Informe Desafio 2

1st Julián Sánchez Ceballos
Facultad de ingeniería
Universidad de Antioquia
Medellín, Colombia
julian.sanchez6@udea.edu.co

2nd Jaider Bedoya Carmona
Facultad de ingeniería
Universidad de Antioquia
Medellín, Colombia
jaider.bedoya@udea.edu.co

***Abstract*—Este documento presenta la propuesta de solución para el Desafío 2 del curso de Informática II, cuyo objetivo es diseñar e implementar una aplicación en C++ que emule un servicio de streaming musical. La solución se desarrolla bajo el paradigma de Programación Orientada a Objetos (POO), sin el uso de herencia ni librerías STL, promoviendo la construcción manual de estructuras dinámicas mediante plantillas genéricas.**

El sistema permite la gestión de usuarios, artistas, álbumes, canciones y listas de favoritos, además de la reproducción aleatoria de música con soporte para publicidad y funcionalidades diferenciadas entre usuarios estándar y premium. La arquitectura propuesta se organiza en capas modulares que abarcan la gestión de datos, entidades, lógica de usuario, reproducción y utilidades. Finalmente, se presenta un diseño orientado a la eficiencia, con medición del consumo de memoria e iteraciones para evaluar el desempeño del sistema.

***Index Terms*—C++, programación orientada a objetos, streaming musical, estructuras dinámicas, POO, memoria dinámica.**

INTRODUCCIÓN

Como parte de la evaluación del curso de Informática II, se propone un desafío que permita fortalecer las habilidades de programación bajo el paradigma de Programación Orientada a Objetos (POO).

El objetivo del desafío consiste en desarrollar un programa que emule el funcionamiento de un servicio de streaming musical utilizando POO. El sistema debe permitir la gestión eficiente de canciones, álbumes, artistas, usuarios y listas de reproducción, entre otros.

PROPUESTA DE SOLUCIÓN

Condiciones y restricciones

Para el desarrollo del desafío se tienen las siguientes restricciones con las que se debe contar para la propuesta de solución:

- El programa debe ser desarrollado en C++, sin el uso de STL (a excepción de string).
- El programa debe ser desarrollado bajo el paradigma de Programación Orientada a Objetos (POO) sin el uso de herencia.
- Finalmente se debe contar con un repositorio donde se pueda evidenciar el avance del desarrollo de la solución del desafío.

Requerimientos Funcionales

1) **Carga y actualización de datos** Se implementarán algoritmos para leer y actualizar información desde y hacia el almacenamiento permanente.
2) **Ingreso a la plataforma** Permite iniciar sesión con credenciales personales, validando los datos desde el almacenamiento permanente y según el tipo de usuario (estándar o premium), se mostrará el menú correspondiente. No se contempla el registro de nuevos usuarios.
3) **Reproducción aleatoria** Permite reproducir canciones de forma totalmente aleatoria. Durante la reproducción se mostrará:
   - Ruta del archivo de audio y de la portada del álbum.
   - Opciones: *Iniciar* y *Detener*.

   Los usuarios estándar escucharán publicidad durante la reproducción. Los usuarios premium dispondrán además de:
   - *Siguiente*, *Previo* (hasta cuatro canciones hacia atrás) y *Repetir*.

   Se deben validar las condiciones lógicas (por ejemplo, no detener si no hay reproducción activa). Para pruebas, se simulará un temporizador de 3 segundos por canción y un límite de $K = 5$ canciones, utilizando la librería chronos.
4) **Mi lista de favoritos (solo usuarios premium)** Cada usuario podrá mantener una lista personalizada de hasta 10 000 canciones. El menú de esta funcionalidad incluirá:
   a) ***Editar mi lista de favoritos:*** agregar o eliminar canciones según su código `id`. No se permiten duplicados.
   b) ***Seguir otra lista de favoritos:*** permite seguir la lista de otro usuario, fusionándola con la propia.
   c) ***Ejecutar mi lista de favoritos:*** reproduce la lista en orden original o aleatorio. Se permite retroceder hasta $M = 6$ canciones.
5) **Medición del consumo de recursos** Al finalizar cada funcionalidad, el sistema deberá mostrar:
   a) Cantidad total de iteraciones ejecutadas (directas e indirectas).
   b) Memoria total utilizada por las estructuras de datos, objetos, variables locales y parámetros por valor.

