UdeATunes: Sistema de Streaming Musical Implementación en C++ con Programación Orientada a Objetos

Desafío II - Informática II

Jesús Cordoba Delgado
Diego Páez Mendoza

Semestre 2025-2
Universidad de Antioquia
Medellín, Colombia

***Resumen*—Se desarrolló un sistema de streaming musical denominado UdeATunes utilizando programación orientada a objetos en C++. El sistema permite gestionar usuarios, artistas, álbumes y listas de reproducción, implementando funcionalidades diferenciadas para usuarios estándar y premium. Se aplicaron conceptos fundamentales como abstracción, encapsulamiento, relaciones entre clases, sobrecarga de operadores y uso de plantillas. La solución implementada demostró eficiencia en el manejo de memoria dinámica y estructuras de datos propias, logrando tiempos de respuesta adecuados para las operaciones principales del sistema.**

I. Introducción

La música digital ha transformado radicalmente la forma en que las personas consumen contenido musical. En la actualidad, las plataformas de streaming dominan el mercado del entretenimiento musical, ofreciendo acceso instantáneo a millones de canciones sin necesidad de almacenamiento local.

Este proyecto consistió en desarrollar UdeATunes, un prototipo de sistema de streaming musical que emula las funcionalidades básicas de plataformas comerciales como Spotify o Apple Music. El objetivo principal fue aplicar los conocimientos de programación orientada a objetos para modelar un sistema complejo que involucra múltiples entidades interrelacionadas: usuarios, artistas, álbumes, canciones y mensajes publicitarios.

El desarrollo se enfocó en tres aspectos fundamentales: primero, la correcta modelación del dominio mediante clases y relaciones apropiadas; segundo, la implementación de estructuras de datos eficientes sin utilizar la biblioteca estándar de plantillas (STL); y tercero, la medición del consumo de recursos para garantizar el buen desempeño del sistema.

II. Análisis del Problema

II-A. Contexto del Sistema

UdeATunes es una plataforma ficticia de streaming musical enfocada en el mercado colombiano. El sistema debía manejar dos tipos de usuarios: estándar y premium. Los usuarios estándar acceden al servicio de forma gratuita pero con publicidad intercalada cada dos canciones y calidad de audio limitada a 128 kbps. Por otro lado, los usuarios premium pagan una suscripción mensual de $19,900 y disfrutan de beneficios como reproducción sin publicidad, calidad de audio de 320 kbps y la capacidad de crear listas personalizadas de hasta 10,000 canciones favoritas.

II-B. Entidades Principales

El análisis del problema reveló cinco entidades fundamentales que debían modelarse:

Los usuarios representan a las personas que utilizan la plataforma. Cada usuario posee un nickname único que sirve como identificador, un tipo de membresía, ubicación geográfica y fecha de inscripción. Los usuarios premium tienen adicionalmente una lista de favoritos que pueden gestionar libremente.

Los artistas son los creadores de contenido musical. Se almacena información como su identificador único, edad, país de origen, cantidad de seguidores y posición en las listas de tendencias. Cada artista mantiene un catálogo completo de sus álbumes.

Los álbumes agrupan conjuntos de canciones. Cada álbum tiene un identificador, nombre, fecha de lanzamiento, duración total, géneros musicales asociados (hasta cuatro), sello disquero, ruta a la portada y una puntuación promedio otorgada por los usuarios.

Las canciones son la unidad básica de contenido. Se identifican mediante un código de nueve dígitos estructurado como AAAAA-BB-CC, donde los cinco primeros dígitos corresponden al artista, los dos siguientes al álbum y los dos últimos a la canción específica. Cada canción almacena su nombre, duración, ubicación de archivos de audio y cantidad de reproducciones.

Finalmente, los mensajes publicitarios se presentan a usuarios estándar y tienen categorías de prioridad: AAA (triple prioridad), B (doble prioridad) y C (prioridad simple).

