Universidad de Antioquia
Departamento de Electrónica y Telecomunicaciones
2598521 - Informatica II

Desafio 2 - UdeATunes

Informe de Análisis y Diseño Inicial

Rigoberto Berrio Berrio
Estudiante de Ingeniería Electrónica
Universidad de Antioquia
rigoberto.berrio1@udea.edu.co

Jeisson Stevens Martinez Arevalo
Estudiante de Ingeniería en Telecomunicaciones
Universidad de Antioquia
jeisson.martinez1@udea.edu.co

28 de octubre de 2025

Índice

1. Resumen ejecutivo

Este documento presenta el informe final y detallado del prototipo **UdeATunes** (Desafío 2). El sistema modela un servicio de streaming musical en consola implementado en C++ y organizado en módulos: entidades de dominio (usuarios, artistas, álbumes, canciones), servicios funcionales (reproductor, gestor de anuncios, métricas) y utilidades de soporte (estructuras dinámicas, generador aleatorio, persistencia).

El informe cubre: análisis del problema y requisitos; diseño orientado a objetos; especificaciones de persistencia; descripción de algoritmos y estructuras implementadas; verificación del cumplimiento de los requisitos del enunciado; pruebas sugeridas y resultados de ejemplo; limitaciones encontradas y recomendaciones para la entrega final.

2. Contexto, alcance y objetivos

2.1. Contexto

Las plataformas de streaming dominan el consumo musical moderno. UdeATunes es un ejercicio académico que reproduce, a escala prototipo, las funciones esenciales de una plataforma: catálogo, reproducción, publicidad segmentada según tipo de usuario y métricas de rendimiento.

2.2. Alcance

El prototipo 1.0 implementado cubre:

- Representación en memoria de usuarios (estándar/premium), artistas, álbumes y canciones.
- Reproductor con modos aleatorio, siguiente, previa y repetir; temporizador de 3 segundos para pruebas; reglas de retroceso según tipo de membresía.
- Gestor de anuncios con priorización por categoría (C, B, AAA) y selección proporcional al peso, evitando repetición inmediata.
- Listas de favoritos para usuarios premium (límite configurable hasta 10000) y la posibilidad de seguir listas de otros usuarios premium.

- Persistencia básica mediante un módulo de almacenamiento (FileStore) y medición de métricas (iteraciones y estimación de memoria viva).

2.3. Objetivos

1. Diseñar una arquitectura POO modular y extensible.
2. Implementar las funcionalidades solicitadas por el enunciado con énfasis en eficiencia y trazabilidad.
3. Documentar formatos de persistencia y métricas para la sustentación.
4. Proveer un conjunto de pruebas y un plan de trabajo para completar la entrega final.

3. Análisis del problema y requisitos

3.1. Requisitos funcionales relevantes

Se resumen los requisitos clave que la implementación debe satisfacer:

1. Gestión completa de usuarios: nickname único, tipo de membresía, ciudad, país, fecha de inscripción.
2. Usuarios premium: reproducción sin anuncios, audio 320 kbps, lista de favoritos (hasta 10000), seguir listas de otros premium y retroceder hasta 4 canciones; usuarios estándar: audio 128 kbps y anuncios cada dos canciones.
3. Artistas, álbumes y canciones con metadatos completos; cada canción contiene rutas absolutas a dos archivos `.ogg` (128 y 320).
4. Identificador de canción: entero de 9 dígitos (5 dígitos artista, 2 dígitos álbum, 2 dígitos canción).
5. Créditos por canción divididos en Productores, Músicos y Compositores (cada persona con código SAYCO de 10 caracteres).
6. Publicidad: hasta 50 anuncios, con categorías C (peso 1), B (peso 2), AAA (peso 3); no mostrar mismo anuncio de forma consecutiva.
7. Persistencia: carga y actualización de datos desde almacenamiento permanente.

8. Métricas: contabilización de iteraciones y medición de memoria viva al finalizar cada funcionalidad.

3.2. Requisitos no funcionales

- Implementación en C++ (estructura actual en carpeta `Proyecto_creo`).
- Portabilidad preferente a Linux; medición real de memoria opcional con `/proc/self/statm`.
- Eficiencia: elección de estructuras que optimicen búsquedas y operaciones frecuentes.
- Documentación completa: diagrama de clases, formatos de persistencia, definición exacta de métricas.

