Instituto Tecnológico de Costa Rica
Unidad de Computación

Proyecto de Aplicación de Música Cliente-Servidor

Joshua Esteban Sancho Burgos / 2021108350
Alejandro Jiménez Ulloa / 2021084774

Sede San Carlos
23/09/2023

Contenido

Introducción

Esta documentación se enfoca en un proyecto con el objetivo principal de aplicar los conocimientos de los paradigmas imperativo y funcional para crear una aplicación que realice tareas de reproducción, CRUD (Crear, Leer, Actualizar, Borrar) y búsqueda de música. El proyecto sigue una arquitectura cliente-servidor basada en TCP/IP.

El propósito fundamental de este proyecto es aprovechar los paradigmas de programación imperativo y funcional para diseñar e implementar una aplicación de reproducción de música que ofrezca un conjunto completo de funcionalidades, incluyendo la gestión de canciones y listas de reproducción, así como la búsqueda y reproducción de canciones. Para lograr este propósito, el proyecto se divide en dos componentes esenciales: un servidor desarrollado en el lenguaje de programación Go y un cliente implementado en F#. Estos componentes trabajan en conjunto de manera eficiente y sincronizada para llevar a cabo las operaciones mencionadas. A lo largo de esta documentación, se explorarán en detalle los aspectos técnicos y funcionales de ambos componentes, destacando las decisiones de diseño y las estrategias de programación empleadas en este proyecto.

Objetivos del Proyecto

Objetivo General:

Diseñar, desarrollar y desplegar una aplicación de reproducción de música basada en una arquitectura cliente-servidor TCP/IP, utilizando los paradigmas de programación imperativo y funcional, que permita a los usuarios gestionar canciones y listas de reproducción, realizar búsquedas avanzadas y reproducir canciones de forma eficiente.

Objetivos Específicos:

a) ***Desarrollo del Servidor:*** Implementar un servidor en el lenguaje Go que sea capaz de recibir, procesar y responder a las solicitudes de los clientes a través del protocolo TCP/IP. Este servidor debe gestionar eficazmente la lista de canciones, proporcionando funcionalidades de CRUD (Crear, Leer, Actualizar, Borrar) para canciones y listas de reproducción.

b) ***Desarrollo del Cliente:*** Construir un cliente en F# con una interfaz gráfica de usuario (GUI) intuitiva que permita a los usuarios interactuar de manera efectiva con la aplicación. El cliente debe ofrecer funciones de búsqueda avanzada que permitan a los usuarios buscar canciones según múltiples criterios personalizables.

c) Integración Sincronizada: Garantizar que el servidor y el cliente trabajen de manera sincronizada y concurrente, permitiendo a los usuarios gestionar sus listas de reproducción, realizar búsquedas y disfrutar de la música sin interrupciones. La reproducción de canciones debe ser eficiente, con una gestión adecuada de la decodificación y envío de paquetes de audio.

Componentes Principales

Servidor (Implementado en Go):

- El servidor es responsable de atender las solicitudes de los clientes a través del protocolo TCP-IP.
- Administra una lista de canciones, proporcionando funciones para agregar y eliminar canciones, así como para buscar canciones según diversos criterios.
- Permite la reproducción local de canciones, lo que implica la decodificación y el envío de paquetes de audio a los clientes.

Cliente (Implementado en F#):

- El cliente proporciona una interfaz gráfica de usuario (GUI) intuitiva para que los usuarios administren sus listas de reproducción y puedan reproducir su música.
- Ofrece funciones de búsqueda avanzada que permiten a los usuarios buscar canciones según múltiples criterios personalizables (álbum, año y duración).

