Manual del Programador

Integrantes:

Santiago Ibarra

Agustin Colman

Carlos Insaurralde

Gabriel Beneitez

Curso: 7°2

Materia: Proyecto de Desarrollo de Software para Plataformas Móviles

Escuela de Educación Secundaria Técnica N°1
Año 2025

Índice

1. Introducción

Este manual está dirigido a programadores que deseen comprender, mantener o extender el sistema de Reproductor Inteligente de música desarrollado para la materia "Proyecto de Desarrollo de Software para Plataformas Móviles". Aquí se documentan la estructura, los archivos principales, la base de datos y las recomendaciones para el desarrollo.

2. Estructura del Proyecto

El proyecto se compone principalmente de los siguientes archivos y carpetas:

- **ppm_music.html / ppm_music.js**: Interfaz y lógica principal del reproductor.
- **list_songs.php**: Lista todas las canciones desde la base de datos.
- **get_songs.php**: Devuelve las canciones en formato JSON para la app.
- **songs_database.sql**: Script SQL para la base de datos de canciones.
- **Manuales/**: Carpeta de documentación.

3. Base de Datos

La base de datos se encuentra en el archivo `songs_database.sql`. Contiene la tabla `songs` con los campos principales: id, artist, name, year, ppm, audioUrl, entre otros.

4. Explicación de Archivos

Esta sección describe el propósito y funcionamiento de los archivos principales que componen el sistema. Se detallan tanto los archivos de interfaz (HTML/JS) como los del backend (PHP y SQL), explicando su rol, interacción y ejemplos relevantes para el programador.

4.1. index.html

El archivo `index.html` es la página principal del sistema y cumple la función de ser la interfaz de usuario del "Reproductor Inteligente". Este archivo HTML estructura toda la aplicación web y define la experiencia visual para el usuario. A continuación se describe en detalle cada sección del documento:

4.1.1. Estructura General

- **Doctype y Metadatos:**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Reproductor Inteligente</title>
    ...
</head>
```

Define el tipo de documento como HTML5, establece el idioma español y configura metadatos importantes como la codificación UTF-8 y la configuración de viewport para dispositivos móviles.

- **Enlaces a Recursos Externos:**

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.m
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.1
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.b
<script src="ppm_music.js"></script>
```

Importa Bootstrap 5 para el diseño responsivo, Bootstrap Icons para los iconos de la interfaz, y el archivo JavaScript principal `ppm_music.js` que contiene toda la lógica de la aplicación.

4.1.2. Estilos CSS Personalizados

- **Estilo General:**

```
<style>
    body {
        background: linear-gradient(135deg, #1a1a1a, #333);
        color: white;
        min-height: 100vh;
    }
</style>
```

Define un fondo oscuro con degradado para toda la aplicación, texto blanco y altura mínima para ocupar toda la pantalla.

- **Animaciones:**

```
.heart-icon {
    animation: pulse 1s ease infinite;
}
@keyframes pulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.1); }
    100% { transform: scale(1); }
}
```

Crea una animación de latido para el ícono de corazón en el título, utilizando transformaciones CSS y keyframes.

- **Tarjetas de Canciones:**

```
.song-card {
    background: rgba(255, 255, 255, 0.1);
    border: none;
    transition: transform 0.3s ease;
}
.song-card:hover {
    transform: translateY(-5px);
    background: rgba(255, 255, 255, 0.2);
}
```

Estiliza las tarjetas de canciones con un fondo semitransparente y un efecto de elevación al pasar el cursor.

- **Control Deslizante:**

```
.slider-container {
    width: 100%;
    max-width: 500px;
    margin: 2rem auto;
}
.custom-range {
    height: 2rem;
}
.custom-range::-webkit-slider-thumb {
    background: #ff4444;
}
```

Personaliza el control deslizante de PPM, ajustando su tamaño, márgenes y el color del "thumb"(control deslizante).

4.1.3. Componentes Principales de la Interfaz

- **Encabezado (header):**

```
<header class="bg-dark text-white py-3 mb-4">
    <div class="container d-flex justify-content-between align-items-center">
        <h2 class="mb-0">Reproductor Inteligente</h2>
        <nav>
            <a href="list_songs.php" class="btn btn-outline-light ms-2">Mostr
        </nav>
    </div>
</header>
```

Crea una barra de navegación superior con el título de la aplicación y un enlace al listado completo de canciones.

- **Título Principal:**

```
<h1 class="text-center mb-5">
    <span class="heart-icon">[emoji-corazon]</span>
    Reproductor Inteligente
</h1>
```

Muestra el título principal con un ícono de corazón animado.

- **Visualización y Control de PPM:**

```
<div class="text-center mb-4">
    <div class="ppm-display mb-3">
        <span id="ppmValue">140</span> PPM
    </div>
    <div class="slider-container">
        <input type="range" class="form-range custom-range" id="ppmSlider"
               min="100" max="180" value="140">
    </div>
    <button class="btn btn-danger mt-3" id="startSimulation">
        Iniciar Simulación
    </button>
    <button class="btn btn-secondary mt-2 ms-2" id="stopSong" style="display:
        <i class="bi bi-stop-fill"></i> Detener Canción
    </button>
</div>
```

Muestra el valor actual de PPM, un control deslizante para ajustarlo manualmente (rango 100-180), y botones para iniciar/detener la simulación y la reproducción.