Clases y sus relaciones.

TABLE I
RESUMEN DE CLASES DEL SISTEMA

Clase	Atributos	Métodos
Artist	id, age, name, country, albums	addAlbum(), getAlbum()
Album	id, name, label, songs, genres_mask, release_date	addSong(), totalDuration()
Song	id, duration, name, path, playCount, credits	play(), getId(), getName()
Credit	name, surname, code[11]	set()
User	nickname, premium, city, country, signup_date, favorites, following	addFavorite(), playFavorites(), follow(), isPremium(), getNick()
DataBase	users, artists, albums, songs, ads	findUser(), findSong(), loadData(), saveData()
AdMessage	text, category, weight	getWeight(), getText()
PlaybackSession	user, history, repeat, lastAdIndex	playRandom(), playFavorites(), showMetrics()
DynamicArray<T>	data, len, capacity	push_back(), reserve(), clear(), getSize(), operator[]
MemoryTracker	totalBytes	alloc(size_t), free(size_t), getTotal()
Counter	iterations	reset(), inc(), get()

TABLE II
RELACIONES ENTRE CLASES

Clase origen	Relación	Clase destino
Artist	contiene	Album
Album	contiene	Song
Song	usa	Credit
User	contiene	Song (favoritos)
User	sigue	User
DataBase	contiene	User, Artist, Album, Song, AdMessage
PlaybackSession	usa	User, Song, AdMessage
MemoryTracker	monitorea	Todas las clases que usen new y delete
Counter	monitorea	Todas las operaciones logicas
AppMusic	Controla	

A. *Arquitectura de la solución*

La arquitectura del sistema se basa en el paradigma de Programación Orientada a Objetos (POO), sin el uso de herencia ni librerías STL, siguiendo un diseño modular y extensible.

- **Capa de datos:** gestionada por la clase `DataBase`, encargada de la carga, persistencia y actualización de la información desde archivos definidos por los programadores.
- **Capa de entidades:** incluye las clases Artist, Album, Song, Credit y AdMessage, que modelan los elementos principales de la plataforma.
- **Capa de lógica de usuario:** representada por la clase User, que administra las acciones de los usuarios, sus listas de favoritos y las relaciones entre ellos.
- **Capa de reproducción:** implementada mediante la clase PlaybackSession, que controla la reproducción aleatoria, la interacción con publicidad y las métricas de rendimiento.
- **Capa de utilidades:** contiene la plantilla DynamicArray<T>, que provee estructuras de datos dinámicas sin depender de STL.

Estructura de directorios

La estructura de los directorios para la solución del desafio se dispone de la siguiente manera:

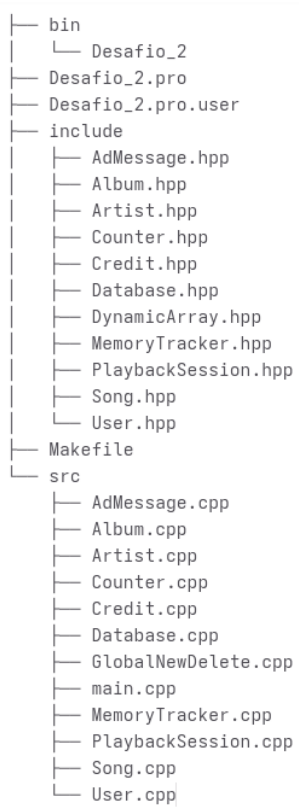

Fig. 1. Estructura final del proyecto

Cada archivo de la las carpetas include y src componen una clase, esto con la finalidad de cumplir con el requisito de programación bajo el dogma de POO.

IMPLEMENTACIÓN DE LA SOLUCIÓN

Cada clase cumple una tarea dentro de la ejecución del programa, en esta sección se describe de manera individual la función que cumple cada una de estas en la implementación del desafío:

Artist:

En ella se almacena la información básica del artista, como su nombre, edad, país de origen y un identificador único, la clase permite administrar los álbumes asociados al artista mediante un arreglo dinámico. Incluye funciones para agregar nuevos álbumes, buscar alguno en específico por su ID, consultar la cantidad total de álbumes y mostrar toda la información del artista junto con sus álbumes.

