



Desarrollo Full Stack

Reporte Individual

Nombre: Frida Sarahi Garza Gálvez	Matricula: al07002961
Profesor: Erik Bethuel Estrada Andrade	
Fecha: 3 de febrero de 2025	

Descripción

En la primera fase del reto, los aprendedores deberán crear la base de una aplicación Full Stack con un enfoque general y de interés amplio (puede estar orientada a sectores como el educativo, automotriz, financiero, industrial, etc.), y centrarse en el desarrollo del Frontend y Backend utilizando Node.js y Express.js. Esta fase incluye la configuración del servidor, la implementación de rutas y middleware, y la gestión de operaciones CRUD en una base de datos. Además, se espera la creación de un diseño básico de interfaz utilizando HTML, CSS y JavaScript.

Objetivo

Configurar el entorno de desarrollo y los módulos iniciales. Desarrollar una estructura básica del Frontend con HTML, CSS y JavaScript, así como un servidor Backend en Node.js y Express que permita realizar operaciones CRUD en una base de datos e implemente autenticación básica mediante JWT. Finalmente, desplegar una versión preliminar de la aplicación en un entorno local o en la nube.

Instrucciones

a. Configuración del proyecto.

1. Inicia un nuevo proyecto con `npm init` en Node.js.
2. Elige la base de datos que se utilizará (MongoDB, MySQL o PostgreSQL).
3. Instala las dependencias necesarias: `express`, `jsonwebtoken`, `bcryptjs`, `cors`. Dependiendo de la base de datos seleccionada, instala también mongoose (para MongoDB) o pg (para PostgreSQL).
4. Configura el proyecto con variables de entorno usando `dotenv`.
5. Consulta un ejemplo de configuración de variables de entorno utilizando `dotenv` para almacenar credenciales y claves sensibles, mediante una herramienta de inteligencia artificial (IA) como ChatGPT, Gemini AI o GitHub Copilot. Adapta el ejemplo proporcionado a las necesidades específicas del proyecto. Esto te permitirá comprender cómo estructurar variables de entorno de forma segura y funcional, además de ahorrar tiempo y adquirir mejores prácticas basadas en una solución generada por IA.

b. Backend (API RESTful con Express.js).

1. Crea un servidor básico en Express.
2. Configura las rutas básicas para realizar operaciones CRUD (*Create, Read, Update, Delete*) utilizando una base de datos local, ya sea MongoDB, MySQL o PostgreSQL.
3. Implementa un sistema de autenticación utilizando JWT. Los usuarios deberán iniciar sesión para realizar ciertas operaciones. Dependiendo de tus reglas de negocio, las operaciones de lectura pueden ser abiertas, mientras que las operaciones de creación, modificación y eliminación pueden quedar protegidas.
4. Utiliza middleware personalizado para manejar errores y validaciones.
5. Consulta una herramienta de inteligencia artificial (IA) para generar un middleware básico de manejo de errores en Express.js. Adapta el middleware a los requerimientos específicos de la aplicación y realiza pruebas para verificar su funcionalidad. Esto te permitirá visualizar ejemplos prácticos de manejo de errores en Express.js, aprender a adaptarlos a las necesidades del proyecto y mejorar la robustez del Backend.

c. Frontend (HTML, CSS, JavaScript).

1. Desarrolla un Frontend básico que incluya una página de inicio de sesión y una página principal donde se muestren los datos gestionados desde el Backend (como lista de tareas, usuarios, productos, etc.).
2. Utiliza **CSS** para el diseño visual (aplicar los principios básicos de diseño UI/UX) y **JavaScript** básico para manipular el DOM, enviar peticiones al servidor y recibir respuestas.
 - Consulta una herramienta de inteligencia artificial (IA) para obtener sugerencias de diseño en CSS para la página de inicio de sesión. Evalúa las propuestas y adapta las más adecuadas para optimizar la experiencia del usuario.
Esto fomentará la creatividad en el diseño y permitirá explorar ideas innovadoras generadas por IA, mejorando la apariencia y usabilidad de la interfaz mientras se alinean con los principios fundamentales de diseño UI/UX.
3. Implementa interacciones dinámicas utilizando JavaScript, como agregar y eliminar elementos de una lista.

d. Despliegue preliminar.

1. Realiza el despliegue de la aplicación en un entorno local.
2. Si es posible, configura un despliegue inicial en un servicio en la nube como **Heroku**, **Render** o **AWS** para el Backend.

e. Entrega avance del reto.

1. Incluye los siguientes elementos en un documento en formato Word o PDF:
 - Código fuente completo del Backend y Frontend, subido a un repositorio en GitHub.
 - Documentación técnica donde expliques la estructura del proyecto y los *endpoints* de la API.
 - Graba un video demostrativo que muestre el funcionamiento del Frontend y de las operaciones CRUD en el Backend.

Reporte individual

Frida Sarahi Garza Gálvez

Dev

En esta actividad obtuve el cargo de Dev, que se encarga de desarrollar varios requisitos de la actividad, como el HTML, CSS y JavaScript que apoyan en el diseño de la página discográfica y las validaciones junto a las acciones de varios aspectos de la página web que buscamos ofrecer.

Se nos solicita la selección de una base de datos, la creación de un proyecto en Node.js, la creación de operaciones CRUD que se asocien a el tema de la página, por ello, decidimos que se hiciera un Login y Registro, ya que después de ello, se abarcaría la edición y eliminación de los datos del usuario.

Requerimientos

Estos son los requerimientos que genere debido a lo que se me solicita a lo largo de la creación del código y necesidades de la actividad:

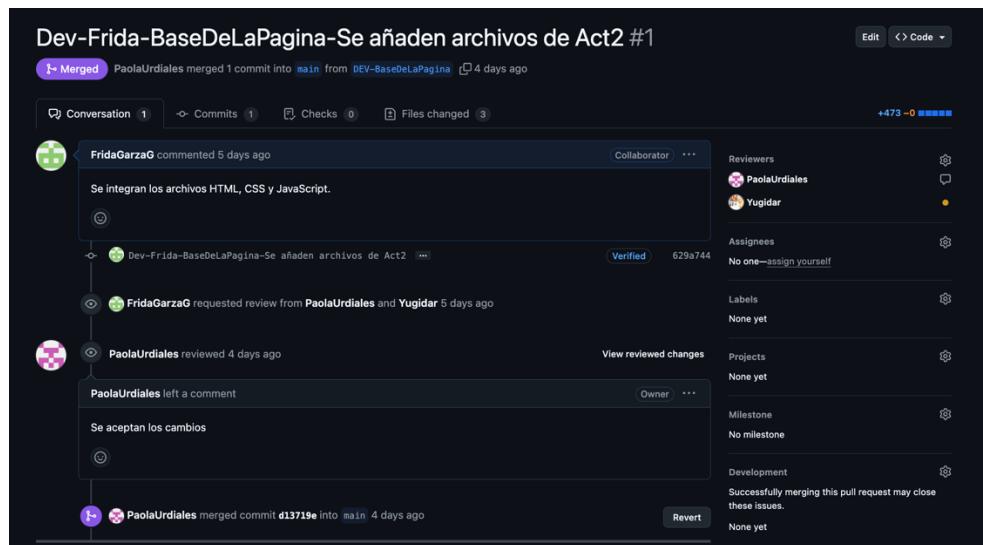
- Añadimiento de archivos HTML,CSS y JavaScript
- Iniciar proyecto en Node.js
- Instalar dependencias necesarias
- Seleccionar base de datos y crear la conexión
- Comprobación de la conexión de la base de datos
- Configurar variables de entorno usando Dotenv
- Crear un servidor básico en Express
- Crear una ruta para CREATE de usuario
- Crear una ruta para READ de usuario
- Crear una ruta para UPDATE de usuario
- Crear una ruta para DELETE de usuario
- Crear un HTML para que el usuario inicie sesión
- Validamos que se ingresen usuarios registrados
- Funcionalidad de botón para registrarse
- Generar un HTML para que el usuario se registre
- Funcionalidad de botón para crear usuario
- Generar mensajes para que el usuario
- Generar un Embedded JavaScript para el Perfil del usuario
- Crear apartado en 'Perfil' para mostrar datos
- En el apartado en mostrar datos agregar un botón de eliminar.
- Funcionalidad de botón para eliminar usuario
- Crear apartado en 'Perfil' para editar datos
- Crear botones que muestren el apartado específico en Perfil
- Se valida que el usuario ingrese un correo valido en el apartado de Ajustes
- Crear botón de cerrar sesión en 'perfil.ejs'
- Generar una barra lateral
- Diseño de barra lateral
- Añadir el diseño del UI/UX
- Configurar los JavaScript para los HTML y EJS
- Enlazar a los JavaScript
- Responsiva de la pagina
- Desplegar de forma local

Realización

Para mi realización de requerimientos fue necesario concordar con mi equipo lo que buscábamos crear y añadir, por ello, vimos correcto ampliar las páginas de la página y agregar nuevos HTML y diseño. Y a continuación los requerimientos y como fueron abarcados.

- Añadimiento de archivos HTML, CSS y JavaScript

Para el requerimiento, fue necesario ingresar al repositorio en GitHub, creado por mi compañera Paola Urdiales, donde creamos una Branch en donde específico que es el añadimientode los archivos ‘index.html’ que le cambia el nombre a ‘listaCanciones.html’, el ‘pagPrincipal’ que es el archivo que se utilizó por nuestro compañero Pedro De León en la primera actividad, junto a esos documentos, se agregaron ‘app.js’ y ‘style.css’, siendo una base sólida de lo que nuestra página llegaría a ser.



Podemos observar en la captura de pantalla como se hizo el Pull Request y como fue aceptado por el CI/CD para así dar comienzo con la creación de la actividad.

- Crear el proyecto en Node.js

Para la creación en el Node.js se tuvieron que seguir ciertos pasos necesarios, como la comprobación de la instalación con el comando **node -v** que nos comenta si está instalado Node.js y **npm -v** que es el gestor de paquetes Node Package Manager.

```
test@Laptop ~ % node -v  
v23.6.1  
test@Laptop ~ % npm -v  
11.0.0
```

Podemos ver en la captura de pantalla como tras ingresar dichos comandos en la terminal nos da la versión instalada.

