Informe análisis: *"Desafío 2"*

Estudiantes:

Harlin David Palacios Bermúdez

Sebastian Arango Sánchez

Docente Informática ll:

Aníbal José Guerra Soler

Universidad De Antioquia

Octubre 10 del 2025

Segundo semestre

a. Análisis del problema y consideraciones para la alternativa de solución propuesta.

Como objetivo de nuestra misión solicitada por el cliente, bajo el marco de la Programación Orientada a Objetos, este requiere la elaboración de una aplicación de streaming musical la cual tenga o posea gran potencial de ser tendencia en la población y alcanzar el top como compañía líder. Por tanto, esta aplicación que diseñaremos debe cumplir diversas funciones que hagan eficiente el manejo de los datos en todo el sistema, y así mismo, esta tenga una interacción práctica con los usuarios, de forma que estos gocen de diversas garantías, funciones especiales, control de su historial de interacciones, entre muchas otras actividades que generen una comodidad en el entorno para el usuario, que se espera goce de la aplicación. Entonces, para el desarrollo de nuestro modelo de solución, establecemos como punto de partida una base estructural para la arquitectura del sistema, lo cual nos permitirá comenzar a identificar y generar convenciones sobre las diversas interpretaciones que nos es posible desarrollar para llevar a cabo un correcto enlace entre toda la base de datos del sistema, y por supuesto, la interacción con el usuario y/o consumidor final, tal que esta sea eficiente, rápida y cumpla la expectativa esperada, bajo las siguientes condiciones inherentes al modelo de Programación Orientada a Objetos a plantear:

- Cada una de las partes fundamentales a interactuar para el intercambio, manejo y modelado de datos, y elementos del sistema serán planteadas en Clases(Usuario, Artistas, Álbum, Canciones, Mensajes Publicitarios, Reproducción, etc).
- Los Objetos a instanciar a partir de las clases, estarán almacenados en Arreglos Tipo objeto.
- La base de datos del Sistema será constantemente extraída, y así mismo, cargada en Archivos, que permitan almacenar los datos más allá de que el Programa finalice su ejecución, lo cual permitirá suponer la conservación de los cambios efectuados en la base de datos durante la ejecución del Programa, caracterizando a éste como garante de confianza de la información que se brinda en la plataforma de streaming.
- Para dar un correcto seguimiento a la gran proporción de actividades, en términos de la eficiencia, se guardará toda la información durante la ejecución del programa en arreglos lineales de forma dinámica, lo cual permitirá plasmar un modelo hacia el crecimiento de los datos que puedan ser añadidos al sistema(Usuarios, Artistas, Álbumes, etc), contando con que el programa debe de brindar el uso y capacidad para realizar una alta gama de opciones y resultados óptimos a entregar al cliente final.
- Debido a que entre los tipos de usuarios se evidencian diferencias, con respecto a sus garantías y funciones especiales, en el ámbito de la reproducción musical se deben recargar operaciones para garantizar que se cumplan las restricciones de reproducción planteadas para los tipos de usuarios especificados.
- Los tipos de usuarios determinarán la calidad de los servicios que se brindarán a estos en términos de las funciones especiales.
- Con respecto a las canciones, se debe garantizar que esta pertenezca a un único Álbum y que puede pertenecer a varios Artistas, así como que, los Álbumes pueden poseer 4 géneros musicales en su categorización. Por tanto, para evitar fallas en este proceso de organización entre Canción, Álbum y Artista se hará un mediático uso de la identificación de la Canción con sus respectivos dígitos numéricos, lo cual

permitirá verificar que la Canción cumpla los estándares propuestos, y así mismo, con los estándares de reproducción gracias al uso de la sobrecarga de los operadores lógicos de igualdad.

- Delimitar la estructura de la base de datos, y a su vez, las operaciones entre dos datos de diferentes clases(Acceder mediante una a información de la otra).
- Para los usuarios premium con Lista de Favoritos generar un balance entre sus propias canciones y las que se les adiciona o elimina según se trate la lista de favoritos de otro usuario.

De forma consecuente que, con todo esto evidenciado, se nos permite interiorizar toda clase de limitaciones a nivel organizacional que nos llevaran a un manejo y correcto desarrollo del programa, procurando que este trabaje como desea nuestro cliente, cumpliendo el estándar de uso esperado respecto a las funciones que el usuario de la aplicación espera usar y garantizando que los datos estén bien regulados y distribuidos durante la implementación de todo el Sistema sin excepción alguna, frente a los datos que recibe tal cual como los que debe arrojar como resultado.

b. Diagrama de clases de la solución planteada. Adicionalmente, describa en alto nivel la lógica de las tareas que usted definió para aquellos subprogramas cuya solución no sea trivial.

*Link:*https://app.diagrams.net/#G19ylc_W4LvGqIh0jLKcgE7Pg8tTMFeHBj#%7B%22pageId%22%3A%22gge4Ijc0jWBcZNEL-cae%22%7D

c. Algoritmos implementados debidamente intra-documentados. No exceda la intra-documentación. No use IA para generar la documentación.

