Universidad de Antioquia
Departamento de Ingeniería Electrónica
Informática II

Informe Preliminar

UdeATunes - Sistema de Streaming Musical

Desafío II: Análisis y Diseño de la Solución

Diego Andrés Páez Mendoza
Jesús Alberto Cordoba Delgado

18 de octubre de 2025

Índice

1. Introducción **3**

2. Análisis del Problema **3**

3. Arquitectura Propuesta **4**

4. Estructuras de Datos **6**

5. Diagrama de Clases **7**

6. Flujo de Ejecución **8**

7. Persistencia de Datos **10**

8. Sistema de Reproducción **11**

9. Medición de Recursos **11**

10. Consideraciones de Implementación **12**

11. Plan de Desarrollo **13**

12. Estrategia de Pruebas **14**

13. Conclusiones **14**

1. Introducción

El presente documento describe el proceso de análisis y diseño desarrollado por el equipo para la implementación de UdeATunes, un sistema de streaming musical que emula el funcionamiento de plataformas comerciales conocidas. El proyecto representa una oportunidad para aplicar los conceptos de Programación Orientada a Objetos mediante el desarrollo de un sistema complejo que gestiona usuarios, contenido musical y listas de reproducción.

El enfoque adoptado por el equipo prioriza la eficiencia en el manejo de datos y la claridad en el diseño arquitectónico. Se ha realizado un análisis exhaustivo del problema planteado, identificando las entidades principales del dominio y las relaciones que existen entre ellas. A partir de este análisis, se propone una arquitectura modular que facilita tanto el desarrollo como el mantenimiento futuro del sistema.

Este informe preliminar presenta los resultados de la fase de análisis y diseño, incluyendo el diagrama de clases completo, la arquitectura general del sistema, las estructuras de datos seleccionadas y las consideraciones de implementación. El documento está organizado para guiar al lector desde la comprensión del problema hasta las decisiones técnicas que fundamentan la propuesta de solución.

2. Análisis del Problema

2.1. Contexto y Alcance

La industria del streaming musical ha transformado la manera en que las personas acceden y consumen contenido musical. UdeATunes busca replicar las funcionalidades básicas de estas plataformas, ofreciendo dos modelos de membresía diferenciados por sus características y beneficios.

El sistema debe gestionar un catálogo musical organizado jerárquicamente: cada artista posee múltiples álbumes, y cada álbum contiene varias canciones. Los usuarios interactúan con este catálogo mediante funcionalidades de reproducción y gestión de favoritos. La arquitectura propuesta debe soportar eficientemente operaciones de búsqueda, inserción y actualización sobre colecciones potencialmente grandes de datos.

El equipo identificó desde el inicio las restricciones técnicas del proyecto. No se permite el uso de la librería estándar de plantillas, con excepción de la clase string. Todas las estructuras de datos deben implementarse manualmente utilizando memoria dinámica. Esta restricción, lejos de ser una limitación, representa una oportunidad para comprender a profundidad los algoritmos fundamentales y optimizar el rendimiento del sistema.

2.2. Entidades del Dominio

Durante la fase de análisis se identificaron las siguientes entidades principales que conforman el modelo de datos:

Los usuarios representan a las personas registradas en la plataforma. Cada usuario posee un identificador único (nickname), información de ubicación geográfica y un tipo de membresía que determina sus privilegios. Los usuarios premium mantienen adicionalmente una lista personalizada de canciones favoritas con capacidad para almacenar hasta diez mil elementos.

Los artistas constituyen la fuente del contenido musical. Se almacena información relevante como edad, país de origen, cantidad de seguidores y posición en tendencias. Cada artista administra su propio catálogo de álbumes mediante una colección dinámica que permite agregar nuevos lanzamientos.

Los álbumes agrupan canciones relacionadas bajo una producción musical cohesiva. Contienen metadatos como fecha de lanzamiento, géneros asociados, sello disquero y portada. El sistema calcula automáticamente la duración total sumando las duraciones individuales de cada canción.

Las canciones representan la unidad básica de contenido reproducible. Cada canción almacena su nombre, duración, ubicación de archivos de audio en dos calidades diferentes y los créditos completos de los colaboradores. El identificador de nueve dígitos permite ubicar rápidamente cualquier canción conociendo su artista y álbum.

Los créditos documentan la participación de cada colaborador en la producción musical. Se organizan en tres categorías: productores, músicos y compositores. Esta información será utilizada en versiones futuras para calcular regalías según las reproducciones.

