Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Lenguajes de Programación

Primer Tarea Programada | Spoticry

Integrantes:

Fabiana Arias Rosales – Carné: 2021028380

Casey Baeza Castrillo – Carné: 2022437750

Profesor:

Oscar Víquez Acuña

Semestre II

2023

Índice

Introducción	2
Descripción del problema	3
Análisis de la solución	4
Resultados	6
Conclusiones	8
Referencias	9

Introducción

A través de la historia, la música siempre ha estado presente en la vida del ser humano. Esta ha sido participe en momentos significativos y ayuda al bienestar en general de las personas [1]. Con la digitalización y nuevas tecnologías ha cambiado la forma en que se escucha y gestiona la música. De manera que, está al alcance la capacidad de acceder y disfrutar de la música favorita de manera sencilla y organizada. En este contexto, surge el Sistema de Gestión de Música SpotiCry, una solución de software diseñada para llevar la gestión de la música local a una experiencia accesible y placentera.

SpotiCry Server es un servidor de música que cuenta con un módulo que se encarga de la atención sincrónica de solicitudes por parte de clientes y otro módulo que implementa un cliente, el cual puede instanciarse en múltiples ocasiones.
Esta aplicación se utiliza para almacenar información sobre canciones, como título, artista, género y ubicación de archivos. Permite a los usuarios buscar canciones en función de varios criterios como el nombre, el nombre del artista o incluso el género, duración y similitud de ritmo. Asimismo, se puede reproducir música, adelantando y retrocediendo al gusto del usuario, y administrar su colección de canciones por medio de Playlist, incluyendo la adición y eliminación de canciones de estas y reproducir la lista de reproducción.

Por último, este documento se sumergirá en los detalles técnicos y funcionales de SpotiCry, proporcionando una visión completa de este sistema de gestión de música. En la sección del Marco teórico se abordarán temas como el manejo de archivos de música, la administración en memoria de la lista de canciones del lado del servidor, manejo de conflictos y excepciones, implementación de sockets y procesos en paralelo, etc. En la sección de Descripción de la solución se incluye información sobre la implementación. En la sección de Análisis de resultados se realizan algunas pruebas de la implementación y en la sección de Conclusiones se detallan las conclusiones a partir de los resultados y agregando, qué elementos se deben tomar en cuenta para mejorar en caso de una implementación robusta de un sistema similar en producción a gran escala.

Descripción del problema

El problema que se aborda en este proyecto es el desarrollo de una aplicación de gestión de música por medio de sockets implementando una arquitectura simple de clientes y servidor a través del TCP-IP, incluyendo la creación y gestión de conexiones, la recepción y envío de datos. El servidor debe atender una serie de solicitudes de clientes que no serán otros que instancias del módulo/aplicación que reproduzca las canciones y realice la interacción en general con el cliente, esta tiene que ser implementada en otro lenguaje de programación siguiendo prácticas y estrategias de programación funcional.

Existen distintos aspectos por tomar en cuenta como el manejo de archivos de música desde archivos locales, la carga y gestión de estos. Además, el manejo de la lista de canciones en memoria para poder responder a las solicitudes de los clientes de manera eficiente, y de manera que se garantice coherencia de datos. También, el manejo de conflictos y excepciones para garantizar la estabilidad y la confiabilidad del sistema.

Análisis de la solución

El servidor se implementará en el lenguaje Go y atenderá solicitudes de múltiples clientes. También es conocido como Golang, es un lenguaje de programación desarrollado por Google. Se caracteriza por su simplicidad y eficiencia, y se utiliza para desarrollar una amplia variedad de aplicaciones, desde programas de línea de comandos hasta sistemas distribuidos y aplicaciones web de alta escala [2]. Además, el cliente se desarrollará en F# y permitirá la administración de listas de canciones y su reproducción. F# es un lenguaje de programación universal para escribir código conciso, sólido y eficaz. F# le permite escribir código ordenado y autoexplicativo cuyo foco permanece centrado en el dominio del problema, en lugar de en los detalles de programación. La velocidad y la compatibilidad no se ven afectadas, ya que es de código abierto, multiplataforma e interoperable [3].