Clase sistema:

Sistema: login(string nickname)

Sinopsis: Autentica un usuario en la plataforma

Entradas:

- nickname: string, nombre de usuario para iniciar sesión

Salidas:

- returns: bool, true si el login fue exitoso, false en caso contrario

Sistema: buscarUsuarioPorNickname(const string nickname)

Sinopsis: Busca un usuario por su nombre único en el sistema.

Entradas:

- nickname: string, nombre de usuario a buscar

Salidas:

- returns: Usuario*, puntero al usuario encontrado o nullptr si no existe

Sistema :buscarCancionPorId(int id)

Sinopsis: Busca una canción por su ID único de 9 dígitos

Entradas:

- id: int, identificador de la canción (9 dígitos)

Salidas:

- returns: Cancion*, puntero a la canción encontrada o nullptr si no existe

Clase usuario:

Usuario: agregarFavorito(Cancion* cancion)

Sinopsis: Agrega una canción a la lista de favoritos del usuario

Entradas:

- cancion: Cancion*, puntero a la canción a agregar

Salidas:

- returns: bool, true si se agregó correctamente, false si ya existe o lista llena

Usuario :eliminarFavorito(Cancion* cancion)

Sinopsis: Elimina una canción de la lista de favoritos

Entradas:

- cancion: Cancion*, puntero a la canción a eliminar

Salidas:

- returns: bool, true si se eliminó correctamente, false si no existía

Usuario: seguirUsuario(Usuario* usuario)

Sinopsis: Permite seguir a otro usuario premium

Entradas:

- usuario: Usuario*, puntero al usuario a seguir

Salidas:

- returns: bool, true si se agregó correctamente, false si ya lo seguía

Usuario: obtenerListaFusionada(Cancion* resultado, int* cantidad)

Sinopsis: Combina la lista propia con listas de usuarios seguidos

Entradas:

- resultado: Cancion*, arreglo de salida para canciones fusionadas

- cantidad: int*, tamaño del arreglo de salida

Salidas:

- void, el resultado se devuelve por parámetros

Clase álbum

Album: agregarCancion(Cancion* cancion)

Sinopsis: Agrega una canción al álbum

Entradas:

- cancion: Cancion*, puntero a la canción a agregar

Salidas:

- returns: bool, true si se agregó correctamente, false si ya existe

Album: buscarCancionPorId(int idCancion)

Sinopsis: Busca una canción dentro del álbum por su ID

Entradas:

- idCancion: int, ID de la canción a buscar

Salidas:

- returns: Cancion*, puntero a la canción encontrada o nullptr

Album: agregarGenero(string genero)

Sinopsis: Agrega un género musical a la lista de géneros del álbum

Entradas:

- genero: string, género musical a agregar

Salidas:

- returns: bool, true si se agregó, false si ya existe o máximo alcanzado

Clase canción:

Cancion: reproducir()

Sinopsis: Incrementa el contador de reproducciones de la canción

Entradas:

- ninguna

Salidas:

- void

Cancion: obtenerRutaAudio(int calidad)

Sinopsis: Genera la ruta del archivo de audio según la calidad

Entradas:

- calidad: int, 128 para estándar, 320 para alta calidad

Salidas:

- returns: string, ruta completa al archivo de audio

Cancion: descomponerId(int* idArtista, int* idAlbum, int* idCancion)

Sinopsis: Descompone el ID de 9 dígitos en sus componentes

Entradas:

- idArtista: int*, salida para ID de artista (5 dígitos)

- idAlbum: int*, salida para ID de álbum (2 dígitos)

- idCancion: int*, salida para ID de canción (2 dígitos)

Salidas:

- void, los resultados se devuelven por parámetros

Clase sesion reproducción:

SesionReproduccion: iniciar()

Sinopsis: Inicia la reproducción de música

Entradas:

- ninguna

Salidas:

- void

SesionReproduccion: siguiente()

Sinopsis: Avanza a la siguiente canción en la lista de reproducción

Entradas:

- ninguna

Salidas:

- returns: bool, true si había siguiente canción, false si era la última

SesionReproduccion: establecerFuenteAleatoria(FuenteAleatoria* fuente)

Sinopsis: Establece la fuente de canciones aleatorias

Entradas:

- fuente: FuenteAleatoria*, puntero a la fuente de canciones aleatorias

Salidas:

- void

Clase fuente aleatoria:

FuenteAleatoria: siguienteCancion()

Sinopsis: Obtiene la siguiente canción de manera aleatoria

Entradas:

- ninguna

Salidas:

- returns: Cancion*, puntero a la siguiente canción aleatoria

Clase medidor de recursos:

Sinopsis: Obtiene el número total de iteraciones realizadas

Entradas:

- ninguna

Salidas:

- returns: int, total de iteraciones acumuladas

d. Problemas de desarrollo que afrontó.