- **Tarjeta de Reproducción Actual:**

```
<div class="card song-card text-white" id="nowPlayingCard" style="display:nor
    <div class="card-body">
        <h5 class="card-title" id="nowPlayingTitle"></h5>
        <h6 class="card-subtitle mb-2 text-muted" id="nowPlayingArtist"></h6>
        <p class="card-text" id="nowPlayingYear"></p>
        <p class="card-text mb-1">
            <span id="nowPlayingCurrent">0:00</span> / <span id="nowPlayingDu
            <span class="ms-3">PPM: <span id="nowPlayingPPM">-</span></span>
        </p>
        <div class="d-flex align-items-center gap-2 mb-2">
            <button class="btn btn-light btn-sm" id="nowPlayingPrev"><i class
            <button class="btn btn-light btn-sm" id="nowPlayingPause"><i clas
```

```
                <button class="btn btn-light btn-sm" id="nowPlayingSkip"><i class
                <input type="range" id="nowPlayingSeek" value="0" min="0" max="10
            </div>
            <div class="d-flex align-items-center gap-2">
                <label for="nowPlayingVolume" class="form-label mb-0">Volumen</la
                <input type="range" class="form-range custom-range" id="nowPlayin
            </div>
        </div>
    </div>
```

Muestra información detallada de la canción en reproducción, incluyendo controles de reproducción (anterior, pausa/play, siguiente), barra de progreso y control de volumen. Esta tarjeta está oculta inicialmente y se muestra cuando se reproduce una canción.

- **Contenedor de Recomendaciones:**

```
<div class="row mt-5" id="songRecommendations" style="display:none">
    <!-- Las recomendaciones de canciones se insertarán aquí -->
</div>
```

Contenedor donde se mostrarán dinámicamente las recomendaciones de canciones según el PPM actual. Inicialmente está oculto y se rellena mediante JavaScript.

- **Historial de Reproducción:**

```
<div class="position-absolute top-0 end-0 p-3" style="z-index:2000; width:320
    <div class="card bg-dark text-white shadow-sm" id="historyCard" style="ma
        <div class="card-header py-2 px-3" style="font-size:1.1rem; font-weig
            Reproducidas
        </div>
        <ul class="list-group list-group-flush" id="historyList" style="backg
    </div>
</div>
```

Muestra un panel flotante en la esquina superior derecha con el historial de canciones reproducidas. Tiene una altura máxima y scroll vertical para manejar listas largas. Inicialmente está oculto y se actualiza dinámicamente.

4.1.4. Integración con JavaScript

El archivo HTML proporciona la estructura y los elementos DOM que serán manipulados por `ppm_music.js`. Los elementos tienen IDs específicos que son referenciados en el código JavaScript para actualizar la interfaz dinámicamente. Por ejemplo:

- `ppmSlider` y `ppmValue` para el control y visualización de PPM

- `startSimulation` y `stopSong` para los botones de control
- `nowPlayingCard` y sus elementos hijos para la información de reproducción actual
- `songRecommendations` para las tarjetas de canciones recomendadas
- `historyCard` y `historyList` para el historial de reproducción

4.2. ppm_music.js

El archivo `ppm_music.js` es el componente principal de JavaScript que controla toda la lógica de la aplicación. Este script maneja la simulación de PPM (pulsaciones por minuto), la recomendación de canciones basada en el PPM actual, y la reproducción de audio.

4.2.1. Índice de Funciones

A continuación se presenta un índice detallado de todas las funciones implementadas en `ppm_music.js`:

1. **setAudioVolume():** Actualiza el volumen del audio actual según el valor del control deslizante.
2. **startAutoPlaySongs():** Inicia la reproducción automática de canciones basada en el PPM actual.
3. **stopAutoPlaySongs():** Detiene la reproducción automática y limpia los temporizadores.
4. **playNextAutoSong():** Selecciona y reproduce la siguiente canción automáticamente basada en el PPM actual.
5. **playSongAuto(audioUrl):** Reproduce una canción automáticamente y configura su comportamiento al finalizar.
6. **startSimulation():** Inicia la simulación de variaciones de PPM, simulando el ritmo cardíaco durante ejercicio.
7. **stopSimulation():** Detiene la simulación de PPM.
8. **updatePPMDisplay(value):** Actualiza el valor mostrado de PPM en la interfaz.
9. **updateRecommendations(currentPPM):** Actualiza las recomendaciones de canciones basadas en el PPM actual.
10. **playSong(audioUrl, buttonElement):** Maneja la reproducción manual de una canción seleccionada por el usuario.
11. **showStopButton(show):** Muestra u oculta el botón para detener la reproducción.
12. **showNowPlaying(song, audio):** Muestra la información de la canción en reproducción y actualiza la interfaz.
13. **hideNowPlaying():** Oculta la información de la canción en reproducción.

14. **formatTime(seconds):** Formatea el tiempo en segundos a formato MM:SS para la visualización.

15. **addToHistory(song):** Añade una canción al historial de reproducción.

16. **renderHistory():** Actualiza la visualización del historial de reproducción en la interfaz.

4.2.2. Detalles de Implementación

- **Gestión de datos de canciones:**

```
// Base de datos de canciones (simulada)
const songs = [];

// Cargar las canciones desde la base de datos
fetch('get_songs.php')
    .then(response => response.json())
    .then(data => {
        if (Array.isArray(data)) {
            songs.push(...data);
        }
    })
```

Inicializa un array `songs` que se llena mediante una petición AJAX a `get_songs.php`. Cada canción tiene información como artista, nombre, PPM, año y URL del audio.