Y con ello confirmado, ahora ingresamos a la carpeta del proyecto, haciendo uso del comando **cd** junto al **nombre de la carpeta del proyecto**, dando el siguiente resultado:

```
test@Laptop ~ % cd AVANCE-DE-PROYECTO  
test@Laptop AVANCE-DE-PROYECTO %
```

Tras ello, haciendo uso del comando **npm init -y**, se nos genera un archivo llamado package.json en nuestra carpeta. En la consola se nos mostrara el contenido del archivo mismo y podremos ver que contiene varios puntos relevantes del proyecto como:

Reporte individual

- Nombre del proyecto
- Versión del proyecto
- Archivo principal del proyecto
- Comandos que podemos ejecutar
- Palabras clave para describir el proyecto
- Autor del proyecto
- Tipo de licencia
- Definición de si se usan módulos o CommonJS

Esto nos ayudara en la funcionalidad del servidor de nuestro proyecto en Node.js

```
test@Laptop AVANCE-DE-PROYECTO % npm init -y
Wrote to /Users/test/AVANCE-DE-PROYECTO/package.json:

{
  "name": "avance-de-proyecto",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "description": ""
}
```

- Instalación de dependencias

Tras la creación del proyecto en Node.js, ahora debemos hacer las descargas de las dependencias que son necesarias en nuestra actividad, siendo muy relevantes para ciertas funciones que buscamos abarcar en nuestro proyecto, algunas de ellas son:

Bcryptjs

```
test@Laptop AVANCE-DE-PROYECTO % npm install bcryptjs
added 1 package, and audited 2 packages in 919ms
```

Librería empeñada en encriptar las contraseñas para antes de almacenarlas en la base de datos.

CORS (Cross-Origin Resource Sharing)

```
test@Laptop AVANCE-DE-PROYECTO % npm install cors
added 3 packages, and audited 5 packages in 721ms
found 0 vulnerabilities
```

Nos ayuda en el permiso de que el servidor de Backend acepte solicitudes desde dominios diferentes al Frontend, evitando que sean bloqueadas por el navegador debido a la seguridad.

Dotenv

```
test@Laptop AVANCE-DE-PROYECTO % npm install dotenv
added 1 package, and audited 6 packages in 624ms
```

Nos permite cargar variables de entorno desde el archivo .env de la página, manteniendo configuraciones sensibles fuera del código.

Ejs

```
test@Laptop AVANCE-DE-PROYECTO % npm install ejs
added 16 packages, and audited 22 packages in 1s
3 packages are looking for funding
```

Es un motor de plantillas para generar en el HTML dinámicas desde el Backend, siendo así que podamos insertar datos directamente en el HTML.

Express

```
test@Laptop AVANCE-DE-PROYECTO % npm install express
added 68 packages, and audited 90 packages in 2s
17 packages are looking for funding
```

Es un framework para la creación de un servidor en node.js, nos da una estructura sencilla para manejar rutas, peticiones, middleware y más funciones para un desarrollo óptimo en el servidor Backend.

Express-session

Nos permite manejar sesiones de usuario en la página, dando la oportunidad de almacenar información en el servidor sobre el estado de un usuario entre las peticiones, dandonos la gestión de autenticación de usuarios.

```
test@Laptop AVANCE-DE-PROYECTO % npm install express-session
added 6 packages, and audited 96 packages in 908ms
```

MySQL

```
test@Laptop AVANCE-DE-PROYECTO % npm install mysql mysql2
added 24 packages, and audited 120 packages in 2s
```

Ambas librerías son para interactuar con la base de datos MySQL, siendo útiles para operaciones CRUD en la base de datos.

Nodemon

```
test@Laptop AVANCE-DE-PROYECTO % npm install nodemon
added 25 packages, and audited 145 packages in 2s
```

Nos facilita el desarrollo en Node.js, siendo que al reiniciar el servidor se hace en automático cuando detecta cambios en los archivos del proyecto.

Sweetalert2

```
test@Laptop AVANCE-DE-PROYECTO % npm install sweetalert2
added 2 packages, and audited 147 packages in 911ms
```

Esta librería nos permite mostrar alertas en el Frontend, dando una mejoría en la experiencia del usuario debido a la forma visualmente atractiva de los mensajes.

Cambios reflejados en el package.json

```
{
  "name": "avance-de-proyecto",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "description": "",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.4.7",
    "ejs": "^3.1.10",
    "express": "^4.21.2",
    "express-session": "^1.18.1",
    "mysql": "^2.18.1",
    "mysql2": "^3.12.0",
    "nodemon": "^3.1.9",
    "sweetalert2": "^11.15.10"
  }
}
```

Tras instalar todas las dependencias necesarias para nuestro avance del proyecto, podemos ver en el archivo ‘package.json’ que se fueron integradas en la sección de dependencias las versiones de cada una de ellas, dandonos así información de este, como el nombre del paquete y la versión de este. Y si notamos, en la versión se tiene el signo ‘^’ que aclara que es la versión más reciente para la compatibilidad con la versión mayor.

Cada una de estas dependencias es necesaria porque son fundamentales para el funcionamiento seguro, eficiente y dinámico de la página.

- Selección y creación de base de datos

Se es necesario el seleccionar una base de datos que sea optima, eficiente y fácil de manejar para las operaciones CRUD que buscamos manejar más adelante, por ello, vemos correcto tomar la base de datos MySQL, creemos que las siguientes características son fundamentales para nuestra aplicación:

- Facilidad de uso: Debido a la facilidad en la gestión de la base de datos.
- Confiabilidad: Debido al manejo que hemos hecho en anteriores materias y sabemos que es de las bases de datos más utilizadas.
- Escalabilidad: Porque permite ampliarse para abarcar las demandas que van surgiendo.
- Flexibilidad: Porque nos permite tener una flexibilidad para desarrollar aplicaciones de bases de datos SQL tradicionales y sin esquema SQL.

Creemos que MySQL es siendo una base de datos relacional se vincula perfectamente en lo que esperamos recibir en nuestra página, dando así almacenamiento de datos en tablas para generar un gran

Reporte individual

almacén ofreciéndonos un modelo de datos lógico, como tablas de datos, vistas, filas y columnas, siendo una programación flexible.

Por ello, vemos correcto crear nuestra base de datos en MySQL, dando así comienzo en lo que buscamos abarcar en el avance del proyecto.

Creación de base de datos

Para la creación de la base de datos fue necesario instalar Homebrew debido a que el dispositivo donde se está realizando el trabajo es una MAC, fue necesario el implementarlo, para después utilizando el comando *brew install mysql*, el cual nos ofrece la descarga e instalación de la última versión que se encuentre en Homebrew de MySQL.

```
test@Laptop AvanceProyecto % brew install mysql  
==> Downloading https://formulae.brew.sh/api/formula.mysql.json
```

Después de la confirmación de la instalación, ingresamos el comando *brew services start mysql*, el cual nos ayuda a iniciar MySQL y le da la configuración de hacer que se ejecute en segundo plano como un servicio.

```
test@Laptop AvanceProyecto % brew services start mysql  
==> Successfully started `mysql` (label: homebrew.mxcl.mysql)
```

Tras ver el mensaje de que se muestra en consola, podemos conectarnos a MySQL con el comando *mysql -u root -p*, debido a que yo ya lo tenía instalado, cree una contraseña y sin ella no me sería posible conectarme a MySQL, y una vez ingresada la contraseña correcta, podemos ver cómo nos da una bienvenida.

```
test@Laptop AvanceProyecto % mysql -u root -p  
Enter password:  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 14  
Server version: 9.0.1 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2025, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Con la confirmación de la entrada a MySQL, creamos la base de datos, y le daremos por nombre 'rhymes_proyecto' siguiendo las normas para la nomenclatura, donde nos pide que no sean con guiones medios.

```
mysql> CREATE DATABASE rhymes_proyecto;  
Query OK, 1 row affected (0,03 sec)
```

Tras la creación de la base de datos, se utiliza el comando USE, para poder seleccionar la base de datos donde deseamos trabajar, y podemos ver como después de dicho comando, se nos comenta que podemos modificar la base de datos.

```
mysql> USE rhymes_proyecto;  
Database changed
```

Después de estar en la posibilidad del cambio en nuestra base de datos, nos planteamos lo que esperamos que contenga y los datos que esperamos manejar en las operaciones CRUD, dando así la siguiente tabla:

Field	Type	Null	Key	Default	Extra
id	int	NO	PRIMARY	NULL	Auto_increment
nombre	varchar(50)	YES		NULL	
correo	varchar(100)	YES	UNIQUE	NULL	
password	varchar(100)	YES		NULL	

Ocasionalmente el comando necesario para crear la base de datos sea el siguiente:

```
mysql> CREATE TABLE usuarios(id INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(50), correo VARCHAR(100) UNIQUE, password VARCHAR(100));
Query OK, 0 rows affected (0,09 sec)
```

Podemos notar como especificamos el nombre, tipo, tamaño y orden de las columnas de la tabla usuarios, dándonos así una base fundamental para lo que haríamos en las operaciones CRUD, tomemos en cuenta que esto es para una base de datos de forma local, mi compañero Pedro De León se encargó de hacer que la base de datos se encuentre en la nube, dando una facilidad de acceso a los datos desde cualquier dispositivo.

- Configuración de variables en Dotenv

Para la configuración de variables en Dotenv fue necesario crear un archivo en la raíz de la carpeta de nuestro proyecto que tome por nombre '.env', siendo este el que contenga las variables de entorno, como podemos ver en la siguiente imagen, se portan datos relevantes como:

```
⚙ .env
1 #Definición de variables
2 DB_HOST=localhost
3 DB_USER=root
4 DB_PASSWORD=Clave1234
5 DB_NAME=rhymes_proyecto
```

- La dirección del servidor de la base de datos.
- El usuario con el que se conecta la base de datos.
- La contraseña del usuario para el ingreso a la base de datos.
- El nombre de la base de datos que se utilizará.

Teniendo las variables definidas, después, cuando creemos un servidor básico en Express, deberemos agregar a ese archivo lo siguiente:

Cargar las variables de entorno

Este comando nos ayuda a importar la librería dotenv, haciendo que cargue las variables de entorno desde el archivo '.env' que se encuentra en la raíz de la carpeta del proyecto.

```
require('dotenv').config();
```

Cargar el paquete de MySQL2

Luego agregamos que se importe el paquete de MySQL2, que nos permitirá interactuar con la base de datos desde Node.js, para así ofrecernos las consultas, conexión y manipulación de la base de datos.

