PROYECTO I – COMMUNITY MUSIC PLAYER 1.0

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computadores
Algoritmos y Estructuras de Datos I (CE 1103)
I Semestre 2024
Valor 25%

ESTE DOCUMENTO ES UN BORRADOR. PUEDE SUFRIR CAMBIOS POSTERIORES

Objetivo general

Implementar un juego que permita la aplicación del Paradigma Orientado a Objetos mediante la utilización de estructuras de datos lineales.

Objetivos específicos

Diseñar una solución que permita resolver el problema descrito en esta especificación aplicando patrones de diseño y utilizando el estándar UML.

Implementar una solución que permita resolver el problema descrito en esta especificación utilizando Programación Orientada a Objetos en Java y haciendo uso de estructuras de datos lineales.

Elaborar la documentación correspondiente a la solución implementada para la evidencia del trabajo desarrollado utilizando estándares de documentación técnica y herramientas de gestión de proyectos

Atributos relacionados

A continuación, se describen los atributos del graduado que se pretenden abordar con el desarrollo del proyecto.

- Diseño (DI): Diseña soluciones creativas para problemas de ingeniería complejos y diseña sistemas, componentes o procesos para satisfacer las necesidades identificadas con la consideración adecuada para la salud y la seguridad públicas, el costo total de la vida, el carbono neto cero, así como las consideraciones de recursos, culturales, sociales y ambientales según sea necesario.

Descripción del problema

A grandes rasgos, CMP es un sistema de software cliente-servidor. El servidor es una aplicación de escritorio (con interfaz gráfica) que funciona como un reproductor de música tradicional, utilizando estructuras de datos lineales. Permite crear *playlists* comunitarios permitiendo a los clientes (aplicaciones de escritorio ejecutándose en computadoras distintas) proponer canciones o votar por las canciones propuestas por otros usuarios.

Requerimientos

Requerimientos del servidor

A continuación, se detallan los requerimientos del componente servidor.

REQ-S001: El servidor es una aplicación de escritorio desarrollada en Java JDK 21 con interfaz gráfica utilizando JavaFX.

REQ-S002: El servidor tiene un archivo de configuración en formato .INI que permite configurar distintos aspectos internos del sistema, por ejemplo, la ruta de la biblioteca musical, el puerto de escucha entre otros. Requerimientos posteriores indicarán cualquier otro ítem de configuración que debe ser configurado. El archivo .INI se coloca en el mismo proyecto donde están los archivos fuentes, para que, a la hora de ejecutar el proyecto, se pueda encontrar el archivo sin especificar una ruta absoluta.

REQ-S003: El servidor carga una biblioteca musical leyendo la ruta (absoluta) desde el archivo de configuración. Con la lista de archivos .MP3 leídos de la carpeta, el servidor crear una lista doblemente enlazada (conocida como la **lista principal**) y la muestra en la interfaz gráfica de manera tradicional. Cada nodo de la lista tiene:

a. Id único en formato GUID autogenerado por cada canción cargada en memoria.
b. Nombre de la canción (extraído de la *metadata* del .MP3)
c. Nombre del artista (extraído de la *metadata* del .MP3)
d. Nombre del álbum (extraído de la *metadata* del .MP3)
e. Género musical (extraído de la *metadata* del .MP3)
f. Cantidad total de up-votes (inicialmente en cero)
g. Cantidad total de down-votes (inicialmente en cero)
h. Referencia al archivo MP3

La interfaz gráfica no muestra el Id autogenerado, pero si muestra todos los datos indicados anteriormente.

REQ-S004: El servidor tiene las funcionalidades tradicionales de un reproductor de música (utilizando cualquier biblioteca Open Source para Java que permita hacer reproducción de audio):

a. Reproducir
b. Pausar
c. Adelantar/Atrasar la canción actual
d. *Scrub* en el timeline de la canción
e. Ir a la siguiente
f. Ir a la anterior
g. Subir/Bajar volumen
h. Eliminar canciones (esto debe actualizar todas las listas que tengan referencias a dicha canción). **Hint**: utilizando el patrón observer, podría resultar en una implementación muy elegante para actualizar cualquier lista interesada.

REQ-S005: La información de la canción actual debe mostrarse claramente en la parte superior del reproductor.

REQ-S006: El servidor muestra a un lado, la lista de todos los artistas diferentes encontrados en la biblioteca música. Esta se implementa con una lista circular, donde el valor de cada nodo incluye:

a. Nombre del artista
b. Lista de canciones del artista. Referenciando los nodos de la lista principal (la doble enlazada creada al cargar el server) sin duplicar el contenido de dichos nodos.

