MusicSearch

1.1.2

Generado por Doxygen 1.16.1

1. Documentación de directorios

1.1. Referencia del directorio BetaProyecto

Directorios

- directorio Helpers
- directorio Models
- directorio obj
- directorio Services
- directorio Singleton
- directorio ViewModels

Archivos

- archivo App.axaml.cs
- archivo Program.cs
- archivo ViewLocator.cs

1.2. Referencia del directorio BetaProyecto.API

Directorios

- directorio Controllers
- directorio obj

Archivos

- archivo Program.cs

1.3. Referencia del directorio BetaProyecto.API/Controllers

Archivos

- archivo MusicController.cs
- archivo StorageController.cs

1.4. Referencia del directorio BetaProyecto.API/obj/Debug

Directorios

- directorio net9.0

1.5. Referencia del directorio BetaProyecto/obj/Debug

Directorios

- directorio net9.0

1.6. Referencia del directorio BetaProyecto/Helpers

Archivos

- archivo ControladorDiccionarios.cs
- archivo Encriptador.cs
- archivo TextoTraducidoConverter.cs

1.7. Referencia del directorio BetaProyecto/Models

Archivos

- archivo Canciones.cs
- archivo Generos.cs
- archivo ListaPersonalizada.cs
- archivo ListaUsuarios.cs
- archivo Reportes.cs
- archivo Roles.cs
- archivo TarjetasCanciones.cs
- archivo TarjetasListas.cs
- archivo Usuarios.cs

1.8. Referencia del directorio BetaProyecto.API/obj/Debug/net9.0

Archivos

- archivo .NETCoreApp,Version=v9.0.AssemblyAttributes.cs
- archivo BetaProyecto.API.AssemblyInfo.cs
- archivo BetaProyecto.API.GlobalUsings.g.cs
- archivo BetaProyecto.API.MvcApplicationPartsAssemblyInfo.cs

1.9. Referencia del directorio BetaProyecto.API/obj/Release/net9.0

Directorios

- directorio win-x64

Archivos

- archivo .NETCoreApp,Version=v9.0.AssemblyAttributes.cs
- archivo BetaProyecto.API.AssemblyInfo.cs
- archivo BetaProyecto.API.GlobalUsings.g.cs

1.10. Referencia del directorio BetaProyecto/obj/Debug/net9.0

Archivos

- archivo .NETCoreApp,Version=v9.0.AssemblyAttributes.cs
- archivo BetaProyecto.AssemblyInfo.cs

1.11. Referencia del directorio BetaProyecto/obj/Release/net9.0

Directorios

- directorio win-x64

Archivos

- archivo .NETCoreApp,Version=v9.0.AssemblyAttributes.cs
- archivo BetaProyecto.AssemblyInfo.cs

1.12. Referencia del directorio BetaProyecto.API/obj

Directorios

- directorio Debug
- directorio Release

1.13. Referencia del directorio BetaProyecto/obj

Directorios

- directorio Debug
- directorio Release

1.14. Referencia del directorio BetaProyecto.API/obj/Release

Directorios

- directorio net9.0

1.15. Referencia del directorio BetaProyecto/obj/Release

Directorios

- directorio net9.0

1.16. Referencia del directorio BetaProyecto/Services

Archivos

- archivo AudioService.cs
- archivo DialogoService.cs
- archivo IDialogoService.cs
- archivo MongoAtlas.cs
- archivo StorageService.cs

1.17. Referencia del directorio BetaProyecto/Singleton

Archivos

- archivo GlobalData.cs
- archivo MongoClientSingleton.cs

1.18. Referencia del directorio BetaProyecto/ViewModels

Archivos

- archivo CentralTabControlViewModel.cs
- archivo INavegable.cs
- archivo LoginViewModel.cs
- archivo MarcoAppViewModel.cs
- archivo PanelUsuarioViewModel.cs
- archivo TabItemBuscadorViewModel.cs
- archivo TabItemInicioViewModel.cs
- archivo TabItemPopularesViewModel.cs
- archivo VentanaAvisoViewModel.cs
- archivo VentanaConfirmacionViewModel.cs
- archivo ViewAyudaViewModel.cs
- archivo ViewCancionesViewModel.cs
- archivo ViewConfiguracionViewModel.cs
- archivo ViewCrearListaPersonalizadaViewModel.cs
- archivo ViewCrearReporteViewModel.cs
- archivo ViewCrearUsuarioViewModel.cs
- archivo ViewCuentaViewModel.cs
- archivo ViewEditarCancionViewModel.cs
- archivo ViewEditarListaPersonalizadaViewModel.cs
- archivo ViewGestionarBDViewModel.cs
- archivo ViewGestionarCuentaViewModel.cs
- archivo ViewGestionarReportesViewModel.cs
- archivo ViewListaPersonalizadaViewModel.cs
- archivo ViewModelBase.cs
- archivo ViewPerfilViewModel.cs
- archivo ViewPublicarCancionViewModel.cs
- archivo ViewSobreNosotrosViewModel.cs
- archivo ViewUsuariosViewModel.cs

1.19. Referencia del directorio BetaProyecto.API/obj/Release/net9.0/win-x64

Archivos

- archivo .NETCoreApp,Version=v9.0.AssemblyAttributes.cs
- archivo BetaProyecto.API.AssemblyInfo.cs
- archivo BetaProyecto.API.GlobalUsings.g.cs
- archivo BetaProyecto.API.MvcApplicationPartsAssemblyInfo.cs

1.20. Referencia del directorio BetaProyecto/obj/Release/net9.0/win-x64

Archivos

- archivo .NETCoreApp,Version=v9.0.AssemblyAttributes.cs
- archivo BetaProyecto.AssemblyInfo.cs

2. Documentación de espacios de nombres

2.1. Referencia del espacio de nombres BetaProyecto

2.2. Referencia del espacio de nombres BetaProyecto.API

2.3. Referencia del espacio de nombres BetaProyecto.API.Controllers

2.4. Referencia del espacio de nombres BetaProyecto.Helpers

2.5. Referencia del espacio de nombres BetaProyecto.Models

2.6. Referencia del espacio de nombres BetaProyecto.Services

2.7. Referencia del espacio de nombres BetaProyecto.Singleton

2.8. Referencia del espacio de nombres BetaProyecto.ViewModels

3. Documentación de clases

3.1. Referencia de la clase BetaProyecto.Services.AudioService

3.1.1. Detalles

Definición en la línea 11 del archivo AudioService.cs.

Métodos

- AudioService ()
- async Task< InfoCancionNube > ObtenerMp3 (string urlYoutube)

  Solicita de forma asíncrona la extracción y el análisis de metadatos de un recurso de YouTube a través de una API externa.
- async Task< string > ObtenerRutaAudioSegura (string url, string idCancion)

  Gestiona la descarga, protección mediante cifrado y recuperación de archivos de audio en el almacenamiento local.

Clases

- class InfoCancionNube

3.1.2. Constructores

AudioService()

BetaProyecto.Services.AudioService.AudioService ()

Definición en la línea 23 del archivo AudioService.cs.

3.1.3. Funciones

ObtenerMp3()

```
async Task< InfoCancionNube > BetaProyecto.Services.AudioService.ObtenerMp3 (
        string urlYoutube)
```

Solicita de forma asíncrona la extracción y el análisis de metadatos de un recurso de YouTube a través de una API externa.

Este método gestiona la comunicación con el microservicio de streaming mediante los siguientes pasos:

1. Construcción: Genera una URI de consulta codificando la urlYoutube como parámetro.
2. Petición: Realiza una llamada GET utilizando el HttpClient configurado.
3. Procesamiento: Si la respuesta es exitosa, deserializa el cuerpo JSON para extraer la URL del flujo de audio y la duración total en segundos.

En caso de fallo en la red o error en el servidor, captura la excepción y devuelve null para evitar cierres inesperados.

Parámetros

urlYoutube	La dirección URL completa del video de YouTube que se desea procesar.

Devuelve

Un objeto InfoCancionNube con la URL de streaming y la duración; devuelve null si la API no responde correctamente o el recurso no es válido.

Definición en la línea 48 del archivo AudioService.cs.

ObtenerRutaAudioSegura()

```
async Task< string > BetaProyecto.Services.AudioService.ObtenerRutaAudioSegura (
        string url,
        string idCancion)
```

Gestiona la descarga, protección mediante cifrado y recuperación de archivos de audio en el almacenamiento local.

Este método implementa un sistema de seguridad y caché distribuido en dos niveles:

1. Caché persistente (.enc): Si el archivo no existe, se descarga de internet, se cifra mediante AES y se guarda en la carpeta de datos locales de la aplicación.
2. Caché temporal (.mp3): Para permitir la reproducción en el motor de audio (VLC), el archivo se desencripta ”on-the-fly” hacia una ruta temporal volátil.
3. Optimización: Si el archivo ya ha sido procesado previamente, evita la descarga redundante y prioriza la recuperación desde el disco.

En caso de error crítico durante el cifrado o acceso a disco, el método retorna la URL original como mecanismo de contingencia.

Parámetros

url	La dirección URL de origen del flujo de audio.
idCancion	Identificador único de la canción, utilizado para nombrar los archivos en el almacenamiento físico.

Devuelve

Una cadena con la ruta local al archivo MP3 desencriptado listo para su reproducción, o la URL original si ocurre una excepción.

Definición en la línea 99 del archivo AudioService.cs.

3.2. Referencia de la clase BetaProyecto.Models.Canciones

3.2.1. Detalles

Definición en la línea 9 del archivo Canciones.cs.

Propiedades

- string Id [get, set]
- string Titulo [get, set]
- List< string > AutoresIds [get, set]
- string NombreArtista = "Artista Desconocido" [get, set]
- List< string > ListaArtistasIndividuales [get]
- string ImagenPortadaUrl [get, set]
- string UrlCancion [get, set]
- DatosCancion Datos = new DatosCancion() [get, set]
- MetricasCancion Metricas = new MetricasCancion() [get, set]

3.2.2. Propiedades

AutoresIds

List<string> BetaProyecto.Models.Canciones.AutoresIds [get], [set]

Definición en la línea 20 del archivo Canciones.cs.

Datos

DatosCancion BetaProyecto.Models.Canciones.Datos = new DatosCancion() [get], [set]

Definición en la línea 47 del archivo Canciones.cs.

Id

string BetaProyecto.Models.Canciones.Id [get], [set]

Definición en la línea 13 del archivo Canciones.cs.

ImagenPortadaUrl

string BetaProyecto.Models.Canciones.ImagenPortadaUrl [get], [set]

Definición en la línea 40 del archivo Canciones.cs.

ListaArtistasIndividuales

List<string> BetaProyecto.Models.Canciones.ListaArtistasIndividuales [get]

Definición en la línea 28 del archivo Canciones.cs.

Metricas

MetricasCancion BetaProyecto.Models.Canciones.Metricas = new MetricasCancion() [get], [set]

Definición en la línea 50 del archivo Canciones.cs.

NombreArtista

string BetaProyecto.Models.Canciones.NombreArtista = "Artista Desconocido" [get], [set]

Definición en la línea 24 del archivo Canciones.cs.

Titulo

string BetaProyecto.Models.Canciones.Titulo [get], [set]

Definición en la línea 16 del archivo Canciones.cs.

UrlCancion

string BetaProyecto.Models.Canciones.UrlCancion [get], [set]

Definición en la línea 43 del archivo Canciones.cs.

3.3. Referencia de la clase BetaProyecto.ViewModels.CentralTabControlViewModel

3.3.1. Detalles

Definición en la línea 11 del archivo CentralTabControlViewModel.cs.

Métodos

- CentralTabControlViewModel ()

Propiedades

- TabItemInicioViewModel InicioVM [get]
- TabItemBuscadorViewModel BuscadorVM [get]
- TabItemPopularesViewModel PopularesVM [get]
- Action? IrAPerfil [get, set]
- Action? IrACuenta [get, set]
- Action? IrAGestionarCuenta [get, set]
- Action? IrAConfig [get, set]
- Action? IrASobreNosotros [get, set]
- Action? IrAAyuda [get, set]
- Action? IrAPublicarCancion [get, set]
- Action? IrACrearPlaylist [get, set]
- Action< Canciones >? IrADetallesCancion [get, set]
- Action< string >? IrAVerArtista [get, set]
- Action< Canciones >? IrACrearReporte [get, set]
- Action< ListaPersonalizada >? IrADetallesPlaylist [get, set]
- Action< Canciones, List< Canciones > > SolicitudCancion [get, set]
- ReactiveCommand< Unit, Unit > BtnPerfil [get]
- ReactiveCommand< Unit, Unit > BtnCuenta [get]
- ReactiveCommand< Unit, Unit > BtnGestionarCuenta [get]
- ReactiveCommand< Unit, Unit > BtnConfiguracion [get]
- ReactiveCommand< Unit, Unit > BtnSobreNosotros [get]
- ReactiveCommand< Unit, Unit > BtnAyuda [get]
- ReactiveCommand< Unit, Unit > BtnPublicarCancion [get]
- ReactiveCommand< Unit, Unit > BtnCrearPlaylist [get, set]
- ReactiveCommand< Canciones, Unit > BtnReproducir [get]
- string ImagenPerfil [get, set]

3.3.2. Constructores

CentralTabControlViewModel()

BetaProyecto.ViewModels.CentralTabControlViewModel.CentralTabControlViewModel ()

Definición en la línea 54 del archivo CentralTabControlViewModel.cs.

3.3.3. Propiedades

BtnAyuda

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnAyuda [get]

Definición en la línea 39 del archivo CentralTabControlViewModel.cs.

BtnConfiguracion

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnConfiguracion [get]

Definición en la línea 37 del archivo CentralTabControlViewModel.cs.

BtnCrearPlaylist

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnCrearPlaylist [get], [set]

Definición en la línea 41 del archivo CentralTabControlViewModel.cs.

BtnCuenta

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnCuenta [get]

Definición en la línea 35 del archivo CentralTabControlViewModel.cs.

BtnGestionarCuenta

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnGestionarCuenta [get]

Definición en la línea 36 del archivo CentralTabControlViewModel.cs.

BtnPerfil

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnPerfil [get]

Definición en la línea 34 del archivo CentralTabControlViewModel.cs.

BtnPublicarCancion

ReactiveCommand<Unit,Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnPublicarCancion [get]

Definición en la línea 40 del archivo CentralTabControlViewModel.cs.

BtnReproducir

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnReproducir [get]

Definición en la línea 44 del archivo CentralTabControlViewModel.cs.

BtnSobreNosotros

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.CentralTabControlViewModel.BtnSobreNosotros [get]

Definición en la línea 38 del archivo CentralTabControlViewModel.cs.

BuscadorVM

TabItemBuscadorViewModel BetaProyecto.ViewModels.CentralTabControlViewModel.BuscadorVM [get]

Definición en la línea 15 del archivo CentralTabControlViewModel.cs.

ImagenPerfil

string BetaProyecto.ViewModels.CentralTabControlViewModel.ImagenPerfil [get], [set]

Definición en la línea 48 del archivo CentralTabControlViewModel.cs.

InicioVM

TabItemInicioViewModel BetaProyecto.ViewModels.CentralTabControlViewModel.InicioVM [get]

Definición en la línea 14 del archivo CentralTabControlViewModel.cs.

IrAAyuda

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrAAyuda [get], [set]

Definición en la línea 24 del archivo CentralTabControlViewModel.cs.

IrAConfig

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrAConfig [get], [set]

Definición en la línea 22 del archivo CentralTabControlViewModel.cs.

IrACrearPlaylist

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrACrearPlaylist [get], [set]

Definición en la línea 26 del archivo CentralTabControlViewModel.cs.

IrACrearReporte

Action<Canciones>? BetaProyecto.ViewModels.CentralTabControlViewModel.IrACrearReporte [get], [set]

Definición en la línea 29 del archivo CentralTabControlViewModel.cs.

IrACuenta

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrACuenta [get], [set]

Definición en la línea 20 del archivo CentralTabControlViewModel.cs.

IrADetallesCancion

Action<Canciones>? BetaProyecto.ViewModels.CentralTabControlViewModel.IrADetallesCancion [get], [set]

Definición en la línea 27 del archivo CentralTabControlViewModel.cs.

IrADetallesPlaylist

Action<ListaPersonalizada>? BetaProyecto.ViewModels.CentralTabControlViewModel.IrADetallesPlaylist [get], [set]

Definición en la línea 30 del archivo CentralTabControlViewModel.cs.

IrAGestionarCuenta

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrAGestionarCuenta [get], [set]

Definición en la línea 21 del archivo CentralTabControlViewModel.cs.

IrAPerfil

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrAPerfil [get], [set]

Definición en la línea 19 del archivo CentralTabControlViewModel.cs.

IrAPublicarCancion

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrAPublicarCancion [get], [set]

Definición en la línea 25 del archivo CentralTabControlViewModel.cs.

IrASobreNosotros

Action? BetaProyecto.ViewModels.CentralTabControlViewModel.IrASobreNosotros [get], [set]

Definición en la línea 23 del archivo CentralTabControlViewModel.cs.

IrAVerArtista

Action<string>? BetaProyecto.ViewModels.CentralTabControlViewModel.IrAVerArtista [get], [set]

Definición en la línea 28 del archivo CentralTabControlViewModel.cs.

PopularesVM

TabItemPopularesViewModel BetaProyecto.ViewModels.CentralTabControlViewModel.PopularesVM [get]

Definición en la línea 16 del archivo CentralTabControlViewModel.cs.

SolicitudCancion

Action<Canciones, List<Canciones> > BetaProyecto.ViewModels.CentralTabControlViewModel.SolicitudCancion [get], [set]

Definición en la línea 31 del archivo CentralTabControlViewModel.cs.

3.4. Referencia de la clase BetaProyecto.Models.ConfiguracionUser

3.4.1. Detalles

Definición en la línea 81 del archivo Usuarios.cs.

Propiedades

- string DiccionarioTema = ”ModoClaro” [get, set]
- string DiccionarioIdioma = ”Español” [get, set]
- string DiccionarioFuente = ”Lexend” [get, set]

3.4.2. Propiedades

DiccionarioFuente

string BetaProyecto.Models.ConfiguracionUser.DiccionarioFuente = ”Lexend” [get], [set]

Definición en la línea 90 del archivo Usuarios.cs.

DiccionarioIdioma

string BetaProyecto.Models.ConfiguracionUser.DiccionarioIdioma = ”Español” [get], [set]

Definición en la línea 87 del archivo Usuarios.cs.

DiccionarioTema

string BetaProyecto.Models.ConfiguracionUser.DiccionarioTema = ”ModoClaro” [get], [set]

Definición en la línea 84 del archivo Usuarios.cs.

3.5. Referencia de la clase BetaProyecto.Helpers.ControladorDiccionarios

3.5.1. Detalles

Definición en la línea 9 del archivo ControladorDiccionarios.cs.

3.5.2. Funciones

AplicarFuente()

void BetaProyecto.Helpers.ControladorDiccionarios.AplicarFuente (
string fuente) [static]

Cambia dinámicamente la familia tipográfica global de la aplicación cargando el diccionario de estilos correspondiente.

Este método gestiona la identidad visual a través de las fuentes mediante los siguientes pasos:

1. Asignación por Defecto: Si el parámetro fuente es nulo o vacío, se utiliza ”Lexend” como tipografía base del sistema.
2. Normalización de Nombre: Verifica si el nombre de la fuente incluye el prefijo ”Fuente”. Si no es así, lo concatena para coincidir con la nomenclatura de los archivos .axaml de estilos.
3. Construcción de URI: Genera la ruta de acceso al recurso dentro de la carpeta Assets/Styles/.
4. Aplicación de Estilo: Invoca a ReemplazarRecurso para sustituir el diccionario con el alias ”Styles”, actualizando la fuente en toda la interfaz de usuario en tiempo de ejecución.

