Proyecto "*UN MAR DE MÚSICAS*"

-Descripción general:

Esta aplicación gestiona a los artistas y/o bandas que se registren en el sitio, permitiéndoles subir sus canciones, crear álbumes, gestionar su contenido y su perfil. Por el lado del oyente puede acceder a las canciones, mediante un registro y log-in previo, para contar con un buscador general que busca entre los álbumes, canciones o artistas que se deseen, además les permite reproducir las canciones y acceder a los perfiles de los artistas/bandas.

-Link del repositorio de GitHub:
https://github.com/Tomy142/Reg_Log

-Arquitectura:

Este proyecto utiliza el Modelo Vista Controlador, para facilitar mantenimiento, escalabilidad del sistema y separar responsabilidades de forma adecuada.

Backend:

Lenguaje: JavaScript ejecutado sobre Node.js.

Base de datos: MySQL ejecutada con XAMPP.

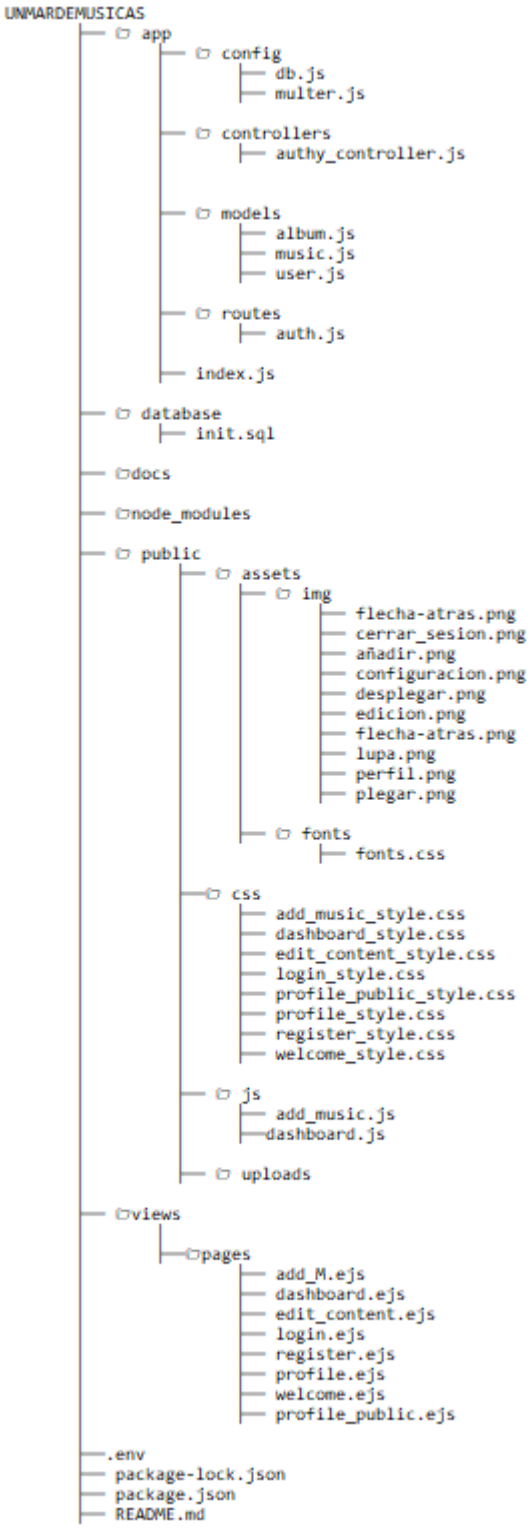
Frontend:

Lenguajes: HTML, CSS y JavaScript.

Frameworks: utiliza Ejs para el renderizado dinámico de las vistas.

Diseño: Simple, cuenta con iconos y descripciones fáciles de identificar.