Análisis del Problema

Al desarrollar una aplicación de reproducción de música con una arquitectura cliente-servidor TCP/IP, pueden existir varios desafíos técnicos y funcionales. Estos desafíos pueden incluir:

a) *Gestión de Canciones:*

- El servidor debe ser capaz de agregar canciones desde archivos de audio (por ejemplo, MP3) a una lista de reproducción.
- Debe ser posible eliminar canciones de la lista de reproducción.
- Los metadatos de cada canción, como el nombre y la ubicación del archivo, deben ser almacenados y gestionados de forma eficiente.

b) *Búsqueda de Canciones:*

- El servidor debe proporcionar al menos tres criterios diferentes para buscar canciones, como duración, álbum, año, etc.
- Los resultados de búsqueda deben ser precisos y presentados de manera clara.

c) *Reproducción de Música:*

- El servidor debe ser capaz de reproducir canciones de forma local, lo que implica la decodificación y el envío de paquetes de audio al cliente.
- Debe ser posible iniciar y detener la reproducción de canciones en cualquier momento.

d) *Interfaz de Texto del Servidor:*

- El servidor debe tener su propia interfaz de texto para que los administradores puedan agregar canciones desde archivos locales y eliminar canciones de la lista de reproducción superior.

e) *Interfaz Gráfica del Cliente:*

- El cliente debe tener una interfaz gráfica de usuario (GUI) que permita a los usuarios administrar listas de reproducción y reproducir canciones.
- La interfaz debe ser intuitiva y fácil de usar.

f) *Búsqueda Avanzada en el Cliente:*

- El cliente debe permitir a los usuarios buscar canciones utilizando al menos tres criterios diferentes, que deben ser suficientemente diversos para requerir diferentes enfoques técnicos en el servidor.
- Los resultados de búsqueda deben ser mostrados de forma clara en la GUI del cliente.

g) *Reproducción de Música en el Cliente:*

- El cliente debe ser capaz de reproducir canciones en distintas playlists.

Solución del Problema

Arquitectura del Manejo de Archivos MP3 y del Servidor

En esta sección, se proporcionará una descripción detallada de la arquitectura y los componentes clave relacionadas con el manejo de archivos MP3 y el funcionamiento del servidor. Se explicarán las librerías utilizadas y las funciones desarrolladas específicamente para estas tareas.

Componentes del Servidor

El servidor de música se compone de varios componentes clave que trabajan en conjunto para permitir la gestión de canciones y listas de reproducción. Estos componentes incluyen:

a) *Servidor TCP/IP:* El servidor se implementa como una aplicación de servidor TCP/IP que escucha en un puerto específico en la dirección "localhost". Utiliza la librería estándar net de Go para crear y gestionar conexiones con los clientes.

b) *Recepción de Solicitudes:* El servidor acepta solicitudes entrantes de clientes que se conectan a través del protocolo TCP/IP. Utiliza goroutines para manejar múltiples conexiones de clientes de manera concurrente, lo que permite un acceso eficiente y simultáneo a las funcionalidades del servidor.

c) *Gestión de Listas de Reproducción:* El servidor administra las listas de reproducción de los clientes, lo que incluye agregar, eliminar y buscar listas de reproducción. Cada cliente tiene su propio conjunto de listas de reproducción que se almacenan en memoria durante la ejecución del servidor.

d) *Gestión de Canciones:* El servidor maneja la administración de canciones, incluida la búsqueda de canciones en una lista de reproducción, la adición y eliminación de canciones, y la obtención de datos de metadatos de las canciones.

e) *Comunicación con el Cliente:* La comunicación entre el servidor y el cliente se basa en un protocolo simple de mensajes de texto. El servidor recibe solicitudes del cliente en forma de cadenas de texto, las procesa y responde según corresponda.

Librerías y Módulos Utilizados

a) *Librería net:* La librería estándar net de Go se utiliza para la implementación de la comunicación a través de sockets TCP/IP. Proporciona las funciones necesarias para crear un servidor que escuche conexiones entrantes y establezca conexiones con los clientes.

b) *Librerías Externas:* Para el manejo de archivos MP3 y la extracción de metadatos de canciones, se utilizan las siguientes librerías externas:

- *github.com/dhowden/tag:* Esta librería se utiliza para leer los metadatos de los archivos MP3, como el título, artista, álbum, año, etc.
- *github.com/hajimehoshi/go-mp3:* Esta librería se utiliza para decodificar archivos MP3 y calcular su duración en segundos.