Parámetros

fuente	El nombre de la fuente o del archivo de estilo (ej. ”Lexend”, ”Roboto”, ”FuenteInter”).

Definición en la línea 80 del archivo ControladorDiccionarios.cs.

AplicarIdioma()

void BetaProyecto.Helpers.ControladorDiccionarios.AplicarIdioma (
string idioma) [static]

Cambia dinámicamente el idioma de la interfaz de usuario cargando el diccionario de recursos de traducción correspondiente.

Este método orquestra la localización de la aplicación mediante los siguientes pasos:

1. Normalización: Si el parámetro idioma no está definido, se establece ”Spanish” como idioma base predeterminado.
2. Construcción de Ruta: Genera una URI de recurso de Avalonia que apunta a los diccionarios de idiomas situados en la carpeta Assets/Language/.
3. Actualización de Recursos: Invoca a ReemplazarRecurso para sustituir el diccionario identificado con la clave ”Language”, desencadenando la actualización inmediata de todas las etiquetas vinculadas mediante el convertidor de localización.

Parámetros

idioma	El nombre del archivo de idioma (sin extensión) que se desea aplicar (ej. "Spanish", "English").

Definición en la línea 61 del archivo ControladorDiccionarios.cs.

AplicarTema()

```
void BetaProyecto.Helpers.ControladorDiccionarios.AplicarTema (
        string tema)   [static]
```

Cambia dinámicamente el aspecto visual de la aplicación cargando y aplicando un diccionario de recursos de tema específico.

Este método gestiona el motor de estilos mediante los siguientes pasos:

1. Validación: Si el parámetro tema es nulo o vacío, se establece "ModoClaro" como valor predeterminado por seguridad.
2. Construcción de URI: Genera una ruta de recurso de Avalonia (URI) apuntando a los archivos .axaml de la carpeta Assets/Interfaces/.
3. Inyección de Estilos: Delega en ReemplazarRecurso para sustituir el diccionario actual identificado con el alias "Interfaces" por el nuevo tema cargado.

Parámetros

tema	El nombre del tema que se desea aplicar (ej. "ModoOscuro", "ModoClaro"). Este debe coincidir con el nombre del archivo .axaml en los activos.

Definición en la línea 43 del archivo ControladorDiccionarios.cs.

CargarConfiguracionInicial()

```
void BetaProyecto.Helpers.ControladorDiccionarios.CargarConfiguracionInicial (
        string tema,
        string idioma,
        string fuente)   [static]
```

Establece el entorno visual y regional de la aplicación al iniciar.

Este método centraliza la carga de preferencias del usuario, invocando secuencialmente las funciones de localización (idioma), estilo (tema oscuro/claro) y tipografía.

Parámetros

tema	Nombre del recurso de estilo a aplicar (ej. "Dark" o "Light").
idioma	Código de cultura o nombre del archivo de traducción.
fuente	Nombre de la familia tipográfica global de la interfaz.

Definición en la línea 22 del archivo ControladorDiccionarios.cs.

ReemplazarRecurso()

void BetaProyecto.Helpers.ControladorDiccionarios.ReemplazarRecurso (
string uriNueva,
string carpetaIdentificadora) [static], [private]

Localiza y sustituye un diccionario de recursos específico dentro de la colección global de la aplicación.

Este método es el motor de la personalización dinámica y opera mediante los siguientes pasos:

1. Acceso Global: Obtiene la instancia actual de la aplicación y accede a su colección Resource↩ Dictionary.MergedDictionaries.
2. Identificación: Escanea los diccionarios cargados buscando un objeto ResourceInclude cuya propiedad Source contenga la cadena definida en carpetaIdentificadora (ej. "Language", "Interfaces", "Styles").
3. Limpieza: Si se encuentra un recurso previo del mismo tipo, se elimina de la colección para evitar conflictos de claves de recursos.
4. Inyección: Instancia un nuevo ResourceInclude con la uriNueva y lo añade a la colección global.

Si la URI es inválida o el archivo no existe, el error se captura en el bloque catch para mantener la estabilidad de la interfaz de usuario.

Parámetros

uriNueva	La ruta absoluta del nuevo archivo AXAML que se desea cargar.
carpetaIdentificadora	La palabra clave (nombre de la subcarpeta) que identifica qué tipo de recurso se está reemplazando.

Definición en la línea 107 del archivo ControladorDiccionarios.cs.

3.6. Referencia de la clase BetaProyecto.Models.DatosCancion

3.6.1. Detalles

Definición en la línea 53 del archivo Canciones.cs.

Propiedades

- int DuracionSegundos [get, set]
- List< string > Generos = new List<string>() [get, set]
- string GenerosTexto [get]
- DateTime FechaLanzamiento [get, set]

3.6.2. Propiedades

DuracionSegundos

int BetaProyecto.Models.DatosCancion.DuracionSegundos [get], [set]

Definición en la línea 56 del archivo Canciones.cs.

FechaLanzamiento

DateTime BetaProyecto.Models.DatosCancion.FechaLanzamiento [get], [set]

Definición en la línea 65 del archivo Canciones.cs.

Generos

List<string> BetaProyecto.Models.DatosCancion.Generos = new List<string>() [get], [set]

Definición en la línea 59 del archivo Canciones.cs.

GenerosTexto

string BetaProyecto.Models.DatosCancion.GenerosTexto [get]

Definición en la línea 62 del archivo Canciones.cs.

3.7. Referencia de la clase BetaProyecto.Services.DialogoService

3.7.1. Detalles

Definición en la línea 9 del archivo DialogoService.cs.

Métodos

- void MostrarAlerta (string mensaje)
- async Task< bool > Preguntar (string titulo, string mensaje, string textoSi, string textoNo)

3.7.2. Funciones

MostrarAlerta()

void BetaProyecto.Services.DialogoService.MostrarAlerta (
string mensaje)

Implementa BetaProyecto.Services.IDialogoService.

Definición en la línea 11 del archivo DialogoService.cs.

Preguntar()

async Task< bool > BetaProyecto.Services.DialogoService.Preguntar (
string titulo,
string mensaje,
string textoSi,
string textoNo)

Implementa BetaProyecto.Services.IDialogoService.

Definición en la línea 20 del archivo DialogoService.cs.

3.8. Referencia de la clase BetaProyecto.Helpers.Encriptador

3.8.1. Detalles

Definición en la línea 12 del archivo Encriptador.cs.

3.8.2. Funciones

DesencriptarArchivo()

```
async Task BetaProyecto.Helpers.Encriptador.DesencriptarArchivo (
            string rutaEntrada,
            string rutaSalida)    [static]
```

Lee un archivo cifrado del almacenamiento local, lo descifra utilizando el algoritmo AES y guarda el resultado en una ubicación temporal.

Este método es el pilar de la recuperación de datos protegidos y opera bajo los siguientes pasos:

1. Carga de Datos: Recupera los bytes cifrados del disco mediante File.ReadAllBytesAsync.
2. Configuración Criptográfica: Reinstancia el motor Aes asegurando el uso de la misma clave y vector de inicialización (IV) empleados durante la encriptación.
3. Procesamiento de Flujo: Utiliza un CryptoStream en modo lectura (CryptoStreamMode.Read) que actúa como un filtro de transformación, convirtiendo los bytes cifrados en datos originales.
4. Persistencia Temporal: Vuelca el flujo descifrado en un nuevo archivo físico (normalmente un .mp3 temporal) para que sea accesible por los servicios de reproducción.

Al utilizar flujos asíncronos, se garantiza que la interfaz de usuario no se bloquee durante el procesamiento de archivos de gran tamaño.

Parámetros

rutaEntrada	La ruta del archivo cifrado (habitualmente con extensión .enc) que se desea procesar.
rutaSalida	La ruta de destino donde se escribirá el archivo resultante ya descifrado.

Devuelve

Una tarea asíncrona que representa el proceso de lectura, descifrado y escritura.

Definición en la línea 123 del archivo Encriptador.cs.

EncriptarBytes()

byte[] BetaProyecto.Helpers.Encriptador.EncriptarBytes (
byte[] byteSinEncrip) [static]

Realiza un cifrado simétrico AES sobre una secuencia de bytes para proteger el contenido de archivos multimedia.

Este proceso de encriptación transforma los datos originales en un formato ilegible mediante los siguientes pasos técnicos:

1. Inicialización: Se instancia el algoritmo Aes y se configuran las propiedades Key (clave secreta) e IV (vector de inicialización).
2. Canalización (Streaming): Se utiliza un CryptoStream como intermediario para procesar los datos a través de un transformador de cifrado.
3. Escritura Segura: Los bytes originales se escriben en el flujo de memoria, donde se aplican las operaciones matemáticas del estándar AES.
4. Finalización: Se ejecuta FlushFinalBlock para procesar los bytes restantes y garantizar la integridad del bloque cifrado.

El resultado es un array de bytes que solo puede ser recuperado mediante el método de desencriptación correspondiente utilizando la misma clave e IV.

Parámetros

byteSinEncrip	El array de bytes original (en texto plano o formato multimedia crudo) que se desea proteger.

Devuelve

Un array de bytes cifrados mediante el estándar AES, listos para ser almacenados de forma segura en el almacenamiento persistente.

Definición en la línea 77 del archivo Encriptador.cs.

HashPassword()

string BetaProyecto.Helpers.Encriptador.HashPassword (
string password) [static]

Aplica un algoritmo de hashing criptográfico SHA-256 a una cadena de texto para proteger información sensible.

Este proceso de seguridad transforma la contraseña mediante los siguientes pasos:

1. Codificación: Convierte la cadena original en una secuencia de bytes utilizando el estándar UTF-8.
2. Cifrado: Utiliza una instancia de SHA256 para calcular un resumen único (hash) de 256 bits.
3. Representación: Transforma los bytes resultantes en una cadena hexadecimal de longitud fija (64 caracteres) mediante un StringBuilder.

4. Gestión de Memoria: Emplea la sentencia using para garantizar la liberación inmediata de los recursos criptográficos en la memoria RAM.

Nota: El hashing es una operación unidireccional; no es posible revertir el resultado para obtener la contraseña original.

Parámetros

password	La contraseña en texto plano que se desea anonimizar.

Devuelve

Una cadena de texto en formato hexadecimal que representa el hash único de la contraseña.

Definición en la línea 36 del archivo Encriptador.cs.

3.9. Referencia de la clase BetaProyecto.Models.EstadisticasUsuario

3.9.1. Detalles

Definición en la línea 62 del archivo Usuarios.cs.

Propiedades

- int NumCancionesSubidas [get, set]

3.9.2. Propiedades

NumCancionesSubidas

int BetaProyecto.Models.EstadisticasUsuario.NumCancionesSubidas [get], [set]

Definición en la línea 65 del archivo Usuarios.cs.

3.10. Referencia de la clase BetaProyecto.Models.Generos

3.10.1. Detalles

Definición en la línea 11 del archivo Generos.cs.

Propiedades

- string Id [get, set]
- string Nombre [get, set]

3.10.2. Propiedades

Id

string BetaProyecto.Models.Generos.Id [get], [set]

Definición en la línea 15 del archivo Generos.cs.

Nombre

string BetaProyecto.Models.Generos.Nombre [get], [set]

Definición en la línea 18 del archivo Generos.cs.

3.11. Referencia de la clase BetaProyecto.Singleton.GlobalData

3.11.1. Detalles

Definición en la línea 7 del archivo GlobalData.cs.

Métodos

- void SetUserData (Usuarios user)

  Sincroniza y mapea la información completa de un objeto Usuarios hacia las propiedades globales de la sesión actual.
- void ClearUserData ()
- Usuarios GetUsuarioObject ()

  Reconstruye y devuelve un objeto de tipo Usuarios integrando todas las propiedades almacenadas en la sesión global.

Propiedades

- static GlobalData Instance [get]
- string UserIdGD [get, set]
- string UsernameGD [get, set]
- string EmailGD [get, set]
- string PasswordGD [get, set]
- string RolGD [get, set]
- string UrlFotoPerfilGD [get, set]
- DateTime FechaNacimientoGD [get, set]
- bool Es_PrivadaGD [get, set]
- string PaisGD [get, set]
- int Num_canciones_subidasGD [get, set]
- List< string > SeguidoresGD [get, set]
- List< string > FavoritosGD [get, set]
- string DiccionarioTemaGD [get, set]
- string DiccionarioIdiomaGD [get, set]
- string DiccionarioFuenteGD [get, set]
- DateTime Fecha_registroGD [get, set]

3.11.2. Constructores

GlobalData()

BetaProyecto.Singleton.GlobalData.GlobalData () [private]

Definición en la línea 180 del archivo GlobalData.cs.

3.11.3. Funciones

ClearUserData()

void BetaProyecto.Singleton.GlobalData.ClearUserData ()

Definición en la línea 106 del archivo GlobalData.cs.

GetUsuarioObject()

Usuarios BetaProyecto.Singleton.GlobalData.GetUsuarioObject ()

Reconstruye y devuelve un objeto de tipo Usuarios integrando todas las propiedades almacenadas en la sesión global.

Este método realiza una operación de ensamblado para convertir las propiedades planas de GlobalData en una estructura jerárquica compleja. Es fundamental para operaciones de persistencia, permitiendo que otros servicios (como el cliente de base de datos) reciban una entidad completa con sus objetos anidados de PerfilUsuario, EstadisticasUsuario, ListasUsuario y ConfiguracionUser.

Devuelve

Una nueva instancia de Usuarios que refleja el estado actual de la sesión del usuario, incluyendo sus preferencias de configuración y listas sociales.

Definición en la línea 137 del archivo GlobalData.cs.

SetUserData()

void BetaProyecto.Singleton.GlobalData.SetUserData (
Usuarios user)

Sincroniza y mapea la información completa de un objeto Usuarios hacia las propiedades globales de la sesión actual.

Este método actúa como un adaptador que distribuye los datos del usuario autenticado en diferentes categorías:

- Datos de Identidad: Mapea ID, nombre, correo y rol directamente desde la raíz del objeto.
- Perfil y Preferencias: Extrae información geográfica, imagen de perfil y estado de privacidad.
- Actividad y Social: Inicializa contadores de estadísticas y asegura que las listas de seguidores y favoritos no sean nulas.

- Configuración de Entorno: Carga los diccionarios de tema, idioma y fuente, aplicando valores por defecto si no existen preferencias guardadas.

Se utiliza principalmente durante el inicio de sesión o tras una actualización exitosa del perfil del usuario para mantener la consistencia en toda la aplicación.

Parámetros

user	El objeto Usuarios recuperado de la base de datos que contiene la información maestra.

Definición en la línea 45 del archivo GlobalData.cs.

3.11.4. Propiedades

DiccionarioFuenteGD

string BetaProyecto.Singleton.GlobalData.DiccionarioFuenteGD [get], [set]

Definición en la línea 27 del archivo GlobalData.cs.

DiccionarioIdiomaGD

string BetaProyecto.Singleton.GlobalData.DiccionarioIdiomaGD [get], [set]

Definición en la línea 26 del archivo GlobalData.cs.

DiccionarioTemaGD

string BetaProyecto.Singleton.GlobalData.DiccionarioTemaGD [get], [set]

Definición en la línea 25 del archivo GlobalData.cs.

EmailGD

string BetaProyecto.Singleton.GlobalData.EmailGD [get], [set]

Definición en la línea 15 del archivo GlobalData.cs.

Es_PrivadaGD

bool BetaProyecto.Singleton.GlobalData.Es_PrivadaGD [get], [set]

Definición en la línea 20 del archivo GlobalData.cs.

FavoritosGD

List<string> BetaProyecto.Singleton.GlobalData.FavoritosGD [get], [set]

Definición en la línea 24 del archivo GlobalData.cs.

Fecha_registroGD

DateTime BetaProyecto.Singleton.GlobalData.Fecha_registroGD [get], [set]

Definición en la línea 28 del archivo GlobalData.cs.

FechaNacimientoGD

DateTime BetaProyecto.Singleton.GlobalData.FechaNacimientoGD [get], [set]

Definición en la línea 19 del archivo GlobalData.cs.

Instance

GlobalData BetaProyecto.Singleton.GlobalData.Instance [static], [get]

Definición en la línea 10 del archivo GlobalData.cs.

Num_canciones_subidasGD

int BetaProyecto.Singleton.GlobalData.Num_canciones_subidasGD [get], [set]

Definición en la línea 22 del archivo GlobalData.cs.

PaisGD

string BetaProyecto.Singleton.GlobalData.PaisGD [get], [set]

Definición en la línea 21 del archivo GlobalData.cs.

PasswordGD

string BetaProyecto.Singleton.GlobalData.PasswordGD [get], [set]

Definición en la línea 16 del archivo GlobalData.cs.

RolGD

string BetaProyecto.Singleton.GlobalData.RolGD [get], [set]

Definición en la línea 17 del archivo GlobalData.cs.

SeguidoresGD

List<string> BetaProyecto.Singleton.GlobalData.SeguidoresGD [get], [set]

Definición en la línea 23 del archivo GlobalData.cs.

UrlFotoPerfilGD

string BetaProyecto.Singleton.GlobalData.UrlFotoPerfilGD [get], [set]

Definición en la línea 18 del archivo GlobalData.cs.

UserIdGD

string BetaProyecto.Singleton.GlobalData.UserIdGD [get], [set]

Definición en la línea 13 del archivo GlobalData.cs.

UsernameGD

string BetaProyecto.Singleton.GlobalData.UsernameGD [get], [set]

Definición en la línea 14 del archivo GlobalData.cs.

3.12. Referencia de la interface BetaProyecto.Services.IDialogoService

3.12.1. Detalles

Definición en la línea 6 del archivo IDialogoService.cs.

Métodos

- void MostrarAlerta (string mensaje)
- Task< bool > Preguntar (string titulo, string mensaje, string textoSi, string textoNo)

3.12.2. Funciones

MostrarAlerta()

void BetaProyecto.Services.IDialogoService.MostrarAlerta (
string mensaje)

Implementado en BetaProyecto.Services.DialogoService.

Preguntar()

Task< bool > BetaProyecto.Services.IDialogoService.Preguntar (
string titulo,
string mensaje,
string textoSi,
string textoNo)

Implementado en BetaProyecto.Services.DialogoService.

3.13. Referencia de la interface BetaProyecto.ViewModels.INavegable

3.13.1. Detalles

Definición en la línea 5 del archivo INavegable.cs.

Propiedades

- Action VolverAtras [get, set]

3.13.2. Propiedades

VolverAtras

Action BetaProyecto.ViewModels.INavegable.VolverAtras [get], [set]

Implementado en BetaProyecto.ViewModels.PanelUsuarioViewModel, BetaProyecto.ViewModels.ViewAyudaViewModel y BetaProyecto.ViewModels.ViewSobreNosotrosViewModel.

Definición en la línea 8 del archivo INavegable.cs.

3.14. Referencia de la clase BetaProyecto.Services.AudioService.InfoCancionNube

3.14.1. Detalles

Definición en la línea 17 del archivo AudioService.cs.

Propiedades

- string Url [get, set]
- int DuracionSegundos [get, set]

3.14.2. Propiedades

DuracionSegundos

int BetaProyecto.Services.AudioService.InfoCancionNube.DuracionSegundos [get], [set]

Definición en la línea 20 del archivo AudioService.cs.

Url

string BetaProyecto.Services.AudioService.InfoCancionNube.Url [get], [set]

Definición en la línea 19 del archivo AudioService.cs.

3.15. Referencia de la clase BetaProyecto.Models.ListaPersonalizada

3.15.1. Detalles

Definición en la línea 11 del archivo ListaPersonalizada.cs.

Propiedades

- string Id [get, set]
- string Nombre [get, set]
- string Descripcion [get, set]
- string UrlPortada [get, set]
- List< string > IdsCanciones = new List<string>() [get, set]
- string IdUsuario [get, set]
- List< Canciones > CancionesCompletas = new List<Canciones>() [get, set]

3.15.2. Propiedades

CancionesCompletas

List<Canciones> BetaProyecto.Models.ListaPersonalizada.CancionesCompletas = new List<Canciones>() [get], [set]

Definición en la línea 37 del archivo ListaPersonalizada.cs.