```
const mysql = require('mysql2');
```

Crear la conexión a la base de datos

Agregamos el método que nos ayuda a conectar la base de datos de MySQL, siendo basada en los parámetros pasados con las variables de entorno para hacer la conexión correcta a la base de datos. Podemos notar como se define la dirección del servidor, el nombre del usuario, la contraseña del usuario y el nombre de la base de datos.

```
//Conexión a MySQL con variables de entorno
const connection = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME
});
```

Reporte individual

Exportar la conexión de la base de datos

Esto nos ayuda a poder exportar desde el archivo actual, hacia otros, para hacerle uso desde otro archivo.

```
module.exports = connection;
```

Iniciar la conexión a la base de datos

Este método nos ayuda a comprobar que al iniciar la conexión con la base de datos nos dé un mensaje en consola basado en si hubo un fallo en la conexión o si fue exitosa, dando así, que, si ocurre un error, se mande un mensaje de que hubo una problemática en la conexión y la muestra detalles del error, y si fue exitosa, nos da un mensaje en consola donde nos afirma que todo fue exitoso.

```
//Prueba de la conexión a la base de datos
connection.connect((err) => {
  if (err) {
    console.error('Error al conectar a la base de datos:', err);
    return;
  }
  console.log('Conexión exitosa a la base de datos.');
});
```

- Creación de un servidor básico en Express

Iniciamos con la creación de un archivo llamado ‘server.js’, el cual contendrá las importaciones de las librerías y módulos que necesitamos, junto a las rutas de las páginas que esperamos manejar más adelante.

Notemos en las siguientes imágenes, como ya incluimos lo que hablamos en la configuración de las variables de entorno respecto a la conexión de la base de datos, pero podemos ver como se fueron agregados más líneas de código en el archivo.

```
ver.js > ...
//Variables de entorno
require('dotenv').config();

//Librerías
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2');
const bcryptjs = require('bcryptjs');
const path = require('path');
const expressSession = require('express-session');
const methodOverride = require('method-override'); // Para manejar el método PATCH

const app = express(); //Iniciar app
```

En la sección de librerías que podemos notar por el comentario, podemos ver como se importan los módulos como Express, CORS, MySQL2, Bcryptjs, Path, Express-session y el Method Override.

Ya habíamos hablado de casi todos esos métodos, excepto Method Override que trata

acerca de poder importar un middleware que permita sobrescribir el método HTTP en solicitudes, junto al método Path, que se encarga de manejar y utilizar las rutas de los archivos de manera segura. Y junto a ello, se agrega la instancia para crear la aplicación Express, que permite definir rutas, middleware y la escucha de solicitudes.

Luego, hacemos uso de Middlewares para manejar cuerpos de solicitudes JSON, habilitación de CORS, habilitación de que los formularios HTML usen métodos como PATCH y

```
//Middleware
app.use(express.json());
app.use(cors());
app.use(methodOverride('_method')); //Method-override para sobrescribir el método HTTP
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'fronted')));
app.use('/js', express.static(path.join(__dirname, 'fronted', 'js')));
```

DELETE, y la habilitación de análisis de solicitudes cuando se envían formularios HTML.

Después de ello, se asegura el definir que usaremos EJS (Embedded JavaScript) como motor de plantillas, dándonos así un HTML dinámico con código JavaScript implementado, con eso hecho, establecemos la carpeta donde se podrán encontrar las plantillas, dándonos la ruta del directorio.

```
//Motor de plantillas
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
```

Con la definición del motor de plantillas, se agrega lo que habíamos comentado en la configuración de las variables de entorno, siendo la conexión y la prueba de si fue correcta la conexión con la base de datos, y

```
//Conexión a MySQL con variables de entorno
const connection = mysql.createConnection({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME
});

module.exports = connection;

//Prueba de la conexión a la base de datos
connection.connect((err) => {
  if (err) {
    console.error('Error al conectar a la base de datos:', err);
    return;
  }
  console.log('Conexión exitosa a la base de datos.');
});

//Configuración de la sesión
app.use(expressSession({
  secret: 'claveSecreta',
  resave: false,
  saveUninitialized: true,
  cookie: { secure: false }
}));

app.use(cors({
  origin: '*',
  methods: ['GET', 'POST', 'PATCH', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization']
}));
```

siendo así que podamos hacer solicitudes desde cualquier dominio, que pueda ser utilizado por el cliente ya sea el GET, POST, PATCH y DELETE, junto al definir las cabeceras que podrían ser enviadas en las solicitudes del usuario.

Con eso definido, se agregan las rutas para servir los archivos HTML, donde ponemos como inicio el HTML que tendrá por nombre 'sesion', después se tendrán las demás rutas como para la página después del Login, siendo 'pagPrincipal' basándonos en el hecho de que si el usuario no está autenticado mediante la propiedadloggedin de la sesión lo redirige a la página de inicio; y agregamos las demás rutas de los HTML, como 'listaCanciones' y 'registrarse'.

tras ello, continuamos con la configuración de la sesión con express-session, la cual es un middleware que nos ayuda a manejar las sesión en la página, haciendo que el usuario este autenticado a lo largo de varias páginas en las que se encuentre, para esto, necesitamos definir la cadena secreta que nos ayuda a firmar el ID de la sesión, haciendo que el usuario no la pueda manipular, el agregar que no se guardara la sesión nueva si no ha habido cambios provocando que evitemos operaciones innecesarias de guardado, para después agregar que se guardara la sesión nueva incluso si no ha sido modificada, y especificamos que la cookie este el false debido a que no necesariamente debemos meternos tanto en ese aspecto.

Después de eso, configuramos el CORS, que es un middleware que nos permite aceptar solicitudes desde dominios diferentes,

```
//HTML "sesion" como principal
app.get('/', (req, res) =>{
  res.sendFile(path.join(__dirname, 'fronted', 'sesion.html'))
})

//HTML "Pagina principal"
app.get('/pagPrincipal', (req, res) => {
  if (req.session.loggedin) {
    res.sendFile(path.join(__dirname, 'fronted', 'pagPrincipal.html'));
  } else {
    res.redirect('/');
  }
});

//HTML "Lista de canciones"
app.get('/listaCanciones', (req, res) => {
  res.sendFile(path.join(__dirname, 'fronted', 'listaCanciones.html'));
});

//HTML "Registrarse"
app.get('/registrarse', (req, res) => {
  res.sendFile(path.join(__dirname, 'fronted', 'registrarse.html'));
});
```

Reporte individual

Por último, lo que se debería tener al momento, es la línea que nos ayuda a inicializar el servidor estando en el puerto 3000, donde usando Express nos apoya en que se inicie el servidor en nuestra maquina local.

```
app.listen(3000, ()=>{
  console.log('Server funciona en http://localhost:3000')
})
```

Para asegurar el funcionamiento, hacemos que se mande un mensaje a la consola donde nos dice que funciona y nos da la URL para poder tratar con la aplicación.

- Creación de ruta *CREATED* de usuario

Con el servidor creado, ahora nos empeñamos en la creación de rutas básicas para operaciones CRUD, comenzando con la operación de *CREATED*, donde debemos poder crear un usuario y sea agregado a la tabla de usuarios de la base de datos llamada ‘rhymes_proyecto’, tomando en cuenta que debemos ingresar datos como nombre, correo y contraseña.

Iniciamos con definir que la ruta sea POST en Express, haciendo que, si el usuario envíe un formulario a ‘/registrarse’, esta función sea llamada para manejar dicha solicitud, continuando en las siguientes líneas, podemos ver que capturamos los valores que ingresa el usuario en el formulario, siendo el nombre que

```
app.post('/registrarse', validacionRegistro, async(req, res)=>{
  const usuario = req.body.usuario;
  const email = req.body.email;
  const pass = req.body.pass;

  console.log('Usuario:', usuario);
  console.log('Email:', email);
  console.log('Contraseña:', pass);
```

definimos como usuario, junto al correo y contraseña almacenándolos en las constantes que tienen nombres como ‘usuario’, ‘email’ y ‘pass’.

Después de ello, hacemos que se impriman en la consola los valores que fueron ingresados para así

comprobar que todo está funcionando correctamente.

Tras eso hecho, hacemos uso de la librería Bcryptjs para poder cifrar la contraseña antes de almacenarla en la base de datos, siendo así que la función bcryptjs.hash tome la contraseña sin cifrar y el hash la cifra, tomemos en cuenta que especificamos que el número es 8, ocasionando esa cantidad de rondas de cifrado que se debía aplicar en la contraseña.

```
let passwordHash = await bcryptjs.hash(pass, 8);
connection.query('INSERT INTO usuarios (nombre, correo, password) VALUES (?, ?, ?)', [usuario, email, passwordHash], async(error, results)=>{
```

Continuando con el código, ahora hacemos un connection.query, siendo un método que nos ayuda a realizar consultas a la base de datos, y esta vez, estamos insertando el nuevo usuario a la tabla de usuario que habíamos creado con anterioridad; para poder hacer la inserción a la tabla debemos hacer la consulta SQL, como podemos ver en las letras verdes, especificando en que tabla y los valores de las columnas, agregando que los valores con signos de interrogación para marcar la posición de los valores que se insertaran, siendo que los valores [usuario, email, password] sean los insertados en la tabla en su campo correspondiente; y el async(error, results) es una función callback, donde después de realizar la consulta a la base de datos se ejecuta, y recibe dos parámetros ya sea por si ocurrió un error en la consulta o si los resultados fueron correctos y todo fue exitoso.

Después de eso, dentro del connection.query(), generamos un if(error) en donde si ocurre un error durante la consulta se ejecuta lo que está dentro del bloque, siendo así que se imprima en la consola el error y que de un mensaje de alerta donde tras decir que se debió a el email repetido, lo regresa a la página de registrarse para que intente de nuevo el registro de usuario.