En el aspecto de manejo de archivos de música, el programa servidor utiliza la biblioteca github.com/hajimehoshi/go-mp3 para obtener información sobre las canciones, como su duración en segundos. Este inicialmente decodifica el io.Reader dado y devuelve una secuencia decodificada. El flujo de datos siempre se formatea como 16 bits (little endian) y 2 canales, incluso si la fuente es un archivo MP3 de un solo canal. Por lo tanto, una muestra siempre consiste en 4 bytes. La función Length devuelve el tamaño total en bytes y la función SampleRate devuelve la tasa de muestreo [4]. También utiliza la biblioteca Os para trabajar con el sistema de archivos y encontrar la ubicación de las canciones.

Además, para la administración en memoria de la lista de canciones del lado del servidor, se almacena la información de las canciones en una lista de estructuras song. Cada estructura “song” cuenta con el nombre de la canción, artista, género, ruta del archivo y duración en segundos. Luego, se define un nuevo tipo llamado "songList," que es un "slice" de estructuras "song." Es decir, "songList" es una colección que puede contener múltiples canciones, y cada elemento de esta lista es una estructura "song." Por último, se declara una variable llamada "songCollection" que es de tipo "songList." Esta variable se utiliza para almacenar y gestionar una colección de canciones en el programa.

Hablando sobre detalles sobre manejo de conflictos y excepciones durante la ejecución de los módulos, el servidor maneja errores al intentar abrir y decodificar archivos de música.

También se verifica si una canción ya existe antes de agregarla. El cliente maneja excepciones al conectarse al servidor y al interactuar con él. Además, verifica la existencia de canciones antes de agregarlas a una lista de reproducción.

En el aspecto de la implementación de sockets y procesos en paralelo. El servidor utiliza goroutines para manejar múltiples conexiones de clientes de forma concurrente. Una goroutine es un hilo de implementación simple en el Golang que se ejecuta simultáneamente con el resto del programa. La goroutina principal controla todas las demás goroutinas; por lo tanto, si la goroutina principal termina, todas las demás goroutinas del script también lo hacen [5]. Cada conexión se maneja en su propia goroutine, lo que permite que múltiples clientes se conecten y realicen operaciones de manera simultánea. El cliente utiliza múltiples hilos para manejar las operaciones de entrada y salida, lo que permite al usuario interactuar con el servidor mientras se reproduce la música en segundo plano [6].

De igual manera, el programa cliente y servidor no implementan ninguna sincronización explícita. Esto podría llevar a problemas en escenarios de concurrencia, como la manipulación de listas de reproducción por múltiples clientes simultáneamente. En caso de un sistema a gran escala, se debe considerar la implementación de mecanismos de sincronización para garantizar la coherencia de los datos compartidos, como la lista de reproducción. Para esto, se puede utilizar los canales ofrecen una forma sencilla pero efectiva de establecer comunicación y coordinación entre goroutines. Al utilizar canales, los programadores pueden prevenir errores típicos asociados con el uso de memoria compartida y disminuir la posibilidad de enfrentar situaciones de conflicto de datos y otros desafíos relacionados con la programación concurrente [7].

Resultados

El servidor se implementa en Go y utiliza sockets para la comunicación con los clientes a través del protocolo TCP-IP. Este puede gestionar solicitudes concurrentes de múltiples clientes de manera eficiente. Además, puede administrar una lista de canciones en memoria, permitiendo la adición y eliminación de canciones. Proporciona una interfaz de texto para agregar canciones desde archivos locales en la máquina del servidor y permite la búsqueda de canciones según criterios definidos.

El cliente se desarrolla en F# y se ejecuta localmente. Este ofrece una interfaz gráfica de usuario para la administración de listas de canciones y la reproducción de música. Asimismo, permite buscar canciones utilizando tres criterios diferentes como es criterio (nombre, género y artista), similitud y duración. También, puede crear listas de reproducción y agregar canciones buscadas desde el servidor a estas listas. Además, proporciona controles de reproducción, incluyendo avance y retroceso en una canción, y de una canción.