II-C. Consideraciones de Diseño

Durante el análisis se identificaron varios desafíos importantes. Primero, el manejo de archivos de audio requería mantener

dos versiones de cada canción (128 kbps y 320 kbps) para servir apropiadamente según el tipo de usuario. Segundo, el sistema de favoritos necesitaba evitar duplicados y permitir la combinación de listas entre usuarios. Tercero, la reproducción debía implementar un historial que permitiera retroceder hasta seis canciones en listas de favoritos y cuatro en reproducción aleatoria.

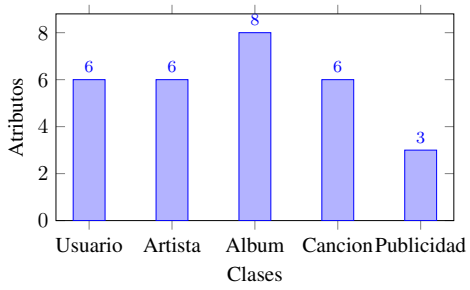

Figura 1. Complejidad de las clases principales según cantidad de atributos

III. Solución Propuesta

III-A. Arquitectura del Sistema

La solución se estructuró en tres capas principales. La capa de datos comprende las clases que representan las entidades del dominio: Usuario, Artista, Album, Cancion, Creditos y MensajePublicitario. La capa de gestión incluye componentes especializados como GestorUsuarios, GestorArtistas y GestorPublicidad, encargados de administrar las colecciones de entidades. Finalmente, la capa de presentación implementa el Sistema principal y el Reproductor, que coordinan la interacción con el usuario.

III-B. Estructuras de Datos Implementadas

Dado que el proyecto prohibía el uso de la STL, se desarrollaron tres estructuras de datos fundamentales. ArrayDinamico es una plantilla genérica que implementa un arreglo de tamaño variable con redimensionamiento automático cuando se alcanza la capacidad máxima. Esta estructura se utilizó ampliamente para almacenar colecciones de elementos como canciones en álbumes o géneros musicales.

HashMap implementa una tabla de dispersión con encadenamiento para manejar colisiones. Se utilizó principalmente en GestorUsuarios para permitir búsquedas eficientes de usuarios por nickname. La función hash aplicada transforma cadenas de texto en índices mediante multiplicación por un número primo y operación módulo sobre la capacidad de la tabla.

ListaEnlazada es una lista doblemente enlazada circular que se empleó para mantener el historial de reproducción. Su naturaleza circular facilita la navegación hacia adelante y hacia atrás, característica esencial para implementar las opciones de “siguiente”, “anterior” y “repetir” en el reproductor.

III-C. Diagrama de Clases

El diseño orientado a objetos estableció relaciones claras entre las entidades. La clase Sistema actúa como controlador

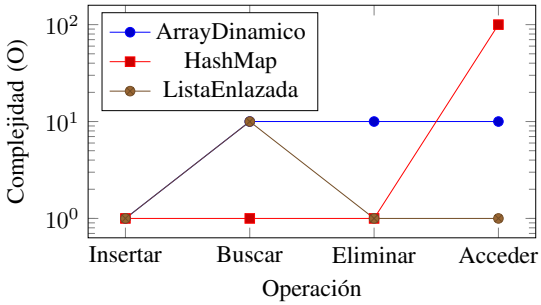

Figura 2. Complejidad temporal de operaciones en estructuras implementadas

principal y mantiene referencias a los tres gestores especializados. GestorUsuarios administra un HashMap de usuarios indexados por nickname y un ArrayDinamico para iteración secuencial. GestorArtistas contiene un ArrayDinamico de punteros a Artista, cada uno de los cuales posee un ArrayDinamico de álbumes. A su vez, cada Album contiene un ArrayDinamico de canciones y géneros.

