UdeATunes — Informe final desafio 2
Informática II — 2025-2
Cristian bravo y josé manuel giraldo

1. Requisitos y trazabilidad

Requisitos funcionales clave:

- Carga/actualización persistente (no visible en menú) para usuarios, artistas, álbumes, canciones, favoritos y anuncios.
- Ingreso por nickname y menú condicionado por tipo de usuario.
- Reproducción aleatoria con temporizador de 3 s y corte K=5, anuncios ponderados para estándar, controles premium (siguiente, previa hasta 4, repetir).
- Módulo “Mi lista de favoritos” (solo premium) con: editar por id, seguir otra lista (fusión continua) y ejecutar en orden original o aleatorio con M=6 previas.
- Medición por funcionalidad: iteraciones (directas/indirectas) y memoria dinámica total en el instante de reporte.

Restricciones:

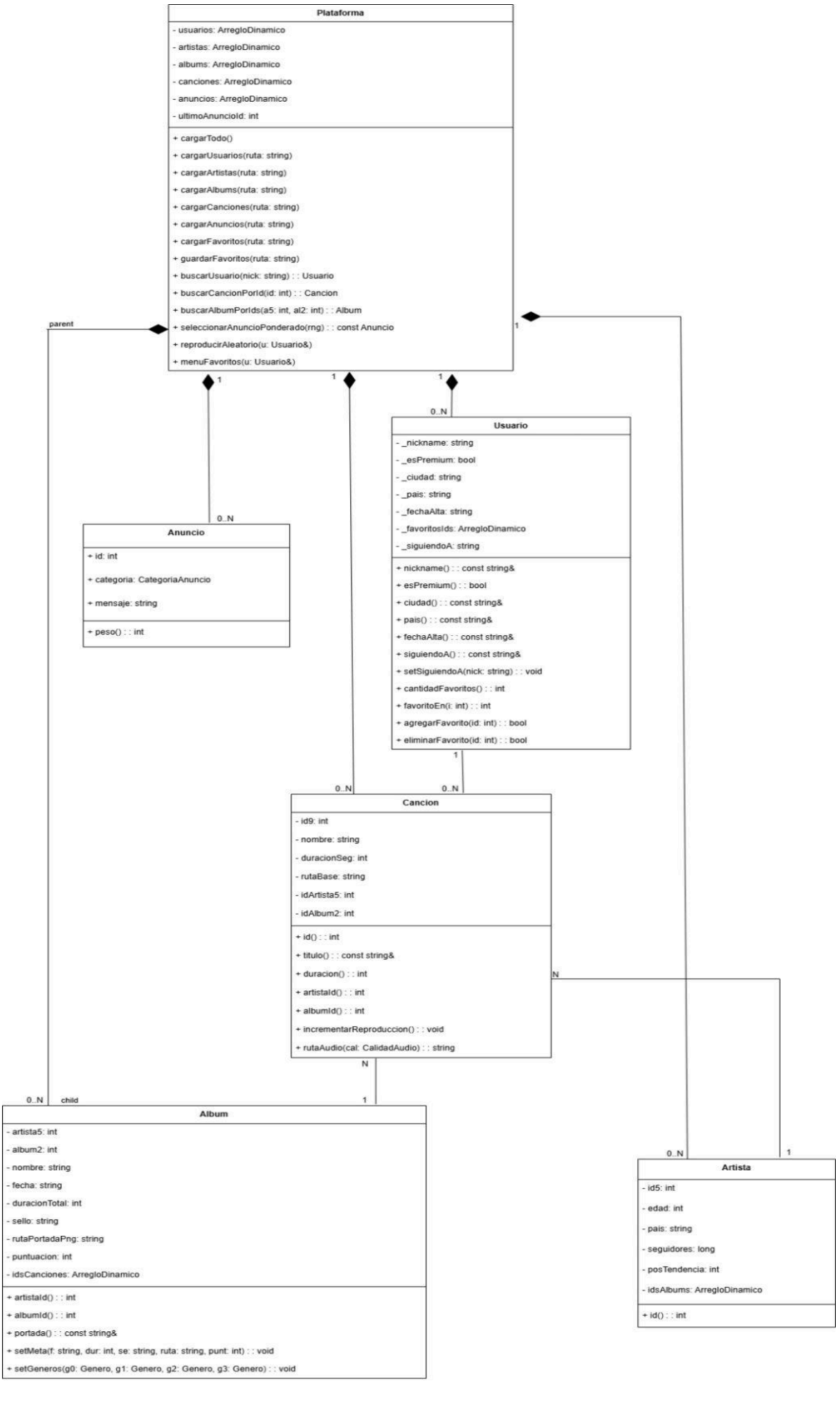
- Sin STL de contenedores, sin herencia; uso de memoria dinámica propia; temporizador con chrono; selección de anuncios ponderada; módulos .h/.cpp y encapsulamiento.
- Se adopta UML simplificado conforme a la guía de la cátedra para el diagrama de clases.