Descripcion

string BetaProyecto.Models.ListaPersonalizada.Descripcion [get], [set]

Definición en la línea 21 del archivo ListaPersonalizada.cs.

Id

string BetaProyecto.Models.ListaPersonalizada.Id [get], [set]

Definición en la línea 15 del archivo ListaPersonalizada.cs.

IdsCanciones

List<string> BetaProyecto.Models.ListaPersonalizada.IdsCanciones = new List<string>() [get], [set]

Definición en la línea 29 del archivo ListaPersonalizada.cs.

IdUsuario

string BetaProyecto.Models.ListaPersonalizada.IdUsuario [get], [set]

Definición en la línea 33 del archivo ListaPersonalizada.cs.

Nombre

string BetaProyecto.Models.ListaPersonalizada.Nombre [get], [set]

Definición en la línea 18 del archivo ListaPersonalizada.cs.

UrlPortada

string BetaProyecto.Models.ListaPersonalizada.UrlPortada [get], [set]

Definición en la línea 24 del archivo ListaPersonalizada.cs.

3.16. Referencia de la clase BetaProyecto.Models.ListasUsuario

3.16.1. Detalles

Definición en la línea 69 del archivo Usuarios.cs.

Propiedades

- List< string > Seguidores = new List<string>() [get, set]
- List< string > Favoritos = new List<string>() [get, set]

3.16.2. Propiedades

Favoritos

List<string> BetaProyecto.Models.ListasUsuario.Favoritos = new List<string>() [get], [set]

Definición en la línea 78 del archivo Usuarios.cs.

Seguidores

List<string> BetaProyecto.Models.ListasUsuario.Seguidores = new List<string>() [get], [set]

Definición en la línea 74 del archivo Usuarios.cs.

3.17. Referencia de la clase BetaProyecto.Models.ListaUsuarios

3.17.1. Detalles

Definición en la línea 10 del archivo ListaUsuarios.cs.

Métodos

- ListaUsuarios (string titulo, ObservableCollection< Usuarios > lista)

Propiedades

- string Titulo [get, set]
- ObservableCollection< Usuarios > Lista [get, set]

3.17.2. Constructores

ListaUsuarios()

```
BetaProyecto.Models.ListaUsuarios.ListaUsuarios (
            string titulo,
            ObservableCollection< Usuarios > lista)
```

Definición en la línea 15 del archivo ListaUsuarios.cs.

3.17.3. Propiedades

Lista

```
ObservableCollection<Usuarios> BetaProyecto.Models.ListaUsuarios.Lista [get], [set]
```

Definición en la línea 13 del archivo ListaUsuarios.cs.

Titulo

```
string BetaProyecto.Models.ListaUsuarios.Titulo [get], [set]
```

Definición en la línea 12 del archivo ListaUsuarios.cs.

3.18. Referencia de la clase BetaProyecto.ViewModels.LoginViewModel

3.18.1. Detalles

Definición en la línea 11 del archivo LoginViewModel.cs.

Métodos

- LoginViewModel ()
- LoginViewModel (IDialogoService dialogoService)

Propiedades

- Action? AlCompletarLogin [get, set]
- Action? IrARegistarUser [get, set]
- string TxtUsuario [get, set]
- string TxtPass [get, set]
- ReactiveCommand< Unit, Unit > Login [get]
- ReactiveCommand< Unit, Unit > BtnRegistarUser [get]

3.18.2. Constructores

LoginViewModel() [1/2]

BetaProyecto.ViewModels.LoginViewModel.LoginViewModel ()

Definición en la línea 41 del archivo LoginViewModel.cs.

LoginViewModel() [2/2]

BetaProyecto.ViewModels.LoginViewModel.LoginViewModel (
IDialogoService dialogoService)

Definición en la línea 45 del archivo LoginViewModel.cs.

3.18.3. Funciones

IntentarLogin()

async Task BetaProyecto.ViewModels.LoginViewModel.IntentarLogin () [private]

Intenta iniciar sesión al usuario conectándose a la base de datos y validando las credenciales proporcionadas.

Si el inicio de sesión tiene éxito, se cargan los datos del usuario y la configuración, y la vista es notificado para proceder. Si el inicio de sesión falla debido a credenciales incorrectas o problemas de conexión, una alerta es se muestra para informar al usuario. Este método no devuelve un resultado; realiza efectos secundarios como actualización del estado global y visualización de alertas.

Devuelve

Devuelve una tarea que representa la operación de inicio de sesión asíncrono. La tarea se completa cuando el intento de inicio de sesión tiene terminado, independientemente del éxito o el fracaso.

Definición en la línea 72 del archivo LoginViewModel.cs.

3.18.4. Propiedades

AlCompletarLogin

Action? BetaProyecto.ViewModels.LoginViewModel.AlCompletarLogin [get], [set]

Definición en la línea 17 del archivo LoginViewModel.cs.

BtnRegistarUser

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.LoginViewModel.BtnRegistarUser [get]

Definición en la línea 38 del archivo LoginViewModel.cs.

IrARegistarUser

Action? BetaProyecto.ViewModels.LoginViewModel.IrARegistarUser [get], [set]

Definición en la línea 18 del archivo LoginViewModel.cs.

Login

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.LoginViewModel.Login [get]

Definición en la línea 37 del archivo LoginViewModel.cs.

TxtPass

string BetaProyecto.ViewModels.LoginViewModel.TxtPass [get], [set]

Definición en la línea 29 del archivo LoginViewModel.cs.

TxtUsuario

string BetaProyecto.ViewModels.LoginViewModel.TxtUsuario [get], [set]

Definición en la línea 22 del archivo LoginViewModel.cs.

3.19. Referencia de la clase BetaProyecto.ViewModels.MarcoAppViewModel

3.19.1. Detalles

Definición en la línea 18 del archivo MarcoAppViewModel.cs.

Métodos

- MarcoAppViewModel ()
- void IrACrearUsuario ()

  Muestra la vista de creación del usuario como una ventana emergente, lo que permite al usuario crear una nueva cuenta.
- void IrAlCentralTabControl ()

  Navega a la vista de control de pestaña central, inicializándola si es necesario y configurándola como la vista actual.
- void IrAPanelUsuario (int pestania)

  Navega al panel de usuario y muestra la pestaña especificada.
- void IrASobreNosotros ()

  Navega a la vista 'Sobre Nosotros', creándola y configurándola si no existe.
- void IrAAyuda ()

  Muestra la vista de ayuda en la aplicación, creándola y configurándola si no existe.
- void ActivarVolverAtras (INavegable vm)

  Asigna una llamada de retorno al modelo de vista especificado que permite la navegación de regreso a la vista principal.
- void ReproducirCancion (Canciones cancion, List< Canciones >? listaOrigen=null)

  Inicia la reproducción de la canción especificada, opcionalmente usando una lista de reproducción proporcionada como cola de reproducción.
- async void CargarYReproducir (Canciones cancion)

  Carga la canción especificada y comienza la reproducción, actualizando la interfaz del reproductor y el estado relacionado en consecuencia.

Propiedades

- string NombreCancionActual [get, set]
- string NombreArtistaActual [get, set]
- string ImagenCancionActual [get, set]
- string IconoPlayPause [get, set]
- string IconoNext [get, set]
- string IconoBack [get, set]
- string IconoAleatorio [get, set]
- string IconoLike [get, set]
- string TiempoActualCancion [get, set]
- string TiempoTotalCancion [get, set]
- double ValorSliderCancion [get, set]
- double ValorSliderVolumen [get, set]
- ViewModelBase VistaActual [get, set]
- ViewModelBase PopupActual [get, set]
- bool PopupVisible [get, set]
- bool BarraVisible [get, set]
- ReactiveCommand< Unit, Unit > BtnPlayPauseCommand [get]
- ReactiveCommand< Unit, Unit > BtnFavCommand [get]
- ReactiveCommand< Unit, Unit > BtnNextCommand [get]
- ReactiveCommand< Unit, Unit > BtnBackCommand [get]
- ReactiveCommand< Unit, Unit > BtnAleatorioCommand [get]

3.19.2. Constructores

MarcoAppViewModel()

BetaProyecto.ViewModels.MarcoAppViewModel.MarcoAppViewModel ()

Definición en la línea 194 del archivo MarcoAppViewModel.cs.

3.19.3. Funciones

AccionarPlayPause()

void BetaProyecto.ViewModels.MarcoAppViewModel.AccionarPlayPause () [private]

Cambia la reproducción del reproductor multimedia entre los estados de reproducción y pausa.

Si el reproductor multimedia está reproduciéndose, este método detiene la reproducción y actualiza el reproducir/pausa el icono en consecuencia. Si el reproductor multimedia se detiene, este método inicia la reproducción y actualiza el icono, y inicia el temporizador asociado. Este método no genera excepciones y asume que el reproductor multimedia y los temporizadores están correctamente inicializados.

Definición en la línea 685 del archivo MarcoAppViewModel.cs.

ActivarVolverAtras()

void BetaProyecto.ViewModels.MarcoAppViewModel.ActivarVolverAtras (
INavegable vm)

Asigna una llamada de retorno al modelo de vista especificado que permite la navegación de regreso a la vista principal.

Utilice este método para proporcionar un comportamiento de retorno estándar para los modelos de vista que admiten navegación. Después de invocar el callback asignado, la barra de navegación principal se vuelve visible.

Parámetros

vm	El modelo de vista que implementa la interfaz INavegable. El método establece su acción Volver↩ Atras para navegar de vuelta al control central de pestañas.

Definición en la línea 534 del archivo MarcoAppViewModel.cs.

ActualizarIconoAleatorio()

void BetaProyecto.ViewModels.MarcoAppViewModel.ActualizarIconoAleatorio (
Canciones cancionInicio) [private]

Actualiza el icono de reproducción aleatoria según la cola de reproducción actual y el estado del modo aleatorio.

Si la cola de reproducción contiene una o ninguna canción, el icono aleatorio está desactivado. Cuando el el modo aleatorio está activo y la cola cambia, el historial de reproducción aleatoria se restablece para comenzar desde el canción especificada.

Parámetros

cancionInicio	La canción que se usará como punto de partida en el historial de reproducción aleatoria cuando el modo aleatorio esté activo.

Definición en la línea 959 del archivo MarcoAppViewModel.cs.

ActualizarIconoNextBack()

void BetaProyecto.ViewModels.MarcoAppViewModel.ActualizarIconoNextBack () [private]

Actualiza los iconos de los botones de navegación Siguiente y Atrás en función del modo y la posición de reproducción actuales en la lista de reproducción.

Este método habilita o deshabilita los iconos de los botones Siguiente y Atrás, dependiendo de si la lista de reproducción está en modo de mezcla y la posición actual de la canción. En el modo de mezcla, el botón Siguiente permanece activado, mientras que el botón Atrás solo está activado si hay un historial de reproducción. En modo normal, los botones están activado o desactivado según si la canción actual es la primera o la última en la lista de reproducción.

Definición en la línea 907 del archivo MarcoAppViewModel.cs.

AlterarFavorito()

async Task BetaProyecto.ViewModels.MarcoAppViewModel.AlterarFavorito (
Canciones cancion) [private]

Añade o elimina la canción especificada de la lista de favoritos del usuario, actualizando el estado favorito en consecuencia.

Si la canción ya está en la lista de favoritos, se elimina; de lo contrario, se añade. El método también actualiza el icono de like y ajusta la métrica de like count de la canción. No se realiza ninguna acción si el la canción proporcionada es nula.

Parámetros

cancion	La canción a añadir o quitar de la lista de favoritos. Si es nula, no se realiza la operación.

Devuelve

Devuelve una tarea que representa la operación asíncrona.

Definición en la línea 755 del archivo MarcoAppViewModel.cs.

AlternarAleatorio()

void BetaProyecto.ViewModels.MarcoAppViewModel.AlternarAleatorio () [private]

Cambia el modo de reproducción aleatoria para la lista de reproducción actual. Cuando está activado, el orden de reproducción se aleatoriza; cuando está desactivado, la reproducción se reanuda en el orden original de la canción actual.

Este método no tiene efecto si la lista de reproducción es nula o contiene uno o menos elementos. Cuando se activa el modo aleatorio, la canción actual se añade al historial de reproducción aleatoria para permitir su retorno a ella. Cuando se desactiva, la reproducción continúa desde la posición actual de la canción en la lista de reproducción original orden.

Definición en la línea 707 del archivo MarcoAppViewModel.cs.

BackCancion()

void BetaProyecto.ViewModels.MarcoAppViewModel.BackCancion () [private]

Mueve la reproducción a la pista anterior en la lista de reproducción o al historial de reproducción, dependiendo de la reproducción actual modo.

En el modo de mezcla, este método navega hacia atrás a través del historial de reproducción si posible. En el modo normal, se mueve a la pista anterior en la lista de reproducción a menos que ya esté en la primera pista. No se toma ninguna medida si no hay pistas en la lista de reproducción o si ya está al principio del historial o lista de reproducción.

Definición en la línea 871 del archivo MarcoAppViewModel.cs.

CargarYReproducir()

async void BetaProyecto.ViewModels.MarcoAppViewModel.CargarYReproducir (
Canciones cancion)

Carga la canción especificada y comienza la reproducción, actualizando la interfaz del reproductor y el estado relacionado en consecuencia.

Este método actualiza la información actual de la canción, los controles de reproducción y el usuario elementos de interfaz para reflejar la canción cargada. Determina la fuente de audio apropiada en función del URL de la canción, que admite enlaces a archivos en la nube tanto de YouTube como directos. Si la canción está marcada como favorita, el el icono like se actualiza. La reproducción se inicia de forma asíncrona y las métricas de reproducción de canciones se incrementan en el fondo. Si no se puede cargar el audio, se muestra una alerta al usuario. Este método es asincrónico. pero devuelve nulo; las excepciones se capturan y registran internamente.

Parámetros

cancion	La canción a cargar y reproducir. No debe ser nula y debe contener metadatos válidos y una URL reproducible.

Definición en la línea 586 del archivo MarcoAppViewModel.cs.

CerrarAplicacion()

void BetaProyecto.ViewModels.MarcoAppViewModel.CerrarAplicacion () [private]

Realiza un cierre limpio de la aplicación, finalizando los procesos relacionados, liberando recursos, y cerrando la ventana de la aplicación.

Este método termina por la fuerza cualquier instancia en ejecución de 'BetaProyecto.API' proceso, descarte de recursos multimedia, detiene temporizadores internos y cierra la aplicación. Si el la aplicación se ejecuta con una vida útil de escritorio clásica, utiliza el mecanismo de apagado apropiado; de lo contrario, abandona el proceso. Utilice este método para asegurarse de que todos los recursos se liberan y la aplicación salidas limpias.

Definición en la línea 1109 del archivo MarcoAppViewModel.cs.

CerrarSesion()

void BetaProyecto.ViewModels.MarcoAppViewModel.CerrarSesion () [private]

Cierra la sesión actual del usuario y restablece el estado de la aplicación a la vista de inicio de sesión.

Este método detiene cualquier reproducción de medios activa, borra datos específicos del usuario y restablece la interfaz de usuario. elementos y elimina la información de usuario almacenada en caché para mayor seguridad. Después de la ejecución, la aplicación regresa al pantalla de inicio de sesión, asegurándose de que ningún dato de sesiones anteriores permanezca accesible. Este método debe ser llamado cuando un el usuario se desconecta o cuando una sesión debe finalizar de forma segura.

Definición en la línea 1070 del archivo MarcoAppViewModel.cs.

IrAAyuda()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAAyuda ()

Muestra la vista de ayuda en la aplicación, creándola y configurándola si no existe.

Cuando se invoca, este método cambia la vista actual a la vista de ayuda y oculta el barra de aplicaciones. Si la vista de ayuda no se ha creado previamente, se inicializa y configura antes se está mostrando.

Definición en la línea 372 del archivo MarcoAppViewModel.cs.

IrACrearPlaylist()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrACrearPlaylist () [private]

Navega a la vista para crear una nueva lista de reproducción personalizada y restablece el formulario de creación de la lista de reproducción.

Este método reemplaza la vista actual con la vista de creación de listas de reproducción y oculta los barra de navegación. Cuando el usuario regresa desde la vista de creación de listas de reproducción, la vista original y la barra de navegación se restauran.

Definición en la línea 413 del archivo MarcoAppViewModel.cs.

IrACrearReporte()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrACrearReporte (
Canciones cancion) [private]

Muestra la vista de creación del informe para la canción especificada.

Parámetros

cancion	La canción para la que se mostrará la vista de creación del informe. No puede ser nula.

Definición en la línea 468 del archivo MarcoAppViewModel.cs.

IrACrearUsuario()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrACrearUsuario ()

Muestra la vista de creación del usuario como una ventana emergente, lo que permite al usuario crear una nueva cuenta.

Este método reemplaza la ventana emergente actual con la vista de creación del usuario. Para cerrar el popup y volver al estado anterior, usar la acción de retroalimentación proporcionada dentro de la creación del usuario

Definición en la línea 264 del archivo MarcoAppViewModel.cs.

IrADetallesCancion()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrADetallesCancion (
Canciones cancion) [private]

Muestra la vista de detalles para la canción especificada y establece acciones relacionadas como reproducir, marcar como favorita y devolver.

Parámetros

cancion	La canción para la que se muestran los detalles. No puede ser nula.

Definición en la línea 433 del archivo MarcoAppViewModel.cs.

IrADetallesPlaylist()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrADetallesPlaylist (
ListaPersonalizada playlist) [private]

Definición en la línea 480 del archivo MarcoAppViewModel.cs.

IrAEditarCancion()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAEditarCancion (
Canciones cancion) [private]

Muestra la vista de edición de canciones para la canción especificada.

Parámetros

cancion	La canción a editar. No puede ser nula.

Definición en la línea 496 del archivo MarcoAppViewModel.cs.

IrAEditarPlaylist()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAEditarPlaylist (
ListaPersonalizada playlist) [private]

Muestra la vista de edición de la lista de reproducción personalizada especificada, permitiendo al usuario modificar sus detalles.

Parámetros

playlist	La lista de reproducción personalizada que se va a editar. No puede ser nula.

Definición en la línea 512 del archivo MarcoAppViewModel.cs.

IrAlCentralTabControl()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAlCentralTabControl ()

Navega a la vista de control de pestaña central, inicializándola si es necesario y configurándola como la vista actual.

Si la vista de control de pestaña central no se ha creado, este método la inicializa y configura sus acciones de navegación. Las llamadas posteriores reutilizarán la instancia de vista existente. Este método también actualiza los iconos de la interfaz como parte del proceso de navegación.

Definición en la línea 280 del archivo MarcoAppViewModel.cs.

IrAPanelUsuario()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAPanelUsuario (
int pestania)

Navega al panel de usuario y muestra la pestaña especificada.

Si el panel de usuario no se ha creado, este método lo inicializa y configura acciones relacionadas. Si el panel ya existe, actualiza la pestaña mostrada. La navegación oculta los principales barra mientras el panel de usuario está activo.

Parámetros

pestania	El índice de la pestaña que se mostrará en el panel de usuario. Debe ser un índice de pestañas válido y compatible con el panel.

Definición en la línea 315 del archivo MarcoAppViewModel.cs.

IrAPublicarCancion()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAPublicarCancion () [private]

Navega a la vista para publicar una nueva canción y actualiza el estado actual de la vista en consecuencia.

Este método reemplaza la vista actual con la vista de publicación de canciones y oculta los barra de navegación. Cuando el usuario regresa desde la vista de publicación, se restaura la vista original del control de pestañas. y la barra de navegación vuelve a ser visible.

