# Aplicativo Web - Reproductor de música

Daniel Santiago Calero Beltrán

Héctor Iván Díaz Villa

Laura López Arbeláez

Santiago Sánchez Muñoz.

Facultad de Ingeniería, Universidad Santiago de Cali

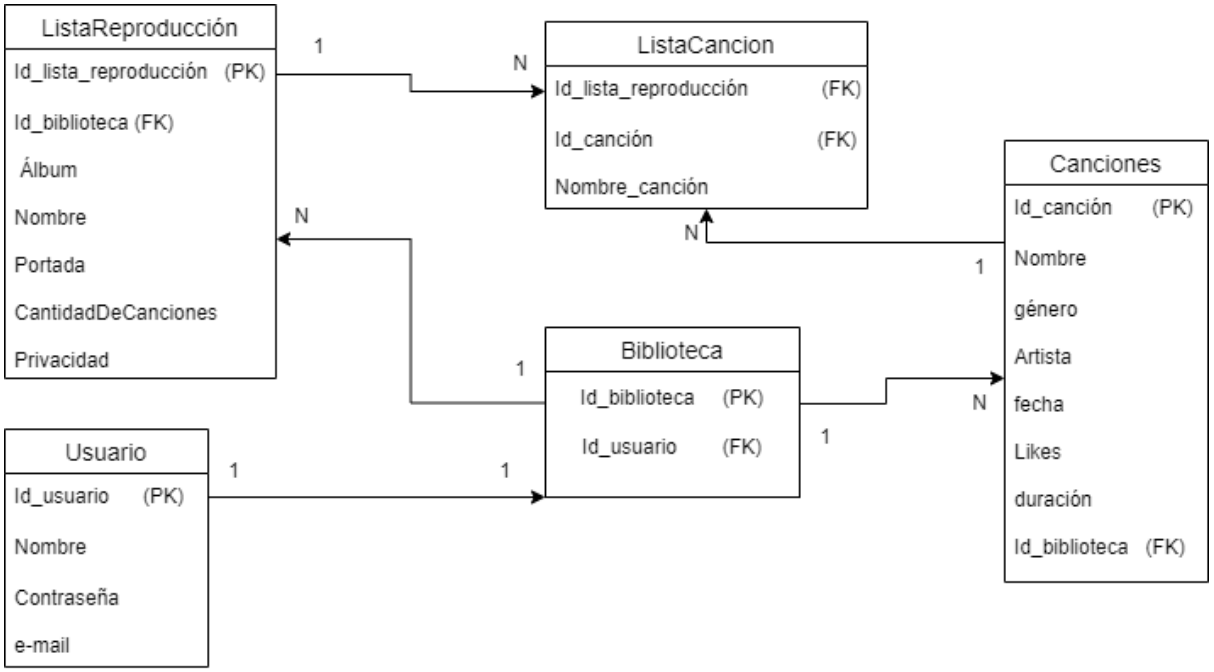
Programación Orientada a la Web, POW01

Ing. Diego Fernando Loaiza Buitrago

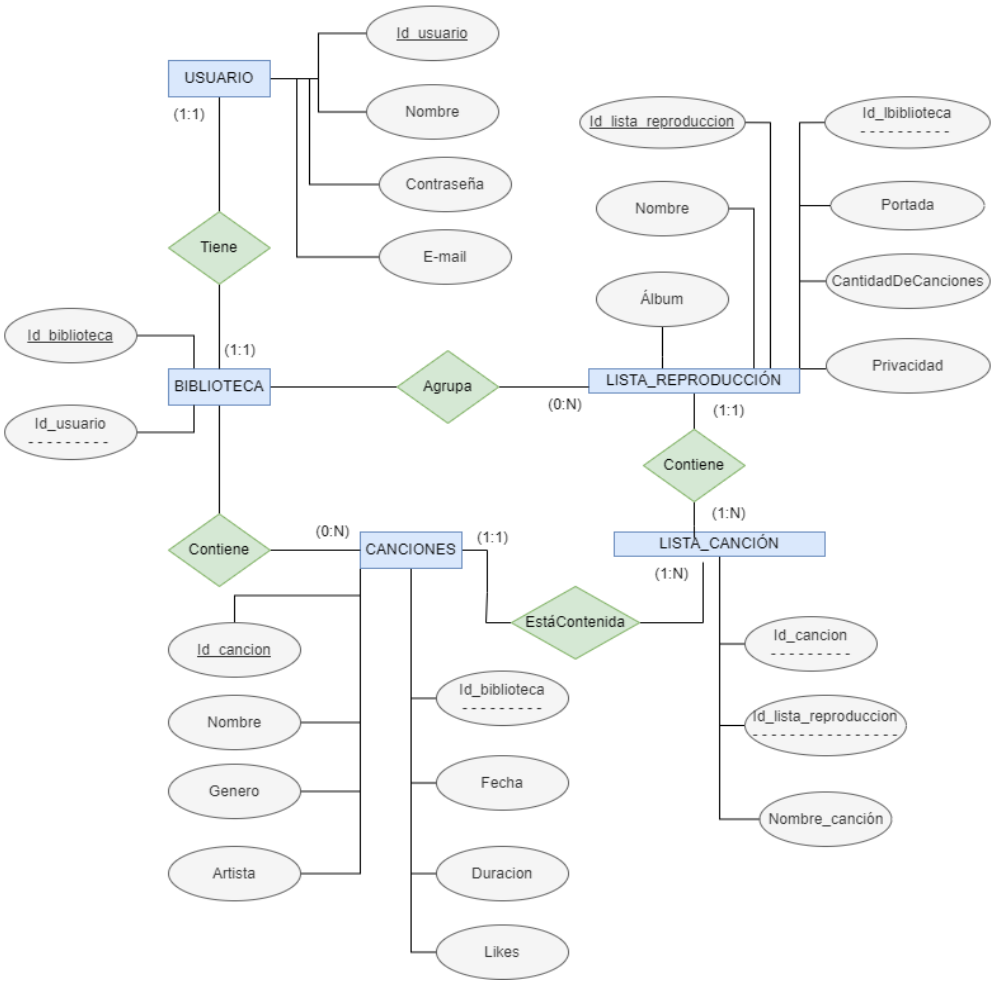
Santiago de Cali, 01/Noviembre del 2022

## Contenido

## Relación entre las Entidades

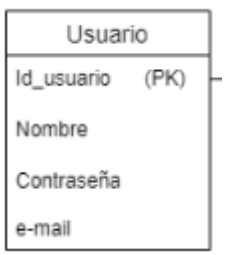


## Diagrama Entidad Relación




## Explicación Entidad Relación

### Entidad Usuario:





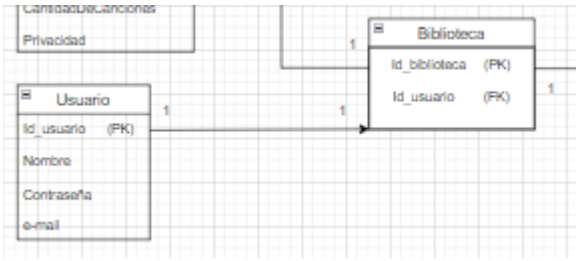
En las imágenes anteriores, se evidencia la entidad usuario y sus respectivos atributos a los que la tabla que se visualizará a continuación relaciona.

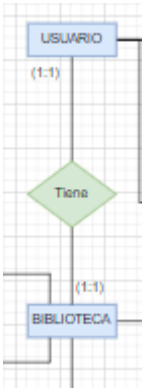
- Donde es posible analizar que *Id_usuario* se considera la primary key de la entidad, con el que es posible identificar únicamente a un usuario que use el reproductor de música.

- El atributo *Nombre* de tipo String, que sería el nombre del usuario en cuestión.

- La *Contraseña* de tipo String, con la cual se accede y se convierte en credenciales de inicio para acceder al reproductor junto con el atributo *e-mail*, de carácter también String para poder acceder a una única cuenta asociada a una persona.