-Estructura del proyecto:



-Experiencia del usuario:

Artista/Banda: Puede subir su canción, con nombre y día de lanzamiento y el archivo de reproducción, gestiona el contenido de la misma permitiendo modificar sus campos y eliminar la canción. Además, puede crear álbumes, vincular canciones ya subidas para sus álbumes, modificar su nombre, fecha de lanzamiento, eliminar álbum.

Por otro lado, le permite gestionar su perfil, añadir una foto de perfil, sus links a redes sociales y su cvu/cbu si desea colaboraciones.

Oyente: Le permite navegar a través de un buscador de contenido, donde puede acceder a canciones, álbumes, artistas/bandas relacionadas, le permite reproducir el tema que desee, la descarga está restringida, además puede acceder a los perfiles de los artistas/bandas, redireccionándolos a sus perfiles donde aparecen los detalles del artista/banda, nombre, redes sociales, cvu/cbu para donaciones, canciones, álbumes.

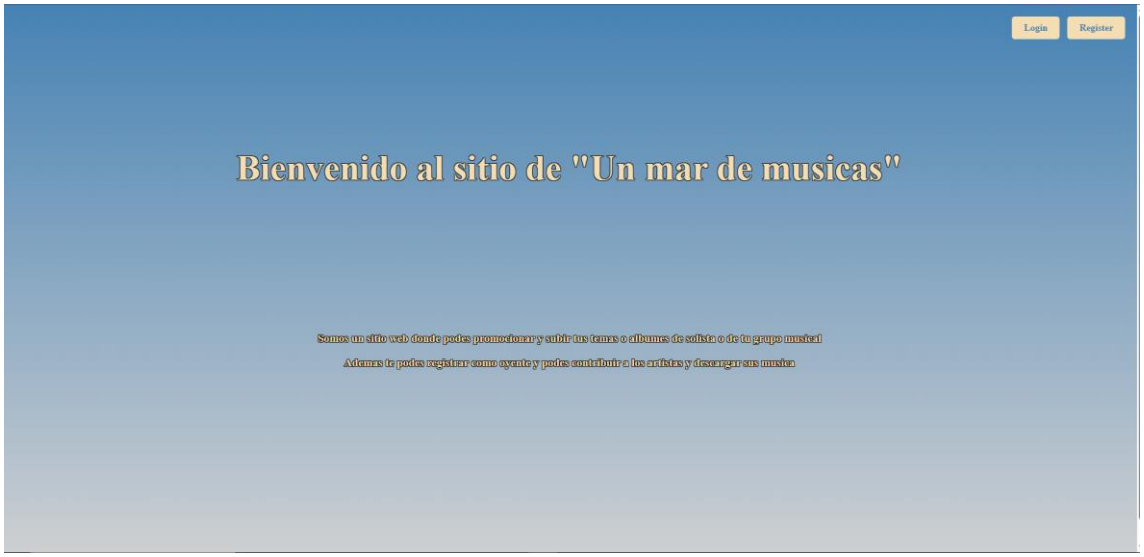
-Tecnologías usadas:

- Node.js: entorno de ejecución para el backend.
- MySQL: sistema de gestión de bases de datos.
- XAMPP: herramienta para administrar MySQL de forma local.
- EJS: motor de plantillas para renderizar las vistas dinámicamente.
- Multer: Middleware para manejar la subida de archivos.

GUI del sitio

Página de Bienvenida:

Cuenta con dos botones, uno de register, para usuarios nuevos y otro de log-in para usuarios ya creados



Formulario de registro:

Al registrase pide como condición elegir un “rol”, ya sea oyente, banda o artista, determina las opciones disponibles en la página.

