DESAFIO 2

INFORMATICA II

REALIZADO POR:

JUAN DIEGO ARBELAEZ MALPICA

SANTIAGO HENAO ZULUAGA

UNIVERSIDAD DE ANTIOQUIA

2025

ANALISIS DEL PROBLEMA

CONTEXTUALIZACION:

La solución del problema planteado es desarrollar un sistema que simule un programa básico de una plataforma de musical llamada UdeATunes, utilizando POO en C++.

El programa debe permitir editar usuarios y que se pueda cambiar de una membresía a otra, debe contener artistas, álbumes, canciones y listas de reproducción, gestionando además aspectos como reproducción de publicidad según el tipo de usuario.

El análisis del problema nos permitió identificar las siguientes clases y relaciones principales:

- **Usuario:** Representa a los clientes de la plataforma. Se distinguen dos tipos de usuarios (estándar y premium), con funcionalidades diferentes (por ejemplo, los premium pueden tener listas de favoritos y reproducir canciones sin publicidad).
- **Artista:** Contiene información del artista, como país, edad y número de seguidores.
- **Álbum:** Conjunto de canciones asociadas a un artista. Incluyendo datos como nombre, género, duración y la ruta de su portada.
- **Canción:** Pista musical con atributos como el nombre, duración, identificador, rutas a los archivos de audio (128 kbps y 320 kbps), número de reproducciones y créditos.
- **Créditos:** Contiene información de los productores, artistas y compositores asociados a una canción.
- **Playlist:** Grupo de canciones seleccionadas por el usuario, con métodos de reproducción normal o aleatoria.
- **Interfaz:** Encargada del control general que administra y coordina las operaciones como autenticación, reproducción y carga de datos.

El siguiente diagrama representa la estructura del sistema:

DIAGRAMA DE CLASES

Presentamos nuestro diagrama de clases con el cual llegamos a nuestra solución planteada.