Funciones Clave

a) *NewServer:* La función NewServer se encarga de crear una instancia del servidor. Toma como parámetro la dirección en la que escuchará el servidor.
b) *Start:* La función Start inicia el servidor y comienza a escuchar conexiones entrantes en la dirección especificada. Cada conexión entrante se maneja en una goroutine separada.
c) *readLoop:* La función readLoop se ejecuta en una goroutine para cada conexión de cliente. Esta función lee los mensajes entrantes del cliente, los procesa y ejecuta las acciones correspondientes en el servidor.

Arquitectura del Manejo de Canciones y Funciones para Listas de Reproducción

Componentes del Manejo de Canciones

El manejo de canciones es una parte central de la aplicación de servidor de música y consta de los siguientes componentes:

a) *Estructura de Datos de Canción (Song):* La estructura de datos Song representa una canción e incluye información como el título, artista, álbum, género, año y duración. Estos datos se obtienen de los metadatos de los archivos MP3 utilizando librerías externas.
b) *Estructura de Datos de Lista de Reproducción (Playlist):* La estructura Playlist es una representación de una lista de reproducción y contiene un nombre y una lista de canciones (Song) asociadas. Los clientes pueden crear, eliminar y administrar listas de reproducción.
c) *Funciones de Búsqueda de Canciones:* Las funciones permiten buscar canciones en una lista de reproducción según varios criterios, como el título, el artista, el álbum, el año y la duración.
d) *Funciones de Adición y Eliminación de Canciones:* Estas funciones permiten a los clientes agregar y eliminar canciones de una lista de reproducción específica. Las canciones deben estar disponibles en una lista de reproducción general antes de poder agregarse a listas de reproducción personalizadas.

Librerías y Módulos Utilizados

a) *Librería github.com/dhowden/tag:* Esta librería se utiliza para extraer metadatos de archivos MP3, como título, artista, álbum, año, género, etc. Permite llenar la estructura de datos Song con información detallada de la canción.

Funciones Clave

a) *SearchSong:* La función SearchSong permite buscar una canción en una lista de reproducción específica por su título. Devuelve la canción encontrada, su posición en la lista y un posible error si no se encuentra la canción.

b) *AddSong:* La función AddSong permite a los clientes agregar una canción a una lista de reproducción específica. Verifica si la canción existe en la lista general "SUPERPLAYLIST" antes de agregarla a la lista personalizada del cliente. Devuelve la lista de reproducción modificada con la nueva canción.

c) *DeleteSong:* La función DeleteSong permite a los clientes eliminar una canción de una lista de reproducción específica. Devuelve la lista de reproducción modificada sin la canción eliminada.

d) *FilterByYear:* La función FilterByYear filtra las canciones de una lista de reproducción por año y devuelve una lista de canciones que cumplen con el criterio de filtrado.

e) *FilterByDuration:* La función FilterByDuration filtra las canciones de una lista de reproducción por duración máxima en minutos y devuelve una lista de canciones que cumplen con el criterio de filtrado.

Interfaz Gráfica de Usuario:

Para la implementación de la interfaz gráfica de usuario (GUI) se decidió utilizar el framework Avalonia UI, esto debido a que es compatible con F#, lo que facilita su implementación al no tener que utilizar otro lenguaje como C# para programarla. Algunas ventajas de Avalonia son las siguientes:

a) *Multiplataforma:* Está diseñado para ser verdaderamente multiplataforma, lo que significa que puede utilizarse para desarrollar aplicaciones de escritorio que se ejecutan en Windows, macOS y Linux sin necesidad de realizar cambios importantes en el código fuente. Esto facilita la creación de aplicaciones que funcionen en una variedad de sistemas operativos sin tener que mantener múltiples bases de código.