4. Diseño orientado a objetos

4.1. Visión general

La arquitectura propuesta se organiza en tres capas conceptuales:

- **Capa de dominio**: clases que modelan el negocio (Usuario, Artista, Album, Cancion, Creditos, Persona).
- **Capa funcional**: servicios que realizan operaciones activas (Reproductor, GestorAnuncios, Metricas, FileStore).
- **Capa de soporte**: utilidades y estructuras (DynArray, Ring, RandomLCG, StringUtil).

4.2. Diagrama de clases (vista de alto nivel)

4.3. Descripción de responsabilidades

Usuario. Mantiene datos de cuenta, estado de membresía y referencias a su lista de favoritos y a las listas que sigue. Expone operaciones para consultar privilegios y modificar su lista de favoritos.

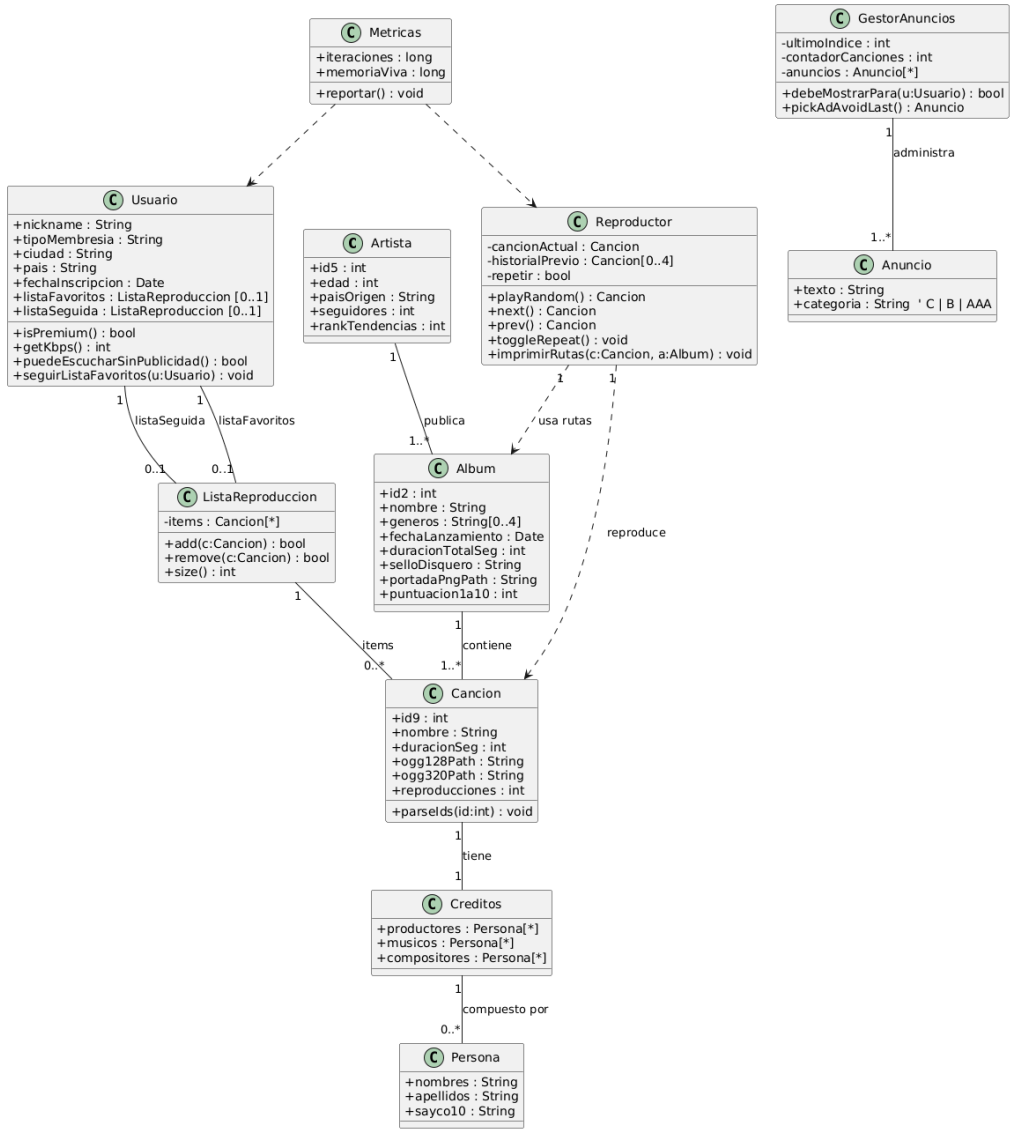

Figura 1: Diagrama conceptual de clases del sistema UdeATunes.

FavoriteList. Estructura que almacena identificadores de canciones sin duplicados. Soporta operaciones de inserción, eliminación y consulta. Implementada para operar en tiempo eficiente en las búsquedas de duplicados (uso de `DynArray` con lógica para evitar duplicados).

Canción, Álbum y Artista. Modelan la jerarquía musical. `Canción` registra rutas a ambos archivos de audio (`_128.ogg` y `_320.ogg`) y contiene la lógica para parsear su id de 9 dígitos en artista/álbum/pista.

GestorAnuncios. Administra la cola de anuncios y su selección ponderada evitando la repetición inmediata del último anuncio mostrado.

Reproductor. Implementa la lógica central de reproducción: selección aleatoria de canciones, opciones de control (siguiente, previa, repetir), interacción con gestor de anuncios y registro en métricas (iteraciones y memoria).

5. Especificaciones de persistencia

5.1. Recomendación general

Para la entrega y facilidad de auditoría se recomienda usar **JSON** como formato de persistencia durante la fase de pruebas y sustentación. Si en el código actual existe un formato propietario, proveer adaptadores que conviertan entre JSON y el formato interno.

5.2. Convenciones importantes