```

} else{
  res.render('registrarse',{
    alert: true,
    alertTitle: "Registro",
    alertMessage: "El Registro exitoso!",
    alertIcon: "success",
    showConfirmButton: false,
    timer: 1500,
    ruta: ''
  })
}

```

```

if(error){
  console.log(error);
  res.render('sesion', {
    alert: true,
    alertTitle: "Error",
    alertMessage: "Email repetido, favor de intentar de nuevo",
    alertIcon: "warning",
    showConfirmButton: true,
    timer: 1500,
    ruta: "registrar"
  });
}

```

Con ese if, ahora checamos si la consulta fue exitosa, provocando que el registro se haya podido realizar, se utiliza mismamente un mensaje, pero esta vez de que fue exitoso, en donde nos dice que el registro fue exitoso y a la par lo redirige a la página de sesion, cuando entras por primera vez a la página, para ahora sí, ingresar su usuario y su contraseña creada, vimos correcto hacer eso, para mayor seguridad y manejo de entrada,

Este código abarca una ruta CRUD, que se empeñó en el registro de un nuevo usuario en el HTML que se llamará ‘registrar.html’.

- Creación de ruta READ de usuario

Para realizar la ruta READ, necesitamos primero hacer una ruta donde el usuario ingrese sus datos para validar su inicio de sesión y después de ello, poder mostrar los datos de lectura del usuario, siendo así que se cree la ruta GET para leer los datos como nombre y email del usuario que ingreso en sesión.

Empezamos con el generar la ruta POST, para manejar el inicio de sesión y hacemos uso de la función async porque usaremos await para comparar las contraseñas, con eso, ahora se extraen los valores que se enviaron en el formulario que se tendrá en el HTML ‘sesion’, y los datos que tomaremos serán el usuario y contraseña.

```

app.post('/sesion', async (req, res) => {
  const usuario = req.body.usuario;
  const pass = req.body.pass;
})

```

Con eso hecho, ahora continuamos con hacer uso de un if, donde se verifica que los dos campos tengan valores, si los tienen, ahora se hace una consulta a la base de datos con connection.query, donde consultamos la base de datos para encontrar el nombre de usuario ingresado, recordemos que el signo de interrogación es para marcar la posición para injectar el valor que tenemos, en async en results mandaremos los datos obtenidos del usuario si se obtuvieron.

```

if (usuario && pass) {
  connection.query('SELECT * FROM usuarios WHERE nombre = ?', [usuario], async (error, results) => {
}

```

Tras hacer la consulta, generamos un if para manejar el error en la consulta, donde notifica que hubo un error en la consulta en la consola y en la página donde muestra un mensaje llamativo notificándolo y redirigiéndolo a la página de inicio.

```

if (error) {
  console.log(error);
  return res.render('sesion', {
    alert: true,
    alertTitle: "Error",
    alertMessage: "Error en la base de datos. Intenta nuevamente.",
    alertIcon: "error",
    showConfirmButton: true,
    timer: null,
    ruta: ''
  });
}

```

Reporte individual

Después se hace una verificación de usuario y contraseña, donde si no se encuentra el usuario en la base de datos, y si la contraseña ingresada no está almacenada en la base de datos le comenta que usuario y/o contraseña son incorrectos.

Dando así un mensaje dinámico donde le da a conocer al usuario su error.

```
if (results.length == 0 || !(await bcryptjs.compare(pass, results[0].password))) {
  return res.render('sesion', {
    alert: true,
    alertTitle: "Error de autenticación",
    alertMessage: "Usuario y/o contraseña incorrectos.",
    alertIcon: "error",
    showConfirmButton: true,
    timer: null,
    ruta: ''
  });
}
```

```
} else {
  //Datos del usuario
  req.session.loggedin = true;
  req.session.userId = results[0].id;
  req.session.nombre = results[0].nombre;
  req.session.email = results[0].correo;

  //Mensaje de que se logro la sesion
  return res.render('sesion', {
    alert: true,
    alertTitle: "Inicio de sesión exitoso",
    alertMessage: "[Bienvenido de nuevo!]",
    alertIcon: "success",
    showConfirmButton: false,
    timer: 1500,
    ruta: 'perfil' //Manda a perfil
  });
}
```

else contendrá el mensaje de advertencia debido a que los campos están vacíos porque no ingreso nombre o contraseña.

Después, de ese if, surge el else donde si los datos son correctos, se inicia la sesión, almacenando los datos del usuario para poder acceder a ellos en las próximas rutas a comentar; y tras almacenar los datos, se le da un mensaje de bienvenida al usuario y lo manda al archivo ‘perfil.ejs’, el cual detallaremos más adelante.

Y para finalizar esta parte de iniciar sesión, se genera un else del primer if donde validaba que se ingresaran datos y no estuvieran sin valores, y este

```
} else {
  return res.render('sesion', {
    alert: true,
    alertTitle: "Campos vacíos",
    alertMessage: "Por favor ingrese usuario y contraseña.",
    alertIcon: "warning",
    showConfirmButton: true,
    timer: null,
    ruta: ''
  });
}
```

Con la ruta para iniciar sesión hecha, ahora podemos hacer la ruta para mostrar el perfil, siendo una ruta

```
app.get('/perfil', (req, res) => {
  const userId = req.session.userId;
  //Verificar si el usuario en la sesión
  if (!userId) {
    return res.redirect('/');
  }
})
```

GET, en donde obtenemos el ID del usuario que fue almacenado en la sesión, y si el userID no llega a estar, nos redirigiría a la página de iniciar sesión.

Después se hace una consulta a la base de datos, basándonos en el ID que habíamos almacenado para obtener los datos como nombre y correo electrónico; y se hace el uso de un if en donde si el usuario fue

```
connection.query('SELECT nombre, correo FROM usuarios WHERE id = ?', [userId], (error, results) => {
  if (error || results.length === 0) {
    //En caso que ya haya sido eliminado.
    req.session.destroy(err) => {
      if (err) {
        console.log('Error al cerrar sesión después de eliminar la cuenta:', err);
      }
      return res.redirect('/');
    });
})
```

eliminado de la base de datos se destruiría la sesión y luego redirige a la página para iniciar sesión.

Y si el usuario existe se mostraría en el perfil los datos, siendo solo el nombre y correo por seguridad, esto se abarca en el else, donde se extraen los datos del usuario y se envían a la vista que se generara en ‘perfil.ejs’.

```
} else {
  //Si hay usuario se debe mostrar el perfil
  const user = results[0];
  res.render('perfil', {
    nombre: user.nombre,
    email: user.correo
  });
}
```

Creación de ruta UPDATE de usuario

Continuando con la creación de rutas para operación CRUD, ahora toca la ruta UPDATE, la cual nos ayudara a actualizar los datos del usuario, siendo desde el nombre, correo y la misma contraseña, y para darle comienzo, hacemos la ruta que tomara por nombre '/actualizarPerfil', la cual nos ayuda a definir la ruta como PATCH, y sirve para actualizar parcialmente los datos del usuario.

```
app.patch('/actualizarPerfil', async (req, res) => {
  const { newName, newPassword, newEmail } = req.body;
  const userId = req.session.userId; //ID del usuario desde la sesión
```

Con la ruta definida, ahora extraemos los valores que se enviaran desde el formulario que tendrá la página 'perfil.ejs', la cual puede recibir el nombre, correo o contraseña; y a la par se obtiene el ID desde la sesion, para asegurar que este autenticado.

```
if (!newName) {
  return res.render('perfil', {
    nombre: req.session.nombre || '',
    email: req.session.email || '',
    success: false,
    alert: true,
    alertTitle: "El nuevo nombre es obligatorio.",
    alertIcon: "error",
    showConfirmButton: false,
    timer: 1000,
    ruta: 'perfil'
  });
}

let updateValues = [newName];
```

Continuamos con la validación del correo electrónico, donde pedimos que si el usuario ingresa un nuevo correo se debe validar para que tenga el formato correcto, haciendo que porte @ y un punto; si el formato es invalido se le daría un mensaje de alerta sobre el error, pero si llegara a ser valido, se es agregado el newEmail el cual se incluye en updateValues que portara los valores que se están actualizando para la base de datos.

Tras la validación de email y nombre, ahora validamos la nueva contraseña, que será cifrada con bcryptjs para así, podrán almacenarla en la base de datos, y se hace uso de un await, ya que es un proceso asíncrono, el poder cifrar la contraseña, siendo así que después descifrarla, podamos agregarla al arreglo que tiene por nombre 'updateValues'.

```
//Verificar y encriptar la contraseña nueva si se ingresa una
let passwordHash = null;
if (newPassword) {
  passwordHash = await bcryptjs.hash(newPassword, 8);
  updateValues.push(passwordHash);
}

//Actualizar los datos en la base de datos
let updateQuery = 'UPDATE usuarios SET nombre = ?';
if (newEmail) {
  updateQuery += ', correo = ?';
}
if (passwordHash) {
  updateQuery += ', password = ?';
}
updateQuery += ' WHERE id = ?';
updateValues.push(userId);
```

Después validamos la entrada del nombre, donde si no proporciona un nuevo nombre, se le da una alerta indicando que es obligatorio, pero se mantienen los datos como nombre y email de la sesion para evitar que desaparezcan en la vista, y los redirige a perfil, para que lo vuelvan a intentar. Si se ingresa un nuevo nombre, se inicializa un arreglo que contenga el nuevo nombre como podemos ver en la linea let updateValues = [newName].

```
if (newEmail) {
  const emailRegex = /^[^@\s]+@[^\s@]+\.\[^@\s]+$/;
  if (!emailRegex.test(newEmail)) {
    return res.render('perfil', {
      nombre: req.session.nombre || '',
      email: req.session.email || '',
      success: false,
      alert: true,
      alertTitle: "Correo electrónico inválido.",
      alertIcon: "error",
      showConfirmButton: false,
      timer: 1000,
      ruta: 'perfil'
    });
  }
  updateValues.push(newEmail);
```

```
//Verificar y encriptar la contraseña nueva si se ingresa una
let passwordHash = null;
if (newPassword) {
  passwordHash = await bcryptjs.hash(newPassword, 8);
  updateValues.push(passwordHash);
}
```

Con los valores actualizados en el arreglo ahora hacemos la creación de la consulta hacia la base de datos, la cual hace uso del comando que se muestra en la imagen el cual queda aclarar que los signos de interrogación son para dejar en claro la posición de donde deberá ser agregado en el nuevo valor, dicha actualización sólo podrá ser llevada a cabo si el usuario está autenticado y tenemos su ID correspondiente.

Reporte individual

Para la actualización de la base de datos usamos ‘connection.query’, el cual ejecute la consulta a MySQL con los valores que fueron generados y agregados en el arreglo anteriormente comentado, en caso de qué llega a haber un error en la actualización, como un correo duplicado, se le daría un mensaje de error y especificando que fue lo que ocurrió.