Los mensajes publicitarios constituyen el contenido promocional mostrado a usuarios estándar. Cada mensaje pertenece a una categoría que determina su probabilidad de visualización, implementando un sistema de priorización que favorece mensajes de mayor relevancia comercial.

2.3. Relaciones Identificadas

El análisis reveló relaciones de composición y agregación entre las entidades. Un artista posee sus álbumes mediante una relación de composición: cuando se elimina un artista, todos sus álbumes se eliminan también. De manera similar, un álbum posee sus canciones.

Los usuarios premium mantienen una relación de agregación con las canciones en su lista de favoritos. La lista almacena referencias a canciones que existen independientemente, por lo que eliminar un usuario no afecta el catálogo musical. Esta distinción entre composición y agregación resulta fundamental para la correcta gestión de memoria.

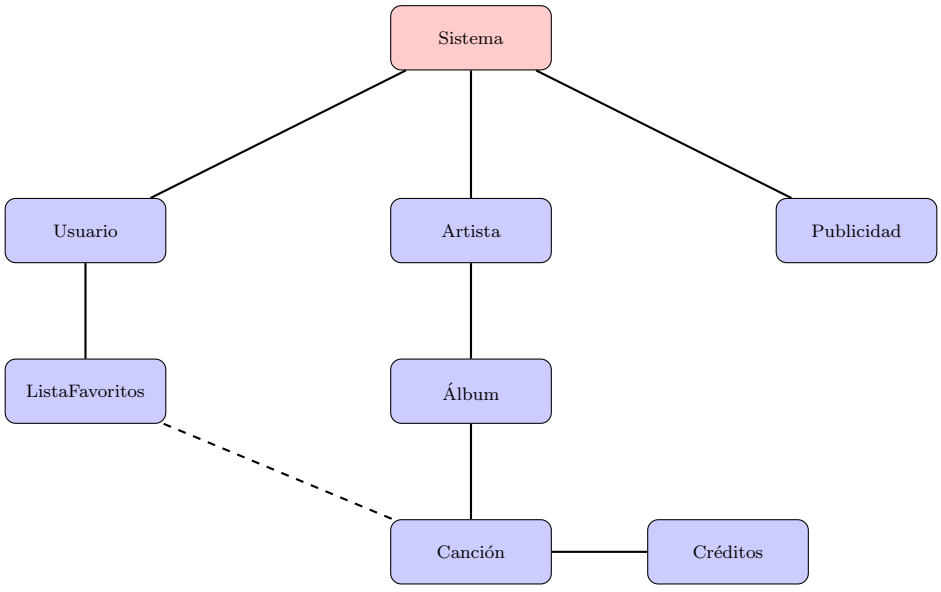

Figura 1: Relaciones principales entre entidades del sistema

3. Arquitectura Propuesta

3.1. Visión General

La arquitectura diseñada por el equipo se organiza en capas conceptuales que separan responsabilidades. En el nivel superior se encuentra el módulo Sistema que actúa como orquestador, inicializando componentes

y coordinando el flujo de ejecución.

La capa de gestión contiene módulos especializados que administran colecciones homogéneas de objetos. Cada gestor encapsula la lógica específica de su dominio, ofreciendo interfaces claras para operaciones como búsqueda, inserción y eliminación. Esta separación permite que los cambios en la implementación interna de un gestor no afecten a otros componentes.

La capa de modelo agrupa las clases que representan entidades del negocio. Estas clases son principalmente contenedores de datos con métodos de acceso controlado. No contienen lógica compleja de negocio, delegando las operaciones complejas a los gestores correspondientes.

La capa de infraestructura proporciona estructuras de datos reutilizables implementadas desde cero. Estas estructuras se diseñaron como plantillas genéricas que pueden almacenar cualquier tipo de dato, promoviendo la reutilización de código y reduciendo duplicación.