- Rutas absolutas en notación Linux para archivos de audio y portadas (escapar espacios según sea necesario al usarse en consola).
- Los identificadores (IDs) deben mantenerse consistentes: artista (5 dígitos), álbum (2 dígitos), pista (2 dígitos), concatenados para formar el id9.
- Límite de anuncios: máximo 50. Límite de favoritos para premium: hasta 10000.

6. Algoritmos y políticas de diseño

6.1. Selección de anuncios (política)

Se asigna peso por categoría: C=1, B=2, AAA=3. La selección se realiza de forma proporcional al peso, excluyendo siempre el anuncio mostrado en la reproducción previa para evitar repetición inmediata. Esta política cumple el requisito de no mostrar el mismo mensaje de forma consecutiva y mantiene diversidad.

6.2. Reproducción aleatoria (reglas)

1. Selección verdaderamente aleatoria entre todas las canciones disponibles (uso de `RandomLCG` para reproducibilidad en pruebas).
2. Para usuarios estándar: cada dos canciones reproducidas se debe mostrar un anuncio (después de la segunda, cuarta, etc.). No se permite patrón fijo de anuncios porque la selección es aleatoria ponderada (con exclusión del último).
3. Para usuarios premium: no se muestran anuncios; se habilitan controles adicionales (previa hasta 4 canciones, repetir).
4. Temporizador de demostración: 3 segundos por canción. En pruebas, detener después de K canciones (K = 5).

6.3. Métricas: definición y contabilización

Iteraciones. Se define como unidad contable la ejecución de pasos lógicos significativos: comparaciones en búsquedas lineales, pasos de bucles que recorren colecciones relevantes, inserciones/eliminaciones en estructuras propietarias (contadas como 1, más 1 adicional si se produce realocación). Esta definición garantiza replicabilidad entre ejecuciones.

Memoria viva (estimación programática). Se estima sumando:

- `sizeof` de objetos estáticos y estructuras.
- Memoria dinámica reservada por contenedores (`capacity * sizeof(element)` para cada `DynArray`).
- Tamaño de buffers activos y variables locales relevantes al momento del muestreo.

