|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Tecnologías de Desarrollo de Software**  **C:\Users\tonto_000\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\B4DA7523.tmp**  **Práctica final:** AppMusic | | |

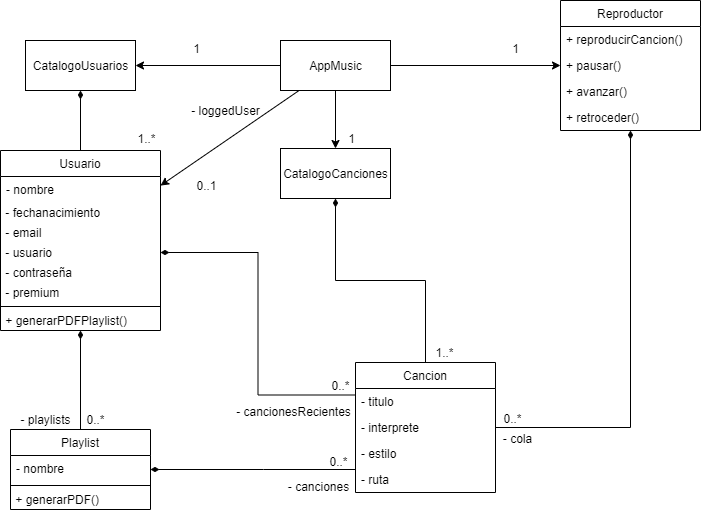
Grupo 3.1

**Profesor/a:** Jesús Joaquín García Molina

**Realizado por:**   
Alberto Cuadrado López (alberto.cuadradol@um.es) y Sergio Escudero Manzano ([sergio.escuderom@um.es](mailto:sergio.escuderom@um.es)).

**Convocatoria:** Junio 2021

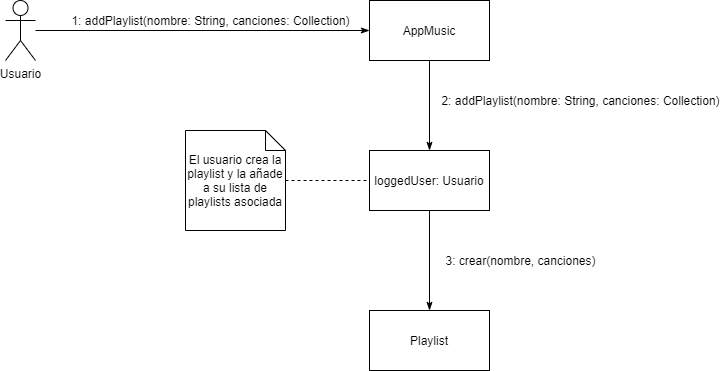
# Diagrama de clases del dominio



## Elementos destacables del dominio

* loggedUser: esta variable estará en el controlador y guardará el usuario que está usando la aplicación en ese momento, puesto que solo va a haber hasta un usuario usando la aplicación al mismo tiempo. Esta variable se establece al ejecutar login(username, password) o logout() de modo que preservamos la seguridad de cara a los usuarios.
* Reproductor: para nuestro caso, hemos decidido que el reproductor gestione las canciones y colas, de modo que trabaja independientemente del controlador una vez se le da una canción o lista de canciones. Además, se encarga de invocar el método del usuario de playedSong(canción), para notificar que se ha reproducido una canción. Esto sucede cuando se acaba una canción y se reproduce la siguiente en la cola.
* Catálogos: los catálogos hacen uso del patrón singleton al igual que el controlador, de modo que son entidades independientes.

# Diagrama de colaboración (añadir playlist)



En este caso no hace falta acceder al catálogo de usuarios por lo mencionado anteriormente, y es que el controlador contiene una variable con el usuario que haya iniciado sesión, de modo que no hay que acceder siempre a dicho catálogo.

# Arquitectura de la aplicación

La aplicación sigue el patrón Modelo-Vista-Controlador, de modo que identificamos 3 partes diferentes:

* Vista: GUI programada haciendo uso de Swing.
* Controlador: clase AppMusic encargada de interactuar con el modelo.
* Modelo: conjunto de clases que representan la información del sistema.

Por otro lado, el modelo es persistente gracias al uso del patrón DAO, de modo que las clases Usuario, Canción y Playlist no se pierden al reiniciar la aplicación. Esto es gracias a la base de datos H2 proporcionada y su respectivo controlador.

