INFORME DEL DESAFIO 2

JEAN CARLOS PARRA SERRANO

UNIVERSIDAD DE ANTIOQUIA

INFORMÁTICA 2

OCTUBRE DEL 2025

MEDELLÍN, ANTIOQUIA

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

El desafío 2 nos plantea una aplicación de los conceptos dados a una situación real de nuestro contexto. Es bien sabido que las plataformas de música en streaming son famosas por acercar a las personas amantes de la música a un entorno donde se pueda disfrutar del arte que es la música; escuchando sus canciones favoritas, sus álbumes y demás. El objetivo en este desafío es realizar una plataforma llamada UdeATunes que muestre en consola algunas de las funcionalidades planteadas en el documento son las siguientes:

* Carga / actualización de datos.
* Ingreso a la plataforma
* Reproducción aleatoria
* Mi lista de favoritos
* Editar mi lista de favoritos
* Seguir otra lista de favoritos
* Ejecutar mi lista de favoritos
* Medición del consumo de recursos

Lo anterior indica que el sistema permitirá a los usuarios iniciar sesión, reproducir canciones de manera aleatoria o desde sus listas de favoritos, mostrando las rutas completas de los archivos de audio y portadas según su tipo de membresía (estándar o premium). Los usuarios estándar verán publicidad cada dos canciones, seleccionada de forma aleatoria y ponderada por prioridad, mientras que los usuarios premium accederán a audio en alta calidad, sin anuncios y con funciones adicionales como seguir listas de otros usuarios y reproducir con control de historial. La aplicación registrará y medirá en cada ejecución el consumo de memoria y el número de iteraciones realizadas, garantizando un diseño eficiente, modular y sin duplicación de datos en el almacenamiento.

Para llegar a la solución de este desafío, se debe de aclarar la lectura de archivos:

Anuncio.txt

Archivo que contiene la categoría del mensaje y el anuncio. Tiene la siguiente estructura *categoría,anuncio.*

Usuario.txt

Archivo que contiene información sobre el usuario y la lista a la que sigue(si es premium). Tiene la siguiente estructura *nickname, membresía, ciudad, país, fecha de inscripción, lista a la que sigue en favoritos(solo si es premium).*

Artista.txt

Archivo que contiene información sobre el artista. Tiene la siguiente estructura *nickname del artista, id\_artista, edad, país, cantidad de seguidores, ranking global, catálogo de álbumes.*

Álbum.txt

Archivo que contiene información sobre los álbumes. Tiene la siguiente estructura *name del álbum, id\_album, duración total, fecha de lanzamiento, ruta de portada, géneros del álbum, puntuación del álbum, sello disquero.*

Canciones.txt

Archivo que contiene información sobre las canciones. Tiene la siguiente estructura *name de la canción, id\_cancion, duración, ruta\_128 de la canción, ruta\_320 de la canción, créditos, reproducciones.*

Listafavoritos.txt

Archivo que contiene información sobre la lista de favoritos. Tiene la siguiente estructura:

*name del álbum, id\_canciones separadas por “;”.*

Cada identidad (usuario, canción, álbum, artista, lista de favoritos y anuncio) se modela mediante una clase. Las relaciones entre ellas se realizan a través de punteros para evitar replicación de datos. Por ejemplo, una canción conoce el ID de su álbum y su artista, y el usuario mantiene punteros a su lista de favoritos o a la lista seguida.

Diagrama de clases Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.

Clases:

Anuncio: clase que contendrá los atributos que son de tipo string para la categoría del anuncio y el anuncio. Nos ayuda a obtener los datos que trabajaremos con ellos. Se hace la carga de estos datos.

Album: clase que contendrá los atributos del archivo álbum.txt, su carga de datos y sus implementaciones, métodos, para obtener los valores.

Artista: clase que contendrá los atributos del archivo artista.txt, su carga de datos y sus implementaciones, métodos para obtener los valores.

Cancion: clase que contendrá los atributos del archivo cancion.txt, su carga de datos y sus implementaciones, métodos para obtener los valores.

Lista de favoritos: clase que contendrá los atributos del archivo listafavoritos.txt y sus métodos básicos, donde enlazamos con punteros a canciones para evitar replicaciones.

Existe un login y un menú:

Se logea el usuario, obteniendo su membresía y dependiendo de ella se le permitirá las opciones(menú).

Para evitar la replicación de datos en lista de favoritos, se plantearon punteros que señalen a las canciones en sus posiciones y a partir de ahí hacer las operaciones que el usuario requiera hacer.

Además, se usan librerías como chrono y random para tiempos y generar números aleatorios respectivamente.

Se crean .h para el conteo de iteraciones y las reservas requeridas, donde a partir de plantillas, se hacen estos conteos. Por otro lado, está el .h de las liberaciones de memoria para cada tipo de estructura que se implementó.