2. Arquitectura y decisiones de diseño

Estilo arquitectónico: “orquestador + entidades puras”. La plataforma coordina colecciones y flujos, las entidades (Usuario/Cancion/Album/Artista/Anuncio) encapsulan datos y exponen APIs mínimas.
Encapsulamiento: todos los campos son privados; Usuario expone nickname/esPremium y una API de favoritos (agregar/eliminar/consultar) que garantiza unicidad y límite de 10 000.
Persistencia: formatos de texto legibles en “./data”, los favoritos se guardan sólo en acciones que cambian el estado (editar, seguir y limpiar).
Reproducción: dos fases para aleatorio (una tanda K=5 y luego uno a uno), ademas de tambien implementar uno a uno en favoritos, lo que facilita probar las funcionalidades de “previa”, “repetir” y publicidad sin entrada no bloqueante ni hilos.
Eficiencia: en la versión base se emplean búsquedas lineales y arreglos dinámicos propios; las métricas se instrumentan en pasos algorítmicos (no en espera de 3 s en chrono pues esta representaba un numero de iteraciones muy significativo que

distorsionaba el numero real de las mismas) para reflejar complejidad real.

3. Modelo de datos y diagrama de clases (UML simplificado)


Plataforma
- usuarios: ArregloDinamico
- artistas: ArregloDinamico
- albums: ArregloDinamico
- canciones: ArregloDinamico
- anuncios: ArregloDinamico
- ultimoAnuncioId: int
+ cargarTodo()
+ cargarUsuarios(ruta: string)
+ cargarArtistas(ruta: string)
+ cargarAlbums(ruta: string)
+ cargarCanciones(ruta: string)
+ cargarAnuncios(ruta: string)
+ cargarFavoritos(ruta: string)
+ guardarFavoritos(ruta: string)
+ buscarUsuario(nick: string) : : Usuario
+ buscarCancionPorId(id: int) : : Cancion
+ buscarAlbumPorIds(a5: int, al2: int) : : Album
+ seleccionarAnuncioPonderado(rng) : : const Anuncio
+ reproducirAleatorio(u: Usuario&)
+ menuFavoritos(u: Usuario&)
parent
1
1
1
1
0..N
0..N
Anuncio
+ id: int
+ categoria: CategoriaAnuncio
+ mensaje: string
+ peso() : : int
Usuario
- _nickname: string
- _esPremium: bool
- _ciudad: string
- _pais: string
- _fechaAlta: string
- _favoritosIds: ArregloDinamico
- _siguiendoA: string
+ nickname() : : const string&
+ esPremium() : : bool
+ ciudad() : : const string&
+ pais() : : const string&
+ fechaAlta() : : const string&
+ siguiendoA() : : const string&
+ setSiguiendoA(nick: string) : : void
+ cantidadFavoritos() : : int
+ favoritoEn(i: int) : : int
+ agregarFavorito(id: int) : : bool
+ eliminarFavorito(id: int) : : bool
1
0..N
0..N
Cancion
- id9: int
- nombre: string
- duracionSeg: int
- rutaBase: string
- idArtista5: int
- idAlbum2: int
+ id() : : int
+ titulo() : : const string&
+ duracion() : : int
+ artistaId() : : int
+ albumId() : : int
+ incrementarReproduccion() : : void
+ rutaAudio(cal: CalidadAudio) : : string
N
N
0..N
child
1
Album
- artista5: int
- album2: int
- nombre: string
- fecha: string
- duracionTotal: int
- sello: string
- rutaPortadaPng: string
- puntuacion: int
- idsCanciones: ArregloDinamico
+ artistaId() : : int
+ albumId() : : int
+ portada() : : const string&
+ setMeta(f: string, dur: int, se: string, ruta: string, punt: int) : : void
+ setGeneros(g0: Genero, g1: Genero, g2: Genero, g3: Genero) : : void
0..N
1
Artista
- id5: int
- edad: int
- pais: string
- seguidores: long
- posTendencia: int
- idsAlbums: ArregloDinamico
+ id() : : int