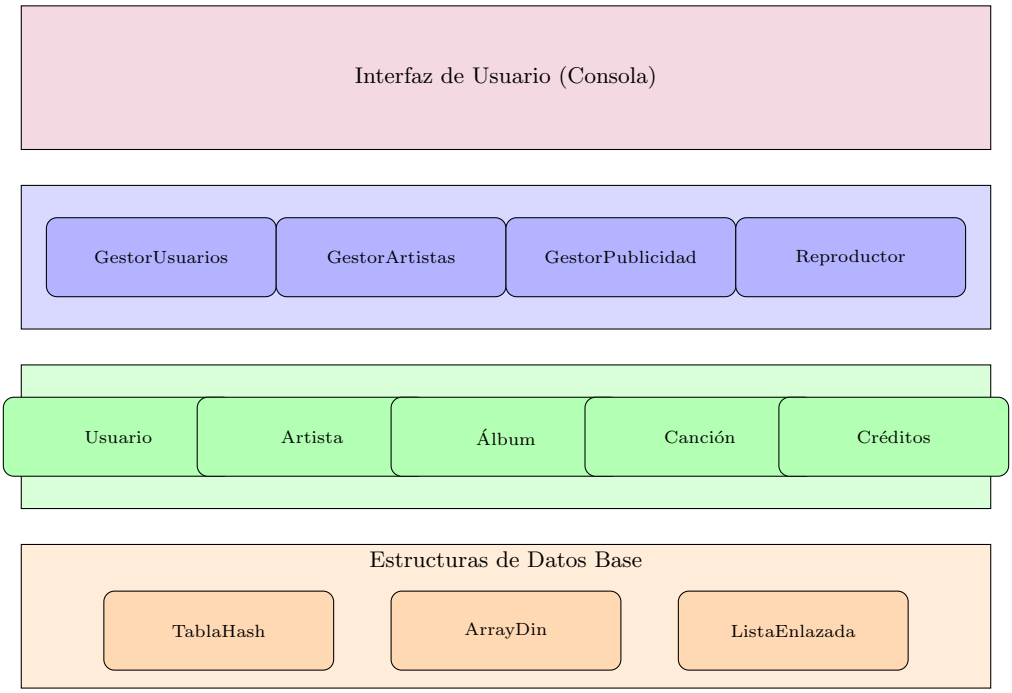

Figura 2: Arquitectura en capas del sistema UdeATunes

3.2. Componentes Principales

El módulo Sistema coordina la inicialización y el ciclo de vida de todos los componentes. Al arrancar, carga los datos desde archivos de persistencia, crea instancias de los gestores e inicializa las estructuras necesarias. Presenta el menú principal y delega las operaciones solicitadas a los gestores apropiados.

GestorUsuarios administra la colección completa de usuarios mediante una tabla hash indexada por nickname. Proporciona métodos para validar credenciales durante el inicio de sesión y buscar usuarios específicos cuando un usuario premium desea seguir listas de favoritos. Coordina la persistencia de información de usuarios.

GestorArtistas mantiene el catálogo musical completo. Utiliza una tabla hash de artistas indexada por su código identificador. Implementa búsquedas eficientes de canciones mediante el identificador de nueve dígitos, descomponiendo este en artista, álbum y canción. Soporta la selección aleatoria necesaria para la

reproducción.

GestorPublicidad controla los mensajes promocionales y su visualización. Calcula pesos acumulados según categorías y selecciona mensajes mediante un algoritmo probabilístico. Mantiene registro del último mensaje mostrado para prevenir repeticiones consecutivas.

Reproductor gestiona toda la lógica de reproducción musical. Mantiene una lista enlazada con el historial de canciones reproducidas, permitiendo navegación hacia adelante y hacia atrás. Coordina la interacción con GestorPublicidad para usuarios estándar e implementa controles específicos para usuarios premium.

4. Estructuras de Datos

4.1. Selección y Justificación

La elección de estructuras de datos apropiadas representa una decisión crítica que impacta directamente en la eficiencia del sistema. El equipo evaluó cuidadosamente las operaciones más frecuentes sobre cada colección para seleccionar la estructura óptima.

TablaHash se implementa para colecciones que requieren búsquedas frecuentes por clave. Los usuarios se indexan por nickname y los artistas por código identificador. La tabla utiliza encadenamiento separado para resolver colisiones, manteniendo listas enlazadas en cada posición. El factor de carga se monitorea continuamente y la tabla se redimensiona cuando supera un umbral, garantizando operaciones en tiempo constante promedio.

ArrayDinamico almacena colecciones que requieren acceso por índice y crecimiento dinámico. Los álbumes de un artista, las canciones de un álbum y los favoritos de un usuario se almacenan en arrays que duplican su capacidad al llenarse. Esta estrategia de crecimiento geométrico garantiza que el número de realocaciones sea logarítmico respecto al tamaño final, manteniendo eficiencia en las inserciones.