Cabe aclarar que en utilidades se hacen las funciones que nos servirán para trabajar sobre estas características para la obtención de datos y su manejo. En utilidades se hace el manejo de validaciones, búsqueda, reproducción, flujo de login y los menús.

**Algunas de las relaciones importantes:**

* usuario —(asociación)→ lista\_favoritos (propia) y —(asociación)→ lista\_favoritos (seguida).
* lista\_favoritos contiene arreglo de punteros a cancion para evitar copia.
* cancion se asocia con un album y un artista por derivación del id (9 dígitos).
* anuncio se usa en reproducción estándar.

Las estructuras de datos utilizadas especialmente para eficiencia:

Arreglos de punteros (\*\*) para colecciones; tamaño ajustado por conteo previo de líneas.

Sin duplicación: las listas guardan punteros a canciones cargadas en memoria.

Fusión de favoritos: arreglo temporal sin duplicados por id, tomamos los punteros a la lista a la lista de favoritos del usuario.

Historial premium:

Aleatorio general y de favoritos usan estructuras tipo pila (máx 6) de punteros.

Medición: plantillas registrarMemoria y liberarMemoria y contador de iteraciones global.

**Funcionalidades implementadas:**

Ingreso y menús

* Búsqueda de usuario por nickname y salida a menú estándar/premium dependiendo del tipo de usuario.

Reproducción aleatoria

* Estándar: muestra 5 canciones con temporizador de 3s, luego permite siguiente o salir; inserta anuncio cada 2 canciones con prioridad 1:2:3 y sin repetición consecutiva, siempre el audio en 128 kbps.
* Premium: controles [N] siguiente, [P] previa (hasta 6), [R] repetir, [S] salir; evita elegir del historial reciente; siempre audio 320 kbps.

Mi lista de favoritos (premium)

* Editar: agregar/eliminar por id; se guardan los cambios en listafavoritos.txt.
* Seguir: enlaza por puntero la lista del seguido; se guarda en usuarios.txt si se agrega a una persona o se elimina.
* Ejecutar:
  + Secuencial: controles N/P/R/S; combinación de listas si se sigue otra lista, sin duplicados.
  + Aleatoria: si existe seguido se combina a mi lista; historial circular máx 6; evita repetir recientes.

*Algunos algoritmos utilizados*

* Selección de anuncios (estándar).  
   Se genera un entero r entre 1 y 6.
* R es 1 → C, r está entre 2 y 3 → B, r está entre 4,5 y 6 → AAA.  
  Se verifica no repetir el mismo anuncio consecutivamente. Si la categoría elegida no tiene tipo, se dirige hacia las otras disponibles manteniendo la regla anti-repetición.
* Fusión de listas (seguir).  
  Dadas Lista de favoritos propia y Lista del usuario a seguir, se recorre en orden y se insertan en un buffer temporal punteros a canciones, evitando duplicados por id. La ejecución usa este arreglo fusionado sin guardar copias para hacer el recorrido secuencial y aleatorio.
* Historial premium (con controles).
* Siguiente: escoge aleatoria evitando los *k* más recientes del historial.
* Previo: navega hacia atrás en el anillo.
* Repetir: vuelve a reproducir el actual sin alterar el historial.
* Capacidad fija (=6) para acotar memoria y simplificar límites.

*Dificultades en el desarrollo*

Muchas de las dificultades fueron en la reproducción, la estándar tiene su complejidad al saber cómo se debe de mostrar los anuncios. La solución a esta fue gracias a la probabilidad del dado, simplemente generamos un numero entre 1 y 6, si es 1 es de tipo C, si es de 2 o 3, es de tipo B y si es 4, 5 o 6 es de tipo C. Luego se buscaba en los anuncios según lo que diera el número. La reproducción de los premium también tuvo su dificultad, puesto que estos tenían más opciones, se crearon “buffers” en anillo que guardaban las últimas canciones para después mostrarlas en caso de ser necesario.

Las búsquedas y validaciones fueron sencillas sabiendo los gets de cada clase, y con los sets cambiábamos los valores necesarios. En el registro de memoria, se buscó crear una plantilla para hacer la cuenta de todas estas iteraciones.

Siempre es bueno resaltar en qué se debe de mejorar. Siento que se podría de reducir la cantidad de líneas de código y la lógica de este si se sabe analizar detalladamente el desafío. Queda aplicar esto a futuras implementaciones.

*Conclusión:*

Este desafío permitió profundizar el manejo de estructuras dinámicas y relaciones entre clases sin duplicar datos o replicación de datos para manejar eficiencia. Implementar el seguimiento de listas mediante punteros garantizó eficiencia y consistencia. Una posible mejora futura sería incluir los siguientes prototipos para la reproducción de las listas, hacer métodos más precisos para la simplificación del código.