4. Subprogramas no triviales y flujo de control

4.1 Carga y actualización de datos

- Módulo: Plataforma::cargarTodo. Orquesta la lectura de usuarios, artistas, álbumes, canciones, anuncios y favoritos en ese orden para garantizar que, al cargar favoritos, ya existan canciones y álbums y pueda validarse consistencia mínima. La carga no aparece en menús del usuario, en línea con el enunciado.
- Lectores de archivo: cargarUsuarios/cargarArtistas/cargarAlbums/cargarCanciones/cargarAnuncios; cada uno lee líneas "|" separadas, valida cardinalidades mínimas, instancia objetos y, cuando aplica, completa referencias cruzadas por id. En canciones se enlaza (a5, al2) al álbum recorriendo albums y agregando id9 a idsCanciones del álbum.
- Favoritos: cargarFavoritos lee "nickname|id1,id2,...|siguiendoA"; incorpora ids a la lista del usuario y configura siguiendoA. La escritura se limita a guardarFavoritos y se dispara tras editar o seguir/dejar de seguir, cumpliendo con "actualización restringida".

4.2 Reproducción aleatoria (dos fases)

- Fase automática K=5: selección uniforme entre todas las canciones mediante std::mt19937 + uniform_int_distribution; se muestra tarjeta con datos y se espera 3 s usando std::chrono::steady_clock. Para estándar, se intercala publicidad cada dos canciones; la selección de anuncio usa ponderación y evita repetir el último.
- Fase interactiva uno-a-uno: tras K canciones, se presenta una pista a la vez con opciones. Premium: siguiente, previa (historial hasta 4), repetir on/off. Estándar: reproducir/detener/salir; no se permite saltar ni consultar previas. Se valida que no haya previa al inicio y que el modo repetir no cree entradas duplicadas.

4.3 Mi lista de favoritos (premium)

- Editar: al entrar, se imprimen catálogo e ids favoritos; el usuario ingresa un id. Si existe, puede agregarse o quitarse, evitando duplicados y respetando el límite de 10 000. Al terminar, se persiste en favoritos.txt.
- Seguir otra lista: al indicar nickname, si existe y es premium, se registra siguiendoA y se fusionan sus ids en la lista propia; además, cada ingreso al módulo y antes de ejecutar se sincroniza: se agregan nuevos ids del seguido y se eliminan los que ya no estén en su lista. Dejar de seguir limpia siguiendoA; en la versión base no se diferencian orígenes de cada id (se mantiene política básica).

- Ejecutar favoritos: se elige orden original o aleatorio. Se imprime tarjeta, se espera 3 s y se ofrecen opciones: siguiente, previa (M=6), repetir y salir; el contador K=5 corta para pruebas.

