INFORME DESAFIO 2

INFORMATICA II

Cristian Camilo Murillo Jiménez

C.C 1017926405

Juan Felipe Pérez Salazar

C.C1017922201

UNIVERSIDAD DE ANTIOQUIA

2025

Informe desafío 2

El desafío 2 del curso informática 2, tuvo como objetivo principal modelar y desarrollar un prototipo “funcional” de un servicio de streaming de música llamada UdeATunes. El propósito era aplicar los conceptos fundamentales de la Programación Orientada a Objetos (POO) en C++ para gestionar las entidades y dinámicas propias de este contexto, además se debía de hacer uso de clases, métodos, memoria dinámica y sin usar métodos de STL.

El sistema pedido debe gestionar entidades como usuarios con membresías estándar o premium, artistas, álbumes, canciones, listas de reproducción y mensajes publicitarios que solo serán mostrados a usuarios estándar, incluyendo funcionalidades clave como carga de datos desde archivos, inicio de sesión, reproducción aleatoria con opciones específicas según el tipo de usuario, por ejemplo, listas de favoritos para premium o solo reproducir música en aleatorio para usuarios estándar. Debía contener mediciones de eficiencia (iteraciones y memoria consumida), además, realizarle su respectiva documentación haciendo uso de un diagrama de clases UML, un video explicativo donde se muestre la funcionalidad del codigo y un repositorio con commits regulares, cumpliendo con criterios de eficiencia y buenas prácticas de diseño.

Los requisitos clave extraídos del desafío incluían:

- Manejar diferentes entidades, en nuestro caso, usuarios, artistas, canciones, álbumes, y playlists (solo para usuarios premium).
- Diferenciar dos tipos de usuarios, estandar y Premium.
- Los usuarios estándar solo pueden hacer una reproducción aleatoria de las canciones que contiene UdeATunes con anuncios publicitarios y calidad de audio estándar (128kbps).
- Los usuarios Premium, tienen una amplia variedad de opciones, como lo es tener y poder reproducir una playlist favorita personalizada por ellos, su cuenta estará libre de anuncios, reproducen audio de alta calidad (320kbps), y funciones sociales como lo es seguir una playlist de otro usuario y reproducirla junto con la suya.
- Almacenar todos los datos desde archivos de texto (.txt), para poder tener siempre la información guardada y cargarla cuando sea necesario.

En el análisis correspondiente que se realizó para abordar la complejidad del problema, se estructuró la solución con un enfoque de un diseño modular basado en la abstracción y la separación de responsabilidades.

En la abstracción, se identificaron las entidades clave requeridas y se implementaron en clases de C++, encapsulando sus atributos y métodos como era conveniente.

Cancion (cancion.h): Es la entidad fundamental. Almacena su ID, nombre, duración y, algo muy fundamental, las dos rutas a los archivos de audio (ruta128 y ruta320). Su método obtenerRutaAudio(int esPremium) decide qué ruta devolver según el tipo de usuario.

Artista (artista.h): Modela al creador de la música, con sus datos biográficos y de popularidad.

Album (album.h): Agrupa canciones, vinculando un idArtista con una colección de idCancion.

Anuncio (anuncio.h): Modela los anuncios publicitarios, con un ID, texto y prioridad con la que sale dicho anuncio.

Usuario (usuario.h): Representa al cliente y almacena sus credenciales (nombre de usuario y contraseña), su estado premium (0 o 1) y si es el caso de estar siguiendo la playlist de otro usuario, aparecerá reflejado allí a que usuario se está siguiendo y se incluye el atributo usuarioSeguido.

Playlist (playlist.h): El componente central de la experiencia del usuario, este gestiona una lista de IDs de canciones, las cuales se reproducirán cuando el usuario de la orden de hacerlo. Esta también es responsable de la lógica de reproducción, mantiene un índice actual para la reproducción secuencial, implementa un historial para la función "Anterior" y provee métodos clave como siguiente(), anterior() y reproducirAleatoria().