Archivo función register en authy_controller.js:

Se encarga de validar la contraseña, el rol elegido, el mail, si todo es válido redirige al login.

```
const register =async(req, res)=>{
    const {user, password, confirm_password, email, role} = req.body;
    try{

        if(password!== confirm_password)
        {
            return res.render("register", {error: "Las contraseñas no coinciden."});
        }

        const validRoles =["listener", "artist", "band"];
        if(!validRoles.includes(role)){
            return res.render("register", {error: "Rol inválido."});
        }

        const existingUser = await userModel.findUserByEmail(email);
        if (existingUser){
            return res.render("register",{error: "El email ya está registrado."});
        }
        const [userCheck] = await db.execute("SELECT * FROM users WHERE username = ?",[user]);
        if(userCheck.length >0){
            return res.render("register", {error: "El nombre de usuario ya está registrado."});
        }

        const userId = await userModel.createUser(user, password, email, role);
        res.redirect("/login");
    }catch(error){
        console.error(error);
        res.render("register", {error: "Error al registrar usuario. Intenta de nuevo."});
    }
};
```

En auth.js la función createUser se encarga de hashear la contraseña y enviar al query a la base de datos.

```
const createUser = async(username, password, email, role)=>{
    const hashedPassword = await bcrypt.hash(password, 10);
    const[result] = await db.execute(
        "INSERT INTO users ( username, password, email, role) VALUES (?,?,?,?)",
        [username, hashedPassword, email, role]
    );
    return result.insertId;
};
```

Inicio de Sesión:

Es muy intuitivo, pide los campos de Usuario y contraseña ya creados.

Al iniciar sesión la salida por consola se muestra de la siguiente forma. **Cabe aclarar que, al realizar cualquier cambio, en cuanto a redirección, subida de canciones, edición de las misma, la salida por consola es la misma. Vuelve a validar las credenciales después de cada acción.**

```
Credenciales correctas, generando token
Token valido, usuario: {
  id: 3,
  username: 'Juan Perez',
  role: 'band',
  iat: 1741635815,
  exp: 1741639415
}
```

Función login en authy_controller.js

Valida las credenciales, de ser correctas, redirige al dashboard correspondiente.

```
const login = async (req, res) =>{
    const {username, password} = req.body;
    try{
        const user = await userModel.findUserByUsername(username);
        if(!user || !(await bcrypt.compare(password, user.password))){
            return res.render("login", {error:"Usuario y/o contraseña incorrectos."});
        }
        console.log("Credenciales correctas, generando token");
        const token= jwt.sign(
            {id: user.id, username: user.username, role:user.role},
            process.env.JWT_SECRET,
            {expiresIn: "1h"}
        );
        res.cookie("token", token,{httpOnly:true,secure: false});
        res.redirect("/dashboard");
    }catch(error){
        console.error(error);
        res.render("login", {error:"Usuario y/o contraseña incorrectos."});
    }
}
```

Dashboard para bandas/artistas:

Cuenta con botones de añadir contenido, editar contenido, perfil y cerrar sesión. Además, si cuenta con canciones ya subidas o álbumes creados, se muestran de la siguiente forma:

De lo contrario, aparece de la siguiente forma:

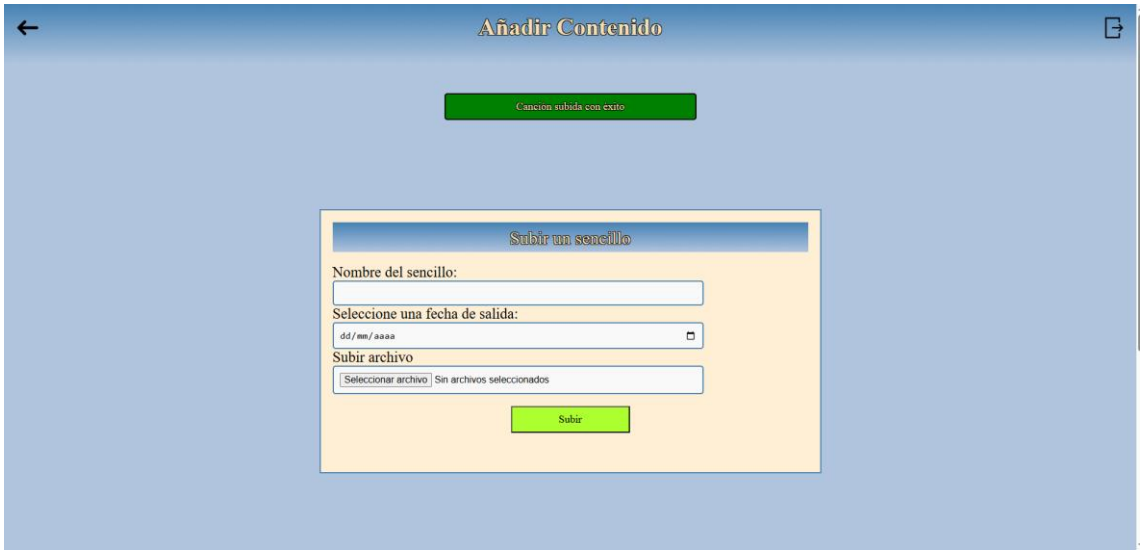
De esta forma valida el rol y despliega las canciones o álbumes si los hay.

```
const dashboard = async (req, res)=>{
    const userId = req.user.id;
    try{
        if (req.user.role === 'listener'){
            res.render("dashboard", {user: req.user});
        } else {
            const songsRaw = await musicModel.getMusicByUser(userId);
            const songs = songsRaw.map(song =>({
                ...song,
                created_at: song.created_at.toISOString().split('T')[0]
            }));

            const albumsRaw = await albumModel.getAlbumsByUser(userId);
            const albums = await Promise.all(albumsRaw.map(async (album) =>{
                const albumSongsRaw = await albumModel.getSongsInAlbum(album.id, userId);
                const albumSongs = albumSongsRaw.map(song =>({
                    ...song,
                    created_at: song.created_at.toISOString().split('T')[0]
                }));
                return {...album, songs: albumSongs};
            }));

            res.render("dashboard", {songs, albums, user: req.user});
        }
    }catch(error){
        console.error("Error al cargar el dashboard:", error);
        res.render("dashboard",{
            songs: [],
            albums: [],
            user: req.user,
            error: "Error al cargar el dashboard: " + error.message
        });
    }
};
```

Sección agregar Contenido(artistas/bandas)

Cuenta con una flecha para volver al dashboard y dos secciones, una para agregar un sencillo, con sus respectivos campos y otra para la creación de álbumes.

Al Subir una canción correctamente muestra el siguiente cartel.



Del lado del backend usamos la librería multer, creando un archivo muler.js para manejar donde se almacenan los sencillos y que tipo de audios son permitidos.

```
import multer from 'multer';
import path from 'path';
import { fileURLToPath } from 'url';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const storage = multer.diskStorage({
    destination: (req, file, cb)=>{
        cb(null, "public/uploads/");
    },
    filename: (req, file, cb)=>{
        const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() *1e9);
        cb(null, `${file.fieldname}-${uniqueSuffix}${path.extname(file.originalname)}`);
    },
});

const musicFileFilter = (req, file, cb) => {
    const allowedTypes = ['audio/mpeg', 'audio/wav'];
    if (allowedTypes.includes(file.mimetype)) {
        cb(null, true);
    } else {
        cb(new Error('Solo se permiten archivos MP3 O WAV'), false);
    }
};
```

Función addMusic en authy_controller.js:

Controla que se haya subido un archivo valido y lo carga en la carpeta uploads.

```
const addMusic = async(req, res) =>{
    const{song_name, release_date}= req.body;
    const file = req.file;
    try{
        if(!file){
            return res.render("add_M", {error: "Debes subir un archivo."});
        }
        const userId = req.user.id;
        const filePath =`/uploads/${file.filename}`;
        await musicModel.createMusic(song_name, filePath, userId);
        res.render("add_M", {success: "Canción subida con éxito"});
    }catch(error){
        console.error("Error al añadir musica:", error.message);
        if(error.code === "LIMIT_FILE_SIZE"){
            res.render("add_M", { error: " El archivo excede el limite de 10 MB."});
        }
        res.render("add_M", { error: error.message});
    }
};
```