- **Control de audio:**

```
function playSong(audioUrl, buttonElement) {
    // Si se está reproduciendo otra canción, detenerla
    if (currentAudio && currentAudioUrl !== audioUrl) {
        currentAudio.pause();
        currentAudio.currentTime = 0;
        if (currentAudioButton) {
            currentAudioButton.innerHTML = '<i class="bi bi-play-fill"></i> P
        }
        currentAudio = null;
    }

    // Si no hay audio o es diferente, crear uno nuevo
    if (!currentAudio) {
        currentAudio = new Audio(audioUrl);
        currentAudioUrl = audioUrl;
        currentAudioButton = buttonElement;
        setAudioVolume();
        currentAudio.play();
        buttonElement.innerHTML = '<i class="bi bi-pause-fill"></i> Pausar';
```

```
            // Buscar canción por URL
            const song = songs.find(s => s.audioUrl === audioUrl);
            if (song) {
                showNowPlaying(song, currentAudio);
                addToHistory(song);
            }

            // Cuando termine la canción
            currentAudio.onended = () => {
                buttonElement.innerHTML = '<i class="bi bi-play-fill"></i> Reproc
                currentAudio = null;
                hideNowPlaying();
            };
        }
        // ...
    }
```

Maneja la reproducción de canciones con funciones como `playSong()`, `playSongAuto()`, controla el volumen mediante un slider, y gestiona la pausa y detención de la reproducción.

- **Simulación de PPM:**

```
function startSimulation() {
    let direction = 1;
    let currentPPM = parseInt(ppmSlider.value);
    let intensityFactor = 1;

    simulationInterval = setInterval(() => {
        // Simular variaciones naturales en el PPM para corredores
        const baseVariation = Math.random() * 5; // Mayor variación para refl
        const variation = baseVariation * intensityFactor;
        currentPPM += variation * direction;

        // Cambiar dirección y ajustar intensidad ocasionalmente
        if (Math.random() < 0.15) {
            direction *= -1;
            // Ajustar factor de intensidad basado en el nivel de ejercicio
            intensityFactor = 0.8 + Math.random() * 0.4; // Varía entre 0.8 y
        }

        // Mantener PPM dentro del rango para corredores (100-180)
        currentPPM = Math.min(Math.max(currentPPM, 100), 180);

        ppmSlider.value = Math.round(currentPPM);
        updatePPMDisplay(currentPPM);
        updateRecommendations(currentPPM);
    }, 1000);
```

```
}
```

Implementa `startSimulation()` y `stopSimulation()` para simular cambios en el ritmo cardíaco. Varía el PPM de forma natural entre 100-180 PPM, simulando el comportamiento durante ejercicio, y actualiza la interfaz en tiempo real con los valores simulados.

- **Sistema de recomendaciones:**

```
function updateRecommendations(currentPPM) {
    // Encontrar canciones con PPM similares (±5 PPM para mayor precisión)
    const matchingSongs = songs
        .filter(song => Math.abs(song.ppm - currentPPM) <= 5)
        .sort((a, b) => Math.abs(a.ppm - currentPPM) - Math.abs(b.ppm - curre
        .slice(0, 3); // Mostrar las 3 canciones más cercanas al PPM actual

    recommendationsContainer.innerHTML = matchingSongs.map(song => `
        <div class="col-md-4 mb-4">
            <div class="card song-card text-white">
                <div class="card-body">
                    <h5 class="card-title">${song.name}</h5>
                    <h6 class="card-subtitle mb-2 text-muted">${song.artist}<
                    <p class="card-text">
                        Año: ${song.year}<br>
                        PPM: ${song.ppm}
                    </p>
                    <button class="btn btn-outline-light mt-2" onclick="playS
                        <i class="bi bi-play-fill"></i> Reproducir
                    </button>
                </div>
            </div>
        </div>
    `).join('');
}
```

La función `updateRecommendations()` encuentra canciones con PPM similar al actual (±5 PPM), ordena las canciones por similitud y muestra las 3 más cercanas, generando tarjetas HTML dinámicamente.

- **Reproducción automática:**

```
function startAutoPlaySongs() {
    autoPlayActive = true;
    playNextAutoSong();
}

function playNextAutoSong() {
```

```
    if (!autoPlayActive) return;
    // Tomar el BPM actual simulado del slider
    const currentPPM = parseInt(ppmSlider.value);
    // Buscar canciones similares
    const matchingSongs = songs
        .filter(song => Math.abs(song.ppm - currentPPM) <= 5)
        .sort((a, b) => Math.abs(a.ppm - currentPPM) - Math.abs(b.ppm - curre
    if (matchingSongs.length === 0) {
        // Si no hay ninguna, buscar la más cercana
        const closest = songs.slice().sort((a, b) =>
            Math.abs(a.ppm - currentPPM) - Math.abs(b.ppm - currentPPM))[0];
        if (!closest) return;
        playSongAuto(closest.audioUrl);
    } else {
        // Elegir una aleatoria entre las más cercanas
        const song = matchingSongs[Math.floor(Math.random() *
            Math.min(3, matchingSongs.length))];
        playSongAuto(song.audioUrl);
    }
}
```

Implementa la reproducción automática de canciones basada en el PPM actual durante la simulación, seleccionando canciones con ritmos similares o la más cercana disponible.

- **Historial de reproducción:**

```
function addToHistory(song) {
    // Evitar duplicados consecutivos
    if (playedHistory.length > 0 && playedHistory[0].name === song.name) retu

    // Añadir al principio y limitar a 10 entradas
    playedHistory.unshift(song);
    if (playedHistory.length > 10) playedHistory.pop();

    renderHistory();
}
```

Mantiene un historial de las últimas 10 canciones reproducidas, evitando duplicados consecutivos y actualizando la visualización en tiempo real.

4.2.3. Eventos y Listeners

El archivo también configura varios event listeners para manejar la interacción del usuario:

- **Slider de volumen:** Actualiza el volumen del audio en tiempo real.