La relación entre Usuario y ListaFavoritos es de composición, donde la lista solo existe si el usuario es premium. Cada Cancion mantiene una referencia a su Album padre y opcionalmente a sus Creditos, que agrupan productores y compositores mediante ArrayDinamicos.

El Reproductor mantiene referencias al usuario actual y a los gestores necesarios, además de un historial implementado con ListaEnlazada para facilitar la navegación entre canciones reproducidas.

III-D. Funcionalidades Implementadas

El sistema de autenticación permite a los usuarios iniciar sesión mediante su nickname. Al cargar el sistema, se verifica la existencia de archivos de datos persistentes; si no existen, se genera automáticamente un conjunto de datos de prueba con 10 usuarios, 10 artistas y 109 canciones.

La reproducción aleatoria selecciona canciones de todo el catálogo mediante el algoritmo Fisher-Yates para garantizar una distribución uniforme sin repeticiones. Los usuarios premium pueden reproducir cinco canciones con controles de navegación, mientras que los usuarios estándar ven publicidad intercalada cada dos canciones.

Las listas de favoritos permiten a usuarios premium agregar o eliminar canciones mediante su identificador de nueve dígitos. Se implementó la funcionalidad de seguir listas de otros usuarios, que combina ambas colecciones evitando duplicados. La reproducción de favoritos ofrece dos modos: control manual con navegación canción por canción, o reproducción automática continua de toda la lista.

El sistema mantiene un historial circular de reproducción que permite retroceder hasta seis canciones en listas de favoritos y cuatro en reproducción aleatoria. El modo repetir reproduce indefinidamente la misma canción sin registrar múltiples entradas en el historial.