Por otro lado, al crear un álbum, aparece este cartel:

Del lado de la consola, imprime los siguientes, datos y como se inserta en la base de datos.

```
Creando álbum para usuario ID: 3
Datos del álbum: { album_name: 'LALALA', release_date: '2025-03-10' }
Ejecutando INSERT con: { title: 'LALALA', releaseDate: '2025-03-10', userId: 3 }
Resultado del INSERT: ResultSetHeader {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 9,
  info: '',
  serverStatus: 2,
  warningStatus: 0,
  changedRows: 0
}
Álbum creado con ID: 9
```

Edición de contenido (Álbumes y sencillos):

Carga las canciones ya subidas, permite modificarles cada uno de sus campos y eliminarlas.

Los álbumes pueden modificarse de la misma forma, pero además cuentan con la vinculación o desvinculación de las canciones a los álbumes.

En la función editContent de authy_controller.js muestra un ejemplo de cómo se actualiza una canción llamando al método updateMusic.

```
if (action === "update_song" && song_id) {
    await musicModel.updateMusic(song_id, userId, song_name, song_date);
    const songsRaw = await musicModel.getMusicByUser(userId);
    const songs = songsRaw.map(song => ({
        ...song,
        created_at: song.created_at.toISOString().split('T')[0]
    }));
    const albumsRaw = await albumModel.getAlbumsByUser(userId);
    const albumsWithSongs = await Promise.all(albumsRaw.map(async (album) => {
        const albumSongsRaw = await albumModel.getSongsInAlbum(album.id, userId);
        const albumSongs = albumSongsRaw.map(song => ({
            ...song,
            created_at: song.created_at.toISOString().split('T')[0]
        }));
        return { ...album, songs: albumSongs };
    }));
    return res.render("edit_content", {
        songs,
        albums: albumsWithSongs,
        allSongs,
        user: req.user,
        success: "Canción actualizada correctamente."
    });
}
```

Perfil: Configuración del perfil del artista/banda:

La salida por consola es la siguiente:

Valida el token del usuario, imprime que datos se recibieron, el dato que se recibió, en este caso solamente agregue una foto, las redes sociales aparecen nulas ya que no se realizó ningún cambio.

```
}
Token valido, usuario: {
  id: 3,
  username: 'Juan Perez',
  role: 'band',
  iat: 1741635815,
  exp: 1741639415
}
Datos recibidos - req.body: [Object: null prototype] {
  instagram: '',
  facebook: '',
  twitter: '',
  youtube: '',
  cbu_cvu: ''
}
Archivo recibido - req.file: {
  fieldname: 'upload_pic',
  originalname: 'IMG_20250307_181544~2.jpg',
  encoding: '7bit',
  mimetype: 'image/jpeg',
  destination: 'public/uploads/',
  filename: 'upload_pic-1741636677167-254235239.jpg',
  path: 'public\\uploads\\upload_pic-1741636677167-254235239.jpg',
  size: 427032
}
profileData antes de iterar: {
  instagram_link: null,
  facebook_link: null,
  twitter_link: null,
  youtube_link: null,
  cvu_cbu: null,
  profile_picture: '/uploads/upload_pic-1741636677167-254235239.jpg'
}
```