Album:

Contiene su nombre, sello discográfico, género, fecha de lanzamiento y una lista dinámica de canciones, cada álbum tiene un identificador único y métodos que permiten agregar canciones, calcular su duración total, acceder a sus datos y mostrar toda su información por consola.

Song:

Guarda información como su nombre, duración, ruta del archivo, cantidad de reproducciones y los créditos de las personas que participaron en su creación (productores, músicos y compositores). También permite reproducir la canción de forma simulada, aumentando su contador de reproducciones y mostrando su información en consola, tiene métodos para agregar créditos y consultar todos los datos de la canción.

User:

esta clase se encarga de lo que pueden hacer los usuarios, tiene su nombre de usuario (nickname), ciudad, país, fecha de registro y si tiene una cuenta premium. Además, permite al usuario guardar canciones como favoritas, reproducirlas (de forma ordenada o aleatoria) y seguir a otros usuarios dentro de la plataforma. También cuenta con métodos para consultar sus datos, eliminar canciones de su lista de favoritos y ver sus estadísticas.

DataBase:

Su función principal es administrar y organizar todos los datos importantes, como los usuarios, artistas, álbumes, canciones y anuncios publicitarios. Permite cargar información de prueba, guardar los datos en archivos y realizar búsquedas dentro del sistema, por ejemplo, encontrar una canción por su identificador o un usuario por su nombre. También ofrece métodos para consultar las listas completas de objetos y mostrar un resumen general del contenido almacenado.

AdMessage:

se encarga de representar los mensajes publicitarios dentro del sistema. Cada anuncio contiene un texto, una categoría (como A, B o C) y un peso o nivel de prioridad que determina su importancia o frecuencia de aparición. Ofrece métodos para definir, modificar y consultar estos valores, así como para mostrar la información del anuncio por consola.

PlayBackSession:

administra una sesión de reproducción musical para un usuario. Permite reproducir canciones (ya sean aleatorias o favoritas), registrar el historial de reproducción y mostrar anuncios de forma ocasional durante la sesión. Además, recopila métricas como el número de canciones reproducidas y anuncios mostrados. También ofrece opciones como el modo de repetición y control para evitar reproducir canciones recientes. Está diseñada para trabajar junto con las clases User, Database, DynamicArray y AdMessage.

DynamicArray:

La clase plantilla DynamicArray implementa un arreglo dinámico genérico, que cumple la función de std::vector ya que este no se puede usar en este desafío, que puede almacenar elementos de cualquier tipo y ajusta automáticamente su capacidad cuando se agregan más elementos. Permite agregar, eliminar y acceder a elementos mediante índices, así como reservar memoria para optimizar el rendimiento. Además, gestiona su propia memoria, asegurando que se libere correctamente al destruir el objeto.

MemoryTracker:

se encarga de monitorear el uso global de memoria dentro de una aplicación. Registra cuántos bytes se han asignado y liberado, calcula la memoria actualmente en uso, y permite mostrar reportes detallados sobre el estado de la memoria. Todos sus métodos son estáticos, por lo que no necesita instanciarse, y utiliza un mutex para garantizar seguridad en entornos con múltiples hilos. Es útil para detectar fugas de memoria y analizar el consumo.

Counter:

implementa un contador que permite registrar el número de operaciones o eventos dentro de un sistema, tiene métodos para incrementar, reiniciar y consultar el valor actual del contador, además de una función para mostrar métricas en consola junto con un contexto opcional que indique su propósito.

Credit:

se encarga de almacenar y mostrar información de crédito o autoría de una persona. Guarda tres datos principales: el nombre, el apellido y un código identificador de hasta 10 caracteres. Tiene métodos para acceder a cada uno de estos valores y una función para mostrar la información completa en consola.

AppMusic:

Representa el núcleo principal de la aplicación musical, encargada de gestionar la base de datos, las sesiones de usuario y las funciones clave del sistema. Controla procesos como el inicio de sesión, la reproducción de canciones (aleatorias o favoritas), la gestión de listas de favoritos y la interacción entre usuarios (como seguir listas ajenas), coordina componentes como Database, PlaybackSession, MemoryTracker y Counter para mantener un control completo del funcionamiento interno.

