Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Lenguajes de Programación

SpotiCry

Profesor:

Oscar Mario Víquez Acuña

Estudiantes:

Kevin Vinicio Varela Rojas

Anthony Andrés Jiménez Zamora

Sede San Carlos

23/09/2023

Introducción

En el presente proyecto se implementa un sistema bajo el lenguaje de F# y Go, diseñado en la aplicación JetBrains Rider. Este ayudará a un usuario para que pueda reproducir música según las opciones que se le bridan, asimismo, podrá hacer una playlist y personalizarla a su gusto con las canciones que este elija. Además, el usuario puede buscar alguna de las canciones bajo ciertos parámetros que este indique.

La aplicación para el cliente funciona con una interfaz gráfica para una mejor interacción del usuario. Además, esta aplicación se conecta con un servidor, el cual es el que se encarga de brindar el servicio de música.

La finalidad del proyecto es que se le permita al usuario disfrutar de un rato agradable con música de su gusto.

Análisis del problema

Según se plantea el problema, se trata de resolver ciertos puntos que solucionarán el problema:

- **Servidor**

Se necesitará crear un servidor, el cual almacene canciones con cierta información sobre la mismas. También, se espera que se pueda obtener listas de canciones por al menos 3 criterios de búsqueda que se hayan definido. Además, que este pueda decodificar y enviar paquetes al cliente, para que se logre escuchar la canción seleccionada. Y por último, que se puedan añadir y eliminar canciones según el gusto del cliente.

- **Cliente**

Se necesita crear una aplicación que corra de forma local. Esta debe tener una interfaz gráfica para una mejor interacción con el cliente, así este puede administrar las listas de canciones y permitirle escuchar cada canción. Además, se le debe permitir al usuario que realice una búsqueda de alguna canción, con al menos 3 filtros. Asimismo, el cliente podrá crear múltiples listas de reproducción con las canciones que este elija. Finalmente, la aplicación del cliente debería permitirle al cliente poder adelantar y retroceder la misma canción las veces que desee.

Solución del problema

Durante la elaboración del proyecto no se hizo uso de ningún diagrama de flujo o pseudocódigo, se improvisaba sobre la marcha. Se utilizaron archivos de Go y F# de prueba para realizar ciertas funciones que se necesitaron y así no tener que correr todo el programa. También se utilizó información de internet para entender el funcionamiento de varias funciones o biblioteca que contiene F#.

- **Primera etapa**

En esta primera etapa del proyecto se centró en realizar en la programación y el funcionamiento del servidor. Este se trabajó en ciertas partes:

1. Creación del Servidor TCP:
    - El código comienza creando un servidor TCP en la dirección IP 127.0.0.1 (localhost) y el puerto 8000. Esto significa que el servidor estará escuchando en ese puerto para aceptar conexiones entrantes.
2. Manejo de Conexiones de Clientes:
    - El bucle principal del servidor está diseñado para aceptar conexiones de clientes de manera continua. Cada vez que un cliente se conecta, se crea una nueva gorutina (go handleClient(conn)) para manejar esa conexión específica.
3. Funciones del Servidor:
    - Envío de Canciones (enviarCancion):
        - ✓ Si un cliente envía un comando que comienza con *, el servidor intenta enviar la canción correspondiente al cliente.
        - ✓ La canción se busca en la carpeta "listaCanciones/" y se envía al cliente en bloques de 1024 bytes.
        - ✓ Si la canción no se encuentra, el servidor responde con "NO EXISTE".
    - Lista de Canciones (enviarListaCompleta):
        - ✓ Cuando un cliente envía el comando "lista", el servidor recopila la lista de canciones disponibles leyendo desde un archivo llamado "cancionesRegistradas.txt".
        - ✓ Luego, envía esta lista al cliente para que pueda ver qué canciones están disponibles.

- Consulta por Nombre (consultaNombre):
    - ✓ Si el cliente envía un comando que comienza con ?, el servidor busca canciones en "cancionesRegistradas.txt" que coincidan con el nombre proporcionado por el cliente.
    - ✓ Luego, envía los resultados de la búsqueda al cliente para que pueda ver las canciones que coinciden con el nombre.
- Consulta de Likes (consultaLikes):
    - ✓ Cuando un cliente envía un comando que comienza con +, el servidor realiza una consulta para encontrar las canciones con más likes en la lista de canciones registradas.
    - ✓ Luego, envía al cliente las canciones más populares en función de la cantidad de likes.
    - ✓ Utiliza un algoritmo que encuentra las tres canciones con más likes.
- Consulta de Año Antes de (consultaAnnioAntesDe):
    - ✓ Aunque esta función está definida, parece estar incompleta en el código proporcionado. Su objetivo sería buscar canciones registradas antes de un año específico, pero la lógica para hacerlo aún no se ha implementado.

4. Manipulación de Archivos:
    - El servidor maneja la lectura de archivos de canciones almacenados en la carpeta "listaCanciones/". Luego, envía los datos de la canción al cliente en bloques de 1024 bytes hasta que se haya enviado toda la canción.
5. Gestión de Errores:
    - El servidor maneja varios tipos de errores, como conexiones cerradas por el cliente o problemas al abrir archivos.
    - Cuando ocurre un error, el servidor responde al cliente con un mensaje apropiado, como "NO EXISTE" si la canción no se encuentra o "Respuesta no válida" si el comando no es reconocido.