```
connection.query(updateQuery, updateValues, (error, results) => {
  if (error) {
    console.log("Error en la actualización:", error);
    return res.render('perfil', {
      nombre: req.session.nombre || '',
      email: req.session.email || '',
      success: false,
      alert: true,
      alertTitle: "Error al actualizar los datos.",
      alertIcon: "error",
      showConfirmButton: false,
      timer: 1000,
      ruta: 'perfil'
    });
}
```

Después hacemos la obtención de los datos actualizados, el cual misma mente hacemos uso de un ‘connection.query’ para validar que los datos hayan sido actualizados, haciendo que se haga una consulta

```
connection.query('SELECT nombre, correo FROM usuarios WHERE id = ?', [userId], (error, results) => {
  if (error) {
    console.log("Error al obtener los datos actualizados:", error);
    return res.render('perfil', {
      nombre: req.session.nombre || '',
      email: req.session.email || '',
      success: false,
      alert: true,
      alertTitle: "Error al obtener los datos actualizados.",
      alertIcon: "error",
      showConfirmButton: false,
      timer: 1000,
      ruta: 'perfil'
    });
  }
  const updatedUser = results[0];
  console.log('Datos actualizados:', updatedUser);
});
```

En caso de que la actualización haya sido exitosa, se actualizan los datos en la sesión para que se puedan reflejar los cambios en la vista del usuario que se encontrará en ‘perfil.ejs’, asimismo se le dará al usuario un mensaje en el que se le da conocer que la actualización fue completamente exitosa.

con el ID que teníamos del usuario, y en caso de qué no lleguen a ser obtenidos, se le mande un mensaje al usuario, respecto a que los datos no pueden ser dados. Asimismo, se genera un mensaje para que se muestren en consola respecto a qué datos fueron actualizados.

```
req.session.nombre = updatedUser.nombre;
req.session.email = updatedUser.correo;
//Mensaje donde se confirma la actualización
return res.render('perfil', {
  nombre: updatedUser.nombre,
  email: updatedUser.correo,
  alert: true,
  alertTitle: "Actualización exitosa",
  alertIcon: "success",
  showConfirmButton: false,
  timer: 1500,
  ruta: 'perfil'
});
```

- Creación de ruta DELETE de usuario

Traslación de la ruta UPDATE, ahora continuamos con la ruta que deberá encargarse de una de las operaciones CRUD que se nos es faltante, siendo la operación DELETE la cual debemos abarcar; esta operación se debe encargar de poder eliminar de la base de datos, la cuenta en el que se está el usuario, siendo así que una vez se ha eliminado dicha cuenta, el usuario tenga que regresar a la página de iniciar sesión.

```
app.delete('/eliminarUsuario', (req, res) => {
  console.log("Solicitud DELETE recibida para eliminar usuario con ID:", req.session.userId);
  const userId = req.session.userId;
```

Comenzamos con definir la ruta que tendrá por nombre ‘/eliminarUsuario’, y comenzará con imprimir en la consola que se ha solicitado la eliminación de un usuario junto a la ID de solicitante, de a la par obtendremos el usuario que está autenticado guardando el valor en userID.

Y damos comienzo con la verificación de si userID tiene valor, en caso de qué no sea así, se nos imprimirán consola que no hay dicha usuario y a la parte se nos dará un mensaje donde se deja conocer dicha situación y lo redirige a la página de iniciar sesión.

Después de eso, en caso de qué si haya usuario, se ejecuta una consulta a la base de datos para poder eliminarlo, siendo así qué en el área donde se encuentra el signo de interrogación se ha puesto el valor que se tiene en

```
connection.query('DELETE FROM usuarios WHERE id = ?', [userId], (error, results) => {
  if (error) {
    console.log('Error al eliminar la cuenta:', error);
    return res.render('sesion', {
      success: false,
      alert: true,
      alertTitle: "Error al eliminar la cuenta.",
      alertIcon: "error",
      showConfirmButton: false,
      timer: 1000,
      ruta: 'perfil'
    });
  }
})
```

```
if (!userId) {
  console.log("No hay usuario.");
  return res.render('perfil', {
    success: false,
    alert: true,
    alertTitle: "No hay usuario.",
    alertIcon: "error",
    showConfirmButton: false,
    timer: 1000,
    ruta: ''
  });
}
```

userID, tras esa consulta se nos genera un if el cual validará si ha habido un error en la eliminación, y en caso que así sea, se registrará en la consola que ha habido un error al eliminar la cuenta y se dará una muestra de error en un mensaje dinámico en la página de iniciar sesión.

En caso de qué no haya sido así, ahora se destruiría la sesión del usuario para así poder asegurarnos de qué después de eliminar la cuenta ya no pueda seguir accediendo a la página, en donde también se generaría un if el cual abarcaría el error al cerrar la sesión, donde nos ofrecería un mensaje en la consola y a la par un mensaje de error, dando notificación en el Backend y en el Frontend.

```
return res.render('sesion', {
  success: true,
  alert: true,
  alertTitle: "Eliminación exitosa",
  alertIcon: "success",
  showConfirmButton: false,
  timer: 2000,
  ruta: '/'
});
```

En caso de que no se haya llegado a cumplir lo del if, se retornaría un mensaje, dando a conocer que todo fue hecho con éxito y redirigiría al usuario a la página de inicio, provocando así que se destruya la sesión del usuario y a la par, la eliminación de la cuenta del usuario.

```
req.session.destroy((err) => {
  if (err) {
    console.log('Error al cerrar sesión después de eliminar la cuenta:', err);
    return res.render('sesion', {
      success: false,
      alert: true,
      alertTitle: "Error al cerrar sesión.",
      alertIcon: "error",
      showConfirmButton: false,
      timer: 1000,
      ruta: 'perfil'
    });
  }
})
```

Con las operaciones CRUD abarcadas, ahora podemos dar comienzo a la generación de los HTML, para empezar a darle más forma a lo que buscamos abarcar en nuestro avance del proyecto.

- Creación de HTML para iniciar sesión

Tras unas conversaciones con el UI/UX, se concordó que se hiciera una página para iniciar sesión, para registrarse, y para demostrar y editar los datos que tendría el usuario; por ello, daremos comienzo con la creación del HTML para iniciar sesión, y dicho HTML será donde se dirigen apenas ingresen al servidor.

Dicho ya HTML, tendrá por nombre 'sesion.html', ahora, para la creación del HTML, se notó que se usó la misma estructura, tanto para encabezado como para el pie de página, es por ello por lo que se tomó la base de los anteriores HTML que han sido creados en la primera y segunda actividad.

Reporte individual

```
Fronted > sesion.html > ...
1  <!DOCTYPE html>
2  <html lang="es-MX">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style.css">
8      <title>Rhymes Music - Login</title>
9  </head>
10
11 <body>
12     <div class="contenedor">
13         <header id="home">
14             <div class="fondonegro"></div>
15             <span class="nombre">Rhymes Music</span>
16         </header>
17
18
19         <footer>
20             <div class="redessociales">
21                 <a href="https://www.facebook.com"><i class="fa-brands fa-facebook-f"></i></a>
22                 <a href="https://www.tiktok.com"><i class="fa-brands fa-tiktok"></i></a>
23                 <a href="https://www.x.com"><i class="fa-brands fa-x-twitter"></i></a>
24                 <a href="https://www.youtube.com"><i class="fa-brands fa-youtube"></i></a>
25             </div>
26             <span class="copyright">© 2025 Rhymes Music. Todos los derechos reservados.</span>
27         </footer>
28     </div>
29     <script src="https://kit.fontawesome.com/9551891e69.js" crossorigin="anonymous"></script>
30     <script src="/app.js"></script>
31
32
33
34 </body>
</html>
```

Con esa base ya obtenida, ahora agregamos el formulario para poder iniciar sesión, y dicho formulario es el que enviará los datos para la ruta ‘/sesión’ que tiene el método POST, siendo así que empecemos a abarcar la ruta CRUD, específicamente la de READ; a la par le damos un título en el que se especifique que es para iniciar sesión, y después con ello, empezamos el general formulario en donde requerimos el campo de usuario y contraseña, en donde esos datos son obligatorios para ser ingresados, pero en el campo de contraseña hacemos que se oculten los caracteres debido al type que le colocamos, y después de ello generamos un botón que podrá enviar el formulario.

```
<div class = loginForm>
    <h1 id="titulo">Inicia Sesión</h1>
    <form action='/sesion' method="POST">
        <label for="usuario">Usuario</label>
        <input type="text" name="usuario" id="usuario" placeholder="Introduce tu usuario" required>
        <label for="pass">Contraseña</label>
        <input type="password" name="pass" id="pass" placeholder="Introduce tu contraseña" required>
        <input type="submit" class="loginBtn" value="Iniciar sesión">
    </form>
    <a href="/registrarse.html" class="registrarBtn">Registrarse</a>
</div>
```

Después del formulario, agregamos un botón que sea específicamente para poder dirigir al usuario a la página para que se pueda registrar haciendo uso de un enlace para dicha página a la cual se debería de dirigir, y con ello empezaremos abarcar la operación CREATE de CRUD.

- Generación de HTML para registro de usuario

Para la generación del HTML para registrarse, damos primero la creación de dicho archivo dentro de la carpeta ‘fronted’, y dicho archivo tendrá por nombre ‘registrarse.html’, dando así una dirección válida al botón de registrarse en cuál está en la página de iniciar sesión.

Recordemos que estos archivos estarán en una carpeta llamada ‘Frontend’, la cual esperamos que contenga los HTML, los CSS, los JavaScript y las posibles imágenes que puedan llegar a ser utilizadas.

Podemos ver en el Head, que tiene el título de login, dando una especificación al usuario respecto a qué páginas se encuentra; a la par se le da un encabezado con el nombre del sitio y un Good que contendría las redes sociales como Facebook, TikTok, X y YouTube.

Y junto a eso podemos ver que tiene scripts, donde buscamos en enlazar un archivo JS para manejar la interactividad.