Una de las bases principales del diseño fue no mezclar la lógica de una entidad con la lógica de acceso a datos. En lugar de hacer que la clase “cancion” sepa cómo leer canciones.txt, se crearon clases "gestoras" o "lectoras":

- LecturaCanciones (lecturacanciones.h): Actúa como un repositorio de canciones, esta carga todas las canciones de canciones.txt en memoria (en un arreglo dinámico Cancion* listaCanciones) al inicio del programa y ofrece un método público llamado buscarCancionPorID(long idBuscado) que permite al resto del sistema obtener cualquier canción de forma eficiente.
- LecturaArtistas (lecturaartistas.h): Esta cumple la misma función para artistas.txt.
- LecturaUsuario (usuario.h): Gestiona usuarios.txt y es responsable de “cargarDesdeArchivo”, “verificarUsuario” para el inicio de sesión y “guardarEnArchivo” para guardar los cambios, como cuando se sigue a un usuario.

Para mantener el main.cpp limpio y enfocado en el flujo de la aplicación, toda la lógica de mostrar menús y capturar entradas con cin / cout se aisló en funciones independientes con el uso de menu.h y menu.cpp.

Como ya se mencionó, el flujo de la aplicación se implementó en main.cpp, coordinando todos los objetos y clases diseñados.

- Arranque y Carga de Datos: Al ejecutar main(), lo primero que ocurre es la inicialización de los gestores. Se crean instancias de “LecturaUsuario”, “LecturaArtistas”, “LecturaCanciones” y se cargan los anuncios con “Anuncio::cargarDesdeArchivo”. Todos los datos de los .txt como canciones, usuarios, artistas, anuncios, entre otros, quedan disponibles en memoria para ser usados cuando sea necesario.

- Autenticación y Flujo Principal: El programa entra en un bucle “ingreso1” controlado por “menuIngreso()”, el usuario selecciona "Ingresar", “menuIngresoUsuario()” captura el nombre de usuario y contraseña, “lector.verificarUsuario()” valida esta información con la lista de usuarios cargada y

si el ingreso es exitoso, la función devuelve el estado “esPremium” (0 o 1). Esta variable se convierte en el "interruptor" principal que controla el resto de la experiencia.

- Implementación del flujo Estandar “(esPremium == 0)”. Se muestra el “menuFuncionesNoPremium()”, de la cual, la única opción funcional es reproducir 5 canciones aleatorias con anuncios mostrados cada que se reproducen 2 canciones del ciclo. Lo que se hace, es que en cada iteración se obtiene una canción con “LecturaCanciones.obtenerCancionAleatoria()”, se obtiene la ruta de audio con “cancion.obtenerRutaAudio(esPremium)” que, al ser “esPremium = 0”, devuelve la ruta128.

- Implementación del flujo Premium “(esPremium == 1)” . Se muestra el “menuFuncionesPremium()”, se crea un objeto playlist, “playlist.cargarDesdeArchivo(usuarioActual + ".txt")” carga la lista de IDs de canciones. Además, el usuario puede entrar al “menuEditarPlaylist()” para añadir o eliminar canciones por ID, para hacer esto el programa cuenta con “playlist.agregarCancion(id)" o “playlist.eliminarCancion(id)” lo cual modifica el arreglo de IDs. Por Ultimo, “playlist.guardarEnArchivo()” se llama para guardar los cambios en el .txt del usuario.