Opcionalmente, se incluye un método que consulta `/proc/self/statm` para obtener el uso real en sistemas Linux para propósitos de validación.

7. Implementación: observaciones sobre el código entregado

Tras la revisión del paquete `Prueba_D_II/Proyecto_creo`, se identificaron los siguientes módulos implementados y observaciones prácticas:

7.1. Módulos principales detectados

- `Player` (reproductor): controla reproducción, historial y relación con anuncios y métricas.
- `AdManager` / `Advertisement`: gestión de anuncios y selección ponderada.
- `Artist`, `Album`, `Song`: entidades musicales con metadatos y rutas.
- `User`, `FavoriteList`: gestión de usuarios y favoritos.
- `FileStore`: persistencia (lectura/escritura de datos).
- `MemoryTracker`, `Iteration`: contabilización de métricas.
- `DynArray`, `Ring`, `RandomLCG`, `StringUtil`: utilidades y estructuras de soporte.

7.2. Observaciones de calidad

- El código está modular y respeta separación de responsabilidades, facilitando pruebas por módulo.
- Se emplean estructuras propias en vez de STL, lo cual es coherente con el objetivo académico (manejo manual de memoria).
- Falta documentar de forma formal el formato de persistencia usado en el proyecto; se recomienda añadir adaptadores JSON.
- Se sugiere desacoplar la lógica de UI (menús de consola) del motor del reproductor para permitir tests unitarios automáticos.

8. Pruebas, dataset sugerido y validación

8.1. Dataset mínimo recomendado para la sustentación

- 3 artistas con 4 álbumes en total.
- 12–20 canciones con rutas simuladas a archivos `ogg` (128 y 320).
- 8–12 anuncios con mezcla de categorías C/B/AAA.
- 3 usuarios (2 premium, 1 estándar), con listas de favoritos y relaciones de seguimiento.

8.2. Casos de prueba esenciales

1. Inicio de sesión y carga de datos desde persistencia.
2. Reproducción aleatoria por usuario premium: verificar ausencia de anuncios, kbps 320, posibilidad de retroceder hasta 4 canciones.
3. Reproducción aleatoria por usuario estándar: verificar anuncio cada 2 canciones y diversidad en selección de anuncios.
4. Edición de lista de favoritos: agregar y eliminar sin duplicados; límite de 10000 validado.
5. Seguir lista de otro usuario premium: consolidación de listas y reproducción continua.
6. Medición de métricas: ejecutar cada funcionalidad y registrar iteraciones y memoria viva.

8.3. Validación estadística sugerida

Para la selección de anuncios, ejecutar la selección ponderada un gran número de veces (por ejemplo 10 000) y verificar que las frecuencias empíricas se aproximen a las proporciones de peso (AAA 3x C, B 2x C).

9. Archivos .txt

Motivo	Descripción
Funcionalidad práctica	Permiten cargar datos reales (canciones, usuarios, listas) sin necesidad de una base de datos, facilitando la lectura y escritura directa desde archivos simples.
Modularidad del sistema	Cada módulo utiliza los archivos `.txt` de forma coherente: `Song` los interpreta, `User` los asigna y `StringUtil` los procesa, manteniendo una arquitectura clara y separada.
Propósito educativo	Sirven para probar estructuras dinámicas como `Ring` o `DynArray` sin necesidad de interfaces gráficas ni bases de datos complejas, lo que favorece la comprensión del flujo lógico.
Simulación de persistencia	Los datos permanecen entre ejecuciones del programa, simulando el comportamiento de persistencia de una aplicación musical real.

Cuadro 1: Razones de elección de los archivos .txt en la implementación del proyecto.

10. Conclusiones

El prototipo UdeATunes constituye una base sólida que satisface los requisitos principales del enunciado académico. La estructura modular, el uso de POO y las utilidades implementadas permiten cubrir funcionalidades críticas de una plataforma de streaming en formato de consola.

Para garantizar una entrega exitosa se recomienda completar la documentación de persistencia, añadir pruebas automatizadas y generar el dataset de demostración; una vez completadas estas tareas, el proyecto estará listo para la sustentación.