Definición en la línea 391 del archivo MarcoAppViewModel.cs.

IrASobreNosotros()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrASobreNosotros ()

Navega a la vista 'Sobre Nosotros', creándola y configurándola si no existe.

Si no se ha creado la vista 'Sobre Nosotros', este método la inicializa y lo establece como la vista actual. Las llamadas posteriores reutilizarán la instancia de vista existente. La barra de música está oculta mientras esta vista está activa.

Definición en la línea 353 del archivo MarcoAppViewModel.cs.

IrAVerArtista()

void BetaProyecto.ViewModels.MarcoAppViewModel.IrAVerArtista (
string idUsuario) [private]

Muestra los detalles del usuario especificado en una vista emergente.

Parámetros

idUsuario	El identificador único del usuario cuyos detalles se deben mostrar. No puede ser nulo o vacío.

Definición en la línea 452 del archivo MarcoAppViewModel.cs.

LimpiarArchivoTemporal()

void BetaProyecto.ViewModels.MarcoAppViewModel.LimpiarArchivoTemporal () [private]

Elimina el archivo temporal actual si existe y libera cualquier recurso multimedia asociado.

Si el archivo temporal está en uso o no se puede eliminar, el método suprime excepciones y registra un mensaje de depuración. Este método está destinado a ser llamado cuando los archivos multimedia temporales no están se necesita más tiempo para liberar espacio en disco y liberar atajos de archivos.

Definición en la línea 1154 del archivo MarcoAppViewModel.cs.

NextCancion()

void BetaProyecto.ViewModels.MarcoAppViewModel.NextCancion () [private]

Avanza a la reproducción de la siguiente canción en la lista de reproducción, manejando tanto el modo secuencial como el de mezcla.

En el modo de mezcla, este método selecciona aleatoriamente la siguiente canción entre las que aún no están disponibles. jugado, manteniendo un historial para evitar repeticiones hasta que se hayan reproducido todas las canciones. En el modo secuencial, avanza a la siguiente canción en orden si está disponible. Si todas las canciones se han reproducido en modo de mezcla, el historial se restablece excepto para la canción actual, y la reproducción continúa. Este método no tiene efecto si la lista de reproducción es vacío.

Definición en la línea 804 del archivo MarcoAppViewModel.cs.

RefrescarIconos()

void BetaProyecto.ViewModels.MarcoAppViewModel.RefrescarIconos () [private]

Actualiza los iconos y las propiedades relacionadas para reflejar el tema de la aplicación actual.

Llame a este método después de que cambie el tema de la aplicación para asegurarse de que todos los iconos y las propiedades del deslizador se actualizan y notifican cualquier vinculación de datos de los cambios.

Definición en la línea 1052 del archivo MarcoAppViewModel.cs.

ReproducirCancion()

void BetaProyecto.ViewModels.MarcoAppViewModel.ReproducirCancion (
Canciones cancion,
List< Canciones >? listaOrigen = null)

Inicia la reproducción de la canción especificada, opcionalmente usando una lista de reproducción proporcionada como cola de reproducción.

Si se proporciona una lista de reproducción, la reproducción comenzará con la canción especificada dentro de esa lista. lista. De lo contrario, la reproducción se limita a la canción individual proporcionada. La cola de reproducción y el índice de canciones actual se actualizan en consecuencia.

Parámetros

cancion	La canción a reproducir. No puede ser nula.
listaOrigen	Una lista opcional de canciones para usar como cola de reproducción. Si es nula o vacía, solo la canción especificada será jugado.

Definición en la línea 554 del archivo MarcoAppViewModel.cs.

Timer_Tick()

void BetaProyecto.ViewModels.MarcoAppViewModel.Timer_Tick (
object? sender,
EventArgs e) [private]

Maneja las marcas de eventos del temporizador para actualizar el progreso de reproducción, la hora actual y la duración total del medio reproductor, o para avanzar a la siguiente pista cuando termina la reproducción.

Este método está destinado a ser utilizado como un manejador de eventos para un temporizador periódico. asociado con la reproducción de medios. Actualiza los elementos de la interfaz de usuario, como el deslizador de progreso y las pantallas de tiempo. respuesta al estado actual del reproductor multimedia. Si la reproducción ha terminado, avanza automáticamente a la siguiente pista o restablece la interfaz de usuario según sea apropiado.

Parámetros

sender	La fuente del evento, normalmente el temporizador que activó la marca.
e	Un objeto que contiene los datos del evento.

Definición en la línea 998 del archivo MarcoAppViewModel.cs.

3.19.4. Propiedades

BarraVisible

bool BetaProyecto.ViewModels.MarcoAppViewModel.BarraVisible [get], [set]

Definición en la línea 180 del archivo MarcoAppViewModel.cs.

BtnAleatorioCommand

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.MarcoAppViewModel.BtnAleatorioCommand [get]

Definición en la línea 191 del archivo MarcoAppViewModel.cs.

BtnBackCommand

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.MarcoAppViewModel.BtnBackCommand [get]

Definición en la línea 190 del archivo MarcoAppViewModel.cs.

BtnFavCommand

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.MarcoAppViewModel.BtnFavCommand [get]

Definición en la línea 188 del archivo MarcoAppViewModel.cs.

BtnNextCommand

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.MarcoAppViewModel.BtnNextCommand [get]

Definición en la línea 189 del archivo MarcoAppViewModel.cs.

BtnPlayPauseCommand

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.MarcoAppViewModel.BtnPlayPauseCommand [get]

Definición en la línea 187 del archivo MarcoAppViewModel.cs.

IconoAleatorio

string BetaProyecto.ViewModels.MarcoAppViewModel.IconoAleatorio [get], [set]

Definición en la línea 96 del archivo MarcoAppViewModel.cs.

IconoBack

string BetaProyecto.ViewModels.MarcoAppViewModel.IconoBack [get], [set]

Definición en la línea 89 del archivo MarcoAppViewModel.cs.

IconoLike

string BetaProyecto.ViewModels.MarcoAppViewModel.IconoLike [get], [set]

Definición en la línea 103 del archivo MarcoAppViewModel.cs.

IconoNext

string BetaProyecto.ViewModels.MarcoAppViewModel.IconoNext [get], [set]

Definición en la línea 82 del archivo MarcoAppViewModel.cs.

IconoPlayPause

string BetaProyecto.ViewModels.MarcoAppViewModel.IconoPlayPause [get], [set]

Definición en la línea 75 del archivo MarcoAppViewModel.cs.

ImagenCancionActual

string BetaProyecto.ViewModels.MarcoAppViewModel.ImagenCancionActual [get], [set]

Definición en la línea 68 del archivo MarcoAppViewModel.cs.

NombreArtistaActual

string BetaProyecto.ViewModels.MarcoAppViewModel.NombreArtistaActual [get], [set]

Definición en la línea 61 del archivo MarcoAppViewModel.cs.

NombreCancionActual

string BetaProyecto.ViewModels.MarcoAppViewModel.NombreCancionActual [get], [set]

Definición en la línea 54 del archivo MarcoAppViewModel.cs.

PopupActual

ViewModelBase BetaProyecto.ViewModels.MarcoAppViewModel.PopupActual [get], [set]

Definición en la línea 159 del archivo MarcoAppViewModel.cs.

PopupVisible

bool BetaProyecto.ViewModels.MarcoAppViewModel.PopupVisible [get], [set]

Definición en la línea 171 del archivo MarcoAppViewModel.cs.

TiempoActualCancion

string BetaProyecto.ViewModels.MarcoAppViewModel.TiempoActualCancion [get], [set]

Definición en la línea 111 del archivo MarcoAppViewModel.cs.

TiempoTotalCancion

string BetaProyecto.ViewModels.MarcoAppViewModel.TiempoTotalCancion [get], [set]

Definición en la línea 118 del archivo MarcoAppViewModel.cs.

ValorSliderCancion

double BetaProyecto.ViewModels.MarcoAppViewModel.ValorSliderCancion [get], [set]

Definición en la línea 126 del archivo MarcoAppViewModel.cs.

ValorSliderVolumen

double BetaProyecto.ViewModels.MarcoAppViewModel.ValorSliderVolumen [get], [set]

Definición en la línea 135 del archivo MarcoAppViewModel.cs.

VistaActual

ViewModelBase BetaProyecto.ViewModels.MarcoAppViewModel.VistaActual [get], [set]

Definición en la línea 151 del archivo MarcoAppViewModel.cs.

3.20. Referencia de la clase BetaProyecto.Models.MetricasCancion

3.20.1. Detalles

Definición en la línea 68 del archivo Canciones.cs.

Propiedades

- long TotalReproducciones = 0 [get, set]
- long TotalMegustas = 0 [get, set]
- double PuntuacionTendencia = 0.0 [get, set]

3.20.2. Propiedades

PuntuacionTendencia

double BetaProyecto.Models.MetricasCancion.PuntuacionTendencia = 0.0 [get], [set]

Definición en la línea 77 del archivo Canciones.cs.

TotalMegustas

long BetaProyecto.Models.MetricasCancion.TotalMegustas = 0 [get], [set]

Definición en la línea 74 del archivo Canciones.cs.

TotalReproducciones

long BetaProyecto.Models.MetricasCancion.TotalReproducciones = 0 [get], [set]

Definición en la línea 71 del archivo Canciones.cs.

3.21. Referencia de la clase BetaProyecto.Services.MongoAtlas

3.21.1. Detalles

Definición en la línea 13 del archivo MongoAtlas.cs.

Métodos

- MongoAtlas ()
- async Task< bool > Conectar ()
- async Task< Usuarios > LoginUsuario (string username, string password)
- async Task< List< Canciones > > ObtenerCancionesFavoritos ()
- async Task< List< Canciones > > ObtenerCacionesNovedades ()
- async Task< List< Canciones > > ObtenerCancionesPorGenero (string genero)
- async Task< List< Canciones > > ObtenerCanciones ()
- async Task< List< Canciones > > ObtenerCancionesPorListaIds (List< string > ids)
- async Task< List< Canciones > > ObtenerCancionesPorBusqueda (string textoBusqueda)
- async Task< List< Canciones > > ObtenerCancionesPorAutor (string idAutor)
- async Task< List< Usuarios > > ObtenerTodosLosUsuarios ()
- async Task< List< Usuarios > > ObtenerUsuariosPorBusqueda (string textoBusqueda, List< string > idsExcluidos)
- async Task< List< Usuarios > > ObtenerUsuariosPorListaIds (List< string > listaIds)
- async Task< List< ListaPersonalizada > > ObtenerListasReproduccion ()
- async Task< List< ListaPersonalizada > > ObtenerPlaylistsPorCreador (string idUsuario)
- async Task< List< Reportes > > ObtenerReportes ()

  Se ocupa de traer todos los reportes de la base de datos.
- async Task< List< string > > ObtenerNombresGeneros ()

  Obtenemos la lista de nombres de géneros de la base de datos.
- async Task< List< Generos > > ObtenerGenerosCompletos ()

  Obtenemos los objetos completos de géneros de la base de datos.
- async Task< List< Canciones > > ObtenerMixPorGenero (string genero)

  Obtiene una mezcla de canciones por género (veteranos + indies) partir del puntaje de tendencia.
- async Task< bool > AgregarAFavorito (string idUsuario, string idCancion)

  Se ocupa de añadir una canción a la lista de favoritos del usuario en la base de datos.
- async Task< bool > SeguirUsuario (string idUsuario, string idUsuarioASeguir)

  Se ocupa de añadir un usuario a la lista de seguidores del usuario en la base de datos.
- async Task< bool > PublicarCancion (Canciones nuevaCancion)

  Se ocupa de publicar una nueva canción en la base de datos.

- async Task< bool > CrearListaReproduccion (ListaPersonalizada nuevaLista)

  Se ocupa de crear una nueva listapersonalizada en la base de datos.
- async Task< bool > EnviarReporte (Reportes nuevoReporte)

  Se ocupa de enviar un nuevo reporte a la base de datos.
- async Task< bool > CrearGenero (string nuevoGenero)

  Se ocupa de crear un nuevo género en la base de datos.
- async Task< bool > CrearUsuario (Usuarios nuevoUsuario)

  Se ocupa de crear un nuevo usuario en la base de datos.
- async Task< bool > EliminarDeFavorito (string idUsuario, string idCancion)

  Se ocupa de eliminar una canción de la lista de favoritos del usuario en la base de datos.
- async Task< bool > DejarDeSeguirUsuario (string idUsuario, string idUsuarioADejar)

  Se ocupa de eliminar un usuario de la lista de seguidores del usuario en la base de datos.
- async Task< bool > EliminarCancionPorId (string idCancion)

  Se ocupa de eliminar una canción de la base de datos por id.
- async Task< bool > EliminarPlaylistPorId (string idPlaylist)

  Eliminamos la listapersonalizada de la base de datos por id.
- async Task< bool > EliminarGenero (Generos generoAEliminar)

  Eliminamos el genero de la base de datos.
- async Task< bool > EliminarUsuario (string idUsuario)

  Eliminamos al usuarios de la base de datos.
- async Task< bool > EliminarReporte (string idReporte)

  Elimina un reporte del la base de datos.
- async Task< bool > ActualizarPerfilUsuario (string idUsuario, string nuevoNombre, string nuevo↩ Email, string nuevoPais, DateTime nuevaFecha, bool esPrivada)

  Actualizamos el usuario logeado mediantes comprobaciones con el singleton.
- async Task< bool > ActualizarUsuario (string id, string nombre, string email, string password, string rol, string pais, string imagenUrl, DateTime fecha, bool esPrivada)

  Actualiza un usaurio de la base de datos.
- async Task< bool > ActualizarConfiguracionUsuario (string idUsuario, ConfiguracionUser nueva↩ Config)
- async Task< bool > ActualizarPlaylist (string nuevoNombre, string nuevaDesc, List< string > nuevasCanciones, string nuevaPortada, ListaPersonalizada original)
- async Task< bool > ActualizarCancion (string nuevoTitulo, string nuevaPortada, List< string > nuevosAutores, List< string > nuevosGeneros, Canciones original)
- async Task< bool > ActualizarEstadoReporte (string nuevoEstado, string nuevaResolucion, Reportes original)
- async Task IncrementarMetricaCancion (string idCancion, string campo, int cantidad)
- async Task IncrementarContadorCancionesUsuario (string idUsuario, int cantidad)
- async Task< bool > ActualizarGenero (string id, string nuevoNombre)

Propiedades

- IMongoDatabase? Database [get, private set]

3.21.2. Constructores

MongoAtlas()

BetaProyecto.Services.MongoAtlas.MongoAtlas ()

Definición en la línea 24 del archivo MongoAtlas.cs.

3.21.3. Funciones

ActualizarCancion()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarCancion (
                string nuevoTitulo,
                string nuevaPortada,
                List< string > nuevosAutores,
                List< string > nuevosGeneros,
                Canciones original)
```

Definición en la línea 1297 del archivo MongoAtlas.cs.

ActualizarConfiguracionUsuario()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarConfiguracionUsuario (
                string idUsuario,
                ConfiguracionUser nuevaConfig)
```

Definición en la línea 1197 del archivo MongoAtlas.cs.

ActualizarEstadoReporte()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarEstadoReporte (
                string nuevoEstado,
                string nuevaResolucion,
                Reportes original)
```

Definición en la línea 1359 del archivo MongoAtlas.cs.

ActualizarGenero()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarGenero (
                string id,
                string nuevoNombre)
```

Definición en la línea 1512 del archivo MongoAtlas.cs.

ActualizarPerfilUsuario()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarPerfilUsuario (
                string idUsuario,
                string nuevoNombre,
                string nuevoEmail,
                string nuevoPais,
                DateTime nuevaFecha,
                bool esPrivada)
```

Actualizamos el usuario logeado mediantes comprobaciones con el singleton.

Parámetros

idUsuario	
nuevoNombre	
nuevoEmail	
nuevoPais	
nuevaFecha	
esPrivada	

Devuelve

Definición en la línea 1097 del archivo MongoAtlas.cs.

ActualizarPlaylist()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarPlaylist (
                string nuevoNombre,
                string nuevaDesc,
                List< string > nuevasCanciones,
                string nuevaPortada,
                ListaPersonalizada original)
```

Definición en la línea 1241 del archivo MongoAtlas.cs.

ActualizarTendencia()

```
async Task BetaProyecto.Services.MongoAtlas.ActualizarTendencia (
                string idCancion)   [private]
```

Definición en la línea 1439 del archivo MongoAtlas.cs.

ActualizarUsuario()

```
async Task< bool > BetaProyecto.Services.MongoAtlas.ActualizarUsuario (
                string id,
                string nombre,
                string email,
                string password,
                string rol,
                string pais,
                string imagenUrl,
                DateTime fecha,
                bool esPrivada)
```

Actualiza un usaurio de la base de datos.

Parámetros

id	
nombre	
email	
password	
rol	
pais	
imagenUrl	
fecha	
esPrivada	

Devuelve

Definición en la línea 1160 del archivo MongoAtlas.cs.

AgregarAFavorito()

async Task< bool > BetaProyecto.Services.MongoAtlas.AgregarAFavorito (
string idUsuario,
string idCancion)

Se ocupa de añadir una canción a la lista de favoritos del usuario en la base de datos.

Parámetros

idUsuario	
idCancion	

Devuelve

Definición en la línea 630 del archivo MongoAtlas.cs.

Conectar()

async Task< bool > BetaProyecto.Services.MongoAtlas.Conectar ()

Definición en la línea 30 del archivo MongoAtlas.cs.

CrearGenero()

async Task< bool > BetaProyecto.Services.MongoAtlas.CrearGenero (
string nuevoGenero)

Se ocupa de crear un nuevo género en la base de datos.

Parámetros

nuevoGenero	

Devuelve

Definición en la línea 773 del archivo MongoAtlas.cs.

CrearListaReproduccion()

async Task< bool > BetaProyecto.Services.MongoAtlas.CrearListaReproduccion (
ListaPersonalizada nuevaLista)

Se ocupa de crear una nueva listapersonalizada en la base de datos.

Parámetros

nuevaLista	

Devuelve

Definición en la línea 723 del archivo MongoAtlas.cs.

CrearUsuario()

async Task< bool > BetaProyecto.Services.MongoAtlas.CrearUsuario (
Usuarios nuevoUsuario)

Se ocupa de crear un nuevo usuario en la base de datos.

Parámetros

nuevoUsuario	

Devuelve

Definición en la línea 810 del archivo MongoAtlas.cs.

DejarDeSeguirUsuario()

async Task< bool > BetaProyecto.Services.MongoAtlas.DejarDeSeguirUsuario (
string idUsuario,
string idUsuarioADejar)

Se ocupa de eliminar un usuario de la lista de seguidores del usuario en la base de datos.

Parámetros

idUsuario	
idUsuarioADejar	

Devuelve

Definición en la línea 880 del archivo MongoAtlas.cs.

EliminarCancionPorId()

async Task< bool > BetaProyecto.Services.MongoAtlas.EliminarCancionPorId (
string idCancion)

Se ocupa de eliminar una canción de la base de datos por id.

Parámetros

idCancion	

Devuelve

Definición en la línea 912 del archivo MongoAtlas.cs.

EliminarDeFavorito()

async Task< bool > BetaProyecto.Services.MongoAtlas.EliminarDeFavorito (
string idUsuario,
string idCancion)

Se ocupa de eliminar una canción de la lista de favoritos del usuario en la base de datos.

Parámetros

idUsuario	
idCancion	

Devuelve

Definición en la línea 846 del archivo MongoAtlas.cs.

EliminarGenero()

async Task< bool > BetaProyecto.Services.MongoAtlas.EliminarGenero (
Generos generoAEliminar)

Eliminamos el genero de la base de datos.

Parámetros

generoAEliminar	

Devuelve

Definición en la línea 974 del archivo MongoAtlas.cs.