Al seleccionar un artista de la lista de artistas, la lista de canciones mostrada en la UI, es solo la del artista seleccionado. Al ser una lista circular, se reproduce infinitamente.

REQ-S007: El servidor permite activar la función de *playlist* comunitario. Cuando esta función está activa (un toggle en la UI), se crea una cola de prioridad donde la prioridad está dada por la cantidad de votos efectivos (votos a favor **menos** en contra). Inicialmente, el servidor propone 10 canciones aleatorias como parte del playlist. El servidor inicia un socket TCP de escucha para recibir votos de los clientes. El puerto del socket se define en el archivo de configuración.

REQ-S008: El servidor utiliza Log4J 2 para llevar un registro de mensajes de error, debug, traces. No se permite que las excepciones o errores se escriban en el standard output (system.out). Deberá investigar buenas prácticas de logging y aplicarlas. *(Este requerimiento será recurrente en todos los proyectos y tareas programadas)*

REQ-S009: Cuando el *playlist* se termina de reproducir, el servidor detiene el modo comunitario. El usuario del server puede reactivarlo e iniciar el proceso nuevamente.

REQ-S010: El servidor cuando está en modo de *playlist* comunitario recibe y responde a las siguientes peticiones (toda la comunicación es en formato JSON). A continuación, se muestran los mensajes y **ejemplos** de respuestas:

je	esta

command": "Get-Playlist"	mmand": "Get-Playlist",
	atus": "OK",
	st": [
	{
	"id": "f97afe84-54d4-43d0-a0c0-b37c91c1349c",
	"song": "Song-1",
	"artist": "Artist-A"
	},
	{
	"id": "b2d0f740-7837-4595-acf3-ba93f6b6f628"
	"song": "Song-2",
	"artist": "Artist-B"
	}
	
	La lista
	ada, se basa en la cola de prioridad, por lo que la
	sta depende de la cantidad de votos de cada canción,
	r el orden puede variar. Al ser una cola, las canciones
	nente reproducidas no se muestran. No se retorna la
	n actualmente siendo reproducida puesto que no se
	votar por la misma.

{
"command": "Vote-up",
"id": "f97afe84-54d4-43d0-a0c0-b37c91c1349c"
}
	{
	"command": "Vote-up",
	"status": "OK"
	}
	Cada voto debe re-ordenar la cola de prioridad y el playlist por reproducir

{
"command": "Vote-down",
"id": "f97afe84-54d4-43d0-a0c0-b37c91c1349c"
}
	{
	"command": "Vote-down",
	"status": "OK"
	}
	Cada voto debe re-ordenar la cola de prioridad y el playlist por reproducir

Cualquier error del servidor al procesar una respuesta debe retornar:


```
{
    "command": "<Nombre del command>",
    "status": "ERROR"
}
```


REQ-S011: El servidor en modo de playlist comunitario, recibe todos los mensajes en un thread dedicado para recibir y responder peticiones del usuario. De otra forma, la interfaz gráfica se quedaría congelada mientras se procesan las peticiones. Consideraciones importantes:

1. Investigar como disparar una actualización en la interfaz gráfica desde un hilo secundario
2. Utilizar semaforos (o Mutex) para coordinar la actualización de las listas desde el hilo principal (el de la interfaz gráfica) y el hilo secundario que atiende mensajes.

Requerimientos del cliente

REQ-C001: El cliente es una aplicación de escritorio desarrollada en Java con JDK 21 y JavaFX.

REQ-C002: El cliente tiene un archivo de configuración en formato .INI que permite configurar distintos aspectos internos del sistema, por ejemplo, el puerto en el que el servidor escucha

REQ-C003: El cliente al iniciar trata de conectarse con el socket del servidor configurado en el .INI. Si la conexión no se logra establecer, seguirá intentando. El socket estará disponible hasta que el servidor lo inicie, por lo tanto, es esperado que abrir el cliente antes que el playlist comunitario se inicie, no funcione.

REQ-C004: El cliente utiliza Log4J 2 para llevar un registro de mensajes de error, debug, traces. No se permite que las excepciones o errores se escriban en el standard output (system.out). Deberá investigar buenas prácticas de logging y aplicarlas. *(Este requerimiento será recurrente en todos los proyectos y tareas programadas)*

REQ-C005: El cliente recibe actualizaciones del servidor mediante polling. Es decir, cada cierta cantidad de segundo (configurado en el archivo .INI) envía un mensaje al servidor para obtener el playlist

actualizado. La respuesta del server en JSON se convierte a una cola. El playlist actualizado se muestra en la interfaz gráfica del cliente.