- **Segunda etapa**

En la segunda etapa se crea la aplicación del cliente y su funcionalidad. La parte funcional de la aplicación se encarga de tareas como la reproducción de música y la comunicación con un servidor remoto para obtener información sobre canciones y descargarlas. Para lograrlo se dividió en ciertas partes:

1. Reproducción de Música (función musica):
    - Esta función se encarga de la reproducción y pausa de la música. Utiliza la biblioteca NAudio.Wave para cargar y reproducir archivos de música en formato MP3.
    - Si se está reproduciendo una canción y se hace clic en el botón "Play/Pause", la función pausa la canción y mantiene la posición de reproducción.
    - Si no se está reproduciendo una canción o se hace clic en el botón "Play/Pause" mientras está pausado, la función carga y reproduce la canción seleccionada.
2. Comunicación con el Servidor (funciones lista, top3 y descargar):
    - Estas funciones se utilizan para interactuar con un servidor remoto que proporciona información sobre canciones y permite la descarga de canciones.
    - lista: Envía una solicitud al servidor para obtener una lista de canciones disponibles. La respuesta del servidor se muestra en la lista de canciones.
    - top3: Solicita al servidor las tres canciones más populares (con más "likes"). La respuesta del servidor se muestra en la lista de canciones.
    - descargar: Solicita una canción específica al servidor y la descarga para su reproducción en la aplicación. La canción descargada se agrega a la lista de reproducción.

- **Tercera etapa**

La tercera etapa será para la creación la interfaz gráfica. La parte gráfica de la aplicación utiliza la biblioteca Avalonia.FuncUI para crear la interfaz de usuario. Aquí se describe cómo se resuelve la creación de la interfaz y cómo se relaciona con la funcionalidad de la aplicación. Se distribuyó su creación en ciertos puntos:

1. Interfaz Gráfica General:
    - La aplicación utiliza un patrón de diseño de interfaz de usuario funcional para definir su apariencia y comportamiento. Se utiliza un DockPanel como el elemento raíz de la interfaz de usuario, que permite organizar otros elementos secundarios en un diseño de panel.
2. Elementos de la Interfaz de Usuario:
    - Playlist (TextBlock): Muestra la lista de canciones en reproducción actual. Se actualiza automáticamente cuando se agregan canciones a la lista de reproducción.
    - Canción Actual (ListBox): Muestra la canción seleccionada actualmente. Permite al usuario cambiar la canción seleccionada haciendo clic en la lista.
    - Botón Play/Pause (Button): Controla la reproducción y pausa de la música. Cuando se hace clic, inicia o pausa la reproducción de la canción seleccionada.
    - Botón Atrasar 15 seg (Button): Permite retroceder la reproducción en 15 segundos.
    - Slider de Reproducción (Slider): Permite al usuario ajustar la posición de reproducción de la canción. Puede moverse para adelantar o retroceder la canción.
    - Lista de Canciones (ListBox): Muestra las canciones disponibles. El usuario puede seleccionar una canción de esta lista.
    - Botón Descargar Canción (Button): Descarga la canción seleccionada desde un servidor remoto y la agrega a la lista de reproducción.
    - Botón Cargar Lista (Button): Solicita al servidor una lista actualizada de canciones disponibles y la muestra en la lista de canciones.
    - Botón Top 3 (Button): Solicita al servidor las tres canciones más populares (con más "likes") y las muestra en la lista de canciones.
3. Funciones de Control:

- Cada botón y control deslizante tiene funciones asociadas que se ejecutan cuando se hace clic en ellos. Por ejemplo, el botón "Play/Pause" inicia o pausa la reproducción de la música utilizando la función musica. El botón "Descargar Canción" descarga una canción utilizando la función descargar, y así sucesivamente.

Análisis de los resultados

Una vez hemos analizado el problema, buscado solución e implementado la solución, tenemos los resultados finales. Estos serán plasmados en el siguiente cuadro.

Tarea/Requerimiento	Estado	Observaciones
Servidor	Completo	Todas los requerimientos están
Cliente	Incompleto	El cliente no puede crear playlists, solamente una. Además, le falta la consulta por nombre y la consulta “año antes de”, también, no se puede atrasar ni adelantar la canción por medio de la barra de reproducción.

Conclusiones:

- Antes de hacer el software, se debe pensar y describir detalladamente lo que se quiere que se que haga. Esto evita problemas y cambios de último minuto que pueda retrasar el proyecto.
- Para que un proyecto sea exitoso, es importante terminar todas las partes más importantes. En SpotiCry, hay algunas partes que no están listas en el servidor y el cliente, por lo que es necesario priorizar y terminar todas las cosas principales para que los usuarios tengan una experiencia completa.
- Es muy importante probar el software de manera exhaustiva, especialmente cuando el usuario interactúa con un servidor. Si no se hacen suficientes pruebas, podrían ocurrir errores y los usuarios no estarán contentos.

Recomendaciones:

- Dado que el cliente aún no puede crear listas de reproducción, solamente una, ni buscar canciones según los tres criterios definidos en el proyecto, es fundamental priorizar y completar estas funcionalidades de manera que el sistema cumpla con los requisitos establecidos. Esto garantizará que la aplicación sea útil y satisfaga las expectativas de los usuarios.
- El hecho de que no existe un botón para refrescar al servidor provoca que no se actualice en caso de haber eliminado una canción. Por lo que sería bueno lograr su implementación y que sea una mejor experiencia para el cliente.

Bibliografía

gorkys. (05 de Diciembre de 2013). *GitHub*. Obtenido de Avalonia: https://github.com/AvaloniaUI/Avalonia

JaggerJo. (20 de Junio de 2019). *GitHub*. Obtenido de Avalonia.FuncUI: https://github.com/fsprojects/Avalonia.FuncUI

OpenAI. (s.f.). *ChatGPT*. Obtenido de ChatGPT [Modelo de lenguaje GPT-3]: https://chat.openai.com/?model=text-davinci-002-render-sha