EliminarPlaylistPorId()

async Task< bool > BetaProyecto.Services.MongoAtlas.EliminarPlaylistPorId (
string idPlaylist)

Eliminamos la listapersonalizada de la base de datos por id.

Parámetros

idPlaylist	

Devuelve

Definición en la línea 953 del archivo MongoAtlas.cs.

EliminarReporte()

async Task< bool > BetaProyecto.Services.MongoAtlas.EliminarReporte (
string idReporte)

Elimina un reporte del la base de datos.

Parámetros

idReporte	

Devuelve

Definición en la línea 1062 del archivo MongoAtlas.cs.

EliminarUsuario()

async Task< bool > BetaProyecto.Services.MongoAtlas.EliminarUsuario (
string idUsuario)

Eliminamos al usuarios de la base de datos.

Parámetros

idUsuario	

Devuelve

Definición en la línea 1015 del archivo MongoAtlas.cs.

EnviarReporte()

async Task< bool > BetaProyecto.Services.MongoAtlas.EnviarReporte (
Reportes nuevoReporte)

Se ocupa de enviar un nuevo reporte a la base de datos.

Parámetros

nuevoReporte	

Devuelve

Definición en la línea 748 del archivo MongoAtlas.cs.

IncrementarContadorCancionesUsuario()

async Task BetaProyecto.Services.MongoAtlas.IncrementarContadorCancionesUsuario (
string idUsuario,
int cantidad)

Definición en la línea 1490 del archivo MongoAtlas.cs.

IncrementarMetricaCancion()

async Task BetaProyecto.Services.MongoAtlas.IncrementarMetricaCancion (
string idCancion,
string campo,
int cantidad)

Definición en la línea 1412 del archivo MongoAtlas.cs.

LoginUsuario()

async Task< Usuarios > BetaProyecto.Services.MongoAtlas.LoginUsuario (
string username,
string password)

Definición en la línea 64 del archivo MongoAtlas.cs.

ObtenerCacionesNovedades()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCacionesNovedades ()

Definición en la línea 130 del archivo MongoAtlas.cs.

ObtenerCanciones()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCanciones ()

Definición en la línea 188 del archivo MongoAtlas.cs.

ObtenerCancionesFavoritos()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCancionesFavoritos ()

Definición en la línea 92 del archivo MongoAtlas.cs.

ObtenerCancionesPorAutor()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCancionesPorAutor (
string idAutor)

Definición en la línea 264 del archivo MongoAtlas.cs.

ObtenerCancionesPorBusqueda()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCancionesPorBusqueda (
string textoBusqueda)

Definición en la línea 236 del archivo MongoAtlas.cs.

ObtenerCancionesPorGenero()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCancionesPorGenero (
string genero)

Definición en la línea 158 del archivo MongoAtlas.cs.

ObtenerCancionesPorListaIds()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerCancionesPorListaIds (
List< string > ids)

Definición en la línea 216 del archivo MongoAtlas.cs.

ObtenerGenerosCompletos()

async Task< List< Generos > > BetaProyecto.Services.MongoAtlas.ObtenerGenerosCompletos ()

Obtenemos los objetos completos de géneros de la base de datos.

Devuelve

Definición en la línea 537 del archivo MongoAtlas.cs.

ObtenerListasReproduccion()

async Task< List< ListaPersonalizada > > BetaProyecto.Services.MongoAtlas.ObtenerListasReproduccion ()

Definición en la línea 389 del archivo MongoAtlas.cs.

ObtenerMixPorGenero()

async Task< List< Canciones > > BetaProyecto.Services.MongoAtlas.ObtenerMixPorGenero (
string genero)

Obtiene una mezcla de canciones por género (veteranos + indies) partir del puntaje de tendencia.

Parámetros

genero	

Devuelve

Definición en la línea 561 del archivo MongoAtlas.cs.

ObtenerNombresGeneros()

async Task< List< string > > BetaProyecto.Services.MongoAtlas.ObtenerNombresGeneros ()

Obtenemos la lista de nombres de géneros de la base de datos.

Devuelve

Definición en la línea 510 del archivo MongoAtlas.cs.

ObtenerPlaylistsPorCreador()

async Task< List< ListaPersonalizada > > BetaProyecto.Services.MongoAtlas.ObtenerPlaylistsPorCreador (
string idUsuario)

Definición en la línea 419 del archivo MongoAtlas.cs.

ObtenerReportes()

async Task< List< Reportes > > BetaProyecto.Services.MongoAtlas.ObtenerReportes ()

Se ocupa de traer todos los reportes de la base de datos.

Devuelve

Definición en la línea 459 del archivo MongoAtlas.cs.

ObtenerTodosLosUsuarios()

async Task< List< Usuarios > > BetaProyecto.Services.MongoAtlas.ObtenerTodosLosUsuarios ()

Definición en la línea 295 del archivo MongoAtlas.cs.

ObtenerUsuariosPorBusqueda()

async Task< List< Usuarios > > BetaProyecto.Services.MongoAtlas.ObtenerUsuariosPorBusqueda (
string textoBusqueda,
List< string > idsExcluidos)

Definición en la línea 319 del archivo MongoAtlas.cs.

ObtenerUsuariosPorListaIds()

async Task< List< Usuarios > > BetaProyecto.Services.MongoAtlas.ObtenerUsuariosPorListaIds (
List< string > listaIds)

Definición en la línea 355 del archivo MongoAtlas.cs.

PublicarCancion()

async Task< bool > BetaProyecto.Services.MongoAtlas.PublicarCancion (
Canciones nuevaCancion)

Se ocupa de publicar una nueva canción en la base de datos.

Parámetros

nuevaCancion	

Devuelve

Definición en la línea 699 del archivo MongoAtlas.cs.

RellenarNombresDeArtistas()

async Task BetaProyecto.Services.MongoAtlas.RellenarNombresDeArtistas (
List< Canciones > listaCanciones) [private]

Definición en la línea 1558 del archivo MongoAtlas.cs.

SeguirUsuario()

async Task< bool > BetaProyecto.Services.MongoAtlas.SeguirUsuario (
string idUsuario,
string idUsuarioASeguir)

Se ocupa de añadir un usuario a la lista de seguidores del usuario en la base de datos.

Parámetros

idUsuario	
idUsuarioASeguir	

Devuelve

Definición en la línea 666 del archivo MongoAtlas.cs.

3.21.4. Propiedades

Database

IMongoDatabase? BetaProyecto.Services.MongoAtlas.Database [get], [private set]

Definición en la línea 23 del archivo MongoAtlas.cs.

3.22. Referencia de la clase BetaProyecto.Singleton.MongoClientSingleton

3.22.1. Detalles

Definición en la línea 10 del archivo MongoClientSingleton.cs.

Propiedades

- static MongoClientSingleton Instance [get]
- MongoAtlas Cliente [get, private set]

3.22.2. Constructores

MongoClientSingleton()

BetaProyecto.Singleton.MongoClientSingleton.MongoClientSingleton () [private]

Definición en la línea 20 del archivo MongoClientSingleton.cs.

3.22.3. Propiedades

Cliente

MongoAtlas BetaProyecto.Singleton.MongoClientSingleton.Cliente [get], [private set]

Definición en la línea 17 del archivo MongoClientSingleton.cs.

Instance

MongoClientSingleton BetaProyecto.Singleton.MongoClientSingleton.Instance [static], [get]

Definición en la línea 14 del archivo MongoClientSingleton.cs.

3.23. Referencia de la clase BetaProyecto.API.Controllers.MusicController

3.23.1. Detalles

Definición en la línea 10 del archivo MusicController.cs.

Métodos

- MusicController ()
- async Task< IActionResult > GetStreamUrl ([FromQuery] string url)

  Procesa una solicitud HTTP GET para extraer la URL de streaming directo y los metadatos de un video de YouTube.

3.23.2. Constructores

MusicController()

BetaProyecto.API.Controllers.MusicController.MusicController ()

Definición en la línea 15 del archivo MusicController.cs.

3.23.3. Funciones

ActualizarYtDlp()

void BetaProyecto.API.Controllers.MusicController.ActualizarYtDlp () [private]

Ejecuta el comando de auto-actualización del binario yt-dlp de forma silenciosa.

Dado que las plataformas de video cambian sus algoritmos frecuentemente, este método asegura que la herramienta de extracción esté en su versión más reciente mediante los siguientes pasos:

1. Validación: Comprueba la existencia del ejecutable antes de intentar la actualización.
2. Ejecución en segundo plano: Inicia un proceso externo con el argumento –update configurado para no mostrar ventanas (CreateNoWindow).
3. Redirección: Captura las salidas del proceso para evitar bloqueos y espera su finalización síncrona.
4. Resiliencia: El bloque catch ignora silenciosamente fallos (como falta de internet o bloqueos de firewall) para permitir que la aplicación principal siga funcionando incluso si la actualización falla.

Definición en la línea 103 del archivo MusicController.cs.

GetStreamUrl()

async Task< IActionResult > BetaProyecto.API.Controllers.MusicController.GetStreamUrl (
[FromQuery] string url)

Procesa una solicitud HTTP GET para extraer la URL de streaming directo y los metadatos de un video de YouTube.

El flujo de ejecución de este endpoint es el siguiente:

1. Validación: Comprueba que la URL recibida no sea nula y que el binario yt-dlp esté disponible en el servidor.
2. Preparación de Argumentos: Configura yt-dlp con los parámetros –dump-json (para obtener datos en lugar de descargar) y –cookies (usando la versión sanitizada para evitar bloqueos).
3. Ejecución de Proceso: Inicia un proceso externo de forma asíncrona, capturando la salida estándar codificada en UTF-8 para evitar errores con caracteres especiales.
4. Análisis de Datos: Parsea la salida JSON generada por la herramienta para extraer el enlace directo (url) y la duración exacta en segundos (duration).

Parámetros

url	La dirección URL del video de YouTube proporcionada como parámetro de consulta (query string).

Devuelve

Un objeto JSON que contiene la URL de streaming directo y la duración; o un código de error (400 o 500) con el detalle del fallo.

Definición en la línea 146 del archivo MusicController.cs.

InicializarEntorno()

```
void BetaProyecto.API.Controllers.MusicController.InicializarEntorno (
        string appDir,
        string userDir)   [private]
```

Configura el entorno de ejecución local, asegurando la presencia de las dependencias binarias y sanitizando archivos de configuración.

Este método de preparación realiza las siguientes operaciones críticas:

1. Despliegue de Binarios: Verifica la existencia de yt-dlp.exe en la ruta de ejecución. Si no está presente o el archivo está dañado (0 bytes), realiza una copia desde el directorio de instalación.
2. Sanitización de Cookies: Procesa el archivo cookies.txt para eliminar la marca de orden de bytes (BOM). Esto es indispensable ya que yt-dlp requiere una codificación UTF-8 pura para validar sesiones de usuario.
3. Normalización de Rutas: Centraliza los archivos operativos en carpetas de datos de usuario para evitar problemas de permisos de escritura.

Cualquier error durante el acceso a archivos o escritura de disco se captura y se registra en la consola para facilitar el diagnóstico.

Parámetros

appDir	Directorio raíz donde se encuentran los archivos originales de la aplicación.
userDir	Directorio de datos de usuario (AppData) donde se desplegará el entorno de trabajo.

Definición en la línea 48 del archivo MusicController.cs.

3.24. Referencia de la clase BetaProyecto.ViewModels.PanelUsuarioViewModel

3.24.1. Detalles

Definición en la línea 8 del archivo PanelUsuarioViewModel.cs.

Métodos

- PanelUsuarioViewModel (int tabInicial=0)

Propiedades

- ViewPerfilViewModel ViewPerfilVM [get]
- ViewCuentaViewModel ViewCuentaVM [get]
- ViewGestionarCuentaViewModel ViewGestionarCuentaVM [get]
- ViewConfiguracionViewModel ViewConfiguracionVM [get]
- ViewGestionarReportesViewModel ViewGestionarReportesVM [get]
- ViewGestionarBDViewModel ViewGestionarBDVM [get]
- Action VolverAtras [get, set]
- Action AccionLogout [get, set]
- Action AccionSalir [get, set]
- Action< ListaPersonalizada > IrAEditarPlaylist [get, set]
- Action< Canciones > IrAEditarCancion [get, set]
- Action? AccionRefrescarDesdePadre [get, set]
- int IndiceTab [get, set]
- bool PuedeVerBD [get, set]
- bool PuedeVerReportes [get, set]

3.24.2. Constructores

PanelUsuarioViewModel()

BetaProyecto.ViewModels.PanelUsuarioViewModel.PanelUsuarioViewModel (
int tabInicial = 0)

Definición en la línea 51 del archivo PanelUsuarioViewModel.cs.

3.24.3. Funciones

ConfigurarPermisos()

void BetaProyecto.ViewModels.PanelUsuarioViewModel.ConfigurarPermisos () [private]

Evalúa y establece los privilegios de acceso del usuario a las funciones administrativas del panel basándose en su rol.

Este método consulta el rol actual desde GlobalData.Instance.RolGD y actualiza las propiedades de visibilidad de la interfaz. La gestión de base de datos se restringe exclusivamente al rol Roles.SuperAdmin, mientras que el acceso a reportes se habilita tanto para administradores como para superadministradores.

Definición en la línea 89 del archivo PanelUsuarioViewModel.cs.

3.24.4. Propiedades

AccionLogout

Action BetaProyecto.ViewModels.PanelUsuarioViewModel.AccionLogout [get], [set]

Definición en la línea 20 del archivo PanelUsuarioViewModel.cs.

AccionRefrescarDesdePadre

Action? BetaProyecto.ViewModels.PanelUsuarioViewModel.AccionRefrescarDesdePadre [get], [set]

Definición en la línea 24 del archivo PanelUsuarioViewModel.cs.

AccionSalir

Action BetaProyecto.ViewModels.PanelUsuarioViewModel.AccionSalir [get], [set]

Definición en la línea 21 del archivo PanelUsuarioViewModel.cs.

IndiceTab

int BetaProyecto.ViewModels.PanelUsuarioViewModel.IndiceTab [get], [set]

Definición en la línea 28 del archivo PanelUsuarioViewModel.cs.

IrAEditarCancion

Action<Canciones> BetaProyecto.ViewModels.PanelUsuarioViewModel.IrAEditarCancion [get], [set]

Definición en la línea 23 del archivo PanelUsuarioViewModel.cs.

IrAEditarPlaylist

Action<ListaPersonalizada> BetaProyecto.ViewModels.PanelUsuarioViewModel.IrAEditarPlaylist [get], [set]

Definición en la línea 22 del archivo PanelUsuarioViewModel.cs.

PuedeVerBD

bool BetaProyecto.ViewModels.PanelUsuarioViewModel.PuedeVerBD [get], [set]

Definición en la línea 36 del archivo PanelUsuarioViewModel.cs.

PuedeVerReportes

bool BetaProyecto.ViewModels.PanelUsuarioViewModel.PuedeVerReportes [get], [set]

Definición en la línea 44 del archivo PanelUsuarioViewModel.cs.

ViewConfiguracionVM

ViewConfiguracionViewModel BetaProyecto.ViewModels.PanelUsuarioViewModel.ViewConfiguracionVM [get]

Definición en la línea 14 del archivo PanelUsuarioViewModel.cs.

ViewCuentaVM

ViewCuentaViewModel BetaProyecto.ViewModels.PanelUsuarioViewModel.ViewCuentaVM [get]

Definición en la línea 12 del archivo PanelUsuarioViewModel.cs.

ViewGestionarBDVM

ViewGestionarBDViewModel BetaProyecto.ViewModels.PanelUsuarioViewModel.ViewGestionarBDVM [get]

Definición en la línea 16 del archivo PanelUsuarioViewModel.cs.

ViewGestionarCuentaVM

ViewGestionarCuentaViewModel BetaProyecto.ViewModels.PanelUsuarioViewModel.ViewGestionarCuentaVM [get]

Definición en la línea 13 del archivo PanelUsuarioViewModel.cs.

ViewGestionarReportesVM

ViewGestionarReportesViewModel BetaProyecto.ViewModels.PanelUsuarioViewModel.ViewGestionarReportesVM [get]

Definición en la línea 15 del archivo PanelUsuarioViewModel.cs.

ViewPerfilVM

ViewPerfilViewModel BetaProyecto.ViewModels.PanelUsuarioViewModel.ViewPerfilVM [get]

Definición en la línea 11 del archivo PanelUsuarioViewModel.cs.

VolverAtras

Action BetaProyecto.ViewModels.PanelUsuarioViewModel.VolverAtras [get], [set]

Implementa BetaProyecto.ViewModels.INavegable.

Definición en la línea 19 del archivo PanelUsuarioViewModel.cs.

3.25. Referencia de la clase BetaProyecto.Models.PerfilUsuario

3.25.1. Detalles

Definición en la línea 47 del archivo Usuarios.cs.

Propiedades

- string ImagenUrl [get, set]
- DateTime FechaNacimiento [get, set]
- bool EsPrivada [get, set]
- string Pais [get, set]

3.25.2. Propiedades

EsPrivada

bool BetaProyecto.Models.PerfilUsuario.EsPrivada [get], [set]

Definición en la línea 56 del archivo Usuarios.cs.

FechaNacimiento

DateTime BetaProyecto.Models.PerfilUsuario.FechaNacimiento [get], [set]

Definición en la línea 53 del archivo Usuarios.cs.

ImagenUrl

string BetaProyecto.Models.PerfilUsuario.ImagenUrl [get], [set]

Definición en la línea 50 del archivo Usuarios.cs.

Pais

string BetaProyecto.Models.PerfilUsuario.Pais [get], [set]

Definición en la línea 58 del archivo Usuarios.cs.

3.26. Referencia de la clase BetaProyecto.Models.ReferenciasReporte

3.26.1. Detalles

Definición en la línea 52 del archivo Reportes.cs.

Propiedades

- string UsuarioReportanteId [get, set]
- string CancionReportadaId [get, set]

3.26.2. Propiedades

CancionReportadaId

string BetaProyecto.Models.ReferenciasReporte.CancionReportadaId [get], [set]

Definición en la línea 58 del archivo Reportes.cs.

UsuarioReportanteId

string BetaProyecto.Models.ReferenciasReporte.UsuarioReportanteId [get], [set]

Definición en la línea 55 del archivo Reportes.cs.

3.27. Referencia de la clase BetaProyecto.Models.Reportes

3.27.1. Detalles

Definición en la línea 11 del archivo Reportes.cs.

Propiedades

- string Id [get, set]
- string TipoProblema [get, set]
- string Descripcion [get, set]
- string Estado = "Pendiente" [get, set]
- ReferenciasReporte Referencias [get, set]
- DateTime FechaCreacion = DateTime.UtcNow [get, set]
- string Resolucion = "" [get, set]
- string NombreReportante = "Usuario Desconocido" [get, set]
- string TituloCancionReportada = "Canción Desconocida" [get, set]
- string ColorEstado [get]

3.27.2. Propiedades

ColorEstado

string BetaProyecto.Models.Reportes.ColorEstado [get]

Definición en la línea 43 del archivo Reportes.cs.

Descripcion

string BetaProyecto.Models.Reportes.Descripcion [get], [set]

Definición en la línea 21 del archivo Reportes.cs.

Estado

string BetaProyecto.Models.Reportes.Estado = "Pendiente" [get], [set]

Definición en la línea 24 del archivo Reportes.cs.

FechaCreacion

DateTime BetaProyecto.Models.Reportes.FechaCreacion = DateTime.UtcNow [get], [set]

Definición en la línea 30 del archivo Reportes.cs.

Id

string BetaProyecto.Models.Reportes.Id [get], [set]

Definición en la línea 15 del archivo Reportes.cs.

NombreReportante

string BetaProyecto.Models.Reportes.NombreReportante = "Usuario Desconocido" [get], [set]

Definición en la línea 36 del archivo Reportes.cs.