Durante el proceso de plantear la estrategia de implementación, logramos evidenciar diversas formas de llevar a cabo las operaciones en el algoritmo que permitiera concretar la solución y facilitar los medios de establecer parámetros, o bien ya sea variables, que entrelazan y/o modifican los datos cargados en las estructuras cargadas en el Sistema (Arreglos Lineales Dinámicos y Estáticos), pero a su vez esto nos llevó a evidenciar muchas inconsistencias a satisfacer con respecto a los requisitos que implica la aplicación final que se nos solicita entregar, entre las cuales tenemos:

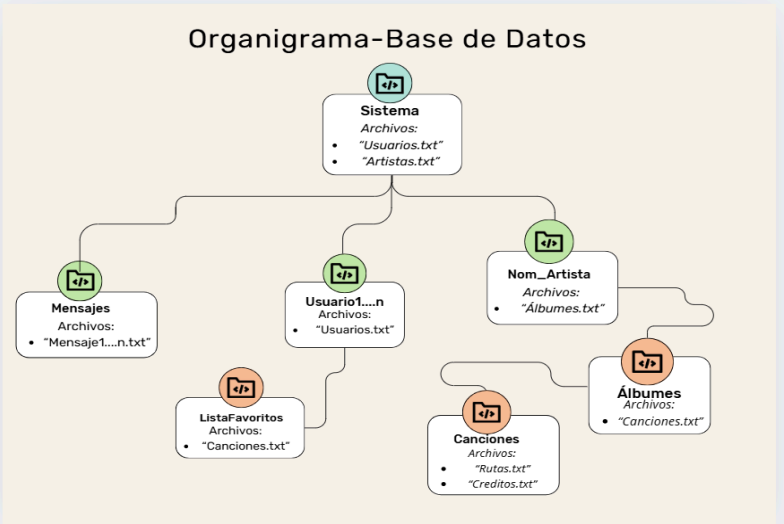
- La suma decisión de cómo usar la información cargada en el sistema, de manera tal que acceder a esta no denote una sustancial copia innecesaria de la información, pérdida, o inclusive, un aumento operacional que ralentice los procesos que solicita realizar el usuario durante el uso de la Aplicación.

- La conexión entre las listas de Favoritos de los Usuarios Premium puesto que esto significa un efecto tipo aliasing entre la lista de Favoritos del seguido con la información que se le agrega a la Lista de Favoritos del Usuario Seguidor.
- La definición de la base de datos permanente de forma que esta permita conservar ordenadamente la información de los Usuarios, Listas de Favoritos, Artistas, Álbumes, Mensajes, etc.
- La caracterización de las relaciones entre las clases, de tal manera que nos tocó tomar diferentes decisiones respecto a la interacción entre estas, y así mismo, los medios de trabajar sobre cada una de ellas respecto al tipo de interacción planteado.
- La optimización de situaciones en que por desgracia las iteraciones y la eficiencia operacional, en conjunto a la de memoria, acrecentaban ser excesivas.
- El establecimiento de una lógica que permitiera llevar a asimilaciones entre los códigos de las canciones, álbumes y por supuesto, el artista.
- Los campos de acción entre los cuales podíamos introducir operaciones por medio de las funciones amigas.
- Establecer la eliminación de memoria dinámica en escenarios óptimos.

e. Evolución de la solución y consideraciones para tener en cuenta en la implementación.

Ya luego de tener claras todas las ideas a plasmar para relacionar cada una de las operaciones del sistema, organizar correctamente el sistema e identificar las clases a usar como bases de toda la información a manejar, y tener presentes las diversas problemáticas que conflictuában a la hora de establecer estructuras diferentes entre cada clase; hemos logrado establecer como solución y consideraciones de la implementación lo siguiente:

1. La estructura de datos la estableceremos de forma que toda la información se encuentre contenida en una sola carpeta con subcarpetas referentes a la respectiva información que debemos guardar en el sistema sobre cada objeto declarado, instanciado o presente en el sistema. Esto nos permitirá llevar a cabo una correcta organización y diferenciación respecto a los datos, en conjunto a una mejor presentación de todo el sistema, de manera tal, que hará posible controlar de forma monitoreada el crecimiento de la información en cada sección, sin esta llegar a afectar los otros datos del sistema.



2. Plantear la lectura y escritura de los archivos a partir de sus rutas, puesto que cada archivo corresponderá a una subcarpeta de otra de orden superior, hasta llegar a la

carpeta madre, la cual será la que modelará en general cómo se compone todo el Sistema o la estructura de una sección en específico.

3. La información será cargada al Sistema en su ejecución en diversas estructuras de datos (Arreglos) de tipo Objeto o punteros, lo que nos permitirá tanto por una parte agilizar la búsqueda de los elementos, como también, una mayor eficacia operacional.
4. Para dar un correcto uso del modelo de Programación Orientada a Objetos, las operaciones de despliegue cumplirán el papel de abarcar la mayor parte de las interacciones del Usuario con el Sistema.
5. Implementar la sobrecarga de métodos u operadores para delimitar el campo de acción de los métodos sobre las estructuras cargadas en el sistema.
6. Abarcar las funciones amigas para la construcción de un modelo de control de la eficiencia e iteradores de operaciones en el sistema.
7. Extrapolar el caso en que el usuario que se loguee en el Sistema no esté registrado, considerando el acto de permitirle gozar de las primitivas básicas del sistema, o simplemente, no darle ingreso.