ListaEnlazada se utiliza para el historial de reproducción. La implementación usa enlaces dobles y estructura circular, facilitando la navegación en ambas direcciones. Los usuarios premium pueden retroceder hasta seis canciones en sus favoritos o cuatro en reproducción aleatoria. La inserción y eliminación de nodos se realiza en tiempo constante sin necesidad de desplazar elementos.

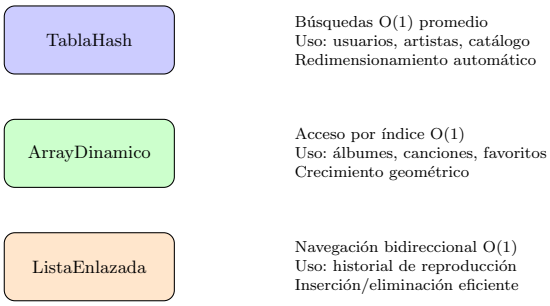

Figura 3: Estructuras de datos implementadas y sus características

4.2. Optimizaciones Implementadas

Para maximizar el rendimiento, el equipo implementó varias optimizaciones en las estructuras de datos. La función hash para cadenas utiliza el algoritmo de Horner con un primo apropiado, distribuyendo uniformemente las claves y minimizando colisiones.

El ArrayDinamico mantiene tanto el tamaño actual como la capacidad reservada. Solo se realoca cuando el tamaño alcanza la capacidad, reduciendo el número de operaciones costosas de copia. Al redimensionar,

se reserva el doble de la capacidad actual, amortizando el costo de futuras inserciones.

La ListaEnlazada mantiene punteros al primer y último nodo, permitiendo inserciones eficientes en ambos extremos. La estructura circular simplifica la lógica de navegación y elimina casos especiales para el inicio y fin de la lista.

5. Diagrama de Clases

El diagrama de clases completo captura todas las entidades del dominio y sus relaciones. Se utilizó notación UML simplificada, enfocándose en la claridad y evitando detalles de implementación innecesarios.