Referencias

ReferenciasReporte BetaProyecto.Models.Reportes.Referencias [get], [set]

Definición en la línea 27 del archivo Reportes.cs.

Resolucion

string BetaProyecto.Models.Reportes.Resolucion = "" [get], [set]

Definición en la línea 33 del archivo Reportes.cs.

TipoProblema

string BetaProyecto.Models.Reportes.TipoProblema [get], [set]

Definición en la línea 18 del archivo Reportes.cs.

TituloCancionReportada

string BetaProyecto.Models.Reportes.TituloCancionReportada = "Canción Desconocida" [get], [set]

Definición en la línea 39 del archivo Reportes.cs.

3.28. Referencia de la clase BetaProyecto.Models.Roles

3.28.1. Detalles

Definición en la línea 9 del archivo Roles.cs.

3.29. Referencia de la clase BetaProyecto.API.Controllers.StorageController

3.29.1. Detalles

Definición en la línea 10 del archivo StorageController.cs.

Métodos

- StorageController ()
- async Task< IActionResult > SubirImagen (IFormFile archivo)

  Recibe un archivo de imagen, lo procesa en memoria y lo carga en el servicio externo ImgBB.
- async Task< IActionResult > SubirAudio (IFormFile archivo)

  Procesa un archivo multimedia de audio y lo carga de forma asíncrona en el servicio de almacenamiento de Cloudinary.
- async Task< IActionResult > EliminarArchivo ([FromQuery] string url)

  Solicita la eliminación permanente de un recurso multimedia alojado en Cloudinary a partir de su URL pública.

3.29.2. Constructores

StorageController()

BetaProyecto.API.Controllers.StorageController.StorageController ()

Definición en la línea 22 del archivo StorageController.cs.

3.29.3. Funciones

EliminarArchivo()

async Task< IActionResult > BetaProyecto.API.Controllers.StorageController.EliminarArchivo (
[FromQuery] string url)

Solicita la eliminación permanente de un recurso multimedia alojado en Cloudinary a partir de su URL pública.

Este endpoint implementa una lógica de filtrado y limpieza:

1. Discriminación de dominio: Solo procesa eliminaciones si la URL pertenece a Cloudinary, ignorando otros proveedores (como ImgBB) para evitar errores de API.

2. Extracción de Identificador: Procesa la URL para obtener el PublicId, que es la clave única que Cloudinary necesita para localizar el archivo.
3. Borrado por tipo de recurso: Define específicamente el ResourceType.Video (usado para audio en tu configuración) para asegurar que el motor de búsqueda de Cloudinary encuentre el objeto.

Parámetros

url	La dirección URL completa del archivo que se desea eliminar.

Devuelve

Un mensaje de confirmación de éxito o un error detallado si la operación falla.

Definición en la línea 169 del archivo StorageController.cs.

ObtenerPublicId()

string BetaProyecto.API.Controllers.StorageController.ObtenerPublicId (
string url) [private]

Analiza una URL de Cloudinary y extrae el identificador único (Public ID) necesario para operaciones de gestión.

El método realiza un ”parseo” quirúrgico de la URL siguiendo este algoritmo:

1. Localización: Busca el segmento /upload/ que separa la configuración del servidor de los datos del archivo.
2. Limpieza de Versión: Omite el componente de versión (ej. v17397...) que Cloudinary genera automáticamente.
3. Extracción de Carpeta y Nombre: Captura la ruta interna y el nombre del archivo.
4. Remoción de Extensión: Elimina el sufijo del formato (ej. .mp3) para obtener el ID limpio que requiere la API de borrado.

Parámetros

url	La dirección URL completa del recurso alojado en Cloudinary.

Devuelve

El Public ID del recurso (incluyendo carpetas) o null si el formato de la URL es inválido.

Definición en la línea 210 del archivo StorageController.cs.

SubirAudio()

async Task< IActionResult > BetaProyecto.API.Controllers.StorageController.SubirAudio (
IFormFile archivo)

Procesa un archivo multimedia de audio y lo carga de forma asíncrona en el servicio de almacenamiento de Cloudinary.

El flujo de este endpoint incluye:

1. Validación: Comprobación de integridad del archivo recibido.
2. Tratamiento de Stream: Apertura del archivo como flujo de datos para evitar la carga total en RAM.
3. Categorización: Uso de parámetros de video (requeridos por Cloudinary para archivos de audio) y asignación de carpeta destino.
4. Persistencia: Obtención de una URL segura (HTTPS) para su almacenamiento en la base de datos.

Parámetros

archivo	El archivo de audio (mp3, wav, etc.) enviado a través de la petición HTTP.

Devuelve

Un objeto JSON con la URL segura del recurso alojado o un mensaje de error detallado.

Definición en la línea 118 del archivo StorageController.cs.

SubirImagen()

async Task< IActionResult > BetaProyecto.API.Controllers.StorageController.SubirImagen (
IFormFile archivo)

Recibe un archivo de imagen, lo procesa en memoria y lo carga en el servicio externo ImgBB.

Este endpoint actúa como un puente (proxy) entre la aplicación cliente y ImgBB:

1. Validación: Asegura que el archivo no sea nulo.
2. Conversión: Transforma la imagen binaria a una cadena Base64 (texto), que es el formato requerido por la API de ImgBB.
3. Transmisión: Realiza una petición POST segura enviando la API Key y el contenido.
4. Resolución: Extrae la URL directa de la respuesta JSON para que la App pueda guardarla en su base de datos.

Parámetros

archivo	El archivo de imagen enviado desde el formulario (IFormFile).

Devuelve

Un objeto JSON con la URL pública de la imagen alojada.

Definición en la línea 47 del archivo StorageController.cs.

3.30. Referencia de la clase BetaProyecto.Services.StorageService

3.30.1. Detalles

Definición en la línea 10 del archivo StorageService.cs.

Métodos

- async Task< string > SubirImagen (string rutaArchivoEnTuPc)

  Carga un archivo de imagen desde el sistema de archivos local hacia el servidor de almacenamiento en la nube.
- async Task< string > SubirCancion (string rutaArchivoEnTuPc)

  Carga un archivo de audio desde el almacenamiento local hacia el servidor de distribución en la nube.
- async Task< bool > EliminarArchivo (string urlCompleta)

  Solicita de forma asíncrona la eliminación de un recurso almacenado en la nube a través de la API de almacenamiento.

3.30.2. Funciones

EliminarArchivo()

async Task< bool > BetaProyecto.Services.StorageService.EliminarArchivo (
string urlCompleta)

Solicita de forma asíncrona la eliminación de un recurso almacenado en la nube a través de la API de almacenamiento.

Este método gestiona la baja de archivos (imágenes o audio) siguiendo este flujo:

1. Validación: Verifica que la URL proporcionada no sea nula o vacía.
2. Seguridad: Configura un HttpClientHandler para omitir la validación de certificados SSL, permitiendo peticiones a entornos localhost con certificados autofirmados.
3. Petición: Ejecuta un verbo HTTP DELETE enviando la URL del archivo como parámetro de consulta.
4. Confirmación: Evalúa la respuesta de la API para confirmar si el archivo fue removido con éxito del proveedor externo (Cloudinary).

Parámetros

urlCompleta	La dirección URL absoluta del recurso que se desea eliminar del almacenamiento remoto.

Devuelve

Una tarea que devuelve true si el servidor confirma la eliminación (Success Status Code); de lo contrario, false si el recurso no existe o hubo un fallo en la comunicación.

Definición en la línea 74 del archivo StorageService.cs.

EnviarA_Api()

async Task< string > BetaProyecto.Services.StorageService.EnviarA_Api (
string ruta,
string endpoint) [private]

Realiza la carga física de un archivo hacia la API de almacenamiento mediante una petición POST de tipo multipart/form-data.

Este método núcleo (core) orquestra la transferencia de archivos multimedia siguiendo este flujo:

1. Verificación local: Valida la existencia del archivo en el sistema de archivos del cliente antes de iniciar la conexión.
2. Configuración de Seguridad: Implementa un bypass de validación SSL para permitir el desarrollo en entornos locales con certificados no firmados.
3. Empaquetado (Multipart): Abre el archivo como un flujo de datos (StreamContent) y lo encapsula en un contenedor compatible con formularios web.
4. Transmisión y Respuesta: Ejecuta la petición asíncrona hacia el endpoint especificado y procesa la respuesta JSON para extraer la URL persistente generada por el servidor.

Parámetros

ruta	La ruta absoluta del archivo en el disco duro local.
endpoint	El segmento final de la URL de la API (ej. ”subir-imagen” o ”subir-audio”).

Devuelve

La URL pública del archivo cargado en el servidor de almacenamiento.

Excepciones

FileNotFoundException	Se lanza si la ruta especificada no apunta a un archivo válido.
Exception	Encapsula errores de conectividad, rechazos de la API (códigos 4xx o 5xx) o fallos en el análisis del JSON.

Definición en la línea 123 del archivo StorageService.cs.

SubirCancion()

async Task< string > BetaProyecto.Services.StorageService.SubirCancion (
string rutaArchivoEnTuPc)

Carga un archivo de audio desde el almacenamiento local hacia el servidor de distribución en la nube.

Este método encapsula la lógica de transferencia de archivos multimedia utilizando el endpoint dedicado para audio. El proceso sigue el siguiente flujo:

1. Empaquetado: Invoca al método core EnviarA_Api para convertir el archivo físico en un flujo de datos (Stream).
2. Transmisión: Envía el recurso mediante una petición POST multipart/form-data al microservicio de almacenamiento.
3. Respuesta: Retorna la URL segura (HTTPS) proporcionada por Cloudinary, necesaria para la persistencia en la base de datos.

Parámetros

rutaArchivoEnTuPc	La ruta absoluta del archivo de audio (ej. .mp3, .wav) en el disco local.

Devuelve

Una tarea que contiene la URL pública del archivo alojado si la operación tiene éxito; de lo contrario, devuelve una cadena vacía o nula.

Definición en la línea 51 del archivo StorageService.cs.

SubirImagen()

async Task< string > BetaProyecto.Services.StorageService.SubirImagen (
string rutaArchivoEnTuPc)

Carga un archivo de imagen desde el sistema de archivos local hacia el servidor de almacenamiento en la nube.

Este método actúa como un envoltorio especializado que invoca la lógica genérica de comunicación con la API utilizando el endpoint específico para el procesamiento de imágenes. Es ideal para la gestión de avatares de usuario, portadas de álbumes y miniaturas de playlists.

Parámetros

rutaArchivoEnTuPc	La ruta absoluta del archivo de imagen en el almacenamiento local.

Devuelve

Una tarea que representa la operación asíncrona. El valor devuelto contiene la URL pública generada por el servicio de almacenamiento (Cloudinary) si la carga fue exitosa.

Definición en la línea 29 del archivo StorageService.cs.

3.31. Referencia de la clase BetaProyecto.ViewModels.TabItemBuscadorViewModel

3.31.1. Detalles

Definición en la línea 10 del archivo TabItemBuscadorViewModel.cs.

Métodos

- TabItemBuscadorViewModel ()

Propiedades

- string TxtBusqueda [get, set]
- string TxtInfoResultado [get, set]
- string TxtContador [get, set]
- ObservableCollection< Canciones > ListaBusqueda [get, set]
- ReactiveCommand< Unit, Unit > BtnBuscar [get]

3.31.2. Constructores

TabItemBuscadorViewModel()

BetaProyecto.ViewModels.TabItemBuscadorViewModel.TabItemBuscadorViewModel ()

Definición en la línea 44 del archivo TabItemBuscadorViewModel.cs.

3.31.3. Funciones

BuscarEnBD()

async Task BetaProyecto.ViewModels.TabItemBuscadorViewModel.BuscarEnBD () [private]

Realiza una búsqueda asíncrona de canciones en la base de datos utilizando el texto de búsqueda actual y actualiza el resultados de búsqueda y propiedades de estado relacionadas.

Si la conexión a la base de datos no está disponible, el método actualiza las propiedades de estado para indicar un error de conexión. Los resultados de búsqueda y las propiedades de estado se actualizan en función de si hay alguno las canciones coincidentes se encuentran.

Devuelve

Devuelve una tarea que representa la operación de búsqueda asíncrona.

Definición en la línea 66 del archivo TabItemBuscadorViewModel.cs.

3.31.4. Propiedades

BtnBuscar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.TabItemBuscadorViewModel.BtnBuscar [get]

Definición en la línea 42 del archivo TabItemBuscadorViewModel.cs.

ListaBusqueda

ObservableCollection<Canciones> BetaProyecto.ViewModels.TabItemBuscadorViewModel.ListaBusqueda [get], [set]

Definición en la línea 35 del archivo TabItemBuscadorViewModel.cs.

TxtBusqueda

string BetaProyecto.ViewModels.TabItemBuscadorViewModel.TxtBusqueda [get], [set]

Definición en la línea 14 del archivo TabItemBuscadorViewModel.cs.

TxtContador

string BetaProyecto.ViewModels.TabItemBuscadorViewModel.TxtContador [get], [set]

Definición en la línea 28 del archivo TabItemBuscadorViewModel.cs.

TxtInfoResultado

string BetaProyecto.ViewModels.TabItemBuscadorViewModel.TxtInfoResultado [get], [set]

Definición en la línea 20 del archivo TabItemBuscadorViewModel.cs.

3.32. Referencia de la clase BetaProyecto.ViewModels.TabItemInicioViewModel

3.32.1. Detalles

Definición en la línea 15 del archivo TabItemInicioViewModel.cs.

Métodos

- TabItemInicioViewModel ()

Propiedades

- Action< Canciones, List< Canciones > >? EnviarReproduccion [get, set]
- Action< Canciones >? SolicitudVerDetalles [get, set]
- Action< string >? SolicitudVerArtista [get, set]
- Action< Canciones >? SolicitudCrearReporte [get, set]
- Action< ListaPersonalizada >? SolicitudVerDetallasPlaylist [get, set]
- ReactiveCommand< object, Unit > BtnReproducirDesdeTarjeta [get]
- ReactiveCommand< ListaPersonalizada, Unit > BtnReproducirPlaylist [get]
- ReactiveCommand< Unit, Unit > BtnRefrescar [get]
- ReactiveCommand< Canciones, Unit > BtnIrADetalleCancion [get]
- ReactiveCommand< object, Unit > BtnIrAArtista [get]
- ReactiveCommand< Canciones, Unit > BtnIrAReportar [get]
- ReactiveCommand< ListaPersonalizada, Unit > BtnIrADetallesPlaylist [get]
- ObservableCollection< TarjetasCanciones > Tarjetas [get, set]
- ObservableCollection< TarjetasListas > Playlists [get, set]
- string TxtFav [get, set]

3.32.2. Constructores

TabItemInicioViewModel()

BetaProyecto.ViewModels.TabItemInicioViewModel.TabItemInicioViewModel ()

Definición en la línea 62 del archivo TabItemInicioViewModel.cs.

3.32.3. Funciones

CargarDatosCanciones()

async Task BetaProyecto.ViewModels.TabItemInicioViewModel.CargarDatosCanciones () [private]

Carga y organiza de forma asíncrona los datos de canciones y listas de reproducción desde la base de datos, actualizando los correspondientes colecciones para la vinculación de datos.

Si la conexión a la base de datos no está disponible, se muestra una alerta y no hay datos cargado. Las canciones y listas de reproducción se agrupan en secciones como favoritas, nuevos lanzamientos, rock, personalizadas listas y listas de comunidades, asignadas a sus respectivas colecciones para la vinculación de la interfaz.

Devuelve

Devuelve una tarea que representa la operación de carga asíncrona.

Definición en la línea 212 del archivo TabItemInicioViewModel.cs.

IrAArtistaDesdeBoton()

void BetaProyecto.ViewModels.TabItemInicioViewModel.IrAArtistaDesdeBoton (
object parametro) [private]

Dirige la navegación a la vista de un artista cuando se activa mediante un botón asociado con el nombre del artista.

Este método se utiliza típicamente como un manejador de comandos para elementos de la interfaz de usuario que representan artistas dentro de un contexto de canción. Recupera el artista y la información de la canción relevante desde el botón jerarquía y plantea una solicitud para mostrar los detalles del artista. El parámetro debe estructurarse como descrito para que la navegación tenga éxito.

Parámetros

parametro	El parametro de comando, que se espera sea un botón cuyo DataContext contiene el nombre del artista y cuya etiqueta hace referencia al botón padre que contiene la información de la canción.

Definición en la línea 107 del archivo TabItemInicioViewModel.cs.

ReproducirDesdeBoton()

void BetaProyecto.ViewModels.TabItemInicioViewModel.ReproducirDesdeBoton (
object parametro) [private]

Maneja el comando de reproducción disparado desde un botón, iniciando la reproducción de la canción seleccionada y su colección asociada.

Este método se utiliza típicamente como un manejador de comandos para botones de reproducción en el usuario interfaz. El DataContext del botón debe hacer referencia a la canción que se va a reproducir, y su etiqueta debe hacer referencia al colección a la que pertenece la canción. Si falta alguno de los valores o es inválido, el método lo hace nada.

Parámetros

parametro	El parámetro de comando, que se espera sea un botón cuyo DataContext es una canción a reproducir y cuya etiqueta contiene la colección de canciones. No debe ser nula y debe ser un botón con valores válidos de DataContext y Tag.

Definición en la línea 151 del archivo TabItemInicioViewModel.cs.

ReproducirPlaylist()

void BetaProyecto.ViewModels.TabItemInicioViewModel.ReproducirPlaylist (
ListaPersonalizada playlist) [private]

Inicia la reproducción de la lista de reproducción especificada, reproduciendo la primera canción y poniendo en fila las canciones restantes para reproducción.

Si la lista de reproducción es nula o no contiene canciones, se muestra una alerta para informar el usuario que la lista de reproducción está vacía.

Parámetros

playlist	La lista de reproducción a reproducir. No debe ser nula y debe contener al menos una canción.

Definición en la línea 185 del archivo TabItemInicioViewModel.cs.

3.32.4. Propiedades

BtnIrAArtista

ReactiveCommand<object, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnIrAArtista [get]

Definición en la línea 34 del archivo TabItemInicioViewModel.cs.

BtnIrADetalleCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnIrADetalleCancion [get]

Definición en la línea 33 del archivo TabItemInicioViewModel.cs.

BtnIrADetallesPlaylist

ReactiveCommand<ListaPersonalizada, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnIrADetallesPlaylist [get]

Definición en la línea 38 del archivo TabItemInicioViewModel.cs.

BtnIrAReportar

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnIrAReportar [get]

Definición en la línea 35 del archivo TabItemInicioViewModel.cs.

BtnRefrescar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnRefrescar [get]

Definición en la línea 30 del archivo TabItemInicioViewModel.cs.

BtnReproducirDesdeTarjeta

ReactiveCommand<object, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnReproducirDesdeTarjeta [get]

Definición en la línea 28 del archivo TabItemInicioViewModel.cs.

BtnReproducirPlaylist

ReactiveCommand<ListaPersonalizada, Unit> BetaProyecto.ViewModels.TabItemInicioViewModel.BtnReproducirPlaylist [get]

Definición en la línea 29 del archivo TabItemInicioViewModel.cs.

EnviarReproduccion

Action<Canciones, List<Canciones> >? BetaProyecto.ViewModels.TabItemInicioViewModel.EnviarReproduccion [get], [set]

Definición en la línea 21 del archivo TabItemInicioViewModel.cs.

Playlists

ObservableCollection<TarjetasListas> BetaProyecto.ViewModels.TabItemInicioViewModel.Playlists [get], [set]

Definición en la línea 49 del archivo TabItemInicioViewModel.cs.

SolicitudCrearReporte

Action<Canciones>? BetaProyecto.ViewModels.TabItemInicioViewModel.SolicitudCrearReporte [get], [set]

Definición en la línea 24 del archivo TabItemInicioViewModel.cs.