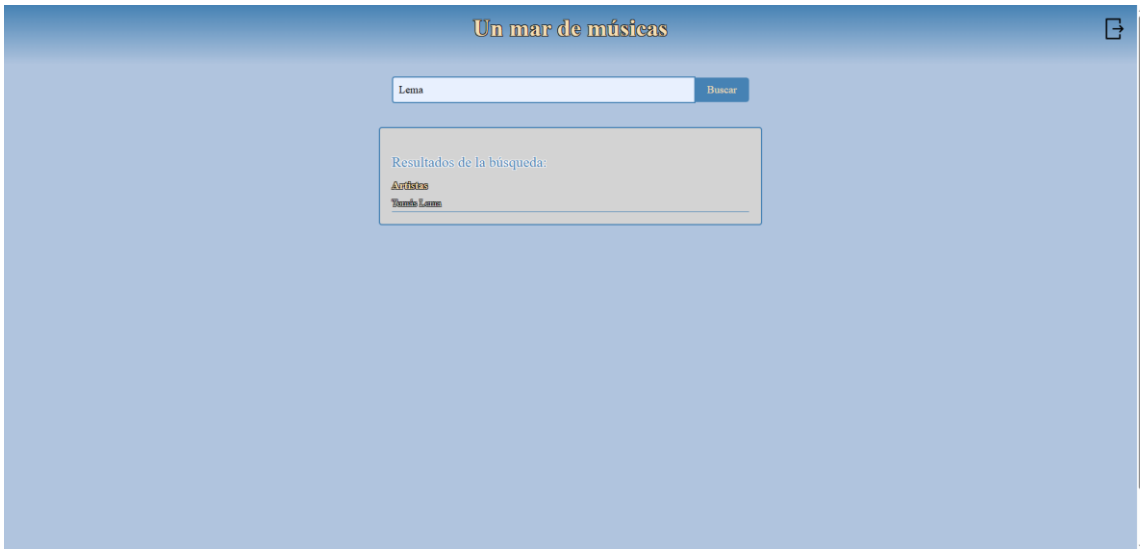
Dashboard del Oyente:

Cuenta con una barra de búsqueda la cual busca entre los artistas, bandas, canciones y álbumes con el criterio deseado, en este ejemplo busco un usuario, y como resultado tiene como rol de artista. Tiene la posibilidad de entrar al perfil clickeando en el nombre.

En la función search de authy_controller.js:

Se realiza la búsqueda del contenido según lo que pide la query y se imprime en pantalla el resultado.