5.1. Capa de Control

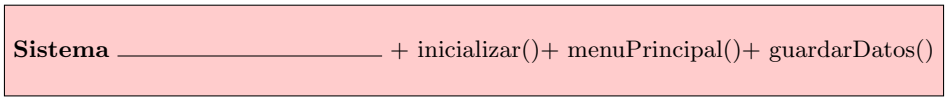

Figura 4: Clase controladora principal

5.2. Capa de Gestión

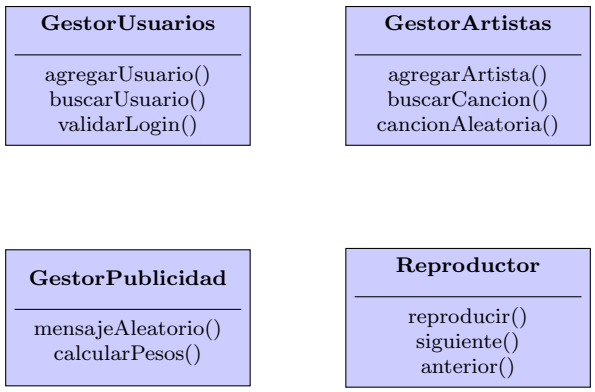

Figura 5: Módulos gestores del sistema

5.3. Capa de Modelo de Datos

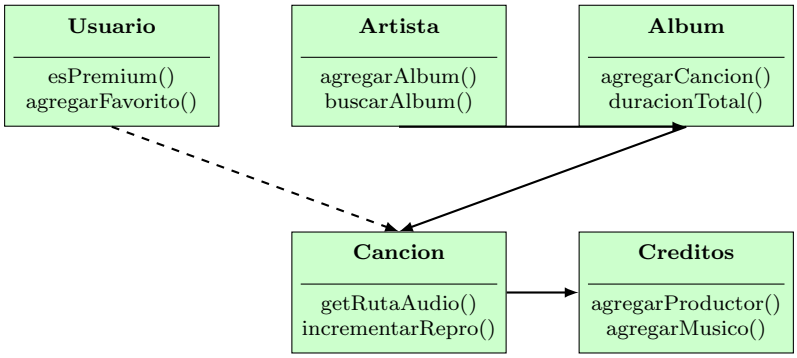

Figura 6: Entidades del modelo de datos y sus relaciones

5.4. Capa de Estructuras de Datos

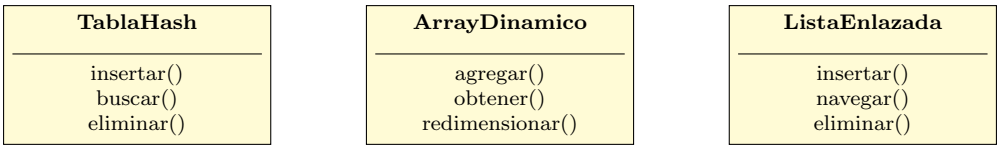

Figura 7: Estructuras de datos genéricas implementadas

Las clases se organizan en cuatro grupos claramente diferenciados. El primero corresponde al controlador principal que coordina todo el sistema. El segundo agrupa los gestores que administran colecciones específicas de objetos. El tercero contiene las entidades del modelo de datos que representan conceptos del dominio. El cuarto incluye las estructuras de datos genéricas reutilizables.

Las relaciones de asociación se representan mediante flechas continuas, indicando que una clase conoce y utiliza a otra. Las relaciones de dependencia se muestran con líneas punteadas, denotando uso temporal o indirecto. No se utilizó herencia en ninguna parte del diseño, siguiendo las restricciones del proyecto.

6. Flujo de Ejecución

6.1. Diagrama de Flujo General

El siguiente diagrama ilustra el flujo de ejecución desde el inicio del programa hasta su finalización, mostrando las decisiones principales que afectan el comportamiento del sistema.

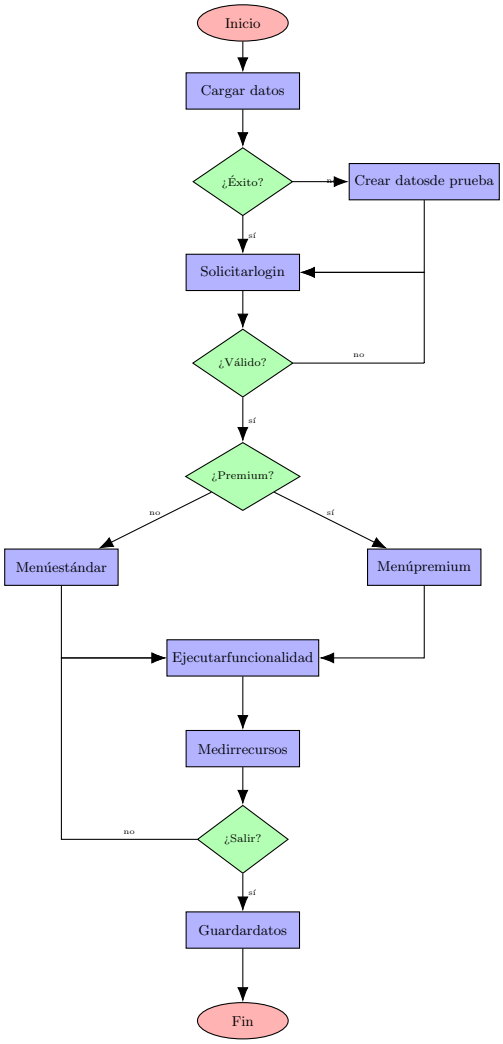

Figura 8: Diagrama de flujo general del sistema

6.2. Descripción del Flujo

El programa inicia intentando cargar los datos desde archivos de persistencia. Si los archivos no existen o contienen errores, se generan datos de prueba que permiten evaluar las funcionalidades básicas del sistema sin requerir configuración previa.

Una vez inicializado el sistema, se presenta la pantalla de autenticación donde el usuario ingresa su nickname. Las credenciales se validan contra la información almacenada en GestorUsuarios. Si la validación falla, se solicitan nuevamente las credenciales hasta obtener un login exitoso.

Después de autenticar al usuario, el sistema determina su tipo de membresía y presenta el menú correspondiente. Los usuarios estándar acceden únicamente a reproducción aleatoria, mientras que los usuarios premium cuentan con opciones adicionales para gestionar favoritos y seguir listas de otros usuarios.

Al finalizar cualquier funcionalidad, el módulo de medición de recursos calcula y muestra las métricas solicitadas: número de iteraciones ejecutadas y memoria total consumida. Estas métricas permiten evaluar

la eficiencia de la implementación y detectar posibles cuellos de botella.

Cuando el usuario decide salir de la aplicación, el sistema persiste todos los cambios realizados durante la sesión. Los datos de usuarios, el catálogo musical y la configuración se guardan en archivos para ser recuperados en la próxima ejecución.