4.4 Interfaz de reproducción

- Tarjeta uniforme: “mensaje publicitario” (si aplica), cantante/álbum/ruta de portada, título/ruta del audio/duración y opciones disponibles según rol y modo. Se centraliza en un helper de UI para mantener consistencia visual entre modos aleatorio y favoritos.
- Validaciones: no detener cuando ya está detenido, no previa sin historial, repetir no avanza ni apila duplicados, estandar no puede saltar en aleatorio, y el corte de K=5 tiene prioridad para finalizar.

5. Algoritmos, estructuras y complejidad

5.1 Estructuras propias

- ArregloDinamico<T>: crecimiento por duplicación, acceso O(1), agregar O(1) amortizado, eliminarEn en O(1) por swap con el último; contabiliza memoria mediante ajustar_memoria en new[]/delete[].
- PilaSimple<T>: implementada sobre ArregloDinamico; push/pop O(1); usada para historiales de 4 (aleatorio) y 6 (favoritos).

5.2 Búsquedas y relaciones

- buscarUsuario/buscarCancionPorId/buscarAlbumPorIds usan barridos O(n) por restricción de no-stl; suficientes para catálogos académicos de prueba.
- Enlace canción→álbum: al cargar canciones, se recorre albums hasta coincidencia (a5, al2) y se agrega id9 a idsCanciones.

5.3 Selección de publicidad ponderada

- Se construye un arreglo temporal de pesos (peso según categoría C/B/AAA) y se usa std::discrete_distribution para muestrear un índice; se evita repetir consecutivo reintentando hasta 20 veces antes de un fallback lineal. Complejidad O(n) para preparar pesos por invocación (en la versión base) y O(1) esperado para muestrear.

5.4 Reproducción y navegación

- Aleatorio: selección uniforme O(1) por pista, historial push/pop O(1), validaciones O(1); K canciones implica O(K).
- Favoritos: avance original O(1); aleatorio sobre favoritos O(1); “previa” pop O(1); sincronización con seguido O(m+n) en cada entrada (m=favoritos propios, n=favoritos del seguido).

6. Medición del consumo de recursos

6.1 Iteraciones (directas e indirectas)

- Se cuentan comparaciones en búsquedas lineales, elementos procesados en parseo de líneas/campos, fusiones/limpiezas en “seguir”, intentos de anuncios y transiciones de pista/historial. Se evita contar espera con chrono y E/S por consola.
- Ejemplo: en buscarCancionPorId, 1 iteración por comparación; en seleccionarAnuncioPonderado, 1 por intento; en cargarCanciones, 1 por registro más 1 por comparación en cada álbum hasta enlazar.

6.2 Memoria dinámica total

- Se suman bytes de todos los arreglos dinámicos de las colecciones (usuarios, artistas, álbums, canciones, anuncios) y de los arreglos internos (idsCanciones, favoritosIds).
- Reporte: al finalizar cada funcionalidad, imprimir_metricas muestra Iteraciones y Memoria dinámica (bytes) con estado consistente en ese instante.

7. Pruebas funcionales y de eficiencia

7.1 Datos y rutas

- Archivos de datos en ./data con formatos “|” para entidades y “,” para ids de favoritos; rutas a portadas en ./covers y audios en ./audio con sufijos _128.ogg/_320.ogg, de modo que se pueda validar presencia física o al menos la impresión de rutas.
- Suficiencia de dataset: varios usuarios premium/estándar, múltiples artistas/álbums/canciones para probar aleatorio y favoritos, y anuncios con categorías para test de ponderación.

7.2 Casos de uso verificados

- Inicio premium vs estándar; edición de favoritos con verificación de duplicados; seguir y sincronizar con limpieza cuando el seguido borra; dejar de seguir.
- Reproducción aleatoria: tanda K=5 sin controles, luego uno-a-uno con controles según perfil; publicidad cada dos en estándar evitando repetición inmediata.
- Reproducción de favoritos: original/aleatorio, previa hasta M=6, repetir toggle, corte en K=5.

7.3 Casos límite

- Favoritos vacíos; usuario inexistente; canción no encontrada al ejecutar; álbum sin portada; archivos de datos faltantes (comportamiento tolerante: no crashea, pero informa).