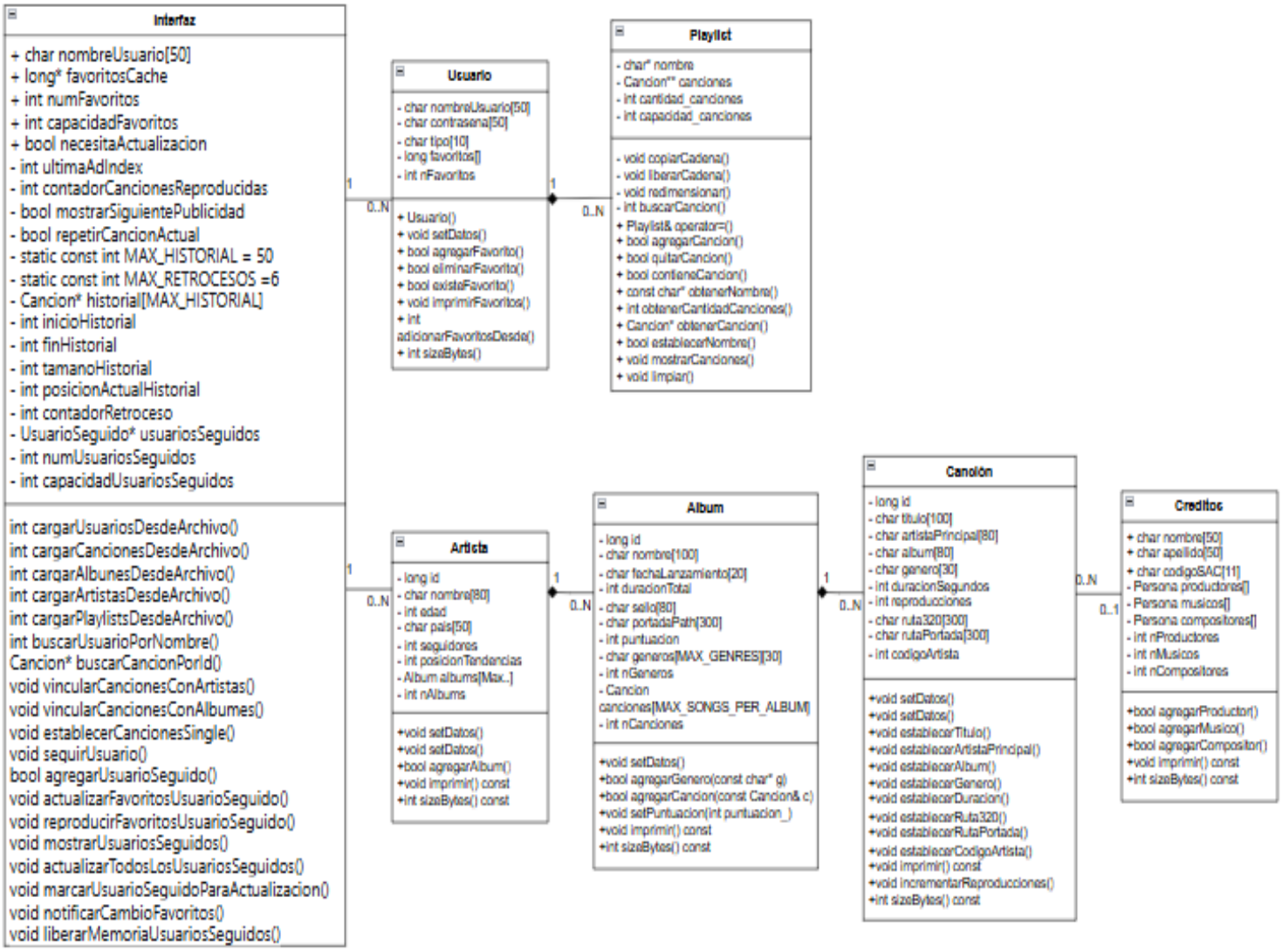

Imagen tomada de nuestro proyecto implementado en drawio

Algoritmos implementados

(intra-documentados)

Presentamos tres algoritmos importantes del sistema, explicados paso a paso dentro del contexto del código.

Reproducción aleatoria

```
// Función que reproduce canciones aleatorias
void Interfaz::reproducirAleatoria(Cancion canciones[], int nCanciones, Usuario* u,bool esPremium, int &iterCount, int &memUsed) {
// Si no hay canciones, mostrar mensaje
   if (nCanciones <= 0) {
      cout << "No hay canciones.\n";
      return;
   }

 // Se eligen hasta 5 canciones máximo
   int K = (nCanciones < 5 ? nCanciones : 5);
   Cancion* sesion[5];
   bool usado[1000] = {false};
   int elegidos = 0;

   srand((unsigned)time(nullptr)); // para los números aleatorios

 // Selecciona canciones sin repetir
   while (elegidos < K) {
```

```
        int r = rand() % nCanciones;
        if (!usado[r]) {
            usado[r] = true;
            sesion[elegidos++] = &canciones[r];
        }
    }

    // Se llama a la función que reproduce la lista
    reproducirLista(sesion, K, u, esPremium, iterCount, memUsed);
}

// En la función reproducirLista se agrega la publicidad si el usuario no es premium
void Interfaz::reproducirLista(Cancion* lista[], int n, Usuario* u, bool esPremium, int
&iterCount, int &memUsed) {
    if (n <= 0) {
        cout << "Lista vacia.\n";
        return;
    }

    for (int i = 0; i < n; i++) {
        reproducirCancion(lista[i], esPremium, iterCount, memUsed);

// Cada dos canciones se muestra un mensaje para usuarios Free
        if (!esPremium && i % 2 == 1)
            mostrarPublicidad("Compra ya el producto X!", 'C');
    }

    cout << "\nReproduccion finalizada.\n";
```

```
}
```

