Pre-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*—**

INTRODUCCIÓN

Como parte de la evaluación del curso Informática 2 se propone un desafío que permita afianzar 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, 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
MemoryTacker	monitorea	Todas las clases que usen new y delete
Counter	monitorea	Todas las operaciones logicas

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 propone de la siguiente manera:

```
Desafio_2/
├── src/
│   ├── main.cpp
│   ├── Database.cpp
│   ├── PlaybackSession.cpp
│   ├── User.cpp
│   ├── Song.cpp
│   ├── Artist.cpp
│   ├── Album.cpp
│   ├── Credit.cpp
│   ├── MemoryTracker.cpp
│   ├── Counter.cpp
│   └── DynamicArray.cpp
├── include/
│   ├── Database.hpp
│   ├── PlaybackSession.hpp
│   ├── User.hpp
│   ├── Song.hpp
│   ├── Artist.hpp
│   ├── Album.hpp
│   ├── Credit.hpp
│   ├── MemoryTracker.hpp
│   ├── Counter.hpp
│   └── DynamicArray.hpp
├── build/Desktop-Debug/bin
│   ├── Desafio_2.bin
│   ├── users.txt
│   ├── artists.txt
│   ├── albums.txt
│   ├── songs.txt
│   └── ads.txt
├── documentación/
│   ├── Diagrama de clases.pdf
│   ├── Informe Final.pdf
│   └── Pre-Informe.pdf
├── .gitignore
├── Desafio_2.pro
└── Desafio_2.pro.user
```

Fig. 1. Estructura del proyecto