```

Fronted > registrarse.html ...
1  <!DOCTYPE html>
2  <html lang="es-MX">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <link rel="stylesheet" href="style.css">
8      <title>Rhymes Music - Registro</title>
9
10 </head>
11 <body>
12     <div class="contenedor">
13         <header id="home">
14             <div class="fondonegro"></div>
15             <span class="nombre">Rhymes Music</span>
16         </header>
17
18         <footer>
19             <div class="redessociales">
20                 <a href="https://www.facebook.com"><i class="fa-brands fa-facebook-f"></i></a>
21                 <a href="https://www.tiktok.com"><i class="fa-brands fa-tiktok"></i></a>
22                 <a href="https://www.x.com"><i class="fa-brands fa-x-twitter"></i></a>
23                 <a href="https://www.youtube.com"><i class="fa-brands fa-youtube"></i></a>
24             </div>
25             <span class="copyright"> 2025 Rhymes Music. Todos los derechos reservados.</span>
26         </footer>
27     </div>
28     <script src="https://kit.fontawesome.com/9551891e69.js" crossorigin="anonymous"></script>
29     <script src="/app.js"></script>
30
31 </body>
32
33 </html>

```

Como comentamos en el anterior requerimiento, hacemos una base que tienen la mayoría de los HTML, es que estábamos manejando, siendo la única diferencia, el título, el cual porta el nombre de registro.

Mismamente contiene un encabezado y pie de página en el cual abarcan tanto el nombre del sitio como las redes sociales que se manejan.

Obteniendo la base del HTML de registro, ahora podemos incluir el formulario que estará empeñado en el registro, el cual enviará los datos

a la ruta ‘/registrarse’ con el método POST, siendo así que abarquemos la operación CREATE de CRUD, para la creación de dicho formulario fue necesario generar varios campos como para ingresar el usuario, el email y la contraseña, si es así que esto sean obligatorios en su ingreso, y por parte del email se tiene contemplado que tenga el formato con @; una vez hecho, eso se genera un botón, el cual tendrá por nombre crear usuario. Generando así, los datos necesarios que deberá enviar el formulario para la ruta, donde se crea el nuevo usuario en la base de datos.

```

<div class = "registroForm">
    <h1 id="registroTitulo">Crear un usuario</h1>
    <form action='/registrarse' method="POST">
        <label for="usuario">Usuario</label>
        <input type="text" name="usuario" id="usuario" placeholder="Crea tu usuario" required>
        <label for="email">Correo</label>
        <input type="email" name="email" id="email" placeholder="Introduce tu email" required>
        <label for="pass">Contraseña</label>
        <input type="password" name="pass" id="pass" placeholder="Introduce tu contraseña" required>
        <input type="submit" class="registrarBtn" value="Crear usuario">
    </form>
</div>

```

- Mensajes de éxito, error y advertencia

He comentado a lo largo de los requerimientos que se usan mensajes para dar a conocerle al usuario lo que ha ocurrido, y dichos mensajes son posibles debido a que se integró SweetAlert2 y EJS como motor de plantillas, obteniendo así, alertas dinámicas para las páginas que se han creado.

Recordemos que en ‘server.js’ si implementó el motor de plantillas y que serían obtenidas en una carpeta llamada ‘views’, dicha carpeta contendría los archivos ‘.ejs’, cada uno de los archivos ‘.ejs’, portan la misma estructura, siendo sólo el cambio en el título en el encabezado, donde define que página es en la que se debe encontrar.

```

//Motor de plantillas
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));

```

Reporte individual

```
<!DOCTYPE html>
<html lang="es-MX">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Rhymes Music - Login</title>
</head>

<body>
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
    <% if(typeof alert != "undefined"){ %>
        <script>
            Swal.fire({
                title: '<%= alertTitle %>',
                text: '<%= alertMessage %>',
                icon: '<%= alertIcon %>',
                showConfirmButton: '<%= showConfirmButton %>',
                timer: '<%= timer %>'
            }).then(()=>{
                | window.location='/<%= ruta %>'
            })
        </script>
    <% } %>
</body>

</html>
```

Podemos ver como en el Body se contiene un script, el cual hace la integración del SweetAlert2, y después de ello se tiene otro script en el cual contiene el código para llamar una función llamada Swal.fire(), siendo esto, una función que nos ayudó a mostrar alertas personalizadas, las cuales pueden contener un título, texto, ícono, botones y el tiempo en el que se mostrará dicha alerta, siendo así que también con ello, pueda ser redirija a alguna ruta después de cerrar la alerta. Esto ocurre en la mayoría de las secciones, donde se generó un mensaje, ya sea de éxito, error y advertencia.

- Generación de EJS para mostrar el perfil del usuario

Con la generación de los HTML para iniciar sesión y registrarse, ahora nos empeñamos en generar el archivo donde se mostrarán los datos del usuario y a la par, el poder editarlos.

```
> ◊ perfil.ejs > ...
<!DOCTYPE html>
<html lang="es-MX">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Rhymes Music - Perfil</title>
</head>

<body>
    <div class="contenedor">
        <header id="home">
            <div class="fondonegro"></div>
            <span class="nombre">Rhymes Music</span>
        </header>

        <footer>
            <div class="redessociales">
                <a href="https://www.facebook.com"><i class="fa-brands fa-facebook-f"></i></a>
                <a href="https://www.tiktok.com"><i class="fa-brands fa-tiktok"></i></a>
                <a href="https://www.x.com"><i class="fa-brands fa-x-twitter"></i></a>
                <a href="https://www.youtube.com"><i class="fa-brands fa-youtube"></i></a>
            </div>
            <span class="copyright">© 2025 Rhymes Music. Todos los derechos reservados.</span>
        </footer>
    </div>

    <script src="https://kit.fontawesome.com/9551891e69.js" crossorigin="anonymous"></script>
    <script src="/app.js"></script>

    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
    <% if(typeof alert != "undefined"){ %>
        <script>
            Swal.fire({
                title: '<%= alertTitle %>',
                icon: '<%= alertIcon %>',
                showConfirmButton: '<%= showConfirmButton %>',
                timer: '<%= timer %>'
            }).then(()=>{
                | window.location='/<%= ruta %>'
            })
        </script>
    <% } %>
</body>

</html>
```

Y para ello se decidió que el archivo sea ‘.ejs’; dicho archivo tiene por nombre, ‘perfil’, y se les ha agregado una base como en los anteriores HTML de iniciar sesión y registrarse, en los cuales se les da un encabezado y un pie de página, lo único que los diferencia de los otros HTML, es que aquí es posible incluir el script, el cual abarca los mensajes dinámicos que se buscan ofrecer al usuario, y dicho archivo estará en la carpeta ‘views’.

Teniendo la base del archivo ‘perfil’, ahora podemos empezar a abarcar los apartados donde se deberán mostrar los datos del perfil, la edición de datos y la barra lateral, en donde se mostrarán las acciones que deberá aportar ‘perfil.ejs’.

- Barra de secciones en ‘perfil.ejs’

Para la barra de secciones, es necesario generar un contenedor para dicha sección, y empezamos a agregar un botón de perfil el cual es requerido por parte de UI/UX siendo un círculo gris, después de ello se generan botones ya sean para la sección de Perfil, Ajustes y Cerrar sesión.

Para la funcionalidad de dichos botones, es necesario ir al archivo ‘app.js’, el cual contendrá un evento que maneje la interacción con los botones en el perfil del usuario, ahora, podemos observar que es un ‘document.addEventListener(‘DOMContentLoaded’,’, dicho evento solo ocurre cuando

```
<div class="perfilLateral">
  <div class="perfilBtn"></div>
  <button id="btnPerfil" class="botonPerfil">Perfil</button>
  <button id="btnAjustes" class="botonPerfil">Ajustes</button>
  <button id="btnCerrar" class="botonPerfil">Cerrar sesión</button>
</div>
```

```
//Para cuando se haga click en alguno de los botones que se encuentran el perfil.ejs
document.addEventListener('DOMContentLoaded', () => {
  console.log('Documento cargado'); //Verifica que este el documento cargado

  // Verifica si los botones existen
  const btnPerfil = document.getElementById('btnPerfil');
  const btnAjustes = document.getElementById('btnAjustes');
  const btnCerrar = document.getElementById('btnCerrar');
  const infoPerfil = document.getElementById('infoPerfil');
  const actuInfo = document.getElementById('actuInfo');

  console.log(btnPerfil, btnAjustes, btnCerrar, infoPerfil, actuInfo); //Verifica que esten

  //Marca en consola que no se encontraron
  if (!btnPerfil || !btnAjustes || !btnCerrar || !infoPerfil || !actuInfo) {
    console.error('Uno o mas elementos no se encontraron en el DOM.');
    return;
  }

  //Funcionalidad de botones, incluyendo un mensaje para que se muestre en la
  //consola que esten siendo usados
  btnPerfil.addEventListener('click', () => {
    console.log('Botón "Perfil" clickeado');
    infoPerfil.style.display = 'block';
    actuInfo.style.display = 'none';
  });

  btnAjustes.addEventListener('click', () => {
    console.log('Botón "Ajustes" clickeado');
    actuInfo.style.display = 'block';
    infoPerfil.style.display = 'none';
  });

  btnCerrar.addEventListener('click', () => {
    window.location.href = '/';
  });
});
```

estructura, donde especificamos que debe ocurrir si se presiona uno, por ejemplo, en el botón de perfil, si este es presionado, se mostrara en consola un mensaje notificándolo, y se hará un cambio en el CSS donde el elemento ya sea de infoPerfil y actuInfo cambien del valor, siendo un cambio para mostrar y ocultar un apartado específico. El botón distinto, es el de cerrar sesión, donde solamente redirige a la página de iniciar sesión.

- Apartado de Perfil en ‘perfil.ejs’

Para el apartado de perfil, solo se hizo un contenedor que tendrá la información y a la par se le da el ID que es clave para que se muestre u oculte; con ello, lo que deberá contener son etiquetas que abarcan la información de nombre y email, y son los datos tomados de la información del usuario de la base de

datos, antes aseguramos que la variable tenga información, en caso de que no contenga, pues se da a conocer con un mensaje que diga ‘No disponible’.