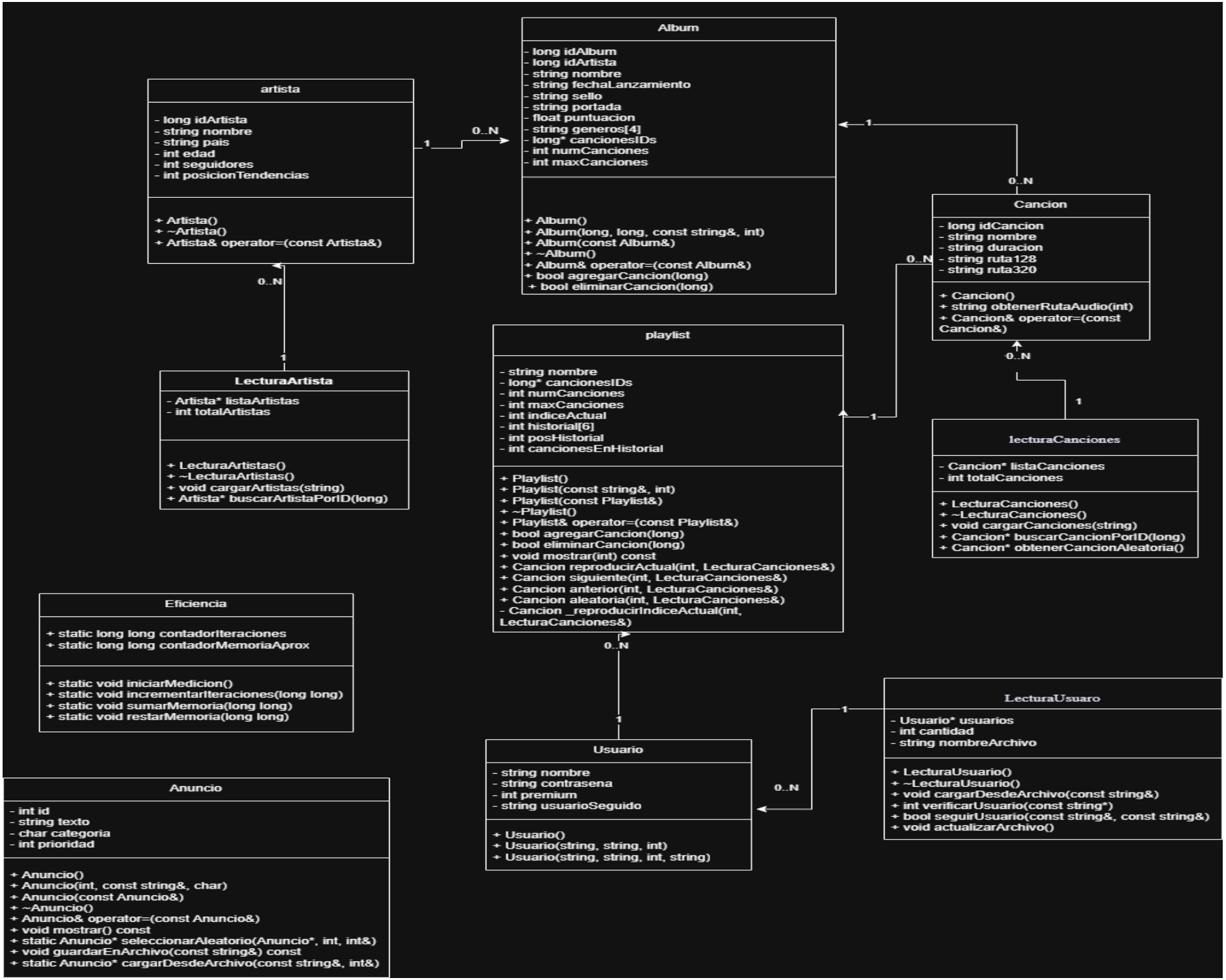
- Para la reproducción de playlist, el usuario interactúa con los controles “menuControlesRepro()”, según la opción seleccionada, se llama a “playlist.siguiente()”, “playlist.anterior()” o “playlist.reproducirAleatoria()”, estos métodos actualizan el índice actual de la playlist y devuelven el objeto “canción” correspondiente que será buscado con “LecturaCanciones.buscarCancionPorID()” y se obtiene la ruta de audio con “cancion.obtenerRutaAudio()” que, al ser “esPremium = 1”, devuelve la ruta320.

- La función Social o seguir una playlist de un usuario premium, el usuario puede seleccionar "Seguir playlist", se hace llamado a “LectoraUsuario.seguirUsuario()”, esta función actualiza el objeto “Usuario” en memoria y reescribe usuarios.txt con la información del usuario seguido.

Durante el desarrollo del desafío propuesto se presentaron diversos inconvenientes que requirieron análisis y ajustes continuos para lograr una solución eficiente. Uno de los principales retos correspondió a la definición y estructuración adecuada de las clases, ya que en un primer momento no se establecieron de manera óptima para responder correctamente a los requerimientos del problema. Esta situación generó dificultades en la organización del código y en la interacción entre los distintos componentes del sistema. En consecuencia, fue necesario replantear el diseño de las clases, sus atributos y métodos.