En cuanto a la estructura de la interfaz, estamos hablando de que tenemos una clase que contiene todos los demás JFrames de la aplicación, en nuestro caso es MainWindow. Esta contiene un border layout el cual se distribuye de la siguiente manera:

* Arriba: Información y botones (mejora de cuenta, carga de canciones y cierre de sesión).
* Abajo: Controles e información de reproducción.
* Izquierda: Panel de navegación y lista de playlists.
* Centro: Paneles de búsqueda, modificación de playlist, etc.

Cabe destacar que el centro contiene varios paneles, pero solo puede tener uno visible al mismo tiempo, de modo que, cambiar entre las diferentes partes de la aplicación es cuestión de ocultar todos los paneles y hacer visible el que hayamos seleccionado.

# Patrones de diseño utilizados

Algunos de los patrones de diseño que hemos empleado son los siguientes:

* Observer (Reproductor): Coordinar el reproductor y la interfaz para realizar el cambio de imágenes de los botones es algo complejo por lo que buscamos una solución a ello y llegamos a la conclusión de que este patrón era ideal, ya que separamos las acciones que realiza el usuario de los cambios en la interfaz, de modo que se implementan por separado. En primer lugar, las acciones se programan como normalmente se haría en swing, pero para que haya cambios en la interfaz, usamos un Listener el cual incluye métodos que suponen la reproducción de una canción o alternar los modos de reproducción, tal que cuando el Reproductor ejecute los eventos, la interfaz se actualizará acorde. Esta solución es debido a que a veces pueden ocurrir errores debido a la reproducción, así que delegamos esta gestión en el Reproductor, de modo que centramos toda su lógica en una única clase.
* Observer (Swing): para conocer cuando el usuario interactúa con la aplicación, se hace uso de este patrón, para introducir código a esas acciones por medio de Listeners, los cuales se ejecutan cuando se realiza una determinada acción, como pulsar un botón.
* Controlador (AppMusic): para la gestión de información del sistema, utilizamos el patrón controlador, que va a ser el encargado de interactuar con las clases del dominio, centrando las acciones en una única clase.
* Experto y creador: dentro del modelo se respetan los patrones experto y creador, los cuales distribuyen las responsabilidades en las clases correspondientes del dominio.
* Factoría abstracta: para la implementación de un servicio de persistencia independiente de la aplicación, hacemos uso de este patrón, de modo que separamos la declaración de la persistencia de la implementación, permitiendo al modelo optar por varías opciones cambiando un parámetro a la hora de obtener la factoría.

# Componentes utilizados

## Cargador de canciones

Este componente se encarga de cargar las canciones de un archivo XML extrayendo sus datos y URL. De por sí, el componente nos permitía cargar las canciones, pero para adaptarlo al sistema y evitar el acoplamiento, se hace uso de una interfaz IBuscadorCanciones y de CancionesListener, de modo que podemos establecer el archivo de canciones y añadir un listener para notificar del evento a nuestra aplicación.

## Pulsador

Este pulsador se ha utilizado en la interfaz para abrir la selección de archivo XML para la carga de canciones. Es simplemente un botón que se enciende o se apaga cada vez que se pulsa.

## JCalendar

JCalendar nos permite añadir el selector de fecha que encontramos en la ventana de registro, de modo que es más accesible y sencillo para el usuario, puesto que no tiene que seguir un patrón concreto al introducir la fecha, sino que el propio componente lo gestiona.

## Tema oscuro

Para el tema oscuro de swing que hemos utilizado, hemos empleado una librería de código abierto disponible en este [enlace](https://github.com/weisJ/darklaf). Su uso es muy sencillo y se ejecuta al inicio de la aplicación, de modo que cambia todo el estilo de Swing automáticamente. Toda la información está en su repositorio de GitHub.

## Generación de PDFs

La librería ITextPDF nos permite generar PDFs de manera sencilla y rápida. En nuestro caso hemos hecho que los archivos generados sean muy simples para no extender demasiado el código, puesto que es una funcionalidad secundaria.

# Tests unitarios

# Manual de usuario

Un pequeño manual de usuario que explique cómo usar la aplicación.

# Observaciones

Observaciones finales que el alumno desee comentar (deben incluir una estimación del tiempo dedicado)