Manejo de favoritos

```
// Parte del menú para manejar favoritos
if (sub == 1) {
    cout << "\nCanciones disponibles:\n";
    for (int i = 0; i < nCanciones; i++) {
        cout << "ID=" << canciones[i].getId() << " - " << canciones[i].getNombre() << "\n";
    }

    cout << "Ingrese ID de cancion para agregar o eliminar: ";
    long id;
    cin >> id;

    Cancion* c = interfaz.buscarCancionPorId(canciones, nCanciones, id);
    if (!c) {
        cout << "No existe ninguna cancion con ese ID.\n";
    } else {
// Si la cancion ya está en favoritos, se elimina
        if (u->existeFavorito(id)) {
            u->eliminarFavorito(id);
            cout << "Eliminado de favoritos: " << c->getNombre() << "\n";
        }
// Si no está, se agrega
        else {
            if (u->agregarFavorito(id)) {
                cout << "Agregado a favoritos: " << c->getNombre() << "\n";
```

```
        } else {
            cout << "No se pudo agregar (lista llena o duplicada)\n";
        }
    }

    // Mostrar la lista actualizada
    cout << "\nFavoritos actuales (" << u->getNumFavoritos() << "):\n";
    for (int i = 0; i < u->getNumFavoritos(); i++) {
        long fid = u->obtenerFavoritoPorIndice(i);
        Cancion* fc = interfaz.buscarCancionPorId(canciones, nCanciones, fid);
        if (fc) cout << "ID=" << fid << " - " << fc->getNombre() << "\n";
    }
  }
}
```

Cálculo del uso de memoria

```
// Función que calcula la memoria usada en el sistema
int Interfaz::calcularMemoriaUsadaEstimado(Usuario usuarios[], int nUsuarios, Cancion canciones[], int nCanciones) {
  int memoria = 0;

// Cada usuario ocupa cierto espacio en memoria
  memoria += nUsuarios * sizeof(Usuario);

// Cada cancion también ocupa su espacio
  memoria += nCanciones * sizeof(Cancion);
```

```
// Se suma el espacio que usa la interfaz
    memoria += sizeof(Interfaz);

// Se retorna el total estimado en bytes
    return memoria;
}
```

Problemas de Desarrollo que afrontamos

Durante el desarrollo del proyecto tuvimos varios inconvenientes, sobre todo al principio. Uno de los primeros problemas fue organizar tantas clases y lograr que todas se comunicaran bien sin usar herencia ni librerías STL.
Tuvimos que pensar bien cómo pasar la información entre archivos, porque a veces se nos cruzaban los tipos de datos o se duplicaban los nombres de variables.

Otro problema fue en la parte de reproducción aleatoria, porque al principio las canciones se repetían o no salían todas las que debían.
Entonces nos tocó ajustar el ciclo que selecciona las canciones para evitar que se repitieran usando un arreglo de control.

También presentamos problemas con el manejo de favoritos, porque cuando intentaba eliminar o agregar canciones, a veces no se actualizaba la lista o mostraba IDs que ya no existían y pues al final pudimos darle solución revisando bien los índices y usando ciclos simples para recorrer los arreglos sin errores.

Otro detalle complicado fue el uso de rutas de archivos, ya que el programa no encontraba los .txt hasta que pusimos las rutas en varias ocasiones y al fin era como problema del compilador.

Por último, el tema del temporizador y la publicidad también nos dio trabajo, porque nos tocó simular el tiempo con usleep() y coordinar cuándo mostrar los mensajes sin que el programa se bloqueara.

Evolución de la solución y consideraciones para tener en cuenta en la implementación

Al inicio del proyecto la idea era solo que el programa pudiera reproducir canciones, pero poco a poco fuimos agregando más cosas como los favoritos, las listas, la medición de memoria y la publicidad para los usuarios Free.
Al principio teníamos todo el código separado en las ramas de cada uno, pero al momento de hacer el merge se nos dañaba o nos presentaba errores en los archivos .user y .user.pro y al final pudimos dejarlo implementado sin ramas previamente hablado con el profesor.

Durante el desarrollo también fuimos corrigiendo errores y simplificando partes del código que se veían muy largas.
Por ejemplo, la reproducción aleatoria primero estaba repetida en varias funciones, pero la reorganizamos dentro de la clase Interfaz para que todo quedara más ordenado.
También agregamos el conteo de iteraciones y memoria para tener una idea de cómo se comporta el programa.

Consideramos que, para futuras implementaciones, sería bueno guardar los cambios al cerrar el programa.
También sería interesante agregar más opciones de reproducción o crear un sistema de búsqueda de canciones.