- **Slider de PPM:** Actualiza el valor mostrado y las recomendaciones cuando el usuario ajusta el PPM manualmente.
- **Botón de simulación:** Alterna entre iniciar y detener la simulación de PPM.
- **Botón de detener:** Detiene la reproducción de la canción actual.

4.2.4. Flujo de Ejecución

El flujo principal de ejecución del script es:

1. Inicialización de variables y carga de datos de canciones desde el servidor.
2. Configuración de event listeners para controles de la interfaz.
3. Espera de interacción del usuario (ajuste manual de PPM o inicio de simulación).
4. Durante la simulación: actualización periódica del PPM, recomendaciones y reproducción automática.
5. Mantenimiento del historial de reproducción y visualización de información de la canción actual.

4.3. list_songs.php

El archivo `list_songs.php` implementa una página web que muestra la lista completa de canciones disponibles en la base de datos. Combina HTML, CSS y PHP para crear una interfaz de tabla con todas las canciones y sus detalles.

4.3.1. Estructura General

- **Encabezado y Metadatos:**

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista de Canciones</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstr
    ...
</head>
```

  Similar a `index.html`, establece el documento HTML5, idioma español y metadatos básicos.

- **Estilos CSS Personalizados:**

```
<style>
    body { background: #222; color: #fff; }
    th, td { vertical-align: middle!important; border: 1px solid #444 !import
    .table thead { background: #333; color: #ff4444; }
    .table-striped > tbody > tr:nth-of-type(odd) { background-color: #292929;
    .table-striped > tbody > tr:nth-of-type(even) { background-color: #232323
    .table-bordered { border: 2px solid #555 !important; }
    .table thead th { border-bottom: 2px solid #ff4444 !important; font-size:
    .table tbody td { font-size: 1.05rem; color: #f8f8f8; }
    .table tbody td.song-title { color: #ffe082; font-weight: bold; }
    .table tbody td.artist { color: #80cbc4; }
    .table tbody td.id { color: #b39ddb; }
    .table tbody td.ppm { color: #ffab91; font-weight: bold; }
    .table-responsive { box-shadow: 0 2px 16px rgba(0,0,0,0.5); border-radius
    .audio-cell audio { background: #111; border-radius: 5px; }
    .table-hover tbody tr:hover { background-color: #444 !important; }
    .container { max-width: 950px; }
</style>
```

Define un tema oscuro para la tabla de canciones, con colores específicos para diferentes tipos de datos (artista, título, PPM, etc.) y efectos visuales como bordes, sombras y estados hover.

4.3.2. Componentes Principales

- **Encabezado (header):**

```
<header class="bg-dark text-white py-3 mb-4">
    <div class="container d-flex justify-content-between align-items-center">
        <h2 class="mb-0">Lista de Canciones</h2>
        <nav>
            <a href="ppm_music.html" class="btn btn-outline-light ms-2">Volve
        </nav>
    </div>
</header>
```

Crea una barra de navegación superior con el título de la página y un botón para volver al reproductor principal.

- **Tabla de Canciones:**

```
<div class="container">
    <h3 class="mb-4">Todas las canciones</h3>
    <div class="table-responsive">
        <table class="table table-striped table-bordered align-middle" id="so
            <thead>
                <tr>
```

```
                        <th>#</th>
                        <th>Artista</th>
                        <th>Nombre</th>
                        <th>Año</th>
                        <th>PPM</th>
                        <th>Audio</th>
                    </tr>
                </thead>
                <tbody>
                    <!-- Contenido dinámico generado por PHP -->
                </tbody>
            </table>
        </div>
    </div>
```

Define la estructura de la tabla con encabezados para cada columna de datos.

4.3.3. Código PHP para Acceso a Base de Datos

```
<?php
$host = 'localhost';
$dbname = 'songs_database';
$user = 'root';
$pass = '';

try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->query("SELECT artist, name, year, PPM as ppm, audioUrl FROM songs")
    $songs = $stmt->fetchAll(PDO::FETCH_ASSOC);
} catch(PDOException $e) {
    echo '<div class="alert alert-danger">Error de base de datos: ' .
         htmlspecialchars($e->getMessage()) . '</div>';
    $songs = [];
}
?>
```

Este bloque PHP:

- Define los parámetros de conexión a la base de datos MySQL
- Establece una conexión PDO con manejo de errores
- Ejecuta una consulta SQL para obtener todas las canciones
- Almacena los resultados en el array `$songs`
- En caso de error, muestra un mensaje y establece un array vacío

4.3.4. Generación Dinámica de Filas de la Tabla

```
<?php foreach ($songs as $idx => $song): ?>
<tr>
    <td class="id"><?= $idx + 1 ?></td>
    <td class="artist"><?= htmlspecialchars($song['artist'] ?? '-') ?></td>
    <td class="song-title"><?= htmlspecialchars($song['name'] ?? '-') ?></td>
    <td><?= htmlspecialchars($song['year'] ?? '-') ?></td>
    <td class="ppm"><?= htmlspecialchars($song['ppm'] ?? '-') ?></td>
    <td class="audio-cell"><?php if (!empty($song['audioUrl'])): ?>
        <audio controls src="<?= htmlspecialchars($song['audioUrl']) ?>" style="width
    <?php else: ?>-
    <?php endif; ?></td>
</tr>
<?php endforeach; ?>
```

Este bloque:

- Itera sobre el array de canciones obtenido de la base de datos
- Genera una fila de tabla para cada canción
- Aplica clases CSS específicas a cada celda según su tipo de dato
- Utiliza `htmlspecialchars()` para prevenir ataques XSS
- Incluye un reproductor de audio HTML5 para cada canción que tenga URL de audio
- Maneja casos donde faltan datos con el operador de fusión null (`??`)

4.4. list_songs.js

El archivo `list_songs.js` es una versión alternativa para cargar las canciones en la página `list_songs.php` utilizando JavaScript en lugar de PHP directo. Este enfoque permite una mayor flexibilidad y la posibilidad de actualizar la tabla sin recargar la página.

4.4.1. Estructura General