7. Persistencia de Datos

7.1. Diseño de Archivos

El equipo propone utilizar tres archivos principales en formato de texto plano con delimitadores. Esta decisión facilita la depuración durante el desarrollo y permite inspeccionar manualmente el contenido cuando sea necesario.

El archivo usuarios.dat almacena un usuario por línea. Los campos se separan mediante el carácter pipe y al final se incluye una lista de identificadores de canciones favoritas entre corchetes. Este formato permite reconstruir eficientemente las listas de favoritos mediante referencias a canciones existentes.

El archivo artistas.dat utiliza un formato jerárquico con indentación. Cada artista inicia una sección, seguida de sus álbumes con un nivel de indentación, y cada canción con dos niveles. Los créditos se agregan como sublíneas adicionales con identificadores de categoría. Esta estructura textual refleja naturalmente las relaciones de composición.

El archivo publicidad.dat contiene un mensaje por línea con la categoría y el contenido separados por pipe. Al cargar estos datos, el sistema calcula automáticamente los pesos para implementar el sistema probabilístico de visualización según las prioridades establecidas.

Tabla 1: Formato de archivos de persistencia

Archivo	**Formato de ejemplo**
usuarios.dat	usuario123—premium—Medellin—Colombia—2024-01-15—[123456789,234567890]
artistas.dat	12345—Nombre—25—Colombia—50000—15 01—AlbumNombre—2023-05—Pop,Rock—... 01—CancionNombre—240—/ruta/...
publicidad.dat	AAA—Disfruta 3 meses gratis

7.2. Estrategia de Carga y Guardado

La carga de datos se realiza en orden específico para respetar las dependencias entre entidades. Primero se cargan los artistas con todos sus álbumes y canciones, construyendo el catálogo completo. Posteriormente se cargan los usuarios, y al procesar sus listas de favoritos se buscan las canciones por identificador en el catálogo previamente construido.

El guardado invierte este proceso. Se recorren todas las estructuras activas en memoria, generando las líneas de texto correspondientes y escribiéndolas en los archivos. El equipo implementó validaciones para detectar archivos corruptos y manejar errores de lectura/escritura de manera robusta.

8. Sistema de Reproducción

8.1. Reproducción Aleatoria

La reproducción aleatoria selecciona canciones del catálogo completo sin seguir ningún orden predefinido. GestorArtistas proporciona un método que retorna una canción aleatoria, seleccionando primero un artista al azar, luego un álbum al azar de ese artista, y finalmente una canción al azar del álbum.

Para usuarios estándar, cada dos canciones reproducidas se muestra un mensaje publicitario. GestorPublicidad selecciona el mensaje mediante un algoritmo ponderado que respeta las prioridades de categoría. Los mensajes AAA tienen probabilidad triple que los de categoría C, implementado mediante pesos acumulados y selección por número aleatorio.

Los usuarios premium cuentan con controles adicionales. Pueden avanzar a la siguiente canción, retroceder hasta cuatro canciones previas, y activar el modo repetir que reproduce indefinidamente la canción actual. El historial se mantiene mediante una lista enlazada que registra cada canción reproducida.

8.2. Reproducción de Favoritos

Los usuarios premium pueden reproducir su lista de favoritos de manera secuencial o aleatoria. En modo secuencial, las canciones se reproducen en el orden en que fueron agregadas. En modo aleatorio, se genera una permutación aleatoria de los índices y se reproducen en ese orden.

Durante la reproducción de favoritos, los usuarios pueden retroceder hasta seis canciones. El límite mayor que en reproducción aleatoria reconoce que los favoritos representan una selección cuidadosa del usuario que merece mayor control de navegación.

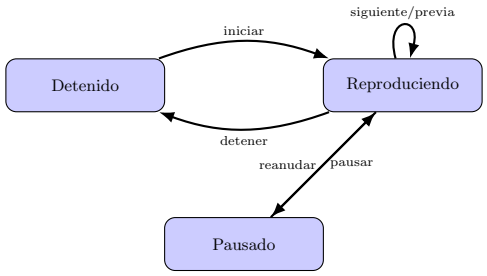

Figura 9: Diagrama de estados del reproductor

9. Medición de Recursos

9.1. Contabilización de Iteraciones

Para contabilizar iteraciones se define una variable global que se incrementa en cada ciclo de todos los algoritmos del sistema. Esta variable se reinicia al inicio de cada funcionalidad y se reporta al finalizar. El equipo documentará claramente en el código cada punto donde se incrementa el contador.