- Las relaciones que presenta con las siguiente entidad conocida como Biblioteca, se visualizan de la siguiente manera:

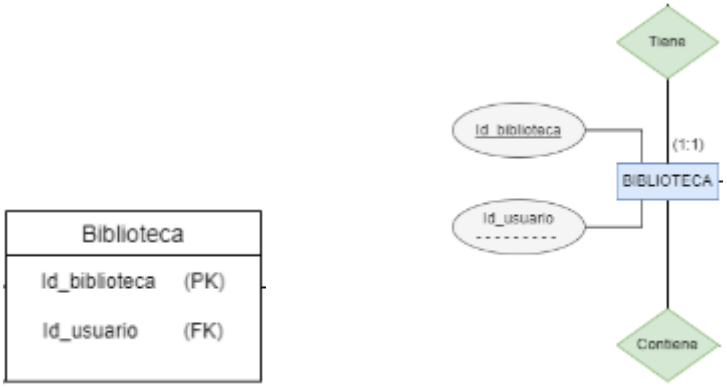



Como se visualiza en las imágenes anteriores, la relación con la entidad biblioteca es de 1 a 1, es decir que, un usuario posee dentro del dominio de la aplicación una biblioteca donde deposita múltiples Listas de reproducción o Playlist que considere de su agrado, interés o desee crear con contenido multimedia propio o canciones de otros artistas registrados.

Por ende se define la entidad Biblioteca con los siguientes atributos:

**Biblioteca:**



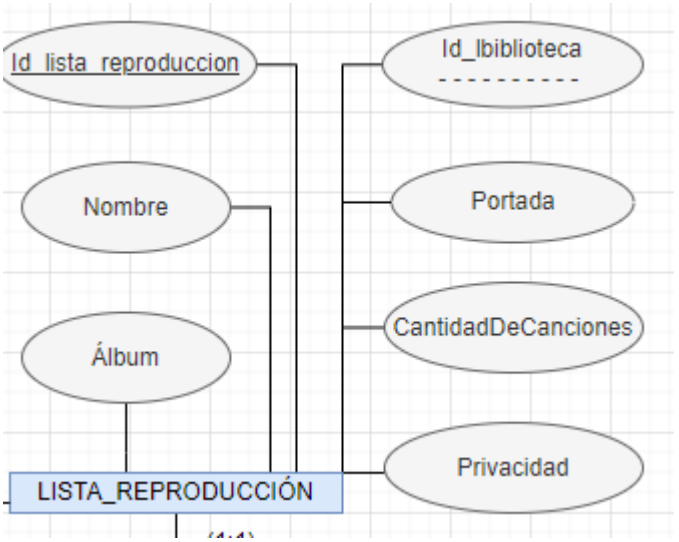

Basándose a las imágenes de anteriores con la tabla *Biblioteca*, se determinan los siguientes atributos:

- *Id_biblioteca* va tener como tipo Int, además va a ser una Primary Key de la tabla, esto para tener un identificador y hacer la conexión con las otras tablas, además este se va desempeñar para diferenciar un álbum de otro.

- *Id_usuario,* va tener como tipo Int y va a ser considerada como una llave foránea para relacionar a un usuario y su respectiva biblioteca dentro de su cuenta..

**Lista de Reproducción:**

La relación existente entre la entidad biblioteca y lista de reproducción, tiene una cardinalidad de 1 a muchos, puesto de que en una biblioteca asociada a un usuario, puede contener múltiples listas de reproducción.


Con base en las imágenes presentadas, se establece una entidad conocida como:

- ListaReproducción, la cual desempeña el papel de las playlist que existen en el reproductor de música ya sea creadas por otros artistas o usuarios y que se desean añadir a la biblioteca o alguna lista de reproducción creada por el usuario.

- *Id_Lista_reproduccion*, como primary Key, se considera a un identificador de tipo int para reconocer únicamente a una lista de reproducción en específico que le pertenezca a algún usuario.

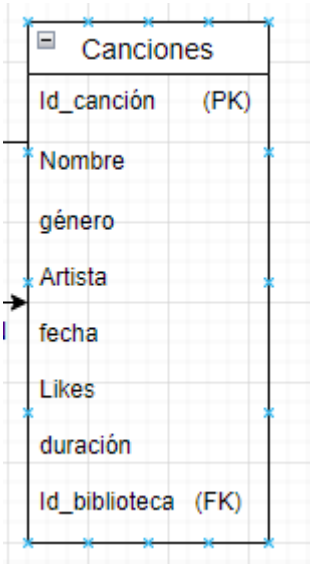
- Una llave foránea conocida como *Id_biblioteca*, que representa a qué biblioteca está siendo asociada o vinculada con un usuario para poder realizar actualizaciones o una respectiva administración.