```
document.addEventListener('DOMContentLoaded', function() {
    fetch('get_songs.php')
        .then(response => response.json())
        .then(data => {
            const tbody = document.querySelector('#songsTable tbody');
            tbody.innerHTML = '';
            if (Array.isArray(data)) {
                data.forEach((song, idx) => {
                    const tr = document.createElement('tr');
                    tr.innerHTML = `
                        <td>${idx + 1}</td>
                        <td>${song.artist || '-'}</td>
                        <td>${song.name || '-'}</td>
```

```
                    <td>${song.year || '-'}</td>
                    <td>${song.ppm || '-'}</td>
                    <td>${song.audioUrl ? `<audio controls src='${song.audioUrl}'
                `;
                tbody.appendChild(tr);
            });
        } else {
            tbody.innerHTML = `<tr><td colspan='6'>No se pudieron cargar las canci
        }
    })
    .catch(err => {
        const tbody = document.querySelector('#songsTable tbody');
        tbody.innerHTML = `<tr><td colspan='6'>Error cargando canciones.</td></tr>
    });
});
```

4.4.2. Análisis del Código

- **Evento DOMContentLoaded:**

```
document.addEventListener('DOMContentLoaded', function() {
    // Código principal
});
```

Asegura que el código se ejecute solo después de que el DOM esté completamente cargado, evitando errores por intentar acceder a elementos que aún no existen.

- **Petición Fetch:**

```
fetch('get_songs.php')
    .then(response => response.json())
    .then(data => {
        // Procesamiento de datos
    })
    .catch(err => {
        // Manejo de errores
    });
```

Utiliza la API Fetch para realizar una petición HTTP al endpoint `get_songs.php`, que devuelve los datos de las canciones en formato JSON. La respuesta se procesa en cadena usando promesas (`.then()`).

- **Manipulación del DOM:**

```
const tbody = document.querySelector('#songsTable tbody');
tbody.innerHTML = '';
```

Obtiene una referencia al elemento `tbody` de la tabla y limpia su contenido actual.

- **Generación Dinámica de Filas:**

```
if (Array.isArray(data)) {
    data.forEach((song, idx) => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${idx + 1}</td>
            <td>${song.artist || '-'}</td>
            <td>${song.name || '-'}</td>
            <td>${song.year || '-'}</td>
            <td>${song.ppm || '-'}</td>
            <td>${song.audioUrl ? `<audio controls src='${song.audioUrl}' sty
        `;
        tbody.appendChild(tr);
    });
}
```

Verifica que los datos recibidos sean un array, luego itera sobre cada canción creando un elemento `tr` (fila de tabla) con celdas para cada propiedad. Utiliza el operador OR lógico (`||`) para manejar valores faltantes y un operador ternario para el reproductor de audio. Finalmente, añade cada fila al `tbody`.

- **Manejo de Errores:**

```
} else {
    tbody.innerHTML = `<tr><td colspan='6'>No se pudieron cargar las cancione
}
// ...
.catch(err => {
    const tbody = document.querySelector('#songsTable tbody');
    tbody.innerHTML = `<tr><td colspan='6'>Error cargando canciones.</td></tr
});
```

Maneja dos tipos de errores: cuando los datos no son un array (respuesta inesperada) y cuando ocurre un error en la petición fetch. En ambos casos, muestra un mensaje de error en la tabla.

4.4.3. Ventajas de este Enfoque

- **Separación de responsabilidades:** El backend (`get_songs.php`) solo proporciona datos en formato JSON, mientras que el frontend (`list_songs.js`) se encarga de la presentación.
- **Experiencia de usuario mejorada:** Permite implementar actualizaciones en tiempo real o filtrado sin recargar la página.

- **Mantenibilidad:** Facilita la modificación independiente de la lógica de datos y la presentación.
- **Reutilización:** El mismo endpoint `get_songs.php` puede ser utilizado por diferentes interfaces.

5. Recomendaciones para el Programador

- Mantener la estructura de carpetas organizada.
- Documentar cambios en los archivos principales.
- Hacer respaldos periódicos de la base de datos.
- Seguir buenas prácticas de seguridad en PHP y JS.
- Probar el sistema en diferentes navegadores y dispositivos.

6. Contacto y Créditos

Desarrollado por: Santiago Ibarra, Agustin Colman, Carlos Insaurralde, Gabriel Beneitez.
Curso: 7°2 – Año 2025.

7. Mejoras Implementadas en list_songs.php

La versión actual de `list_songs.php` ha sido mejorada significativamente para ofrecer una experiencia de usuario más completa y funcional. A continuación, se detallan los componentes principales y su implementación.

7.1. Estructura HTML y Diseño

La página utiliza Bootstrap 5 para el diseño responsivo y Bootstrap Icons para los iconos. La estructura principal incluye:

- Encabezado con navegación
- Sección de filtros y búsqueda
- Tabla de canciones
- Sistema de paginación

7.2. Carga de Datos

La carga de datos se realiza mediante una petición AJAX al archivo `get_songs.php`:

```
// Cargar las canciones desde la base de datos
document.addEventListener('DOMContentLoaded', function() {
    fetch('get_songs.php')
        .then(response => response.json())
        .then(songs => {
            if (Array.isArray(songs)) {
                window.allSongs = songs; // Guardar todas las canciones para la búsque
                filteredSongsGlobal = [...songs]; // Inicializar canciones filtradas
                displaySongs(songs);
                setupSearch();
                setupFilters();
                setupPagination();
            } else {
                console.error('Respuesta inesperada:', songs);
                document.getElementById('songsList').innerHTML = `
                    <tr>
                        <td colspan="5" class="text-center">Error al cargar las cancio
                    </tr>
                `;
            }
        })
        .catch(error => {
            console.error('Error cargando canciones:', error);
            document.getElementById('songsList').innerHTML = `
                <tr>
                    <td colspan="5" class="text-center">Error al cargar las canciones
                </tr>
            `;
        });
});
```

Este código:

- Realiza una petición fetch al endpoint `get_songs.php`
- Almacena las canciones en variables globales para su uso posterior
- Llama a funciones para mostrar las canciones y configurar la funcionalidad
- Maneja errores adecuadamente

7.3. Sistema de Filtrado y Búsqueda

El sistema de filtrado implementa múltiples criterios:

```
// Aplicar todos los filtros y ordenamiento
function applyFilters() {
```

```
    if (!window.allSongs) return;

    let filteredSongs = [...window.allSongs];

    // Aplicar filtro de búsqueda por texto
    const searchTerm = document.getElementById('searchInput').value.toLowerCase().trim
    if (searchTerm !== '') {
        filteredSongs = filteredSongs.filter(song =>
            (song.name && song.name.toLowerCase().includes(searchTerm)) ||
            (song.artist && song.artist.toLowerCase().includes(searchTerm))
        );
    }

    // Aplicar filtro por PPM
    const ppmFilter = document.getElementById('filterPPM').value;
    if (ppmFilter !== '') {
        switch (ppmFilter) {
            case 'low':
                filteredSongs = filteredSongs.filter(song => song.ppm >= 100 && song.p
                break;
            case 'medium':
                filteredSongs = filteredSongs.filter(song => song.ppm > 120 && song.pp
                break;
            case 'high':
                filteredSongs = filteredSongs.filter(song => song.ppm > 150 && song.pp
                break;
        }
    }

    // Aplicar ordenamiento
    const sortOption = document.getElementById('sortBy').value;
    if (sortOption !== '') {
        // Código de ordenamiento según diferentes criterios
        // ...
    }

    // Resetear a la primera página cuando se aplican filtros
    currentPage = 1;

    // Mostrar las canciones filtradas
    displaySongs(filteredSongs);
}
```

La configuración de los filtros se realiza mediante event listeners:

```
// Configurar los filtros
function setupFilters() {
    const filterPPM = document.getElementById('filterPPM');
    const sortBy = document.getElementById('sortBy');
```

```
    const resetFilters = document.getElementById('resetFilters');

    if (filterPPM) {
        filterPPM.addEventListener('change', function() {
            applyFilters();
        });
    }

    if (sortBy) {
        sortBy.addEventListener('change', function() {
            applyFilters();
        });
    }

    if (resetFilters) {
        resetFilters.addEventListener('click', function() {
            // Resetear todos los filtros a sus valores por defecto
            document.getElementById('searchInput').value = '';
            document.getElementById('filterPPM').value = '';
            document.getElementById('sortBy').value = '';

            // Mostrar todas las canciones
            if (window.allSongs) {
                displaySongs(window.allSongs);
            }

            // Mostrar notificación
            showToast('Filtros reseteados');
        });
    }
}
```

7.4. Sistema de Paginación

La paginación se implementa para mostrar un número limitado de canciones por página:

```
// Variables globales para la paginación
let currentPage = 1;
const itemsPerPage = 10; // Número de canciones por página
let filteredSongsGlobal = []; // Almacena las canciones filtradas globalmente
```

La función `displaySongs` se encarga de mostrar solo las canciones correspondientes a la página actual:

```
// Calcular el rango de canciones a mostrar según la página actual
const startIndex = (currentPage - 1) * itemsPerPage;
const endIndex = Math.min(startIndex + itemsPerPage, songs.length);
const songsToDisplay = songs.slice(startIndex, endIndex);
```

La actualización de la paginación se realiza dinámicamente:

```
// Actualizar la paginación basada en el número total de elementos
function updatePagination(totalItems) {
    const totalPages = Math.ceil(totalItems / itemsPerPage);
    const pagination = document.getElementById('pagination');

    // Ocultar paginación si solo hay una página
    if (totalPages <= 1) {
        document.getElementById('paginationContainer').style.display = 'none';
        return;
    } else {
        document.getElementById('paginationContainer').style.display = 'flex';
    }

    // Actualizar estado del botón anterior
    const prevPageItem = document.getElementById('prevPage');
    if (currentPage === 1) {
        prevPageItem.classList.add('disabled');
        prevPageItem.querySelector('a').setAttribute('aria-disabled', 'true');
    } else {
        prevPageItem.classList.remove('disabled');
        prevPageItem.querySelector('a').setAttribute('aria-disabled', 'false');
    }

    // Actualizar estado del botón siguiente
    const nextPageItem = document.getElementById('nextPage');
    if (currentPage === totalPages) {
        nextPageItem.classList.add('disabled');
        nextPageItem.querySelector('a').setAttribute('aria-disabled', 'true');
    } else {
        nextPageItem.classList.remove('disabled');
        nextPageItem.querySelector('a').setAttribute('aria-disabled', 'false');
    }

    // Generar los números de página
    const pageItems = [];