```
<!-- Apartado de Perfil -->
<div class="infoPerfil" id="infoPerfil">
  <p><strong>Nombre</strong><br><%= nombre || 'No disponible' %></p>
  <p><strong>Email</strong><br><%= email || 'No disponible' %></p>

  <button id="btnEliminar" class="botonPerfil">Eliminar Cuenta</button>
</div>
```

Junto a eso, está el botón de eliminar, donde hacemos uso de JavaScript para su funcionalidad.

Reporte individual

- Funcionalidad de botón para eliminar usuario en perfil.ejs

Para la funcionalidad el botón fue necesario hacer uso de un evento en el cual, si se hace click, ocurra la siguiente función, la cual manda un mensaje de confirmación haciendo uso de SweetAlert, que le

```
//Para cuando se haga click en el botón de eliminar en perfil.ejs
document.getElementById("btnEliminar").addEventListener("click", function() {
  Swal.fire({
    title: "Estás seguro",
    text: "Esta acción no se puede deshacer.",
    icon: "warning",
    showCancelButton: true,
    confirmButtonColor: "#d33",
    cancelButtonColor: "#3085d6",
    confirmButtonText: "Sí, eliminar",
    cancelButtonText: "Cancelar"
  }).then((result) => {
    if (result.isConfirmed) {
      fetch("/eliminarUsuario", {
        method: "DELETE",
        headers: {
          "Content-Type": "application/json"
        }
      }).then(response => response.json())
      .then(data => {
        Swal.fire({
          title: data.success ? "Eliminado" : "Error",
          text: data.message,
          icon: data.success ? "success" : "error",
          timer: 2000,
          showConfirmButton: false
        }).then(() => {
          if (data.success) {
            window.location.href = "/"; //Mandara a sesion
          } else {
            window.location.href = "/";
          }
        });
      });
    }
  .catch(error => { //Aunque haya un error, se borra correctamente en la base de datos
    Swal.fire({
      title: "Exitoso",
      text: "Se eliminó la cuenta.",
      icon: "success",
      timer: 3000,
      showConfirmButton: false
    });
    window.location.href = "/";
    console.error("Eliminada la cuenta:", error);
    setTimeout(() => {window.location.href = "/"}, 3000);
  });
  });
});
```

comenta al usuario si está seguro de querer hacer dicha acción, dándole dos botones, ya sean para cancelar o confirmar.

Si el usuario llega a confirmar, se hace la solicitud al DELETE para eliminar el usuario, donde utiliza la ruta '/eliminarUsuario', siendo así que abarquemos la operación de DELETE de CRUD, tras hacer la solicitud se maneja la respuesta haciendo que se convierta en JSON devolviendo los datos por el servidor confirmando si fue exitosa o no la operación. Con eso hecho, se muestra una alerta con el resultado de la eliminación y después si data.success es true o false se muestra la alerta con el título de eliminado o error, y una vez se cierra la sesión la página lo redirige a la página de iniciar sesión.

- Apartado de Ajustes en 'perfil.ejs'

Para el apartado de Ajustes se generó un formulario donde el atributo action abarca la ruta que debe ser usada para enviar los datos del formulario, siendo dirigidos a '/actualizarPerfil', pero se le agrega el parámetro '_method=PATCH' se utilizara para simular el método HTTP PATCH porque ciertos navegadores solo soportan GET y POST; aunque en el método se especifica que el formulario se enviará en método POST y el valor '_method=PATCH'; hace que se trate como una solicitud PATCH en el servidor.

Ahora, ocultamos el campo que nos ayuda a simular el método PATCH al incluirlo en el formulario nos ayuda a que en Backend nos permita que se interprete como una solicitud PATCH; y con ello damos comienzo a crear los campos para que se ingresen nuevo nombre, email y contraseña, especificando en el input sobre el tipo y cuales son obligatorios. Junto a eso, está el botón para enviar el formulario, que se encargara después de ser enviada la información al servidor con los datos ingresados por el usuario.

```
<div class="actuInfo" id="actuInfo">
  <form action="/actualizarPerfil?_method=PATCH" method="POST" id="updateForm">
    <input type="hidden" name="_method" value="PATCH"> 
    <label for="newName">Nuevo Nombre:</label>
    <input type="text" id="newName" name="newName" value="<% nombre %>" required>

    <label for="newEmail">Nueva Email:</label>
    <input type="email" id="newEmail" name="newEmail">

    <label for="newPassword">Nueva Contraseña:</label>
    <input type="password" id="newPassword" name="newPassword">

    <button type="submit" id="btnActu">Actualizar</button>
  </form>
</div>
```

- Funcionalidad del botón de actualizar datos en 'perfil.ejs'

Teniendo encuentra donde se encuentra el botón de actualizar, ahora nos centraremos en ver que nos ayuda en que sea funcional dicho botón.

Mismamente que la funcionalidad del botón para eliminar, hacemos uso de un even listener donde escuchamos el evento cuando se presiona el botón de actualizar, y con eso hecho evitamos que se envié

el formulario de manera tradicional para empezar a manejar la solicitud, ahora, con eso empezamos a obtener los datos de los campos que fueron ingresados haciendo que se guarden y después hacemos la solicitud al método PATCH, siendo así que abarquemos la última operación CRUD, la de UPDATE, cuando se hace la solicitud se deberán enviar los datos con formato JSON. Cuando se enviaron los datos, se empieza a procesar la respuesta del servidor, y se empieza a manejar la respuesta con un if, el cual verifica si todo salió de forma exitosa y si llegara ser así, se actualizarían los datos en la página con los nuevos valores para después redirigir a 'perfil.ejs', en caso de que no sea así, en el else se le daría conocer al usuario con un mensaje que hubo un error en la actualización.

- Creación de Barra lateral

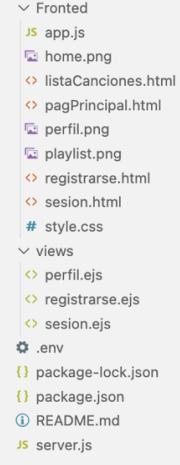
Tras la explicación de lo que conllevo en el Backend y Frontend, ahora tratare de mostrar como abarque la creación de la barra lateral que se me fue pedida por UI/UX.

Reporte individual

Podemos ver en el código que se hace un contenedor principal que abarca toda la barra lateral y después de ello se hace una sección de menú que es usado para cada elemento, podemos ver cómo hacemos uso de imágenes y de texto que tendrá mayor forma en el CSS, pero en total deben ser creados 3 ítems y deben redirigir a su respectiva página.



```
<div id="barraLateral">
  <div class="menu">
    <a href="#" class="icono" id="iconoDistinto">
      
    </a>
    <span class="globo">Perfil</span>
  </div>
  <div class="menu">
    <a href="pagPrincipal.html" class="icono">
      
    </a>
    <span class="globo">Home</span>
  </div>
  <div class="menu">
    <a href="listaCanciones.html" class="icono">
      