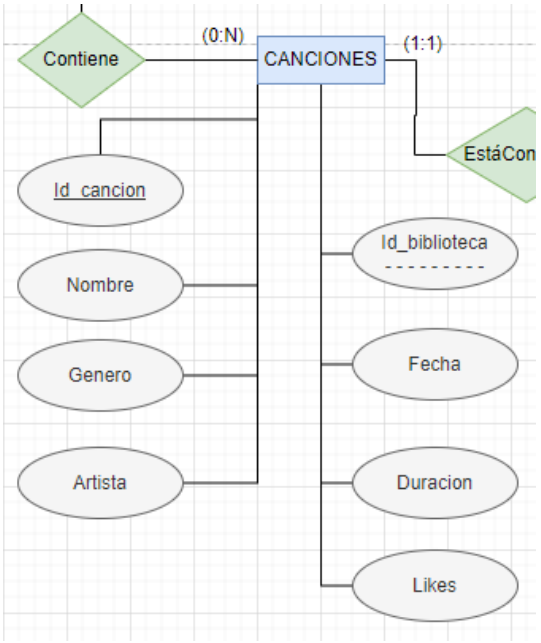
- *Álbum,* un atributo de tipo String que permita tener la descripción del álbum o playlist en cuestión, *Nombre*, en el cual almacena el nombre que se muestra en el álbum creado. *Portada,* Un valor de tipo String que relaciona la dirección o ubicación de la imágen que se le asigna al álbum, el atributo *CantidadDeCanciones,* el cual estipula o contabiliza la cantidad de canciones que tiene el álbum y por último un atributo de

tipo booleano conocido como *Privacidad,* el cual permite definir si la playlist está abierta a los demás usuarios o únicamente a los editores de la playlist.

**Canciones:**

La relación existente entre la entidad biblioteca y canciones, tiene una cardinalidad de 1 a muchos, puesto de que en una biblioteca asociada a un usuario, puede contener múltiples canciones.



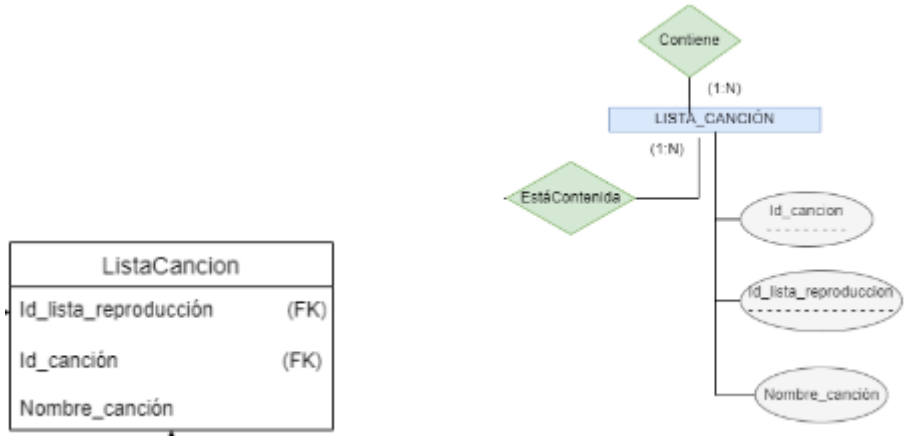

Estableciendo la entidad canciones, se establecen los siguientes atributos:

- *Id_cancion,* atributo de tipo int, considerado como primary key, pues se convierte en identificador principal para una determinada canción dentro de todas las existentes. Nombre, de tipo String que se considera el título de la canción.

- El *género* de la canción de tipo String, su respectivo *artista* de tipo String, la fecha de subida de tipo *String.* Para medir los likes de la canción en particular para un usuario, se considera cuantificarlos por medio de un tipo de dato de tipo Int llamada *Likes.*

- Como tipo de dato float, el tipo de dato *duración,* es considerada dentro de la entidad canción y de llave foránea, tendremos *id_biblioteca.*