b) *Interfaz de Usuario XAML:* Al igual que WPF en el mundo de Microsoft, Avalonia utiliza un lenguaje de marcado llamado XAML (eXtensible Application Markup Language) para definir la interfaz de usuario de las aplicaciones. Los desarrolladores pueden crear la

interfaz de usuario de sus aplicaciones de manera declarativa, lo que facilita la separación de la lógica de la interfaz de usuario de la lógica de la aplicación.

c) *Estilo y Temas Personalizables:* Avalonia UI permite personalizar completamente la apariencia de las aplicaciones a través de estilos y temas. Los desarrolladores pueden definir estilos para controles y componentes específicos, lo que les brinda un alto grado de control sobre la apariencia visual de la aplicación.

d) *Diseño Responsivo:* Avalonia admite un diseño responsivo, lo que significa que las aplicaciones pueden adaptarse automáticamente a diferentes tamaños de pantalla y resoluciones. Esto es especialmente importante en el contexto de aplicaciones que se ejecutan en una variedad de dispositivos y plataformas.

e) *Eventos y Enlaces de Datos:* Al igual que otros frameworks de GUI, Avalonia permite la gestión de eventos y la vinculación de datos, lo que facilita la interacción entre la interfaz de usuario y la lógica de la aplicación.

f) *Comunidad Activa:* Es un proyecto de código abierto con una comunidad activa de desarrolladores y colaboradores. Esto significa que está en constante evolución y mejora, con nuevas características y correcciones de errores que se realizan regularmente.

Manejo de conflictos y excepciones:

a) *Manejo de Errores de Lectura de Archivos MP3:* En la función ProcessMP3File, se abre un archivo MP3 y se intenta leer sus metadatos. Si hay algún error en la apertura del archivo o en la lectura de los metadatos, se captura el error y se devuelve como parte de la estructura de datos Song. Esto asegura que, en caso de problemas al procesar un archivo MP3, se registre un error y se continúe con la siguiente canción.

b) *Manejo de Errores en la Creación del Servidor:* En la función NewServer, si ocurre algún error al intentar abrir el puerto para escuchar conexiones, se devuelve un error. Esto es importante para manejar situaciones en las que el servidor no puede iniciarse correctamente debido a un problema en el puerto o en la configuración de red.

c) *Manejo de Errores en la Comunicación Cliente-Servidor:* En la función readLoop, se manejan errores de lectura cuando se procesan los mensajes enviados por los clientes. Si ocurre algún error en la lectura de datos desde el cliente, se registra el error y se termina la función, lo que permite manejar de manera adecuada problemas de comunicación durante la ejecución.

d) *Manejo de Errores en la Interacción con Listas de Reproducción:* En varias funciones que interactúan con listas de reproducción, como AddSong, DeleteSong, FilterByYear, etc., se manejan errores y se registran mensajes de error en caso de que no se encuentre una lista de reproducción o no se pueda realizar una operación específica.

e) *Manejo de Errores en la Lectura de Metadatos de Canciones:* En la función GetMP3Data, cuando se procesa un directorio en busca de archivos MP3, se manejan errores si la lectura de metadatos de una canción falla. Los errores se registran y se continúa con el procesamiento de otras canciones en lugar de detener todo el proceso.

f) *Manejo de Errores en la Creación de Conexiones TCP:* En la función acceptLoop, se manejan errores al aceptar conexiones entrantes de clientes. Si no se puede aceptar una conexión, se registra el error y se continúa esperando conexiones adicionales.

Implementación de Sockets y Procesos en Paralelo

¿Qué es la sincronización y que cambios existen al implementarla o no en el programa?

La sincronización se refiere a cómo el servidor y el cliente interactúan y gestionan las acciones de los usuarios en la aplicación de reproducción de música.

