitirá a los artistas vender su música directamente a los fans, ya sea en formato digital o mediante mercancía relacionada, como vinilos, camisetas y más. Este modelo no solo proporcionará una nueva vía de ingresos para los artistas, sino que también aumentará el valor percibido de la plataforma como un ecosistema musical completo.

Mejoras Tecnológicas y de Seguridad:

Continuar con el desarrollo y la mejora de la infraestructura técnica es fundamental para soportar el crecimiento esperado de la plataforma. Esto incluye optimizar el sistema de almacenamiento y reproducción de música para manejar grandes volúmenes de datos y asegurar la protección de la propiedad intelectual y la privacidad de los usuarios.

En conclusión, MusicMuse ha establecido una base sólida para revolucionar la manera en que los artistas emergentes interactúan con su audiencia y gestionan su música. Los logros obtenidos y los planes para futuras mejoras reflejan un compromiso continuo con la innovación y el apoyo a la comunidad artística. Este proyecto no solo ha demostrado ser una solución viable y necesaria dentro del ámbito de la música digital, sino que también ha sentado las bases para futuras expansiones que podrían mejorar aún más su rendimiento.

8. REFERENCIAS BIBLIOGRAFICAS

getID3

Descripción: getID3() es una biblioteca de PHP para extraer información de archivos de audio y video. Se usa mucho para obtener datos como la duración de los archivos, su calidad, formato, etc.

Referencia:

James Heinrich. (2023). getID3(). Disponible en <https://www.getid3.org>

PHPMailer

Descripción: PHPMailer es una clase de PHP que proporciona una manera completa de enviar emails de forma segura a través de PHP, incluyendo emails con HTML y con archivos adjuntos.

Referencia:

PHPMailer. (2023). *PHPMailer Library*. Disponible en
<https://github.com/PHPMailer/PHPMailer>