    // Determinar el rango de páginas a mostrar
    let startPage = Math.max(1, currentPage - 1);
    let endPage = Math.min(totalPages, startPage + 2);

    // Ajustar si estamos en las últimas páginas
    if (endPage - startPage < 2) {
        startPage = Math.max(1, endPage - 2);
    }

    // Crear elementos de página
    for (let i = startPage; i <= endPage; i++) {
```

```
        const isActive = i === currentPage;
        pageItems.push(‘
            <li class="page-item ${isActive ? ’active’ : ’’}">
                <a class="page-link" href="#" data-page="${i}">${i}</a>
            </li>
        ’);
    }

    // Actualizar el HTML de la paginación
    pagination.innerHTML = ‘
        <li class="page-item ${currentPage === 1 ? ’disabled’ : ’’}" id="prevPage">
            <a class="page-link" href="#" tabindex="-1" aria-disabled="${currentPage =
        </li>
        ${pageItems.join(’’)}
        <li class="page-item ${currentPage === totalPages ? ’disabled’ : ’’}" id="next
            <a class="page-link" href="#">Siguiente</a>
        </li>
    ’;

    // Volver a configurar los event listeners
    setupPagination();
}
```

7.5. Reproducción de Audio

La reproducción de audio se implementa con controles para reproducir y detener:

```
// Función para reproducir una canción
function playSong(audioUrl, button) {
    // Detener cualquier audio que se esté reproduciendo
    if (window.currentAudio) {
        window.currentAudio.pause();
        window.currentAudio.currentTime = 0;

        // Restablecer todos los botones de reproducción/detención
        document.querySelectorAll(’.stop-btn’).forEach(btn => {
            btn.style.display = ’none’;
            btn.previousElementSibling.style.display = ’inline-flex’;
        });
    }

    // Crear y reproducir el nuevo audio
    window.currentAudio = new Audio(audioUrl);
    window.currentAudio.volume = 0.7; // Volumen predeterminado
    window.currentAudio.play();

    // Actualizar botones de reproducción/detención
    if (button) {
        button.style.display = ’none’;
```

```
        const stopButton = button.nextElementSibling;
        stopButton.style.display = 'inline-flex';

        // Cuando la canción termine, restaurar los botones
        window.currentAudio.onended = function() {
            button.style.display = 'inline-flex';
            stopButton.style.display = 'none';
            window.currentAudio = null;
        };
    }
}

// Función para detener la reproducción
function stopSong(button) {
    if (window.currentAudio) {
        window.currentAudio.pause();
        window.currentAudio.currentTime = 0;
        window.currentAudio = null;
    }

    // Restaurar botones
    button.style.display = 'none';
    button.previousElementSibling.style.display = 'inline-flex';
}
```

7.6. Menú de Opciones

Se implementa un menú contextual para cada canción:

```
// Función para mostrar opciones adicionales
function showOptions(button) {
    // Obtener información de la canción desde la fila
    const row = button.closest('tr');
    const title = row.querySelector('.song-title').textContent;
    const artist = row.querySelector('.song-artist').textContent;

    // Crear y mostrar un menú contextual
    const optionsMenu = document.createElement('div');
    optionsMenu.className = 'options-menu';
    optionsMenu.innerHTML = `
        <div class="card bg-dark text-white" style="position: absolute; z-index: 1000
            <div class="card-header">
                <strong>${title}</strong>
                <button type="button" class="btn-close btn-close-white float-end" oncl
            </div>
            <ul class="list-group list-group-flush">
                <li class="list-group-item bg-dark text-white" onclick="addToPlaylist
                    <i class="bi bi-music-note-list me-2"></i> Añadir a playlist
                </li>
```

```
                <li class="list-group-item bg-dark text-white" onclick="downloadSong(
                    <i class="bi bi-download me-2"></i> Descargar
                </li>
                <li class="list-group-item bg-dark text-white" onclick="shareSong('${t
                    <i class="bi bi-share me-2"></i> Compartir
                </li>
            </ul>
        </div>
    `;

    // Posicionar el menú y configurar eventos
    // ...
}
```

7.7. Notificaciones Toast

Se implementa un sistema de notificaciones para informar al usuario:

```
// Función para mostrar notificaciones toast
function showToast(message) {
    // Crear elemento toast
    const toastEl = document.createElement('div');
    toastEl.className = 'toast-notification';
    toastEl.innerHTML = `
        <div class="toast show" role="alert" aria-live="assertive" aria-atomic="true">
            <div class="toast-header bg-dark text-white">
                <strong class="me-auto">Reproductor Inteligente</strong>
                <button type="button" class="btn-close btn-close-white" data-bs-dismis
            </div>
            <div class="toast-body bg-dark text-white">
                ${message}
            </div>
        </div>
    `;

    // Estilo para el contenedor de toast
    toastEl.style.position = 'fixed';
    toastEl.style.bottom = '20px';
    toastEl.style.right = '20px';
    toastEl.style.zIndex = '1050';

    // Añadir al DOM
    document.body.appendChild(toastEl);