**Lista de canciones:**



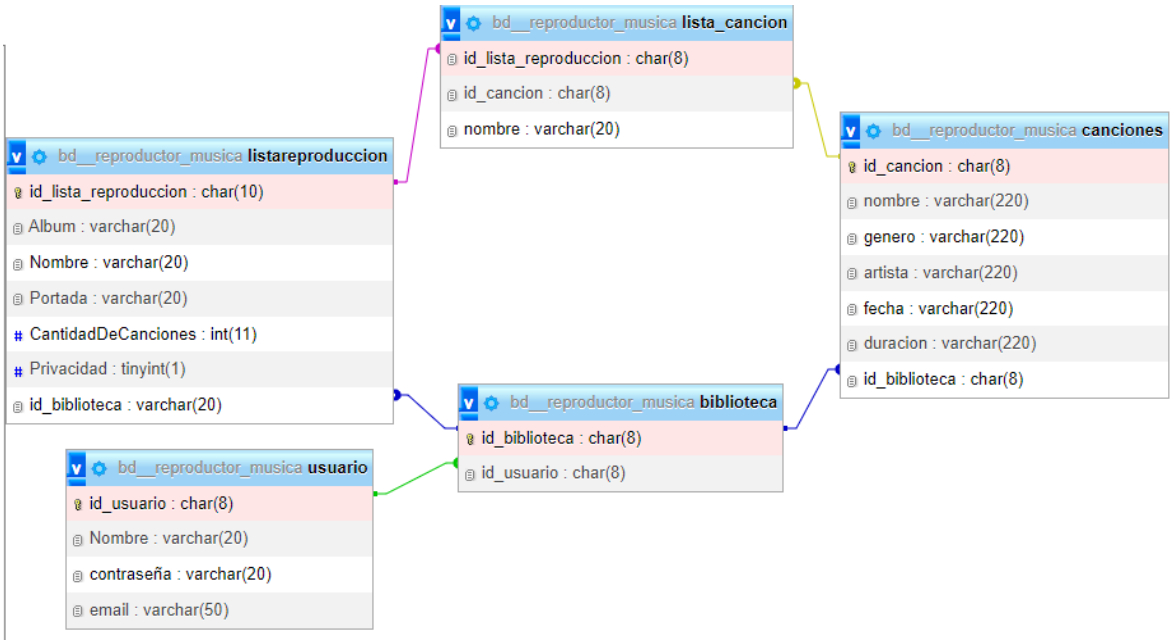

A partir de la figura previa, se establece la tabla *ListaCancion* con las siguientes atributos:

- *Id_lista_reproducion* va ser de tipo Int y además es una Foreign Key, y va hacer la conexión de las tablas ablas *Canciones* y *ListaReproducción.*

- *Id_cancion* va ser de tipo Int y además es una Foreign Key, y va hacer la conexión de las tablas *Canciones* y *ListaReproducción*

- *Nombre_cancion* L va tener como tipo un String, el cual va a ser el nombre de la canción que se esté dentro de la lista, además es la conexión de las tablas *Canciones* y *ListaReproducción* para evitar un ciclo por medio de la regla 2FN.

## Estructura de la Base de Datos




Posteriormente a la creación de la base de datos, se obtienen con satisfacción las relaciones establecidas de manera conceptual.

# Script de Base de Datos

BD__Reproductor_musica.sql

```sql
CREATE DATABASE BD__Reproductor_muscia

go

use BD__Reproductor_musica
go
/* Creando tabla usuario, biblioteca */

Create table usuario(
id_usuario char(8) primary key,
Nombre varchar(20) not null,
contraseña varchar(20) not null,
email varchar(50)

);

/* Creando tabla biblioteca con campos para llaves foraneas */

/* Creando tabla usuario, biblioteca */
/* Creando tabla usuario, biblioteca */
Create table lista_cancion(
id_lista_reproduccion char(8),
 id_cancion char(8),

CONSTRAINT id_lista_reproduccion FOREIGN KEY (id_lista_reproduccion) REFERENCES listaReproduccion (id_lista_reproduccion),
CONSTRAINT id_cancion FOREIGN KEY (id_cancion) REFERENCES canciones (id_cancion)
);

```

```sql
/* Creando tabla canciones */
Create table canciones(
id_cancion char(8) primary key,
nombre varchar(220) not null,
genero varchar(220) not null,
artista varchar(220) not null,
fecha varchar(220) not null,
duracion varchar(220) not null,
id_biblioteca char(8),

CONSTRAINT id_biblioteca FOREIGN KEY (id_biblioteca) REFERENCES biblioteca (id_biblioteca)
);

Create table ListaReproduccion
(
  id_lista_reproduccion char(10) PRIMARY KEY,
  Album varchar(20) not null,
  Nombre varchar(20) not null,
  Portada varchar(20) not null,
  CantidadDeCanciones INT not null,
  Privacidad boolean not null,
  id_biblioteca char(8)
  FOREIGN KEY (id_biblioteca) REFERENCES biblioteca(id_biblioteca)
);


Create table lista_cancion(
id_lista_reproduccion char(8),
 id_cancion char(8),

CONSTRAINT id_lista_reproduccion FOREIGN KEY (id_lista_reproduccion) REFERENCES listaReproduccion (id_lista_reproduccion),
CONSTRAINT id_cancion FOREIGN KEY (id_cancion) REFERENCES canciones (id_cancion)
);
```