Otro de los desafíos más relevantes se relacionó con la medición y análisis de los recursos del sistema, particularmente en lo referente al consumo de memoria. Determinar con precisión la cantidad de memoria utilizada en un momento específico resultó complejo, debido a las variaciones que ocurren durante la ejecución del programa y a las limitaciones de las herramientas de medición disponibles.

En conclusión, el proyecto UdeATunes descrito anteriormente, es una solución robusta y bien diseñada que cumple de buena manera con los requisitos establecidos en el desafío 2, del curso informática 2. El uso correcto de la programación orientada a objetos, especialmente la abstracción como el modelado de cada entidad por separado como lo es usuarios, playlists, álbumes, canciones, entre otras, y la separación de responsabilidades usando clases gestoras o lectoras para el acceso a datos y un módulo respectivo para el "menu", da como resultado un código organizado, mantenible/escalable y que resuelve eficazmente la lógica para poder diferenciar entre usuarios estándar y premium.


Album
- long idAlbum
- long idArtista
- string nombre
- string fechaLanzamiento
- string sello
- string portada
- float puntuacion
- string generos[4]
- long* cancionesIDs
- int numCanciones
- int maxCanciones
+ Album()
+ Album(long, long, const string&, int)
+ Album(const Album&)
+ ~Album()
+ Album& operator=(const Album&)
+ bool agregarCancion(long)
+ bool eliminarCancion(long)
artista
- long idArtista
- string nombre
- string pais
- int edad
- int seguidores
- int posicionTendencias
+ Artista()
+ ~Artista()
+ Artista& operator=(const Artista&)
Cancion
- long idCancion
- string nombre
- string duracion
- string ruta128
- string ruta320
+ Cancion()
+ string obtenerRutaAudio(int)
+ Cancion& operator=(const Cancion&)
LecturaArtista
- Artista* listaArtistas
- int totalArtistas
+ LecturaArtistas()
+ ~LecturaArtistas()
+ void cargarArtistas(string)
+ Artista* buscarArtistaPorID(long)
playlist
- string nombre
- long* cancionesIDs
- int numCanciones
- int maxCanciones
- int indiceActual
- int historial[6]
- int posHistorial
- int cancionesEnHistorial
+ Playlist()
+ Playlist(const string&, int)
+ Playlist(const Playlist&)
+ ~Playlist()
+ Playlist& operator=(const Playlist&)
+ bool agregarCancion(long)
+ bool eliminarCancion(long)
+ void mostrar(int) const
+ Cancion reproducirActual(int, LecturaCanciones&)
+ Cancion siguiente(int, LecturaCanciones&)
+ Cancion anterior(int, LecturaCanciones&)
+ Cancion aleatoria(int, LecturaCanciones&)
- Cancion _reproducirIndiceActual(int, LecturaCanciones&)
lecturaCanciones
- Cancion* listaCanciones
- int totalCanciones
+ LecturaCanciones()
+ ~LecturaCanciones()
+ void cargarCanciones(string)
+ Cancion* buscarCancionPorID(long)
+ Cancion* obtenerCancionAleatoria()
Eficiencia
+ static long long contadorIteraciones
+ static long long contadorMemoriaAprox
+ static void iniciarMedicion()
+ static void incrementarIteraciones(long long)
+ static void sumarMemoria(long long)
+ static void restarMemoria(long long)
Usuario
- string nombre
- string contrasena
- int premium
- string usuarioSeguido
+ Usuario()
+ Usuario(string, string, int)
+ Usuario(string, string, int, string)
LecturaUsuaro
- Usuario* usuarios
- int cantidad
- string nombreArchivo
+ LecturaUsuario()
+ ~LecturaUsuario()
+ void cargarDesdeArchivo(const string&)
+ int verificarUsuario(const string*)
+ bool seguirUsuario(const string&, const string&)
+ void actualizarArchivo()
Anuncio
- int id
- string texto
- char categoria
- int prioridad
+ Anuncio()
+ Anuncio(int, const string&, char)
+ Anuncio(const Anuncio&)
+ ~Anuncio()
+ Anuncio& operator=(const Anuncio&)
+ void mostrar() const
+ static Anuncio* seleccionarAleatorio(Anuncio*, int, int&)
+ void guardarEnArchivo(const string&) const
+ static Anuncio* cargarDesdeArchivo(const string&, int&)
1
0..N