Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Proyecto 1 del Curso de Lenguajes de Programación

Profesor:

Oscar Mario Víquez Acuña

Estudiantes:

René Sánchez Torres - 2020051805

Julián Gutiérrez Susi – 2013111419

Alajuela Septiembre, 2023

TABLA DE CONTENIDOS

1. INTRODUCCIÓN

Este documento proporciona una documentación técnica sobre un proyecto de sistema de música en línea. Su principal propósito es permitir a los usuarios experimentar, organizar y administrar su colección de canciones de manera eficiente, amigable y personalizada.

El proyecto se ha construido utilizando React en el Front-end para crear una interfaz de usuario altamente interactiva y adaptable que garantiza una experiencia de usuario intuitiva y receptiva. Complementariamente, en el Back-end, se ha implementado un servidor en Go que gestiona solicitudes HTTP, dirige la lógica de negocio y facilita la reproducción de archivos de audio. En este documento se ofrecerá una visión detallada de la implementación y funcionamiento de todos los componentes esenciales de este sistema de música.

2. OBJETIVO DEL PROYECTO

El sistema tiene como objetivo principal permitir a los usuarios interactuar con su biblioteca musical de manera intuitiva, permitiendo la creación de playlists, la búsqueda y reproducción de canciones, y la visualización de su colección de música. La visión detrás de este proyecto es proporcionar una solución simple y eficiente que satisfaga las necesidades de los amantes de la música, ofreciendo una experiencia de usuario rica y placentera.

3. ESTRUCTURA DEL SISTEMA

El sistema consta de dos partes principales:

3.1. Front-end

El Front-end se ha desarrollado utilizando React, lo que proporciona una interfaz de usuario interactiva y responsive. Esto significa que los usuarios pueden interactuar de manera intuitiva con el sistema, ya que React permite construir componentes reutilizables y una experiencia de usuario fluida y atractiva.

3.2. Back-end

El Back-end está implementado en Go y se encarga de gestionar las solicitudes HTTP, manejar la lógica del negocio y servir los archivos de audio. Go es conocido por su rendimiento y eficiencia, lo que garantiza respuestas rápidas a las solicitudes de los usuarios y un manejo efectivo de la lógica empresarial y los archivos de audio, asegurando la estabilidad y seguridad del sistema.

4. FUNCIONALIDADES PRINCIPALES

4.1. Visualización de Canciones

Los usuarios tienen la capacidad de explorar de forma exhaustiva la lista completa de canciones disponibles en la plataforma, lo que les permite examinar y seleccionar su música de interés de manera conveniente.

4.2. Reproducción de Canciones

La función de reproducción de canciones brinda a los usuarios la posibilidad de elegir sus canciones preferidas de la lista disponible y disfrutarlas de principio a fin. Esta característica se esfuerza por ofrecer una experiencia de escucha de alta calidad y sin interrupciones.

4.3. Creación de Playlists

Los usuarios pueden ejercer su creatividad al crear listas de reproducción personalizadas, nombrarlas y seleccionar las canciones que deseen incluir en ellas. Esto les permite adaptar su experiencia musical a sus gustos individuales y necesidades.

4.4. Búsqueda de Canciones

La funcionalidad de búsqueda de canciones simplifica la localización de pistas específicas a través de la plataforma, permitiendo a los usuarios encontrar rápida y fácilmente sus canciones favoritas por nombre.

4.5. Adición de Canciones a Playlists

Los usuarios tienen la flexibilidad de ampliar y personalizar sus listas de reproducción personalizadas al agregar nuevas canciones según sus preferencias. Esto les da un control total sobre su experiencia musical y la capacidad de crear listas de reproducción dinámicas.

5. REPRODUCTOR DE MÚSICA

El reproductor de música consta de dos componentes principales. Por un lado, el servidor, implementado en el lenguaje de programación Go y el cliente, desarrollado en JavaScript con React. A continuación se detallan más específicamente estos componentes:

5.1. SERVIDOR

El servidor está implementado en el lenguaje de programación Go, gestiona todas las solicitudes relacionadas con la música y puede ser accedido a través de protocolos HTTP y TCP en los puertos 8081 y 8080 respectivamente. Del lado del servidor muchas funcionalidades del proyecto para poder manejar los archivos de música se hacen a través de endpoints.

5.1.1. Endpoints utilizados

/ver

/reproducir

/crearPlaylist

/agregarCancionAPlaylist

5.1.2. Carga de Canciones

El servidor lee los archivos de música (MP3) desde un directorio específico (el cual está configurado hacia la ruta de una carpeta en nuestra máquina local donde están todas las canciones "*/Users/rsanchez/Documents/MusicProject"*) al recibir una solicitud en el endpoint /ver.