SolicitudVerArtista

Action<string>? BetaProyecto.ViewModels.TabItemInicioViewModel.SolicitudVerArtista [get], [set]

Definición en la línea 23 del archivo TabItemInicioViewModel.cs.

SolicitudVerDetallasPlaylist

Action<ListaPersonalizada>? BetaProyecto.ViewModels.TabItemInicioViewModel.SolicitudVerDetallasPlaylist [get], [set]

Definición en la línea 25 del archivo TabItemInicioViewModel.cs.

SolicitudVerDetalles

Action<Canciones>? BetaProyecto.ViewModels.TabItemInicioViewModel.SolicitudVerDetalles [get], [set]

Definición en la línea 22 del archivo TabItemInicioViewModel.cs.

Tarjetas

ObservableCollection<TarjetasCanciones> BetaProyecto.ViewModels.TabItemInicioViewModel.Tarjetas [get], [set]

Definición en la línea 42 del archivo TabItemInicioViewModel.cs.

TxtFav

string BetaProyecto.ViewModels.TabItemInicioViewModel.TxtFav [get], [set]

Definición en la línea 56 del archivo TabItemInicioViewModel.cs.

3.33. Referencia de la clase BetaProyecto.ViewModels.TabItemPopularesViewModel

3.33.1. Detalles

Definición en la línea 9 del archivo TabItemPopularesViewModel.cs.

Métodos

- TabItemPopularesViewModel ()

Propiedades

- ObservableCollection< string > ListaGeneros [get, set]
- string GeneroSeleccionado [get, set]
- string TxtInfo [get, set]
- string TxtGeneroMostrado [get, set]
- ObservableCollection< Canciones > ListaPopulares [get, set]

3.33.2. Constructores

TabItemPopularesViewModel()

BetaProyecto.ViewModels.TabItemPopularesViewModel.TabItemPopularesViewModel ()

Definición en la línea 55 del archivo TabItemPopularesViewModel.cs.

3.33.3. Funciones

CargarCancionesPorGenero()

async Task BetaProyecto.ViewModels.TabItemPopularesViewModel.CargarCancionesPorGenero (
string genero) [private]

Carga asincrónicamente la lista de canciones populares para el género especificado y actualiza la pantalla relacionada propiedades.

Si no se encuentran canciones para el género especificado, las propiedades de visualización se actualizan a indica que no se encontraron resultados. Si la conexión a la base de datos no está disponible, se establece un mensaje de error en su lugar.

Parámetros

genero	El nombre del género para el que se recuperarán las canciones populares. No puede ser nulo ni vacío.

Devuelve

Devuelve una tarea que representa la operación de carga asíncrona.

Definición en la línea 78 del archivo TabItemPopularesViewModel.cs.

CargarGeneros()

async Task BetaProyecto.ViewModels.TabItemPopularesViewModel.CargarGeneros () [private]

Carga de forma asíncrona la lista de nombres de género desde la base de datos y actualiza la colección utilizada por la vista.

Si la conexión a la base de datos no está disponible, el método escribe un mensaje de error en el salida de depuración y no se actualiza la lista de géneros.

Devuelve

Devuelve una tarea que representa la operación de carga asíncrona.

Definición en la línea 116 del archivo TabItemPopularesViewModel.cs.

3.33.4. Propiedades

GeneroSeleccionado

string BetaProyecto.ViewModels.TabItemPopularesViewModel.GeneroSeleccionado [get], [set]

Definición en la línea 21 del archivo TabItemPopularesViewModel.cs.

ListaGeneros

ObservableCollection<string> BetaProyecto.ViewModels.TabItemPopularesViewModel.ListaGeneros [get], [set]

Definición en la línea 14 del archivo TabItemPopularesViewModel.cs.

ListaPopulares

ObservableCollection<Canciones> BetaProyecto.ViewModels.TabItemPopularesViewModel.ListaPopulares [get], [set]

Definición en la línea 49 del archivo TabItemPopularesViewModel.cs.

TxtGeneroMostrado

string BetaProyecto.ViewModels.TabItemPopularesViewModel.TxtGeneroMostrado [get], [set]

Definición en la línea 41 del archivo TabItemPopularesViewModel.cs.

TxtInfo

string BetaProyecto.ViewModels.TabItemPopularesViewModel.TxtInfo [get], [set]

Definición en la línea 34 del archivo TabItemPopularesViewModel.cs.

3.34. Referencia de la clase BetaProyecto.Models.TarjetasCanciones

3.34.1. Detalles

Definición en la línea 11 del archivo TarjetasCanciones.cs.

Métodos

- TarjetasCanciones (string titulo, ObservableCollection< Canciones > canciones)

Propiedades

- string TituloSeccion [get, set]
- ObservableCollection< Canciones > ListaCanciones [get, set]

3.34.2. Constructores

TarjetasCanciones()

BetaProyecto.Models.TarjetasCanciones.TarjetasCanciones (
string titulo,
ObservableCollection< Canciones > canciones)

Definición en la línea 16 del archivo TarjetasCanciones.cs.

3.34.3. Propiedades

ListaCanciones

ObservableCollection<Canciones> BetaProyecto.Models.TarjetasCanciones.ListaCanciones [get], [set]

Definición en la línea 14 del archivo TarjetasCanciones.cs.

TituloSeccion

string BetaProyecto.Models.TarjetasCanciones.TituloSeccion [get], [set]

Definición en la línea 13 del archivo TarjetasCanciones.cs.

3.35. Referencia de la clase BetaProyecto.Models.TarjetasListas

3.35.1. Detalles

Definición en la línea 10 del archivo TarjetasListas.cs.

Métodos

- TarjetasListas (string titulo, ObservableCollection< ListaPersonalizada > listas)

Propiedades

- string TituloSeccion [get, set]
- ObservableCollection< ListaPersonalizada > Listas [get, set]

3.35.2. Constructores

TarjetasListas()

```
BetaProyecto.Models.TarjetasListas.TarjetasListas (
            string titulo,
            ObservableCollection< ListaPersonalizada > listas)
```

Definición en la línea 15 del archivo TarjetasListas.cs.

3.35.3. Propiedades

Listas

ObservableCollection<ListaPersonalizada> BetaProyecto.Models.TarjetasListas.Listas [get], [set]

Definición en la línea 13 del archivo TarjetasListas.cs.

TituloSeccion

string BetaProyecto.Models.TarjetasListas.TituloSeccion [get], [set]

Definición en la línea 12 del archivo TarjetasListas.cs.

3.36. Referencia de la clase BetaProyecto.Helpers.TextoTraducidoConverter

3.36.1. Detalles

Definición en la línea 14 del archivo TextoTraducidoConverter.cs.

Métodos

- object? Convert (object? value, Type targetType, object? parameter, CultureInfo culture)

  Traduce dinámicamente una clave de recurso en su valor correspondiente definido en los diccionarios de la aplicación.
- object? ConvertBack (object? value, Type targetType, object? parameter, CultureInfo culture)

3.36.2. Funciones

Convert()

```
object? BetaProyecto.Helpers.TextoTraducidoConverter.Convert (
            object? value,
            Type targetType,
            object? parameter,
            CultureInfo culture)
```

Traduce dinámicamente una clave de recurso en su valor correspondiente definido en los diccionarios de la aplicación.

Este convertidor facilita la internacionalización (i18n) en la capa de vista mediante los siguientes pasos:

1. Validación: Verifica si el valor de entrada es una cadena de texto válida (la clave del recurso).
2. Búsqueda: Consulta el diccionario de recursos activo de Application.Current intentando localizar la clave.
3. Resolución: Si encuentra el recurso (ej. una traducción o una ruta de imagen), lo devuelve; de lo contrario, retorna la clave original como valor de respaldo (fallback).

Es ideal para enlazar propiedades de texto en XAML que deben reaccionar a cambios de idioma en tiempo de ejecución.

Parámetros

value	La clave del recurso (string) que se desea localizar.

targetType	El tipo de la propiedad de destino.
parameter	Parámetro opcional (no utilizado).
culture	Información de cultura (no utilizada, se prioriza el diccionario activo).

Devuelve

El objeto localizado encontrado en los recursos o el texto original si no existe coincidencia.

Definición en la línea 33 del archivo TextoTraducidoConverter.cs.

ConvertBack()

```
object? BetaProyecto.Helpers.TextoTraducidoConverter.ConvertBack (
            object? value,
            Type targetType,
            object? parameter,
            CultureInfo culture)
```

Definición en la línea 50 del archivo TextoTraducidoConverter.cs.

3.37. Referencia de la clase BetaProyecto.Models.Usuarios

3.37.1. Detalles

Definición en la línea 8 del archivo Usuarios.cs.

Propiedades

- string Id [get, set]
- string Username [get, set]
- string Email [get, set]
- string Password [get, set]
- string Rol [get, set]
- PerfilUsuario Perfil = new PerfilUsuario() [get, set]
- EstadisticasUsuario Estadisticas = new EstadisticasUsuario() [get, set]
- ListasUsuario Listas = new ListasUsuario() [get, set]
- ConfiguracionUser Configuracion = new ConfiguracionUser() [get, set]
- DateTime FechaRegistro [get, set]

3.37.2. Propiedades

Configuracion

ConfiguracionUser BetaProyecto.Models.Usuarios.Configuracion = new ConfiguracionUser() [get], [set]

Definición en la línea 40 del archivo Usuarios.cs.

Email

string BetaProyecto.Models.Usuarios.Email [get], [set]

Definición en la línea 19 del archivo Usuarios.cs.

Estadisticas

EstadisticasUsuario BetaProyecto.Models.Usuarios.Estadisticas = new EstadisticasUsuario() [get], [set]

Definición en la línea 34 del archivo Usuarios.cs.

FechaRegistro

DateTime BetaProyecto.Models.Usuarios.FechaRegistro [get], [set]

Definición en la línea 43 del archivo Usuarios.cs.

Id

string BetaProyecto.Models.Usuarios.Id [get], [set]

Definición en la línea 13 del archivo Usuarios.cs.

Listas

ListasUsuario BetaProyecto.Models.Usuarios.Listas = new ListasUsuario() [get], [set]

Definición en la línea 37 del archivo Usuarios.cs.

Password

string BetaProyecto.Models.Usuarios.Password [get], [set]

Definición en la línea 22 del archivo Usuarios.cs.

Perfil

PerfilUsuario BetaProyecto.Models.Usuarios.Perfil = new PerfilUsuario() [get], [set]

Definición en la línea 31 del archivo Usuarios.cs.

Rol

string BetaProyecto.Models.Usuarios.Rol [get], [set]

Definición en la línea 25 del archivo Usuarios.cs.

Username

string BetaProyecto.Models.Usuarios.Username [get], [set]

Definición en la línea 16 del archivo Usuarios.cs.

3.38. Referencia de la clase BetaProyecto.ViewModels.VentanaAvisoViewModel

3.38.1. Detalles

Definición en la línea 7 del archivo VentanaAvisoViewModel.cs.

Métodos

- VentanaAvisoViewModel (string mensaje, Action cerrarVentana)

Propiedades

- string Mensaje [get]
- ReactiveCommand< Unit, Unit > BtnAceptar [get]

3.38.2. Constructores

VentanaAvisoViewModel()

```
BetaProyecto.ViewModels.VentanaAvisoViewModel.VentanaAvisoViewModel (
                string mensaje,
                Action cerrarVentana)
```

Definición en la línea 16 del archivo VentanaAvisoViewModel.cs.

3.38.3. Propiedades

BtnAceptar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.VentanaAvisoViewModel.BtnAceptar [get]

Definición en la línea 13 del archivo VentanaAvisoViewModel.cs.

Mensaje

string BetaProyecto.ViewModels.VentanaAvisoViewModel.Mensaje [get]

Definición en la línea 10 del archivo VentanaAvisoViewModel.cs.

3.39. Referencia de la clase BetaProyecto.ViewModels.VentanaConfirmacionViewModel

3.39.1. Detalles

Definición en la línea 7 del archivo VentanaConfirmacionViewModel.cs.

Métodos

- VentanaConfirmacionViewModel (string titulo, string mensaje, string textoSi, string textoNo, Action< bool > cerrarConResultado)

Propiedades

- string TituloCabecera [get]
- string MensajeCuerpo [get]
- string TextoBotonSi [get]
- string TextoBotonNo [get]
- ReactiveCommand< Unit, Unit > BtnSi [get]
- ReactiveCommand< Unit, Unit > BtnNo [get]

3.39.2. Constructores

VentanaConfirmacionViewModel()

```
BetaProyecto.ViewModels.VentanaConfirmacionViewModel.VentanaConfirmacionViewModel (
                string titulo,
                string mensaje,
                string textoSi,
                string textoNo,
                Action< bool > cerrarConResultado)
```

Definición en la línea 20 del archivo VentanaConfirmacionViewModel.cs.

3.39.3. Propiedades

BtnNo

```
ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.VentanaConfirmacionViewModel.BtnNo [get]
```

Definición en la línea 17 del archivo VentanaConfirmacionViewModel.cs.

BtnSi

```
ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.VentanaConfirmacionViewModel.BtnSi [get]
```

Definición en la línea 16 del archivo VentanaConfirmacionViewModel.cs.

MensajeCuerpo

string BetaProyecto.ViewModels.VentanaConfirmacionViewModel.MensajeCuerpo [get]

Definición en la línea 11 del archivo VentanaConfirmacionViewModel.cs.

TextoBotonNo

string BetaProyecto.ViewModels.VentanaConfirmacionViewModel.TextoBotonNo [get]

Definición en la línea 13 del archivo VentanaConfirmacionViewModel.cs.

TextoBotonSi

string BetaProyecto.ViewModels.VentanaConfirmacionViewModel.TextoBotonSi [get]

Definición en la línea 12 del archivo VentanaConfirmacionViewModel.cs.

TituloCabecera

string BetaProyecto.ViewModels.VentanaConfirmacionViewModel.TituloCabecera [get]

Definición en la línea 10 del archivo VentanaConfirmacionViewModel.cs.

3.40. Referencia de la clase BetaProyecto.ViewModels.ViewAyudaViewModel

3.40.1. Detalles

Definición en la línea 8 del archivo ViewAyudaViewModel.cs.

Métodos

- ViewAyudaViewModel ()

Propiedades

- Action? VolverAtras [get, set]
- ReactiveCommand< Unit, Unit > btnVolverAtras [get]

3.40.2. Constructores

ViewAyudaViewModel()

BetaProyecto.ViewModels.ViewAyudaViewModel.ViewAyudaViewModel ()

Definición en la línea 14 del archivo ViewAyudaViewModel.cs.

3.40.3. Propiedades

btnVolverAtras

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewAyudaViewModel.btnVolverAtras [get]

Definición en la línea 13 del archivo ViewAyudaViewModel.cs.

VolverAtras

Action? BetaProyecto.ViewModels.ViewAyudaViewModel.VolverAtras [get], [set]

Implementa BetaProyecto.ViewModels.INavegable.

Definición en la línea 11 del archivo ViewAyudaViewModel.cs.

3.41. Referencia de la clase BetaProyecto.ViewModels.ViewCancionesViewModel

3.41.1. Detalles

Definición en la línea 14 del archivo ViewCancionesViewModel.cs.

Métodos

- ViewCancionesViewModel (Canciones cancion, Action accionVolver, Action< Canciones >? accion↩ Reproducir, Action< Canciones > accionLike)

Propiedades

- Canciones Cancion [get, set]
- string IconoLike [get, set]
- string TxtMensajeTimer [get, set]
- string TxtVariableTimer [get, set]
- ReactiveCommand< Unit, Unit > BtnVolver [get]
- ReactiveCommand< Unit, Unit > BtnReproducir [get]
- ReactiveCommand< Unit, Unit > BtnLike [get]
- string DuracionFormateada [get]
- string FechaLanzamientoFormateada [get]

3.41.2. Constructores

ViewCancionesViewModel()

```
BetaProyecto.ViewModels.ViewCancionesViewModel.ViewCancionesViewModel (
                Canciones cancion,
                Action accionVolver,
                Action< Canciones >? accionReproducir,
                Action< Canciones > accionLike)
```

Definición en la línea 66 del archivo ViewCancionesViewModel.cs.

3.41.3. Funciones

ActualizarIconoLike()

void BetaProyecto.ViewModels.ViewCancionesViewModel.ActualizarIconoLike () [private]

Actualiza el icono de like para indicar si la canción actual está marcada como favorita.

Este método establece el icono similar en función de la presencia del identificador de la canción actual en la lista global de favoritos. Debe llamarse siempre que haya cambiado el estado favorito de la canción para asegurarse de que el icono siga sincronizado con los favoritos del usuario.

Definición en la línea 162 del archivo ViewCancionesViewModel.cs.

IniciarHiloActualizacion()

void BetaProyecto.ViewModels.ViewCancionesViewModel.IniciarHiloActualizacion (
CancellationToken token) [private]

Inicia un bucle de actualización en segundo plano que actualiza periódicamente la información actual de la canción hasta su cancelación es solicitado.

Este método ejecuta la lógica de actualización en un hilo en segundo plano y actualiza los elementos de la interfaz de usuario. usando el despachador. El ciclo de actualización continúa hasta que se indica el token de cancelación proporcionado. Intención para uso interno para mantener la interfaz de usuario sincronizada con los datos más recientes de las canciones.

Parámetros

token	Un token de cancelación que se puede usar para solicitar la finalización del ciclo de actualización.

Definición en la línea 103 del archivo ViewCancionesViewModel.cs.

3.41.4. Propiedades

BtnLike

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCancionesViewModel.BtnLike [get]

Definición en la línea 49 del archivo ViewCancionesViewModel.cs.

BtnReproducir

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCancionesViewModel.BtnReproducir [get]

Definición en la línea 48 del archivo ViewCancionesViewModel.cs.

BtnVolver

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCancionesViewModel.BtnVolver [get]

Definición en la línea 47 del archivo ViewCancionesViewModel.cs.

Cancion

Canciones BetaProyecto.ViewModels.ViewCancionesViewModel.Cancion [get], [set]

Definición en la línea 18 del archivo ViewCancionesViewModel.cs.

DuracionFormateada

string BetaProyecto.ViewModels.ViewCancionesViewModel.DuracionFormateada [get]

Definición en la línea 55 del archivo ViewCancionesViewModel.cs.

FechaLanzamientoFormateada

string BetaProyecto.ViewModels.ViewCancionesViewModel.FechaLanzamientoFormateada [get]

Definición en la línea 63 del archivo ViewCancionesViewModel.cs.

IconoLike

string BetaProyecto.ViewModels.ViewCancionesViewModel.IconoLike [get], [set]

Definición en la línea 26 del archivo ViewCancionesViewModel.cs.

TxtMensajeTimer

string BetaProyecto.ViewModels.ViewCancionesViewModel.TxtMensajeTimer [get], [set]

Definición en la línea 33 del archivo ViewCancionesViewModel.cs.

TxtVariableTimer

string BetaProyecto.ViewModels.ViewCancionesViewModel.TxtVariableTimer [get], [set]

Definición en la línea 40 del archivo ViewCancionesViewModel.cs.

3.42. Referencia de la clase BetaProyecto.ViewModels.ViewConfiguracionViewModel

3.42.1. Detalles

Definición en la línea 10 del archivo ViewConfiguracionViewModel.cs.

Métodos

- ViewConfiguracionViewModel (Action accionVolver, Action accionLogout, Action accionSalir, Action accionRefrescar)

Propiedades

- int IndiceFuente [get, set]
- int IndiceIdioma [get, set]
- bool IndiceTema [get, set]
- bool IndiceTemaOscuro [get, set]
- ReactiveCommand< Unit, Unit > BtnVolverAtras [get]
- ReactiveCommand< Unit, Unit > BtnCerrarSesion [get]
- ReactiveCommand< Unit, Unit > BtnSalirApp [get]

3.42.2. Constructores

ViewConfiguracionViewModel()

```
BetaProyecto.ViewModels.ViewConfiguracionViewModel.ViewConfiguracionViewModel (
                Action accionVolver,
                Action accionLogout,
                Action accionSalir,
                Action accionRefrescar)
```

Definición en la línea 70 del archivo ViewConfiguracionViewModel.cs.