Los ciclos anidados incrementan el contador en el bucle más interno, capturando así la cantidad real de operaciones elementales ejecutadas. Esta métrica permite comparar la eficiencia de diferentes implementaciones y validar que se alcancen las complejidades teóricas esperadas.

9.2. Cálculo de Memoria

El cálculo de memoria se realiza mediante un recorrido exhaustivo de todas las estructuras activas. Cada clase implementa un método calcularMemoria que retorna el tamaño total ocupado, incluyendo el objeto en sí y toda la memoria dinámica reservada.

Para estructuras como TablaHash, se suma el tamaño del array de buckets más la memoria de todas las listas de colisiones. Para ArrayDinamico se reporta la capacidad reservada, no solo el tamaño utilizado. Para objetos complejos como Artista, se suma recursivamente la memoria de todos sus álbumes y canciones.

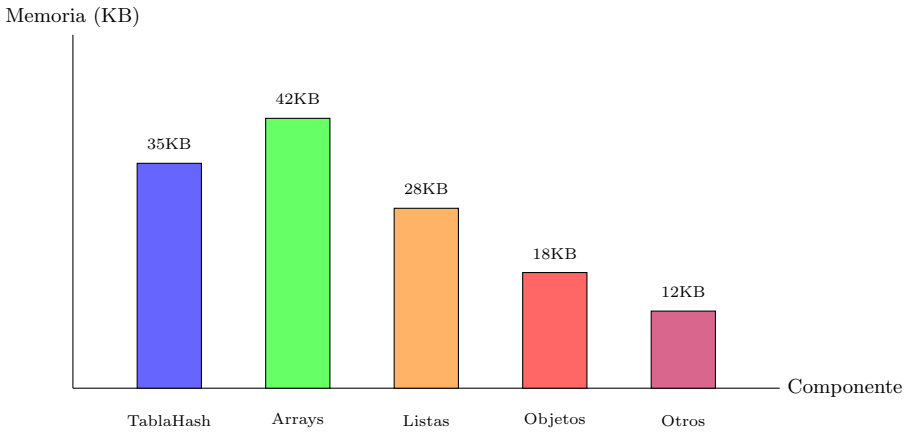

Figura 10: Distribución estimada del consumo de memoria por componente

10. Consideraciones de Implementación

10.1. Gestión de Memoria

La gestión manual de memoria representa uno de los aspectos más críticos del proyecto. El equipo adoptó la política de que cada objeto es responsable de liberar la memoria que reserva. Los destructores de todas las clases liberan explícitamente los recursos dinámicos mediante operador delete.

Para prevenir fugas de memoria, se establece que los gestores son los propietarios de los objetos que administran. Cuando un gestor se destruye, libera todos los objetos de su colección. Las listas de favoritos mantienen únicamente punteros a canciones existentes sin poseerlas, por lo que no las destruyen.

Los punteros nulos se verifican sistemáticamente antes de desreferenciarlos. Las funciones que retornan punteros documentan claramente si transfieren la propiedad del objeto o si el llamador obtiene solo una referencia temporal. Esta disciplina previene tanto fugas como dobles liberaciones.

10.2. Manejo de Referencias

El paso de parámetros se realiza mediante referencias constantes para objetos grandes, evitando copias innecesarias. Por ejemplo, cuando se busca una canción en un álbum, el método recibe una referencia constante al objeto Album en lugar de una copia completa.

Los métodos que modifican objetos reciben referencias no constantes. Los métodos que solo consultan información reciben referencias constantes y se declaran como const. Esta distinción ayuda al compilador a detectar errores donde se intenta modificar objetos que deberían ser inmutables.

10.3. Optimización de Búsquedas

Las búsquedas representan operaciones frecuentes que deben optimizarse. La tabla hash para usuarios utiliza una función hash que distribuye uniformemente los nicknames, minimizando colisiones. El factor de carga se mantiene por debajo de 0.75 para garantizar eficiencia.

La búsqueda de canciones por identificador se optimiza descomponiendo el identificador en sus partes. Los primeros cinco dígitos localizan el artista en la tabla hash, los siguientes dos localizan el álbum en el array del artista, y los últimos dos localizan la canción en el array del álbum. Esta búsqueda jerárquica resulta más eficiente que recorrer linealmente todo el catálogo.