8. Problemas de desarrollo

- Métricas infladas: se excluyó la espera con chrono y prompts de consola del conteo de iteraciones para medir complejidad y no tiempo de pared.
- Conflictos en “seguir”: la definición de limpieza cuando el seguido elimina elementos se implementó al ingresar al módulo o previo a ejecutar, para consistencia “continua” sin reactividad compleja.
- Duplicados/consistencia: la API de Usuario válida unicidad en agregar/eliminar; al seguir, la fusión respeta invariantes.
- Confusión con data sets de desarrollo: Los data sets de desarrollo conforman una parte fundamental de la implementación del código y el no tener en un principio un formato claro representó un problema de estructuración dado que se debían implementar versiones propias sin conocer las métricas de instrucciones para la fabricación de los mismos.
- Definición de tareas: La definición de las tareas en este desafio en particular representó un reto de análisis mas elevado en comparacion al desafio anterior, puesto que al ser un programa que abarca un cantidad de funcionalidades significativas se tuvo que analizar cuidadosamente las estructuras, metodos y atributos de cada clase para el correcto funcionamiento de las mismas.
- Librerías y propio lenguaje de programación: La utilización de librerías externas como chrono o random, supuso un tiempo de investigación a parte para la comprensión de las funcionalidades que estas nos proveen para poder integrarlas de manera satisfactoria en nuestro sistema

9. Evolución de la solución

- prototipo mínimo: se definieron las entidades base (Usuario, Cancion, Album, Artista y Anuncio) y un orquestador Plataforma sin persistencia, con reproducción simple de una pista y sin distinción entre perfiles de usuario, preparando la estructura para cumplir los requisitos posteriores del enunciado.
- carga/actualización persistente oculta: se diseñaron archivos de texto “|”-separados para usuarios, artistas, álbumes, canciones, anuncios y favoritos, y se implementó cargarTodo para leerlos al inicio sin mostrar opciones de carga en el menú del usuario, tal como exige que la carga no aparezca en la interfaz principal.

- inicio de sesión y menú condicionado: se agregó el login por nickname y se activó un menú que limitaba funcionalidades según el tipo de usuarios, dejando que “Mi lista de favoritos”funcionara únicamente para usuarios premium como establece el enunciado.
- reproducción aleatoria con 3 s y K=5 canciones: se incorporó un temporizador de 3 segundos por pista y una detención automática tras K=5 canciones para las pruebas, además de eso se implementaron anuncios ponderados en estándar y controles premium (siguiente, previa hasta cuatro y repetir), respetando que usuarios estándar no pueden consultar previas ni saltar.
- módulo de favoritos: se completó “Mi lista de favoritos” con editar por id (sin duplicados), para seguir otra lista de un usuario premium, y ejecutar en orden original o aleatorio con M=6 previas, garantizando unicidad y persistencia de cambios al finalizar cada operación.
- interfaz de reproducción unificada: se estandarizó la tarjeta de reproducción que muestra los cantantes y álbums, ruta de portada, títulos, ruta de audios y duración, y después las opciones válidas según perfil y modo de reproducción, favoreciendo consistencia y testabilidad.
- medición de recursos: se añadió un reporte por funcionalidad con iteraciones directas e indirectas y la memoria dinámica total del sistema en el instante del informe, siguiendo la exigencia de medir eficiencia y sin contabilizar esperas ni I/O para no distorsionar el indicador.
- pulido y validaciones: se consolidaron reglas de navegación implícitas (no previa al inicio, no detener si no hay reproducción activa, repetir sin duplicar historial) y se ajustó la experiencia para que la fase aleatoria sea primero automática (K=5) y luego uno-a-uno, facilitando la evaluación de “previa” y “repetir” en condiciones controladas.
- La solución evoluciono de ser un prototipo con simples funcionalidades básicas y un diagrama de clases incompleto a uno completamente consistente y completo que presenta con total firmeza todas las funcionalidades exigidas