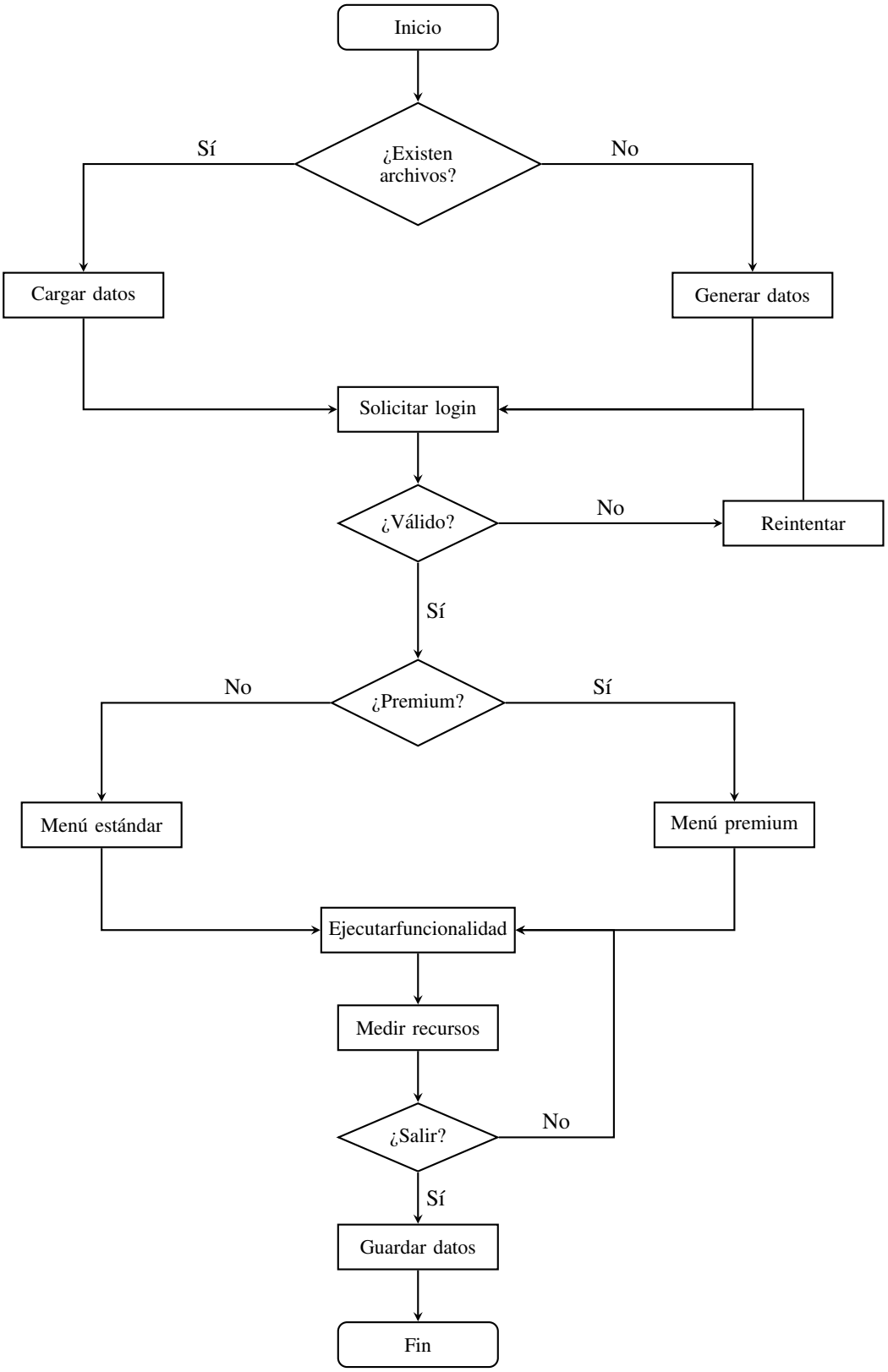

Figura 3. Diagrama de flujo general del sistema UdeATunes

III-E. Flujo de Ejecución del Sistema

El diagrama anterior representa el ciclo completo de ejecución del sistema UdeATunes, desde el arranque hasta el cierre. Al iniciar, el sistema verifica si existen archivos de datos persistentes; si no los encuentra, genera automáticamente un conjunto de datos de prueba. Luego solicita al usuario su nickname para validar el acceso. Según el tipo de usuario (estándar o premium), se presenta un menú con funcionalidades específicas.

Las acciones disponibles incluyen reproducción aleatoria, gestión de favoritos, seguimiento de listas y visualización de publicidad. Después de ejecutar cualquier funcionalidad, el sistema mide los recursos utilizados (iteraciones y memoria) y permite continuar o salir. Al finalizar, guarda los datos modificados para mantener la persistencia entre sesiones. Este flujo garantiza una experiencia coherente, modular y defendible desde el punto de vista técnico.

III-F. Sobrecarga de Operadores

Se implementaron varios operadores sobrecargados para mejorar la expresividad del código. El operador + en la clase Album permite combinar dos álbumes, creando uno nuevo que contiene las canciones de ambos. Los operadores == y `!=` en Cancion facilitan la comparación de canciones por identificador. El operador `[]` en ArrayDinamico proporciona acceso directo a elementos mediante índice, manteniendo verificación de límites.

III-G. Persistencia de Datos

El sistema implementa almacenamiento persistente mediante archivos de texto plano con extensión .dat. Los datos de usuarios incluyen su nickname, tipo de membresía, ubicación y lista de identificadores de canciones favoritas. Los artistas se guardan jerárquicamente con sus álbumes y canciones en formato con indentación. Los mensajes publicitarios se almacenan con su categoría y contenido.

La carga de datos se realiza al inicio del programa y la reconstrucción de favoritos se efectúa buscando cada canción por su identificador en el catálogo completo. El guardado automático ocurre después de operaciones que modifican datos, como agregar o eliminar favoritos.