- *Funcionamiento sin sincronización:* En el programa sin sincronización, tanto el servidor como el cliente funcionan de manera independiente y asincrónica. Los usuarios pueden realizar acciones en el cliente, como buscar canciones, agregarlas a listas de reproducción o reproducirlas, sin tener en cuenta el estado actual del servidor o de otros clientes. Esto significa que las acciones de un usuario en el cliente no afectan inmediatamente a otros usuarios o al servidor. Por ejemplo, si un usuario inicia la reproducción de una canción, el cliente comienza a reproducirla sin esperar una confirmación o respuesta del servidor. Si otro usuario realiza una acción relacionada con la misma canción, como eliminarla de una lista de reproducción, es posible que se produzcan conflictos o problemas de sincronización, ya que el servidor y el cliente no están coordinados.
- *Funcionamiento con sincronización:* En el programa con sincronización, se implementan mecanismos para coordinar y sincronizar las acciones de los usuarios en el cliente y el servidor. Esto implica que el servidor y el cliente se mantienen informados sobre el estado actual de la reproducción, las listas de reproducción y las canciones. Por ejemplo, cuando un usuario inicia la reproducción de una canción en el cliente, el cliente envía una solicitud al servidor para que comience a reproducir la misma canción. El servidor confirma la solicitud y comienza a reproducir la canción, lo que garantiza que la reproducción se

mantenga sincronizada entre el servidor y el cliente. Si otro usuario realiza una acción relacionada con la misma canción, como eliminarla de una lista de reproducción, el servidor puede notificar al cliente de esta acción y tomar las medidas adecuadas para mantener la coherencia de los datos.

¿Qué cambios deberían implementarse en un sistema de producción a gran escala?

Para lograr una implementación robusta de un sistema similar en producción a gran escala, se deben considerar varios aspectos clave en la arquitectura y el diseño del software:

1. *Escalabilidad Horizontal:* El sistema actualmente funciona con múltiples clientes, pero para una implementación a gran escala, se debe garantizar la capacidad de escalar horizontalmente. Esto implica diseñar el sistema de manera que pueda manejar un gran número de conexiones concurrentes y distribuir la carga de manera efectiva en servidores múltiples.
2. *Balanceo de Carga:* Implementar un mecanismo de balanceo de carga que distribuya las solicitudes de los clientes de manera equitativa entre los servidores. Esto asegura una distribución uniforme de la carga y evita la sobrecarga de un servidor específico.
3. *Gestión de Sesiones:* Para un sistema de música en tiempo real, es fundamental implementar una gestión de sesiones robusta para llevar un registro de los clientes conectados y sus estados. Esto permite realizar operaciones de forma coherente, como la pausa o el avance de una canción, en todos los clientes conectados.
4. *Almacenamiento de Datos:* Para manejar grandes cantidades de canciones y metadatos, se debe considerar el uso de una base de datos escalable y de alto rendimiento. Esto permitirá una gestión eficiente de la información de las canciones y las listas de reproducción.
5. *Seguridad:* Implementar medidas de seguridad sólidas es esencial, especialmente en un entorno de producción a gran escala. Esto incluye la autenticación de usuarios, la autorización de acceso a recursos y la protección contra ataques comunes, como ataques DDoS.
6. *Monitorización y Registro:* Incorporar herramientas de monitorización y registro para supervisar el rendimiento del sistema, identificar cuellos de botella y solucionar problemas de manera proactiva. Los registros detallados son esenciales para la depuración y la seguridad.

7. *Recuperación ante Fallos:* Implementar mecanismos de recuperación ante fallos que permitan al sistema recuperarse de manera automática en caso de errores o caídas. Esto incluye la replicación de datos y la capacidad de conmutación por error.

8. *Optimización de Redes:* A medida que el tráfico aumenta, es importante optimizar la comunicación entre el servidor y los clientes. Esto puede incluir la compresión de datos, la minimización del ancho de banda y el uso eficiente de los recursos de red.

9. *Actualizaciones y Mantenimiento:* Planificar un proceso sólido de actualización y mantenimiento del sistema. Esto puede incluir la implementación de actualizaciones sin tiempo de inactividad y la gestión de versiones del software.