## Script Controlador

Conocido como crud_usuario

```
controlador > JS crud_usuario.js > cud > nombre
1   import { conectar } from "../modelo/db_conectar.js";
2
3   var crud_usuario =({});
4   crud_usuario.leer = (req,res)=>{
5
6       // Select * from usuarios;
7       conectar.query('SELECT usuario.id_usuario,usuario.Nombre,usuario.contraseña, usuario.email',(error,results)=>{
8           if (error){
9               throw error;
10
11      }else{
12      res.render('usuario/index',{resultado:results})
13      }
14  })
15  };
```

En el script anterior, se puede visualizar la función de read del CRUD, en donde se adquieren, toman o leen los datos almacenados en la base de datos, tomando todos y cada uno de los parámetros que se adquieren a través del ingreso de los datos mediante unas variables declaradas en las cuales se explican posteriormente y son inicializadas en el nuevo usuario. En el caso que algo no esperado suceda (Error), se maneja y se mostrará el respectivo mensaje de error. En caso contrario, devolvemos el resultado, el cual es la creación de un nuevo elemento conocido como usuario.

```
crud_usuario.cud = (req,res)=>{
    const btn_crear = req.body.btn_crear;
    const btn_actualizar = req.body.btn_actualizar;
    const btn_borrar = req.body.btn_borrar;
    const id_usuario = req.body.txt_id;
    const nombre = req.body.txt_nombres;
    const contraseña = req.body.txt_contraseña;
    const email = req.body.txt_email;

    if (btn_crear){
        conectar.query('insertar información de usuario ?',[{id_usuario:id_usuario,nombre:nombre, contraseña:contraseña ,email:email}], (error, results)=>{
            if(error){
                console.log(error);
            }else{
                //console.log(results);
                res.redirect('/');
            }
        });

    }
```

```
(btn_actualizar){
 conectar.query('update usuario SET ? where id_usuario = ?',[{id_usuario:id_usuario,nombre:nombre, contraseña:contraseña ,email:email},id], (error, resu
    if(error){
        console.log(error);
    }else{
        res.redirect('/');
    }
 });


(btn_borrar){
 conectar.query('delete from usuario  where id_usuario = ?',[id_usuario], (error, results)=>{
    if(error){
        console.log(error);
    }else{
        res.redirect('/');
    }
 });
```

Dentro de la función CUD, se encontrarán condicionales que permiten identificar la acción requerida. Si se refiere a un create (crear elemento en la base de datos), update (Actualizar elementos en la base de datos) y un delete (eliminar elementos de la base de datos).

Para ello se identifican los botones que desempeñarán dichas funciones.

Botón crear, botón actualizar y botón eliminar o borrar.

A su vez se maneja un respectivo error. Es decir, si existe un error en alguno de estos procesos, este se maneja y dará una respectiva respuesta; en caso contrario, seguirá con las operaciones que son permitidas.

*Dentro de la opción Create:*

Se crea mediante un query, una inserción de los datos en la base de datos, vinculada a la tabla usuario, donde se pasan como parámetro o se establecen que los valores a ingresar en la base de datos, son los previamente ingresados. Posteriormente a ello, se procede a refrescar la página.

*Dentro de la opción Update:*

Mediante un query, se solicita nuevamente cuál será la información a actualizar, una vez se tenga esto, se procede a refrescar la página.

*Dentro de la opción Delete:*

Una vez se encuentre un respectivo usuario con un determinado Id estipulado, la información o tupla en dicha tabla, será eliminada inmediatamente. En caso de poseer algún error, este sería encapsulado.