11. Plan de Desarrollo

11.1. Cronograma

El equipo estableció un plan de trabajo estructurado en fases que permite avanzar ordenadamente desde los componentes básicos hasta la integración completa del sistema. Se destinan los primeros días a implementar las estructuras de datos fundamentales, verificando su correcto funcionamiento mediante pruebas unitarias antes de continuar.

La siguiente fase desarrolla el modelo de datos, implementando las clases que representan entidades del dominio. Se verifica que los constructores, destructores y métodos básicos funcionen correctamente. Se presta especial atención a la gestión de memoria en clases que contienen punteros.

Los módulos gestores y la persistencia se desarrollan posteriormente, utilizando las estructuras y clases ya probadas. Se realizan pruebas de carga y guardado verificando que los datos se mantengan correctamente entre ejecuciones del programa.

La implementación del reproductor y las funcionalidades de reproducción se aborda enseguida, integrando todos los componentes previos. Se desarrolla la lógica de reproducción aleatoria, los controles de navegación y la visualización de publicidad.

Finalmente se integra el sistema completo y se implementa el módulo de medición de recursos. Se realizan pruebas exhaustivas verificando el correcto funcionamiento de todas las funcionalidades y refinando detalles de la interfaz de consola.

Tabla 2: Cronograma de desarrollo por fase

Fase	**Días**	**Componentes**
Estructuras base	2	TablaHash, ArrayDinamico, ListaEnlazada
Modelo de datos	2	Usuario, Artista, Album, Cancion, Creditos
Gestores	2	GestorUsuarios, GestorArtistas, GestorPublicidad
Persistencia	1	Carga y guardado de archivos
Reproductor	2	Lógica de reproducción y controles
Integración	1	Sistema completo y medición
Total	**10**	

11.2. Distribución de Tareas

El trabajo se distribuye entre los miembros del equipo según sus fortalezas. Un integrante se enfoca en las estructuras de datos base, garantizando su correcta implementación y gestión de memoria. El otro integrante

desarrolla el modelo de datos y los gestores, asegurando la coherencia de las relaciones entre entidades.

Ambos integrantes colaboran en la implementación del reproductor y la integración final. Las pruebas se realizan de manera conjunta, permitiendo que cada integrante revise el código del otro y detecte posibles errores. Esta metodología de trabajo promueve la calidad del código y el aprendizaje mutuo.

12. Estrategia de Pruebas

Durante el desarrollo se aplicará una estrategia de pruebas incremental. Cada componente se prueba aisladamente antes de integrarlo al sistema. Las estructuras de datos se verifican con casos extremos como inserción en tabla vacía, búsqueda de elementos inexistentes y redimensionamiento de arrays.

Las funcionalidades de reproducción se prueban con diferentes escenarios: usuarios estándar y premium, listas de favoritos vacías y llenas, navegación en los límites del historial. La persistencia se valida guardando datos, cerrando el programa, reiniciando y verificando que la información se recupere correctamente.

El sistema de medición se valida comparando manualmente los valores reportados con cálculos independientes para un conjunto pequeño de datos. Una vez verificada la correctitud, se confía en las métricas para evaluar la eficiencia con volúmenes mayores de información.

13. Conclusiones

El análisis y diseño presentados establecen una base sólida para la implementación del sistema UdeATunes. La arquitectura modular facilita el desarrollo en equipo al separar claramente las responsabilidades de cada componente. La selección cuidadosa de estructuras de datos garantiza que el sistema operará eficientemente incluso con grandes volúmenes de información.

El modelo de clases captura adecuadamente las entidades y relaciones del dominio del problema. La ausencia de herencia simplifica el diseño y evita complejidades innecesarias para este contexto. Las relaciones de composición y agregación reflejan naturalmente cómo se organiza el contenido musical.

Las restricciones técnicas del proyecto representan oportunidades de aprendizaje. Implementar estructuras de datos propias profundiza la comprensión de los algoritmos fundamentales y la gestión de memoria dinámica. Estas habilidades resultan esenciales para desarrollar sistemas eficientes y robustos.

El equipo se encuentra preparado para iniciar la fase de implementación con confianza. El plan de desarrollo establece una ruta clara con hitos verificables. Los próximos días se dedicarán a traducir este diseño en código funcional, manteniendo el foco en la eficiencia y la correctitud.

Referencias

[1] Stroustrup, B. (2013). *The C++ Programming Language* (4th ed.). Addison-Wesley.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[3] Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.