10. *Cumplimiento Normativo:* Asegurarse de que el sistema cumple con las regulaciones y normativas aplicables, especialmente en lo que respecta a la gestión de derechos de autor y datos personales.

11. *Pruebas de Estrés y Rendimiento:* Realizar pruebas exhaustivas de estrés y rendimiento para evaluar cómo se comporta el sistema bajo cargas extremas y garantizar que pueda manejar la demanda esperada.

Conclusiones y Recomendaciones

Matriz de Requerimientos

Requerimiento	Servidor	Cliente	GUI
Conexión cliente-servidor	100%	100%	N/A
Envío de paquetes de bytes	100%	N/A	N/A
Manejo de solicitudes	100%	N/A	N/A
Lectura de metadatos de mp3	100%	N/A	N/A
CRUD de playlists	100%	##	90%
CRUD de canciones	100%	##	90%
Reproducción de música en cliente	N/A	100%	##
Filtrado de canciones por álbum	100%	##	##

Filtrado de canciones por año	100%	##	##
Filtrado de canciones por duración	100%	##	##

Conclusiones:

a) *Manejo de Archivos:* La implementación del proyecto depende en gran medida del manejo adecuado de archivos MP3, incluyendo la extracción de metadatos y la decodificación de audio. El uso de librerías especializadas para estas tareas facilitó mucho este proceso.

b) *Envío de Paquetes de Bytes por el Servidor:* La comunicación eficiente entre el servidor y el cliente mediante la transmisión de paquetes de bytes a través del protocolo TCP/IP es esencial para garantizar la reproducción de música y permite la conexión entre los datos del cliente y servidor.

c) *Arquitectura Cliente-Servidor:* *La arquitectura* cliente-servidor basada en TCP/IP ofrece una estructura sólida para la interacción de múltiples clientes con un servidor centralizado. Esto es fundamental para la gestión y reproducción efectiva de la música.

d) *Dificultad de Programar en Dos Lenguajes con Paradigmas Distintos:* La programación en dos lenguajes con paradigmas diferentes, Go y F#, presenta un desafío, pero se puede superar mediante la coordinación y sincronización adecuada entre los componentes. La sincronización es esencial para evitar conflictos y garantizar un funcionamiento coherente.

Recomendaciones:

a) *Optimización de la Comunicación:* Se recomienda implementar técnicas de optimización de la comunicación, como la compresión de datos y la minimización del ancho de banda, para mejorar el rendimiento al enviar paquetes de audio.

b) *Documentación y Pruebas Exhaustivas:* La documentación detallada del sistema, protocolos y formatos de mensajes es fundamental. Además, realizar pruebas exhaustivas, incluyendo pruebas de estrés y rendimiento, es esencial para garantizar la escalabilidad y la fiabilidad del sistema.

c) *Interoperabilidad y Gestión de Errores:* Es recomendable asegurarse de que los componentes desarrollados en Go y F# sean interoperables y que exista una gestión de errores sólida en ambos lenguajes para mantener la integridad del sistema.

Investigación de Librerías Externas y Frameworks: Se recomienda realizar una investigación exhaustiva de las librerías externas y frameworks utilizados en el proyecto, como github.com/dhowden/tag, github.com/hajimehoshi/go-mp3 y Avalonia. Mantener estas librerías y frameworks actualizados y verificar su compatibilidad con las últimas versiones de los lenguajes Go y F# es esencial para evitar problemas de seguridad y garantizar un funcionamiento estable.

d) *Sincronización y Coordinación:* Dado que el proyecto involucra la programación en dos lenguajes con paradigmas distintos (Go y F#), se recomienda establecer una estrategia sólida de sincronización y coordinación entre los componentes del cliente y el servidor. Esto puede incluir la implementación de mecanismos de bloqueo, uso de canales de comunicación y una estrategia coherente para la gestión de eventos y acciones en tiempo real.