**Script Modelo**

Conocido como: db_conectar

```
modelo > JS db_conectar.js > [e] conectar > database
1   //https://www.npmjs.com/package/mysql
2   import mysql from 'mysql' // o tambien const mysql = require('mysql');
3   var conectar = mysql.createConnection({
4       host     : 'localhost',
5       user     : 'usr_empresa',
6       password : 'Empres@123',
7       database : 'BD_Reproductor_musica'
8   });
9
10  conectar.connect(function(err) {
11      if (err) {
12          console.error('Error en la conexion: ' + err.stack);
13      return;
14    }
15
16        console.log('conexion exitosa ID: ' + conectar.threadId);
17  });
18
19  export {conectar}
```

En este punto se hace la conexión de la base de datos.Por defecto se utiliza el host: “localHost”. Para este caso en particular, se establecen unas credenciales y la base de datos a la cual se desea acceder.

En el momento de realizar la función de conectar con la base de datos. Se procede a verificar si se realizó correctamente, en este caso se  muestra que la conexión está correcta, en caso contrario, se procede a mostrar el respectivo error.

## Script Vista/Usuarios

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <!--    <h1>Hola mundo desde ejs</h1> -->
    <h1>Formulario Usuario</h1>
    <form action="/crud_c" method="post">
        <input type="text" id="txt_id_usuario" name="txt_id_usuario" placeholder="0">
        <input type="text" id="txt_nombres" name="txt_nombres" placeholder="Nombre">
        <input type="text" id="txt_contraseña" name="txt_contraseña" placeholder="contraseña">
        <input type="text" id="txt_email" name="txt_email" placeholder="email">
       <button type="submit" id="btn_crear" name="btn_crear" value="crear">Crear</button>
       <button type="submit" id="btn_actualizar" name="btn_actualizar" value="actializar">Actualizar</button>
       <button type="submit" id="btn_borrar" name="btn_borrar" value="borrar">Borrar</button>
    </form>
    <table class="table" border ="1px">
        <thead>
            <tr>
                <th>id_usuario</th>
                <th>nit</th>
                <th>nombre</th>
                <th>contraseña</th>
                <th>email</th>
            </tr>
        </thead>
        <tbody>
           <% resultado.forEach((usuario) => { %>
            <tr>
                <td><%= usuario.id_usuario %></td>
                <td><%= usuario.Nombre%></td>
                <td><%= usuario.contraseña %></td>
                <td><%= usuario.email %></td>
            </tr>
            <% }) %>
        </tbody>
    </table>
</body>
</html>
```

En el script anterior, se puede visualizar la creación de las vistas de usuarios mediante la creación de un formulario con unos respectivos id's para la definición de cada parte del formulario; también se adicionan unos botones para crear, actualizar y borrar respetando las funcionalidades del CRUD. Las funcionalidades se encuentran establecidas en crud_usuario. Además, en la parte *table class = "table" border = "1px"* se muestra la creación el encabezado y el cuerpo de la tabla con sus respectivos títulos; el cuerpo de la tabla se realiza de forma dinámica accediendo a los atributos del método *usuario.*

# Index

```
JS index.js > ...
1   import express  from "express" // si no se utiliza type mode se deberia de declarar como const express = require('express');
2   import { crud_usuario } from "./controlador/crud_usuario.js"
3   //Paso 7
4
5   //Paso 1
6   const app_e = express() // crear variable para acceder al expressjs
7   app_e.use(express.urlencoded({extended:false}));
8   app_e.use(express.json());
9   // Paso 4 (directorios estaticas)
10  app_e.use(express.static('./vistas'))
11  app_e.use(express.static('./controlador'))
12  app_e.use(express.static('./modelo'))
13  // Paso 5 configurar el motor vistas
14  app_e.set('views','./vistas')
15  app_e.set('view engine','ejs')
16  //Paso 2
17  app_e.listen('3306',function(){
18      console.log('Aplicacion Iniciada : http://localhost:3306/')
19  })
20  // Paso 3
21  app_e.get('/',crud_usuario.leer);
22  app_e.post('/crud_c',crud_usuario.cud);
23
24
25  app_e.get('/',function(req,res){
26      res.send("Hola mundo")
27          })
```

En primer lugar, se procede a realizar los siguientes pasos:

1. npm init -y
2. npm install express
3. npm install nodemon
4. npm install ejs ó npm install pug
5. npm install mysql

En el código procedemos a declarar las vistas, controlador y el modelo.

Tomando en cuenta los códigos anteriores y las funcionalidades realizadas, procedemos a tomar los métodos app_e.get, para realizar el Read, adquiriendo los elementos que se encuentren en la entidad usuario.

Así mismo encontramos el metodo post, donde ingresamos un nuevo elemento conocido como usuario.