    </a>
    <span class="globo">Playlist</span>
  </div>
</div>
```



Podemos notar como las imágenes son implementadas en el fronted, y las imágenes añadidas son las siguientes:



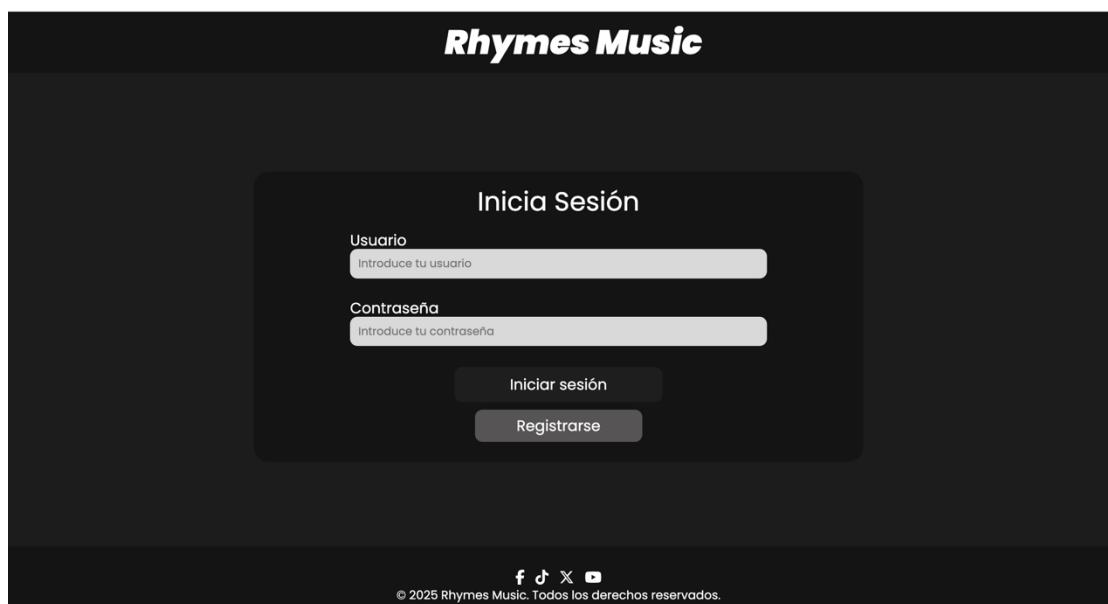
Dicha barra, solo podrá ser vista en los archivos 'pagPrincipal', 'Perfil' y 'listaCanciones'.

-Diseño de UI/UX y responsive en los HTML y EJS

Tras ver lo que fue creado, ahora veremos como quedó el diseño en las nuevas páginas, siguiendo el diseño dado por el UI/UX.

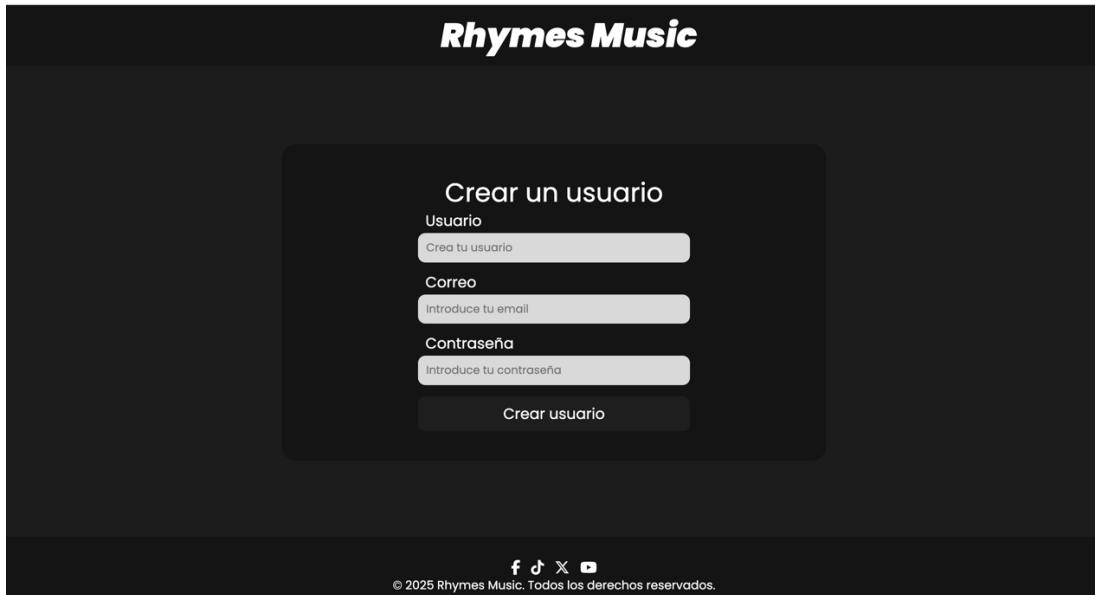
Iniciar sesión

Para la página de iniciar sesión, se da un cambio de color a los botones si el cursor esta encima y el botón de registrarse dirige a donde corresponde.



Registrarse

Para esta página, se podrá ver como el botón se cambia de color si el cursor esta encima.



Barra lateral

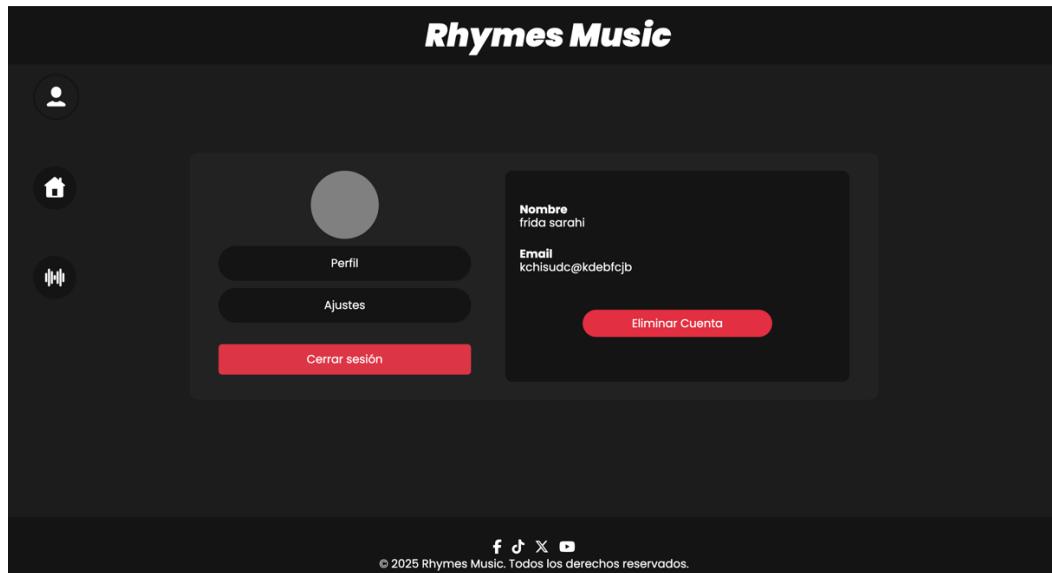
Para la barra lateral, se llevó a cabo que puedan girar si es que el cursor esta encima y a la par muestre un globo de texto, podemos ver en las siguientes imágenes como esta en el lado izquierdo y cumple el que gire y muestre el globo de texto.

A screenshot of the Rhymes Music website. On the left side, there is a vertical sidebar with three icons: a microphone, a house, and a waveform. To the right of the sidebar, there is a "Proyectos" (Projects) section. Within this section, there is a box for the album "Un Verano Sin Ti" by Bad Bunny. The box contains text about the album's success and a small image of Bad Bunny performing on a beach. A text bubble with the URL "localhost:3000/perfil" is shown above the sidebar area.

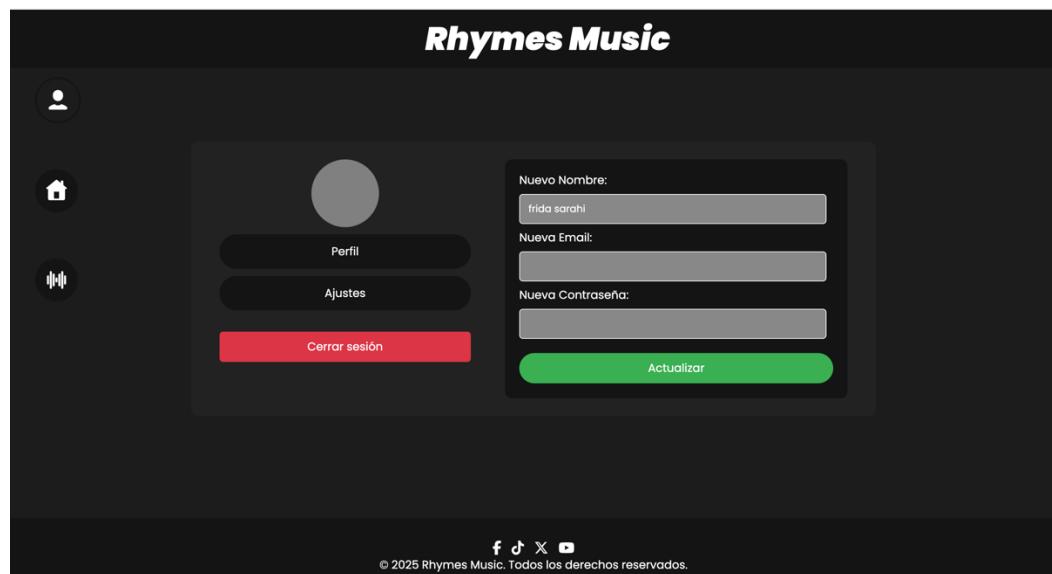
Reporte individual

Perfil del usuario

Para la página de perfil, podemos ver cómo nos muestra las secciones, siendo que si presionas el botón de perfil, te muestre el área de perfil, donde muestra los datos del usuario y el botón de eliminar.



Y después, esta que si presionas el botón de actualizar, se muestre ese apartado donde este la posibilidad de ingresar texto en los campos correspondientes.



Todos los botones mostrados en la página son funcionales, respecto a lo responsive de la página, hubo inconvenientes respecto al tiempo, siendo así que se deba pedir ayuda a mi compañero Pedro De León para que abarque esa área a profundidad, cabe aclarar que el diseño estaba muy complejo para las habilidades que tenemos al momento.

Inconvenientes/Problemas y Solución

Se dio a conocer una problemática gracias al QA, donde notifico que hubo un problema a la validación de que solo se ingrese un usuario y sea único, debido a eso, se tuvo que corregir el código en ciertas áreas, empezando con la solución de corregir los datos de la tabla.

Notemos a continuación como cambian los datos de la tabla:



Field	Type	Null	Key	Default	Extra
id	int	NO	PRIMARY	NULL	Auto_increment
nombre	varchar(50)	YES		NULL	
correo	varchar(100)	YES	UNIQUE	NULL	
password	varchar(100)	YES		NULL	

Field	Type	Null	Key	Default	Extra
id	int	NO	PRIMARY	NULL	Auto_increment
nombre	varchar(50)	YES	UNIQUE	NULL	
correo	varchar(100)	YES	UNIQUE	NULL	
password	varchar(100)	YES		NULL	

Quedando así la creación de la base de datos de forma local corregida:

```
mysql> CREATE TABLE usuarios (    id INT AUTO_INCREMENT PRIMARY KEY,    nombre VARCHAR(50) UNIQUE,    correo VARCHAR(100) UNIQUE,    password VARCHAR(100));  
Query OK, 0 rows affected (0,02 sec)
```

En la corrección del código solo hubo cambio en el código donde abarcamos la operación de registrar un nuevo usuario. Recordemos que una obtenemos los datos mandados por el formulario, y de ahí empezamos con la modificación donde hacemos uso de un Try, que contiene la encriptación de la contraseña, la consulta a la base de datos donde hacemos uso de ‘connection.query()’ donde buscamos insertar el nuevo usuario con los datos dados.

```
app.post('/registrarse', async(req, res)=>{  
  try{  
    let passwordHash = await bcryptjs.hash(pass, 8);  
    connection.query('INSERT INTO usuarios (nombre, correo, password) VALUES (?,?,?)', [usuario, email, passwordHash],(error, results) => {  
      if (error) {  
        console.log(error);  
        let alertMessage = "Ocurrió un error, favor de intentar de nuevo";  
        //Verifica que el error es por entrada de datos duplicados  
        if (error.code === 'ER_DUP_ENTRY') {  
          //Revisamos en el mensaje del error que campo está duplicado  
          if (error.sqlMessage.includes("nombre")) {  
            alertMessage = "El nombre de usuario ya existe, intenta con otro.";  
          } else if (error.sqlMessage.includes("correo")) {  
            alertMessage = "El correo ya está registrado, favor de intentar con otro.";  
          }  
        }  
      }  
    })  
  }  
})
```

Después de ello, se usa un if para manejar el error en la consulta donde genera una alerta para notificar al usuario y a la par dando a conocer el error en la consola, dentro de ese if, habrá otro que manejará los datos duplicados, donde abarca el error de datos duplicados por MySQL siendo dado este error debido a que la columna porta la restricción de único, y si el error llegara a ser porque el nombre o el email fueron repetidos, se le notificaría al usuario específicamente cual se repitió.

Reporte individual

```
return res.render('sesion', {
  alert: true,
  alertTitle: "Error",
  alertMessage: alertMessage,
  alertIcon: "warning",
  showConfirmButton: true,
  timer: 1500,
  ruta: "registrarse"
});
} else {
  return res.render('registrarse', {
    alert: true,
    alertTitle: "Registro",
    alertMessage: "¡Registro exitoso!",
    alertIcon: "success",
    showConfirmButton: false,
    timer: 1500,
    ruta: ''
  });
}
```

Tras eso, se da una respuesta en caso de error, donde muestra una alerta en la página de sesión, donde le da a conocer al usuario el error detectado y siendo una alerta de warning.

Y también puede dar una respuesta de éxito, donde solo da a conocer al usuario con mensaje que el registro fue exitoso y sin inconvenientes.

Para finalizar la corrección del código, se maneja el bloque catch donde manejamos el error del servidor, dando a conocer que ocurrió un problema en el proceso de encriptación

```
  }
} catch(err){
  console.error("Error al encriptar la contraseña:", err);
  return res.render('sesion', {
    alert: true,
    alertTitle: "Error",
    alertMessage: "Ocurrió un error en el servidor, por favor intente de nuevo.",
    alertIcon: "error",
    showConfirmButton: true,
    timer: 1500,
    ruta: "registrarse"
});
```

Referencias

<https://www.oracle.com/mx/mysql/what-is-mysql/#:~:text=MySQL%20es%20r%C3%A1pid%2C%20confiable%2C%20ampliable,de%20funciones%20enriquecido%20y%20%C3%BAtil>