REQ-C006: El cliente es capaz de enviar up-votes o down-votes. La interfaz permite presionar un icono de up-vote o down-vote y esto envia un mensaje al servidor

Requerimientos generales

REQ-G001: El cliente y servidor debe ser tolerante a fallas como cierres inesperados del socket. Deben manejar estos errores de forma correcta para garantizar que la aplicación (cliente o servidor) no se "caigan"

UI/UX Mock-ups

En clase se darán ideas de la UI/UX esperada.

Documentación requerida

Debe entregarse un documento PDF llamado "Documento de diseño" que incluya las siguientes secciones:

1. Introducción
2. Tabla de contenido
3. Breve descripción del problema
4. Descripción de la solución propuesta
5. Decisiones de diseño. Para cada decisión relevante:
    a. Alternativas consideradas
    b. Alternativa seleccionada y razones de la selección
6. Diagrama de clases UML de la solución propuesta (construido previo a la implementación)
7. Problemas encontrados
8. Preguntas abiertas

Deberá entregarse un documento PDF llamado "Planificación del proyecto" que contenga lo siguiente:

1. Lista de historias de usuario (pueden usar Azure DevOps o Jira para llevar la lista de Tareas, pero el documento debe encontrar la lista de las mismas)
2. Plan de iteraciones que agrupen cada bloque de historias de usuario por Sprint, de forma que se vea un desarrollo incremental. Se deberán de crear tres Sprints.
3. Asignación de tareas a cada miembro del equipo.

Aspectos operativos y evaluación

1. **Fecha de entrega: De acuerdo al cronograma del curso**
2. El proyecto tiene un valor de 25% de la nota del curso.
3. El trabajo es **en grupos de 4 personas**.
4. Es obligatorio utilizar un GitHub para el manejo de las versiones. Se debe evidenciar el uso de *commits* frecuentes.
5. Deben entregar en el TEC Digital un documento con el link del repositorio de GitHub y los PDF de las documentaciones.
6. Es obligatorio integrar toda la solución, es decir, debe estar la UI y la lógia de negocios integrada.

7. La presentación funcional tendrá un valor de 70%, la documentación externa 15% y la documentación de diseño 15%.
8. De las notas mencionadas en el punto anterior se calculará la Nota Final del Proyecto.
9. Se evaluará que la documentación sea coherente, acorde a la dificultad/tamaño del proyecto y el trabajo realizado, se recomienda que realicen la documentación conforme se implementa el código.
10. La documentación se revisará según el día de entrega en el cronograma.
11. Las citas de revisión oficiales serán determinadas por el profesor durante las lecciones o mediante algún medio electrónico.
12. Los estudiantes pueden seguir trabajando en el código hasta 15 minutos antes de la cita revisión oficial.
13. Aun cuando el código y la documentación tienen sus notas por separado, se aplican las siguientes restricciones
14. Si no se entrega documentación, automáticamente se obtiene una nota de 0.
15. Si no se utiliza Git se obtiene una nota de 0.
16. Si la documentación no se entrega en la fecha indicada se obtiene una nota de 0.
17. Sí el código no compila se obtendrá una nota de 0.
18. El código debe desarrollarse en Java, si no, se obtendrá una nota de 0.
19. La nota de la documentación debe ser acorde a la completitud del proyecto.
20. Si alguna persona integrante del proyecto no se presenta a la revisión se le asignará una nota de cero en la nota final del proyecto.
21. La revisión de la documentación será realizada por parte del profesor, no durante la defensa del proyecto.
22. Cada grupo tendrá como máximo 30 minutos para exponer su trabajo al profesor y realizar la defensa de éste, es responsabilidad de los estudiantes mostrar todo el trabajo realizado, por lo cual se recomienda tener todo listo antes de ingresar a la defensa.
23. Cada excepción o error que salga durante la ejecución del proyecto y que se considere debió haber sido contemplada durante el desarrollo del proyecto, se castigará con 2 puntos de la nota final del proyecto.
24. Cada estudiante es responsable de llevar los equipos requeridos para la revisión, si no cuentan con estos deberán avisar al menos 2 días antes de la revisión a el profesor para coordinar el préstamo de estos.
25. Durante la revisión únicamente podrán participar el estudiante, asistentes, otros profesores y el coordinador del área.