    // Eliminar después de 3 segundos
    setTimeout(() => {
        toastEl.remove();
    }, 3000);
}
```

7.8. Recomendaciones para Mantenimiento y Mejoras

Para desarrolladores que deseen mantener o extender la funcionalidad de `list_songs.php`, se recomiendan las siguientes prácticas:

- **Separación de Responsabilidades:** Mantener la separación entre la obtención de datos (get_songs.php), la presentación (HTML) y la lógica de interacción (JavaScript).
- **Manejo de Errores:** Implementar manejo de errores robusto tanto en el lado del cliente como del servidor.
- **Optimización de Rendimiento:** Para grandes conjuntos de datos, considerar implementar paginación del lado del servidor.
- **Accesibilidad:** Asegurar que todos los elementos interactivos sean accesibles mediante teclado y lectores de pantalla.
- **Pruebas:** Realizar pruebas exhaustivas en diferentes navegadores y dispositivos.
- **Posibles Mejoras:**
  - Implementar almacenamiento de preferencias de usuario (filtros favoritos)
  - Añadir funcionalidad de arrastrar y soltar para crear playlists
  - Implementar estadísticas de reproducción
  - Añadir visualizaciones de datos (gráficos de PPM vs. popularidad)

8. Integración con el Sistema Principal

La integración del Reproductor Inteligente con el sistema principal de la aplicación es un aspecto crucial para su correcto funcionamiento. Esta sección detalla los aspectos técnicos de esta integración, los puntos de conexión entre componentes y las consideraciones para desarrolladores que necesiten modificar o extender estas interacciones.

8.1. Arquitectura de Integración

El Reproductor Inteligente se integra con el sistema principal a través de una arquitectura cliente-servidor donde:

- **Frontend (Cliente):** Implementado en HTML, CSS y JavaScript puro, se encarga de la interfaz de usuario y la lógica de interacción.
- **Backend (Servidor):** Desarrollado en PHP, gestiona el acceso a la base de datos MySQL y proporciona endpoints para que el frontend pueda obtener y manipular datos.
- **Base de Datos:** Almacena toda la información relacionada con canciones, usuarios y preferencias.

8.2. Puntos de Integración

Los principales puntos de integración entre componentes son:

- **API REST:** El frontend se comunica con el backend a través de llamadas AJAX a endpoints PHP:

```
// Ejemplo de llamada desde el frontend
fetch('get_songs.php')
    .then(response => response.json())
    .then(data => {
        // Procesar datos recibidos
    });
```

- **Eventos del Sistema:** El reproductor responde a eventos del sistema principal como cambios de usuario, actualizaciones de biblioteca o modificaciones en las preferencias:

```
// Suscripción a eventos del sistema principal
window.addEventListener('userPreferencesChanged', function(event) {
    updateUserInterface(event.detail);
});
```

- **Almacenamiento Compartido:** Tanto el reproductor como el sistema principal comparten el acceso a:
  - Base de datos MySQL para datos persistentes
  - LocalStorage para preferencias de usuario en el navegador
  - SessionStorage para datos temporales de la sesión actual

8.3. Flujo de Datos

El flujo de datos entre el Reproductor Inteligente y el sistema principal sigue este patrón:

1. El usuario interactúa con la interfaz del reproductor (ajusta PPM, inicia reproducción, etc.)
2. El frontend JavaScript procesa la interacción y determina qué datos necesita
3. Se realiza una solicitud AJAX al endpoint PHP correspondiente
4. El backend PHP consulta la base de datos y devuelve los resultados en formato JSON
5. El frontend recibe los datos y actualiza la interfaz de usuario
6. Si es necesario, se notifica al sistema principal sobre cambios relevantes

8.4. Consideraciones para Desarrolladores

Al modificar o extender la integración entre el Reproductor Inteligente y el sistema principal, se deben tener en cuenta las siguientes consideraciones:

- **Compatibilidad de Versiones:** Asegurar que las modificaciones en un componente no afecten negativamente a otros.
- **Seguridad:** Validar todas las entradas de usuario tanto en el frontend como en el backend para prevenir inyecciones SQL o XSS.
- **Rendimiento:** Optimizar las consultas a la base de datos y minimizar las llamadas AJAX innecesarias.
- **Escalabilidad:** Diseñar las modificaciones pensando en el crecimiento futuro del sistema.
- **Documentación:** Mantener actualizada la documentación de los endpoints y estructuras de datos compartidas.

8.5. Pruebas de Integración

Para garantizar que las modificaciones no afecten la integración, se recomienda realizar las siguientes pruebas:

- **Pruebas de Endpoints:** Verificar que todos los endpoints devuelvan los datos esperados en el formato correcto.
- **Pruebas de Interfaz:** Comprobar que la interfaz de usuario responda correctamente a los cambios en los datos.
- **Pruebas de Rendimiento:** Medir el tiempo de respuesta de las operaciones críticas bajo diferentes condiciones de carga.
- **Pruebas de Compatibilidad:** Verificar el funcionamiento en diferentes navegadores y dispositivos.

9. Conclusión

Este manual del programador ha proporcionado una documentación detallada del sistema Reproductor Inteligente, abarcando desde su estructura general hasta los detalles técnicos de implementación y las recomendaciones para su mantenimiento y extensión.

Los desarrolladores que trabajen con este sistema ahora cuentan con:

- Una comprensión clara de la arquitectura y componentes del sistema
- Documentación detallada de los archivos principales y sus funcionalidades
- Explicación de la estructura de la base de datos y su interacción con la aplicación
- Guías para la integración con otros sistemas

- Recomendaciones para el mantenimiento y mejora del código

El Reproductor Inteligente es un sistema diseñado con principios de modularidad y escalabilidad, lo que permite su fácil extensión para incorporar nuevas funcionalidades en el futuro. Se recomienda a los desarrolladores seguir las convenciones de código establecidas y documentar adecuadamente cualquier modificación realizada.

La tecnología evoluciona constantemente, por lo que este sistema deberá adaptarse a nuevas necesidades y avances tecnológicos. Con una base sólida y una documentación completa, el Reproductor Inteligente está preparado para crecer y mejorar con el tiempo, proporcionando una experiencia musical cada vez más personalizada y adaptada a las necesidades de los usuarios.