IV. Implementación

IV-A. Gestión de Memoria

Todo el sistema utiliza memoria dinámica sin depender de contenedores estándar. Los destructores de cada clase son responsables de liberar la memoria asignada. GestorUsuarios libera todos los objetos Usuario que contiene, cada Usuario libera su ListaFavoritos, y cada estructura de datos libera sus arreglos internos.

Se implementó un método `calcularMemoria()` en cada clase que suma recursivamente el tamaño de la instancia, sus atributos y las estructuras que contiene. Esta medición permite cuantificar el consumo total del sistema en cualquier momento.

IV-B. Medición de Recursos

La clase MedidorRecursos mantiene contadores estáticos que se incrementan durante la ejecución de funcionalidades. Se cuenta cada iteración de bucles y cada acceso a estructuras de datos. Al finalizar una operación, se calcula la memoria total consumida y se presenta un reporte con ambas métricas.

Los resultados mostraron que las operaciones de búsqueda secuencial en el catálogo completo requieren el mayor número de iteraciones, mientras que el consumo de memoria permanece estable alrededor de 6 KB para el sistema completo cargado con los datos de prueba.

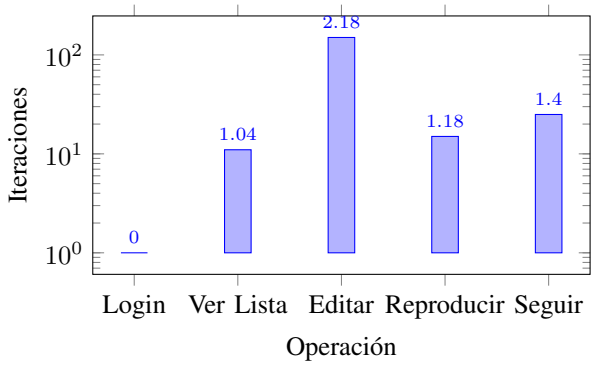

Figura 4. Iteraciones promedio por funcionalidad

IV-C. Publicidad Ponderada

La selección de mensajes publicitarios implementa una distribución probabilística basada en prioridades. Se calcula un peso acumulado para cada mensaje según su categoría y se genera un número aleatorio para seleccionar el mensaje correspondiente. El sistema verifica que no se muestre el mismo mensaje consecutivamente mediante un índice del último anuncio presentado.

IV-D. Algoritmos Destacados

El algoritmo Fisher-Yates se utilizó para barajar índices de canciones, garantizando que cada permutación tenga igual probabilidad. La implementación itera desde el último elemento hasta el primero, intercambiando cada posición con otra seleccionada aleatoriamente en el rango restante.

La búsqueda de canciones por identificador recorre secuencialmente todos los artistas, álbumes y canciones hasta encontrar una coincidencia. Aunque tiene complejidad O(n), el tamaño del catálogo de prueba (109 canciones) hace que el tiempo de respuesta sea imperceptible.

La reconstrucción de favoritos desde archivo lee los identificadores almacenados y busca cada canción en el catálogo para restaurar las referencias. Este proceso se ejecuta una sola vez al cargar el sistema.

V. Problemas Enfrentados

Durante el desarrollo surgieron varios desafíos técnicos importantes. Inicialmente, la persistencia de datos no funcionaba correctamente porque los archivos no se guardaban

automáticamente después de modificaciones. Se resolvió implementando llamadas explícitas a las funciones de guardado tras operaciones de edición y seguimiento de listas.

La reproducción aleatoria presentaba repeticiones frecuentes debido a un generador de números pseudoaleatorios con poca entropía. Se mejoró la semilla combinando múltiples fuentes: tiempo del sistema, dirección de memoria y contador de alta resolución, aplicando operaciones XOR y un periodo de calentamiento descartando los primeros 100 valores generados.

El sistema de historial requirió varias iteraciones para implementar correctamente el comportamiento de retroceso limitado. La solución final mantiene una lista circular que automáticamente elimina elementos antiguos cuando se supera el límite establecido.