Usa el paquete os para listar todos los archivos en el directorio especificado y filtra aquellos que tienen la extensión .mp3.

Responde con la lista de nombres de archivos de música en formato JSON. Esto gracias al import que usamos de "encoding/json".

5.1.3. Reproducción de Canciones

Al recibir una solicitud en el endpoint /reproducir, el servidor abre y decodifica el archivo MP3 solicitado y lo reproduce usando el paquete beep.

El archivo de música se localiza usando la ruta especificada y el nombre del archivo pasado como parámetro.

5.1.4. Creación de Playlists

El servidor permite la creación de playlists mediante el endpoint /crearPlaylist.

La nueva playlist se añade a una variable playlists que almacena todas las playlists creadas durante la ejecución del servidor.

5.1.5. Agregar Canción a Playlist

Se implementa un nuevo endpoint, /agregarCancionAPlaylist, para añadir canciones a una playlist existente.

5.2 CLIENTE

El cliente está desarrollado en JavaScript con React, proporciona una interfaz de usuario atractiva y se conecta al servidor a través de endpoints. Esto permite a los usuarios interactuar con la aplicación y reproducir música de manera efectiva. React se encarga de presentar la interfaz de usuario y de interactuar con el servidor mediante solicitudes HTTP.

5.2.1. Lista de Canciones

Muestra la lista de canciones disponibles en el servidor.

Permite la filtración de canciones mediante un campo de búsqueda.

5.2.2. Creación de Playlists

Proporciona un botón para crear nuevas playlists.

El nombre de la nueva playlist se especifica en un campo de texto.

5.2.3. Agregar Canción a Playlist

Cada canción en la lista tiene un botón o mecanismo para añadirla a una playlist existente.

6. ADMINISTRACIÓN EN MEMORIA DE LA LISTA DE CANCIONES

La lista de canciones y las playlist se están administrando directamente en memoria usando variables. Esto significa que no se está utilizando una base de datos o almacenamiento en disco para guardar esta información. Una vez que el servidor se detiene o se reinicia, se perderá toda la información almacenada en memoria.

6.1. Lista de Canciones

La lista de canciones se construye en el endpoint /ver cada vez que se hace una solicitud a ese endpoint. Este endpoint lee el directorio de música y construye una lista de objetos Canción basado en los archivos .mp3 encontrados en ese directorio. Esta lista de canciones no se almacena en una variable; se construye

dinámicamente cada vez que se solicita el endpoint y se envía directamente en la respuesta.

6.2. Playlists

Las playlists, por otro lado, se almacenan en una variable playlists en el nivel de paquete del servidor Go. Cada vez que se crea una nueva playlist mediante el endpoint /crearPlaylist, se añade un nuevo objeto Playlist a esta variable. La lista de Playlist se mantiene en memoria mientras el servidor esté en ejecución.

7. MANEJO DE CONFLICTOS Y EXCEPCIONES

A continuación, se explican las estrategias para gestionar errores, manejar respuestas de error HTTP y registrar problemas mediante registros de errores. También se examina cómo se manejan los errores relacionados con la decodificación JSON en las solicitudes HTTP.

7.1. HTTP Handlers

Cada HTTP handler en Go debe manejar cualquier error o excepción que pueda ocurrir durante su ejecución. Los errores se manejan típicamente verificando si un error es diferente de nil después de realizar operaciones que pueden fallar, como abrir un archivo o decodificar un stream de MP3. Si no se puede acceder a la ruta muestra un mensaje de error al abrir el archivo. Se envía una respuesta de error HTTP con un código de estado 500 (Internal Server Error) y se retorna del handler.

7.2. HTTP Error Responses

Cuando se detecta un error, se utiliza http.Error para enviar una respuesta de error HTTP con un mensaje de error específico. Esto permite que el cliente que realiza la solicitud sepa exactamente qué salió mal.

7.3. Error Logging

Los errores también se registran en la consola del servidor usando log.Println, lo que ayuda en la detección y diagnóstico de problemas.

A través de este print en consola del lado del servidor podemos ver si la conexión que está realizando el cliente fue exitosa y nos indica desde donde se está conectando el cliente.

7.4. Error en Decodificación JSON

Cuando se decodifica el cuerpo de una solicitud HTTP, también se manejan los errores, por ejemplo, si el cuerpo no contiene JSON válido.

8. IMPLEMENTACIÓN DE SOCKETS Y PROCESOS EN PARALELO

Lamentablemente, en la implementación actual del proyecto de música en línea, no se pudo llevar a cabo la sincronización y gestión adecuada de sockets y procesos en paralelo. Sin embargo, es importante destacar cómo hubiera sido la implementación ideal si se hubieran adoptado estas prácticas.