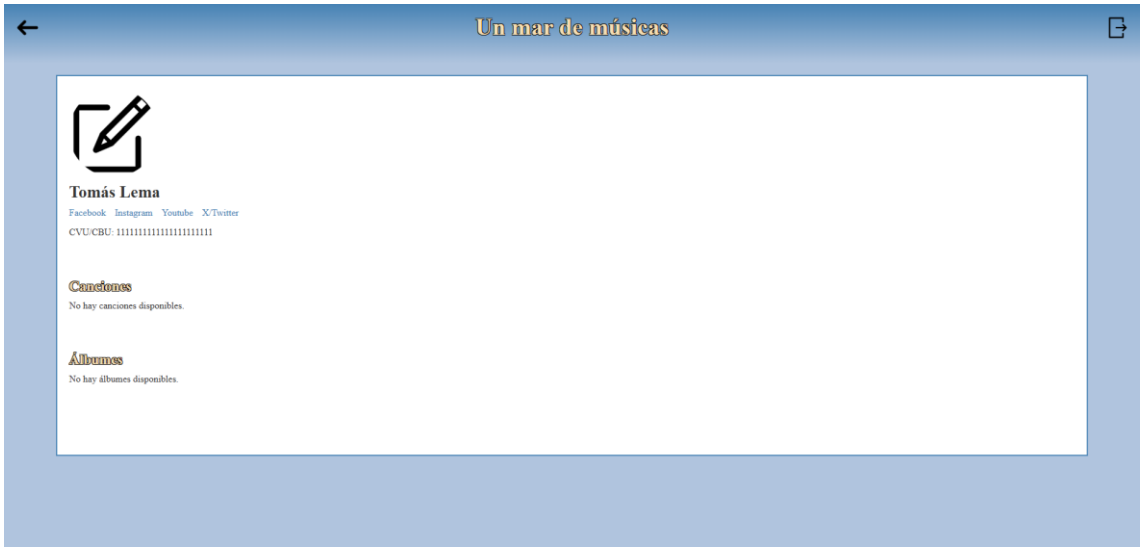
```
const search = async (req, res) =>{
    const { query } = req.query;
    try{
        if (!query || query.trim() === ""){
            return res.json({
                songs: [],
                albums: [],
                artists: [],
                bands: [],
                error: "Ingrese un término de búsqueda."
            });
        }
        const searchTerm = `%${query.trim()}%`;

        const [songsRaw] = await db.execute(
            `SELECT m.*, u.username
            FROM music m
            JOIN users u ON m.user_id = u.id
            WHERE m.title LIKE ? AND (u.role = 'artist' OR u.role = 'band')`,
            [searchTerm]
        );
        const songs = songsRaw.map(song => ({
            ...song,
            created_at: song.created_at.toISOString().split('T')[0]
        }));

        const [albumsRaw] = await db.execute(
            `SELECT a.*, u.username
            FROM albums a
            JOIN users u ON a.user_id = u.id
            WHERE a.title LIKE ? AND (u.role = 'artist' OR u.role = 'band')`,
            [searchTerm]
        );
        const albums = albumsRaw.map(album =>({
            ...album,
            release_date: album.release_date.toISOString().split('T')[0]
        }));

        const [artistsRaw] = await db.execute(
            "SELECT username, profile_picture FROM users WHERE role = 'artist' AND username LIKE ?",
            [searchTerm]
        );
        const artists = artistsRaw;

        const [bandsRaw] = await db.execute(
            "SELECT username, profile_picture FROM users WHERE role = 'band' AND username LIKE ?",
            [searchTerm]
        );
        const bands = bandsRaw;

        res.json({songs, albums, artists, bands });
    } catch (error) {
        console.error("Error en la búsqueda:", error);
        res.json({
            songs: [],
            albums: [],
            artists: [],
            bands: [],
            error: "Error al realizar la búsqueda: " + error.message
        });
    }
};
```



Al entrar al perfil del artista o banda, aparece su foto de perfil, su nombre, sus redes sociales y su CVU/CBU para las donaciones, todo previamente configurado por el artista/banda.

Por último, también muestra todo el contenido subido por el artista ya sean canciones o álbumes, además, permite reproducir las canciones y ajustar la velocidad de reproducción y su volumen.

En caso de no haber subido contenido:



Por el lado de la base de datos:

Utilicé un archivo init.sql donde crea las tablas con cada columna con los datos que requiere.

Indexa el role y user_id ya que son utilizados en constantemente y eso les permite que se realicen las consultas más rápidas.

```sql
USE unmardemusicas;

CREATE TABLE users(
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    role ENUM('listener', 'artist', 'band') NOT NULL,
    profile_picture VARCHAR(255),
    facebook_link VARCHAR(255),
    instagram_link VARCHAR(255),
    youtube_link VARCHAR(255),
    twitter_link VARCHAR(255),
    cvu_cbu VARCHAR(50),
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE music(
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    file_path VARCHAR(255) NOT NULL,
    user_id INT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE albums(
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    release_date DATE NOT NULL,
    user_id INT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(user_id) REFERENCES users(id) ON DELETE CASCADE
);

CREATE TABLE album_songs(
    album_id INT NOT NULL,
    song_id INT NOT NULL,
    PRIMARY KEY (album_id, song_id),
    FOREIGN KEY (album_id) REFERENCES albums(id) ON DELETE CASCADE,
    FOREIGN KEY (song_id) REFERENCES music(id) ON DELETE CASCADE
);

CREATE INDEX idx_users_role ON users(role);
CREATE INDEX idx_music_user ON music(user_id);
CREATE INDEX idx_albums_user ON albums(user_id);
```