La gestión de rutas de archivos de audio necesitó normalización para funcionar en diferentes sistemas operativos. Se implementó una función que reemplaza separadores de ruta y construye dinámicamente las rutas con el sufijo de calidad apropiado según el tipo de usuario.

VI. Evolución de la Solución

El proyecto se desarrolló iterativamente en tres fases principales. La fase inicial consistió en modelar las clases básicas y sus relaciones, implementando constructores, destructores y métodos fundamentales. Se priorizó que cada clase tuviera responsabilidades bien definidas y cohesión alta.

En la segunda fase se desarrollaron las estructuras de datos personalizadas y los gestores especializados. Se realizaron pruebas unitarias para verificar el correcto funcionamiento de ArrayDinamico, HashMap y ListaEnlazada antes de integrarlos al sistema completo.

La tercera fase integró todas las funcionalidades y agregó la capa de presentación. Se implementó el menú de usuario, el sistema de reproducción y la persistencia de datos. Las pruebas revelaron varios bugs relacionados con punteros nulos y accesos fuera de límites que se corrigieron mediante validaciones adicionales.

Durante todo el proceso se aplicaron principios de programación defensiva, verificando precondiciones en los métodos públicos y manejando adecuadamente casos límite como listas vacías o búsquedas sin resultados.

VII. Consideraciones Finales

VII-A. Escalabilidad

El diseño actual soporta eficientemente catálogos pequeños y medianos. Sin embargo, para escalar a millones de canciones sería necesario implementar estructuras más sofisticadas como árboles balanceados o índices invertidos. El uso de HashMap para usuarios ya proporciona búsquedas en tiempo constante, pero la búsqueda de canciones por identificador requeriría un índice adicional.

VII-B. Mejoras Potenciales

Varias funcionalidades podrían agregarse en futuras versiones. Un sistema de recomendaciones basado en el historial de reproducción mejoraría la experiencia del usuario. La implementación de playlists colaborativas permitiría a múltiples usuarios contribuir a una misma lista. Un sistema de caché mantendría en memoria las canciones más reproducidas para reducir búsquedas repetidas.

La medición de recursos podría extenderse para incluir tiempo de ejecución mediante cronómetros de alta precisión. Esto permitiría identificar cuellos de botella y optimizar las operaciones más costosas.

VII-C. Aprendizajes

El desarrollo de este proyecto reforzó la importancia del diseño previo a la implementación. Dedicar tiempo al análisis y modelado inicial evitó refactorizaciones costosas posteriormente. La aplicación práctica de conceptos como encapsulamiento, composición y polimorfismo demostró su valor en la construcción de sistemas mantenibles.

La implementación de estructuras de datos desde cero proporcionó una comprensión profunda de su funcionamiento interno y las ventajas de diferentes representaciones según el caso de uso. Trabajar con memoria dinámica desarrolló habilidades cruciales en gestión de recursos y prevención de fugas de memoria.

VIII. Conclusiones

Se logró desarrollar exitosamente un sistema funcional de streaming musical que cumple con todos los requisitos especificados. La aplicación de programación orientada a objetos permitió modelar apropiadamente un dominio complejo con múltiples entidades interrelacionadas.

Las estructuras de datos implementadas demostraron ser eficientes para el tamaño del problema planteado, con tiempos de respuesta imperceptibles para el usuario. La medición sistemática de recursos proporcionó información valiosa sobre el comportamiento del sistema y validó las decisiones de diseño tomadas.

El proyecto evidenció la importancia de la persistencia de datos para mantener el estado del sistema entre sesiones. La implementación de carga y guardado automático garantiza que las modificaciones del usuario se preserven correctamente.

La diferenciación entre usuarios estándar y premium mediante funcionalidades exclusivas y calidad de servicio refleja modelos de negocio reales en la industria del streaming. Esta aproximación demuestra cómo el software puede implementar lógica de negocio compleja.