En un escenario hipotético donde la sincronización se hubiera implementado de manera efectiva, el proyecto habría alcanzado un nivel superior de eficiencia y confiabilidad. Los aspectos clave habrían sido los siguientes:

8.1. Sincronización de Sockets

Se habrían utilizado mecanismos de sincronización como bloqueos y semáforos para garantizar una comunicación ordenada y segura entre el servidor y el cliente. Esto habría evitado problemas de concurrencia en la manipulación de datos a través de los sockets.

8.2. Procesamiento Paralelo Controlado

La implementación de sincronización habría permitido un procesamiento paralelo más eficiente, asegurando que las solicitudes de los clientes se manejaran de manera ordenada y sin conflictos.

8.3. Manejo de Datos Compartidos

La sincronización habría garantizado la integridad de los datos compartidos, como las listas de canciones y las playlists almacenadas en memoria. Esto habría evitado problemas de concurrencia y posibles conflictos al acceder y modificar estos datos.

8.4. Control de Concurrencia en la Reproducción

Se habría implementado un sistema de control de concurrencia para la reproducción de canciones, lo que habría evitado que múltiples clientes intentaran reproducir la misma canción al mismo tiempo, mejorando la experiencia del usuario.

9. ELEMENTOS DE MEJORA O CAMBIO

Cuando se diseña un sistema a gran escala, existen numerosos aspectos a considerar para asegurar robustez, eficiencia, y escalabilidad. Aquí hay algunos elementos de mejora o cambio con miras en una implementación robusta de un sistema a gran escala de este proyecto:

9.1 Escalabilidad

Horizontal Scaling: Esta técnica permite aumentar la capacidad del sistema agregando más máquinas o nodos para distribuir la carga. Es especialmente útil para manejar un mayor número de usuarios y mantener un alto rendimiento.

Vertical Scaling: Consiste en aumentar los recursos (CPU, memoria, almacenamiento) de un nodo existente. Esto es útil cuando se necesita una mayor capacidad de procesamiento en un servidor específico.

9.2. Alta Disponibilidad y Resiliencia

Redundancia: La redundancia implica tener múltiples instancias del servicio en funcionamiento para evitar que un solo punto de falla detenga todo el sistema. Esto garantiza una mayor disponibilidad y continuidad del servicio.

Balanceo de Carga: Distribuir las solicitudes entrantes entre múltiples instancias de la aplicación garantiza una distribución uniforme de la carga, lo que mejora la resiliencia y la capacidad de respuesta del sistema.

9.3. Manejo de Estado

Base de Datos: Almacenar el estado de la aplicación, como las playlists, en una base de datos robusta y escalable asegura que la información esté disponible y persista incluso en situaciones de fallo.

Caché: Implementar sistemas de caché ayuda a reducir la carga en la base de datos y mejora el tiempo de respuesta al almacenar temporalmente datos que se acceden con frecuencia.

9.4. Seguridad

Autenticación y Autorización: Implementar mecanismos sólidos de autenticación y autorización es esencial para proteger los datos y recursos del sistema. Verificar la identidad del usuario y gestionar sus permisos garantiza un acceso seguro.

Validación y Sanitización: Validar todas las entradas del usuario y sanear las salidas es fundamental para prevenir ataques como la inyección de código y asegurar que los datos sean seguros.

9.5. Monitoreo y Logging

Implementar soluciones de monitoreo y logging permite rastrear el estado del sistema en tiempo real y diagnosticar problemas de manera eficaz. Esto es esencial para mantener un sistema saludable y detectar problemas antes de que afecten a los usuarios.

Estas mejoras son fundamentales para garantizar un sistema a gran escala robusto, eficiente y confiable. Cada uno de estos elementos contribuye al rendimiento general del sistema y a la satisfacción del usuario.

10. CONCLUSIÓN

En resumen, al considerar los diversos elementos mencionados anteriormente, se puede concluir que el diseño y desarrollo de un sistema de música en línea requiere una cuidadosa planificación y atención a múltiples aspectos técnicos. El uso de tecnologías como React y Go proporciona una base sólida para la interfaz de usuario y el backend del sistema.

Sin embargo, es esencial abordar la persistencia de datos y la escalabilidad para garantizar que el sistema sea resistente y pueda crecer para satisfacer las demandas futuras. La gestión de errores y excepciones, junto con el monitoreo en tiempo real, son aspectos cruciales para mantener el sistema en funcionamiento de manera eficiente.