Para tanto el servidor y el cliente se realizaron pruebas exhaustivas para evaluar la funcionalidad de la aplicación Spoticry. Se incluyeron las siguientes pruebas:

- Búsqueda de canciones: Se evaluó la capacidad de la aplicación para buscar canciones en función de los criterios especificados por los usuarios. Se pueden incluir pruebas con diferentes criterios de búsqueda, como nombre de la canción, artista, género, similitud y duración de la canción. Se verificó que la búsqueda generara resultados precisos y que las canciones se devuelvan correctamente de acuerdo con los criterios de búsqueda.
- Reproducción de música: Esta prueba se centró en la funcionalidad de reproducción de música tanto en el servidor como en el cliente. Se comprobó que el servidor puede enviar correctamente los datos de las canciones al cliente para su reproducción. Además, se comprobó funcionalidades como adelantar, retroceder y pausar la canción que se está reproduciendo.
- Gestión de listas de reproducción: Se verificó si el cliente puede administrar sus listas de reproducción de manera efectiva. Esto incluye la creación de nuevas listas de reproducción, la adición y eliminación de canciones de las listas existentes y la

reproducción de canciones desde las listas. Además, se comprobó que las listas de reproducción se actualicen correctamente en caso de eliminar una canción en el servidor, y así garantizar la coherencia de los datos entre el cliente y el servidor.

- Concurrencia: Se evaluó la capacidad del servidor para manejar múltiples solicitudes de clientes de manera concurrente sin errores ni conflictos. Para esto, se simulan múltiples clientes que realizan solicitudes simultáneas al servidor. De manera que, se evaluó la respuesta del servidor ante solicitudes concurrentes, asegurándose de que no se produjeran bloqueos, caídas del servidor o conflictos de acceso a datos compartidos.

Conclusiones

- Se concluye que para un sistema una implementación robusta de un sistema similar en producción a gran escala es necesario que sea capaz de escalar horizontalmente para manejar un mayor número de solicitudes de clientes.
- También se puede mejorar el sistema de almacenamiento de canciones a una manera más eficiente como como una base de datos distribuida, que permita una búsqueda rápida y la recuperación de datos.
- Se pueden mejorar aspectos como la seguridad, más si es en miras a una implementación robusta, siendo fundamental implementar medidas de seguridad, como autenticación y autorización, para proteger los datos y prevenir el acceso no autorizado.
- Por último, el sistema actual se ha diseñado de manera modular con un servidor implementado en Go y un cliente en F#, esto permite flexibilidad y mantenibilidad. Además, este sistema garantiza que se pueda reproducir canciones almacenadas en la máquina del servidor, y se ha diseñado con cierta flexibilidad para permitir futuras mejoras y expansiones.

Referencias

[1] UDEP (2020) *Influencia de la Música en los seres humanos " Udep hoy*, *UDEP Hoy*. Available at: https://www.udep.edu.pe/hoy/2020/08/influencia-de-la-musica-en-los-seres-humanos/ (Accessed: 17 September 2023).

[2] MyTaskPanel (2023a) *Lenguaje de Programación go: Utilidades, Características Y Ventajas*, *MyTaskPanel Consulting*. Available at: https://www.mytaskpanel.com/lenguaje-programacion-go/ (Accessed: 18 September 2023).

[3] Cartermp, M. (no date) *¿Qué es F#? - .NET*, *.NET | Microsoft Learn*. Available at: https://learn.microsoft.com/es-es/dotnet/fsharp/what-is-fsharp (Accessed: 18 September 2023).

[4] Hajime, H. 2022. *go-mp3,* ver v0.3.4. Available at: https://pkg.go.dev/github.com/hajimehoshi/go-mp3#Decoder

[5] Osman, J. (2022) *Goroutines*, *AppMaster*. Available at: https://appmaster.io/es/blog/goroutines-es (Accessed: 19 September 2023).

[6] GO (no date) *Goroutines*, *A Tour of Go*. Available at: https://go.dev/tour/concurrency/1 (Accessed: 19 September 2023).

[7] AppMaster (2023) *Concurrencia en go*, *AppMaster*. Available at: https://appmaster.io/es/blog/concurrencia-ir (Accessed: 20 September 2023).