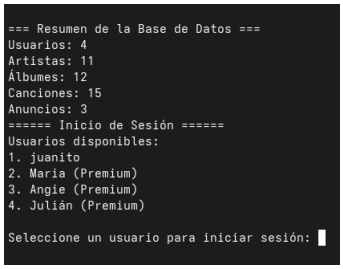

Fig. 2. Manú principal y entrada de la aplicación.

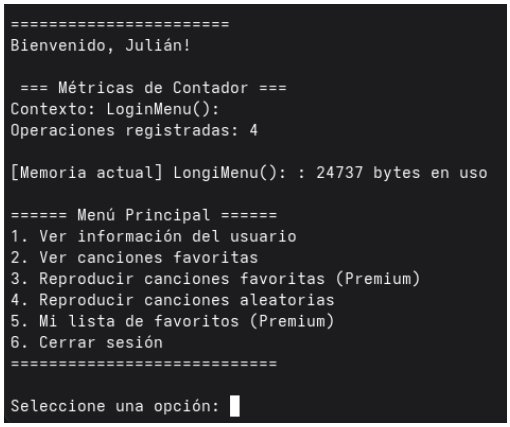

Fig. 3. Manú del usuario.

Es este el objeto mas importante de toda la aplicación, pues es el que dirige la lógica del programa y su orden de ejecución.

Desde aquí se despliegan los dos menús principales que serán la interfaz con el usuario en esta primera versión de la aplicación.

Y finalmente el menú de la lista de favoritos, el que solo es una funcionalidad premium:

Problemas en la Implementación y Posibles Mejoras

Como se ha mencionado anteriormente el programa está hecho en su todalidad basado en el paradigma de POO (Programación Orientada a Objetos) durante la implementación se encontraron problemas como:

- **Gestión de arreglos dinamicos:** al tener la limitación en el uso de STL, no era posible el uso de arreglos dinamicos de la clase std::vector, y al ser este el mas apropiado para los problemas planteados, se tuvo que crear propiamente una clase plantilla la cual ocupase las tareas de la clase vector, llamada **DynamicArray** explicada en la sección anterior, haciendo uso de std::vector se habría evitado implementar manualmente las operaciones de redimensión y manejo de memoria, ya que la clase estándar proporciona estas funcionalidades de forma segura y eficiente.
- **Encapsulamiento de clases:** Dado que se pide explicitamente que el codigo sea desarrollado bajo el dogma de POO no se puede asegurar que el codigo sea el mas eficiente en terminos de rendimiento, pues, aunque presenta una organización impecable esta organización afecta directamente el rendimiento del codigo en comparación de posibles implementaciones usando estructuras en C con miembros publicos
- **Uso de dataset mas amplio:** Una posible mejora sería ampliar el dataset utilizado, incluyendo más usuarios, artistas y canciones. Esto permitiría probar mejor la eficiencia del sistema y su comportamiento con un mayor volumen de datos.
- **Escritura y guardado de datos:** Aunque actualmente los datos se guardan en archivos, una mejora importante sería integrar una base de datos real. Esto facilitaría el acceso concurrente, la persistencia estructurada de la información y una mayor eficiencia en las operaciones de búsqueda y actualización.

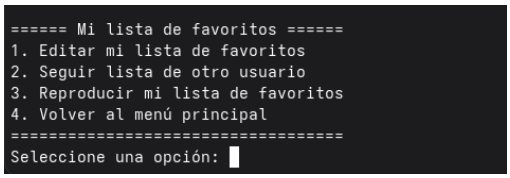

Fig. 4. Menú de favoritos.

Conclusiones

El uso de objetos permitió mantener una estructura organizada y modular dentro del programa, aunque aumentó el nivel de abstracción del código. La decisión de no utilizar la biblioteca STL condujo al desarrollo de una solución más controlada y adaptable, capaz de ejecutarse en sistemas con recursos limitados.

El manejo cuidadoso de la memoria garantiza la estabilidad del sistema, evitando errores o fugas. En conjunto, el proyecto representa una implementación sólida de los principios de la Programación Orientada a Objetos, optimizada para entornos con restricciones de recursos.

References

[1] B. Stroustrup, *The C++ Programming Language*, 4th ed., Addison-Wesley, Boston, MA, USA, 2013.

[2] S. Meyers, *Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14*, O'Reilly Media, Sebastopol, CA, USA, 2014.