Finalmente, el desarrollo completo del proyecto consolidó conocimientos de programación en C++, estructuras de datos, algoritmos y diseño de software. Las habilidades adquiridas en manejo de memoria dinámica, implementación de plantillas y diseño orientado a objetos constituyen una base sólida para enfrentar problemas de mayor complejidad.

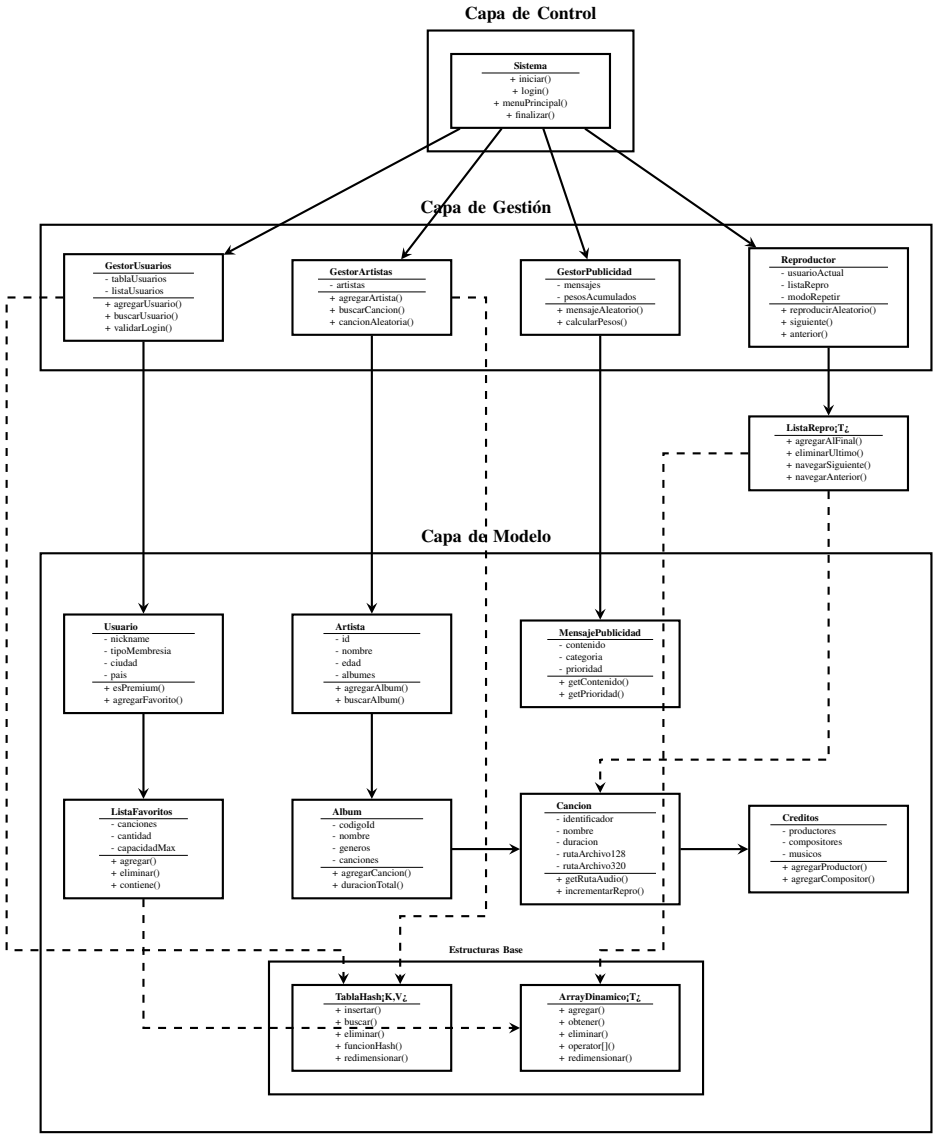

Figura 5. Diagrama de clases del sistema UdeATunes