3.42.3. Funciones

AplicarCambioFuente()

```
void BetaProyecto.ViewModels.ViewConfiguracionViewModel.AplicarCambioFuente (
                int indice)  [private]
```

Ejecuta el cambio de la fuente tipográfica de la aplicación basándose en el índice seleccionado.

Este método gestiona la personalización visual en tres niveles:

1. Visual: Cambia el diccionario de estilos de fuente de forma dinámica mediante ControladorDiccionarios.
2. Persistencia: Si la fuente es diferente a la actual, sincroniza la nueva preferencia en la base de datos MongoDB.
3. Estado Global: Actualiza la referencia en GlobalData.Instance para asegurar que la tipografía se mantenga durante la sesión activa.

Parámetros

indice	El índice del selector que determina la familia tipográfica (0: Lexend, 1: Carlito, 2: Arial, etc.).

Definición en la línea 192 del archivo ViewConfiguracionViewModel.cs.

AplicarCambioIdioma()

void BetaProyecto.ViewModels.ViewConfiguracionViewModel.AplicarCambioIdioma (
int indice) [private]

Ejecuta el cambio de idioma de la interfaz de usuario basándose en el índice seleccionado.

Este método gestiona la internacionalización en tres niveles:

1. Visual: Cambia el diccionario de strings de forma dinámica mediante ControladorDiccionarios.
2. Persistencia: Si el idioma es diferente al actual, sincroniza la nueva preferencia en la base de datos MongoDB.
3. Estado Global: Actualiza la referencia en GlobalData.Instance para asegurar la persistencia durante la sesión activa.

Parámetros

indice	El índice del selector: 0 para ”Spanish” y 1 para ”English”.

Definición en la línea 168 del archivo ViewConfiguracionViewModel.cs.

AplicarCambioTema()

void BetaProyecto.ViewModels.ViewConfiguracionViewModel.AplicarCambioTema (
bool esClaro) [private]

Ejecuta el cambio de apariencia visual de la aplicación entre modo claro y modo oscuro.

Este método gestiona el cambio de tema en tres niveles:

1. Visual: Aplica el diccionario de recursos de forma instantánea mediante ControladorDiccionarios.
2. Persistencia: Si el tema es distinto al actual, sincroniza la preferencia en la base de datos MongoDB.
3. Estado Global: Actualiza la propiedad en GlobalData.Instance para mantener la consistencia en toda la sesión.

Parámetros

esClaro	Indica si se debe aplicar el ”ModoClaro” (true) o el ”ModoOscuro” (false).

Definición en la línea 139 del archivo ViewConfiguracionViewModel.cs.

CargarEstadoInicial()

void BetaProyecto.ViewModels.ViewConfiguracionViewModel.CargarEstadoInicial () [private]

Sincroniza la interfaz de configuración con las preferencias del usuario almacenadas en los datos globales.

Este método recupera los valores actuales de tema, idioma y tipografía desde GlobalData.Instance. Posteriormente, traduce estas cadenas de texto a los índices correspondientes que utilizan los selectores de la vista y fuerza la notificación de cambio de propiedades mediante RaisePropertyChanged para que la UI refleje el estado real de la configuración.

Definición en la línea 98 del archivo ViewConfiguracionViewModel.cs.

GuardarConfiguracionEnMongo()

void BetaProyecto.ViewModels.ViewConfiguracionViewModel.GuardarConfiguracionEnMongo (
string? temaNuevo,
string? idiomaNuevo,
string? fuenteNuevo) [private]

Sincroniza de forma asíncrona las preferencias de personalización del usuario en la base de datos MongoDB.

Este método implementa una lógica de actualización parcial. Construye un objeto de configuración combinando los nuevos valores proporcionados con los valores actuales almacenados en GlobalData.Instance. Si un parámetro se recibe como null, se preserva el valor existente. La actualización se lanza mediante una tarea en segundo plano para no bloquear la interfaz.

Parámetros

temaNuevo	El nombre del nuevo tema visual o null si no ha cambiado.
idiomaNuevo	El nombre del nuevo idioma de la interfaz o null si no ha cambiado.
fuenteNuevo	El nombre de la nueva familia tipográfica o null si no ha cambiado.

Definición en la línea 226 del archivo ViewConfiguracionViewModel.cs.

3.42.4. Propiedades

BtnCerrarSesion

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewConfiguracionViewModel.BtnCerrarSesion [get]

Definición en la línea 66 del archivo ViewConfiguracionViewModel.cs.

BtnSalirApp

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewConfiguracionViewModel.BtnSalirApp [get]

Definición en la línea 67 del archivo ViewConfiguracionViewModel.cs.

BtnVolverAtras

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewConfiguracionViewModel.BtnVolverAtras [get]

Definición en la línea 65 del archivo ViewConfiguracionViewModel.cs.

IndiceFuente

int BetaProyecto.ViewModels.ViewConfiguracionViewModel.IndiceFuente [get], [set]

Definición en la línea 19 del archivo ViewConfiguracionViewModel.cs.

IndiceIdioma

int BetaProyecto.ViewModels.ViewConfiguracionViewModel.IndiceIdioma [get], [set]

Definición en la línea 31 del archivo ViewConfiguracionViewModel.cs.

IndiceTema

bool BetaProyecto.ViewModels.ViewConfiguracionViewModel.IndiceTema [get], [set]

Definición en la línea 43 del archivo ViewConfiguracionViewModel.cs.

IndiceTemaOscuro

bool BetaProyecto.ViewModels.ViewConfiguracionViewModel.IndiceTemaOscuro [get], [set]

Definición en la línea 55 del archivo ViewConfiguracionViewModel.cs.

3.43. Referencia de la clase BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel

3.43.1. Detalles

Definición en la línea 15 del archivo ViewCrearListaPersonalizadaViewModel.cs.

Métodos

- ViewCrearListaPersonalizadaViewModel (Action accionVolver)

Propiedades

- string TxtNombre [get, set]
- string TxtDescripcion [get, set]
- string RutaImagen [get, set]
- bool TieneImagen [get]
- Bitmap? ImagenPortada [get, set]
- string TxtBusqueda [get, set]
- ObservableCollection< Canciones > ListaResultados [get, set]
- ObservableCollection< Canciones > ListaCancionesSeleccionadas [get, set]
- bool EstaCargando [get, set]
- ReactiveCommand< Unit, Unit > BtnVolverAtras [get]
- ReactiveCommand< Unit, Unit > BtnCrear [get]
- ReactiveCommand< Unit, Unit > BtnBuscarCanciones [get]
- ReactiveCommand< Canciones, Unit > BtnAgregarCancion [get]
- ReactiveCommand< Canciones, Unit > BtnEliminarCancion [get]

3.43.2. Constructores

ViewCrearListaPersonalizadaViewModel()

BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.ViewCrearListaPersonalizadaViewModel (
Action accionVolver)

Definición en la línea 97 del archivo ViewCrearListaPersonalizadaViewModel.cs.

3.43.3. Funciones

AgregarCancion()

void BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.AgregarCancion (
Canciones cancion) [private]

Agrega la canción seleccionada a la lista temporal de canciones que formarán parte de la nueva playlist.

Este método realiza tres acciones clave:

1. Verifica que la canción no esté ya añadida para evitar duplicados.
2. Mueve visualmente la canción: la añade a ListaCancionesSeleccionadas y la elimina de ListaResultados.
3. Limpia el campo de búsqueda para facilitar una nueva consulta inmediata.

Parámetros

cancion	El objeto Canciones que el usuario ha seleccionado para añadir.

Definición en la línea 167 del archivo ViewCrearListaPersonalizadaViewModel.cs.

BuscarCanciones()

async void BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.BuscarCanciones () [private]

Realiza una búsqueda asíncrona de canciones en la base de datos utilizando el texto introducido por el usuario.

Este método consulta MongoDB y filtra los resultados obtenidos para excluir aquellas canciones que ya están presentes en la lista de selección (ListaCancionesSeleccionadas). Esto evita duplicados visuales y actualiza la colección de resultados disponibles para añadir.

Definición en la línea 140 del archivo ViewCrearListaPersonalizadaViewModel.cs.

CargarImagenLocal()

void BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.CargarImagenLocal (
string ruta) [private]

Intenta cargar y visualizar una imagen local desde la ruta especificada.

Este método gestiona de forma segura la carga de archivos de imagen. Si el archivo no existe o el formato no es válido (lanzando una excepción), la propiedad ImagenPortada se establece en null para evitar errores visuales.

Al finalizar, fuerza una notificación de cambio en TieneImagen para que la interfaz actualice la visibilidad de los controles dependientes (como el botón de "Quitar imagen").

Parámetros

ruta	La ruta absoluta del sistema de archivos donde se encuentra la imagen.

Definición en la línea 205 del archivo ViewCrearListaPersonalizadaViewModel.cs.

CrearLista()

async Task BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.CrearLista () [private]

Orquesta el proceso completo de creación de una nueva lista de reproducción personalizada de forma asíncrona.

Este método sigue un flujo de transacciones paso a paso:

1. Carga de medios: Sube la imagen de portada seleccionada al servicio de almacenamiento en la nube.
2. Construcción del modelo: Crea una instancia de ListaPersonalizada con los metadatos y la selección de canciones actual.
3. Persistencia: Invoca al cliente de MongoDB para guardar la nueva lista en la base de datos.

Gestiona los estados de carga (EstaCargando) para bloquear la UI durante el proceso y maneja excepciones globales.

Devuelve

Una Task que representa la operación asíncrona.

Definición en la línea 233 del archivo ViewCrearListaPersonalizadaViewModel.cs.

EliminarCancion()

void BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.EliminarCancion (Canciones cancion) [private]

Elimina una canción de la lista de canciones seleccionadas para la nueva playlist.

Permite al usuario rectificar su selección quitando canciones individuales de ListaCancionesSeleccionadas antes de guardar la lista definitiva. La interfaz de usuario refleja el cambio inmediatamente.

Parámetros

cancion	El objeto Canciones que se desea descartar de la selección actual.

Definición en la línea 185 del archivo ViewCrearListaPersonalizadaViewModel.cs.

3.43.4. Propiedades

BtnAgregarCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.BtnAgregarCancion [get]

Definición en la línea 93 del archivo ViewCrearListaPersonalizadaViewModel.cs.

BtnBuscarCanciones

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.BtnBuscarCanciones [get]

Definición en la línea 92 del archivo ViewCrearListaPersonalizadaViewModel.cs.

BtnCrear

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.BtnCrear [get]

Definición en la línea 91 del archivo ViewCrearListaPersonalizadaViewModel.cs.

BtnEliminarCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.BtnEliminarCancion [get]

Definición en la línea 94 del archivo ViewCrearListaPersonalizadaViewModel.cs.

BtnVolverAtras

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.BtnVolverAtras [get]

Definición en la línea 90 del archivo ViewCrearListaPersonalizadaViewModel.cs.

EstaCargando

bool BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.EstaCargando [get], [set]

Definición en la línea 83 del archivo ViewCrearListaPersonalizadaViewModel.cs.

ImagenPortada

Bitmap? BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.ImagenPortada [get], [set]

Definición en la línea 53 del archivo ViewCrearListaPersonalizadaViewModel.cs.

ListaCancionesSeleccionadas

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.ListaCanciones↩ Seleccionadas [get], [set]

Definición en la línea 75 del archivo ViewCrearListaPersonalizadaViewModel.cs.

ListaResultados

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.ListaResultados [get], [set]

Definición en la línea 68 del archivo ViewCrearListaPersonalizadaViewModel.cs.

RutaImagen

string BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.RutaImagen [get], [set]

Definición en la línea 41 del archivo ViewCrearListaPersonalizadaViewModel.cs.

TieneImagen

bool BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.TieneImagen [get]

Definición en la línea 50 del archivo ViewCrearListaPersonalizadaViewModel.cs.

TxtBusqueda

string BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.TxtBusqueda [get], [set]

Definición en la línea 61 del archivo ViewCrearListaPersonalizadaViewModel.cs.

TxtDescripcion

string BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.TxtDescripcion [get], [set]

Definición en la línea 33 del archivo ViewCrearListaPersonalizadaViewModel.cs.

TxtNombre

string BetaProyecto.ViewModels.ViewCrearListaPersonalizadaViewModel.TxtNombre [get], [set]

Definición en la línea 26 del archivo ViewCrearListaPersonalizadaViewModel.cs.

3.44. Referencia de la clase BetaProyecto.ViewModels.ViewCrearReporteViewModel

3.44.1. Detalles

Definición en la línea 11 del archivo ViewCrearReporteViewModel.cs.

Métodos

- ViewCrearReporteViewModel (Canciones cancion, Action accionVolver)

Propiedades

- Canciones CancionAReportar [get]
- List< string > TiposDeProblema [get]
- string TipoSeleccionado [get, set]
- string DescripcionTexto [get, set]
- string MensajeEstado [get, set]
- ReactiveCommand< Unit, Unit > BtnEnviarReporte [get]
- ReactiveCommand< Unit, Unit > BtnCancelar [get]

3.44.2. Constructores

ViewCrearReporteViewModel()

BetaProyecto.ViewModels.ViewCrearReporteViewModel.ViewCrearReporteViewModel (
Canciones cancion,
Action accionVolver)

Definición en la línea 57 del archivo ViewCrearReporteViewModel.cs.

3.44.3. Funciones

EnviarReporteAsync()

async Task BetaProyecto.ViewModels.ViewCrearReporteViewModel.EnviarReporteAsync () [private]

Envía un informe de forma asíncrona utilizando los detalles del informe actual y actualiza el mensaje de estado en función del resultado.

Si el informe se envía con éxito, se actualiza el mensaje de estado para indicar que ha tenido éxito. y el método navega de regreso después de un breve retraso. Si se produce un error, el mensaje de estado se actualiza a indica el fallo.

Devuelve

Devuelve una tarea que representa la operación asíncrona.

Definición en la línea 82 del archivo ViewCrearReporteViewModel.cs.

3.44.4. Propiedades

BtnCancelar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearReporteViewModel.BtnCancelar [get]

Definición en la línea 54 del archivo ViewCrearReporteViewModel.cs.

BtnEnviarReporte

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearReporteViewModel.BtnEnviarReporte [get]

Definición en la línea 53 del archivo ViewCrearReporteViewModel.cs.

CancionAReportar

Canciones BetaProyecto.ViewModels.ViewCrearReporteViewModel.CancionAReportar [get]

Definición en la línea 15 del archivo ViewCrearReporteViewModel.cs.

DescripcionTexto

string BetaProyecto.ViewModels.ViewCrearReporteViewModel.DescripcionTexto [get], [set]

Definición en la línea 39 del archivo ViewCrearReporteViewModel.cs.

MensajeEstado

string BetaProyecto.ViewModels.ViewCrearReporteViewModel.MensajeEstado [get], [set]

Definición en la línea 46 del archivo ViewCrearReporteViewModel.cs.

TiposDeProblema

List<string> BetaProyecto.ViewModels.ViewCrearReporteViewModel.TiposDeProblema [get]

Valor inicial:

```
= new List<string>
      {
         "Copyright / Derechos de autor",
         "Contenido ofensivo o inapropiado",
         "Audio defectuoso o silencio",
         "Spam / Información falsa",
         "Otro"
      }
```

Definición en la línea 22 del archivo ViewCrearReporteViewModel.cs.

TipoSeleccionado

string BetaProyecto.ViewModels.ViewCrearReporteViewModel.TipoSeleccionado [get], [set]

Definición en la línea 32 del archivo ViewCrearReporteViewModel.cs.

3.45. Referencia de la clase BetaProyecto.ViewModels.ViewCrearUsuarioViewModel

3.45.1. Detalles

Definición en la línea 14 del archivo ViewCrearUsuarioViewModel.cs.

Métodos

- ViewCrearUsuarioViewModel (Action accionVolver)
- void CargarImagenPrevia (string ruta)

  Carga una imagen de vista previa desde la ruta del archivo especificada y actualiza la referencia de imagen de perfil del usuario.

Propiedades

- Usuarios NuevoUsuario [get, set]
- string ConfirmarPass [get, set]
- bool EstaCargando [get, set]
- Bitmap? FotoPerfilBitmap [get, set]
- List< string > ListaPaises [get]
- ReactiveCommand< Unit, Unit > BtnRegistrarse [get]
- ReactiveCommand< Unit, Unit > BtnVolver [get]

3.45.2. Constructores

ViewCrearUsuarioViewModel()

BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.ViewCrearUsuarioViewModel (
Action accionVolver)

Definición en la línea 61 del archivo ViewCrearUsuarioViewModel.cs.

3.45.3. Funciones

CargarImagenPrevia()

void BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.CargarImagenPrevia (
string ruta)

Carga una imagen de vista previa desde la ruta del archivo especificada y actualiza la referencia de imagen de perfil del usuario.

Si el archivo no existe o no es una imagen válida, la imagen de previsualización se borra. El método no genera una excepción si la carga falla.

Parámetros

ruta	La ruta del archivo de la imagen a cargar como una vista previa. Debe hacer referencia a un archivo existente.

Definición en la línea 199 del archivo ViewCrearUsuarioViewModel.cs.

RegistrarseTask()

async Task BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.RegistrarseTask () [private]

Gestiona el proceso de registro de usuarios de forma asíncrona, incluida la validación, la carga de imágenes de perfil y creación de cuenta.

Muestra las alertas apropiadas al usuario en caso de errores de validación, problemas de conexión o resultados del registro. Si el registro es exitoso, se notifica al usuario y se lo redirige a la cuenta. pantalla. Este método evita los intentos de registro simultáneos comprobando y configurando una carga estado.

Devuelve

Devuelve una tarea que representa la operación de registro asíncrono.

Definición en la línea 109 del archivo ViewCrearUsuarioViewModel.cs.

3.45.4. Propiedades

BtnRegistrarse

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.BtnRegistrarse [get]

Definición en la línea 58 del archivo ViewCrearUsuarioViewModel.cs.

BtnVolver

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.BtnVolver [get]

Definición en la línea 59 del archivo ViewCrearUsuarioViewModel.cs.

ConfirmarPass

string BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.ConfirmarPass [get], [set]

Definición en la línea 33 del archivo ViewCrearUsuarioViewModel.cs.

EstaCargando

bool BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.EstaCargando [get], [set]

Definición en la línea 41 del archivo ViewCrearUsuarioViewModel.cs.

FotoPerfilBitmap

Bitmap? BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.FotoPerfilBitmap [get], [set]

Definición en la línea 48 del archivo ViewCrearUsuarioViewModel.cs.

ListaPaises

List<string> BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.ListaPaises [get]

Definición en la línea 55 del archivo ViewCrearUsuarioViewModel.cs.

NuevoUsuario

Usuarios BetaProyecto.ViewModels.ViewCrearUsuarioViewModel.NuevoUsuario [get], [set]

Definición en la línea 25 del archivo ViewCrearUsuarioViewModel.cs.

3.46. Referencia de la clase BetaProyecto.ViewModels.ViewCuentaViewModel

3.46.1. Detalles

Definición en la línea 10 del archivo ViewCuentaViewModel.cs.

Métodos

- ViewCuentaViewModel ()

Propiedades

- string NombreUsuario [get, set]
- string Email [get, set]
- string Pais [get, set]
- DateTimeOffset? FechaNacimiento [get, set]
- int IndexPrivacidad [get, set]
- ReactiveCommand< Unit, Unit > BtnGuardar [get]
- ReactiveCommand< Unit, Unit > BtnRefrescar [get]

3.46.2. Constructores

ViewCuentaViewModel()

BetaProyecto.ViewModels.ViewCuentaViewModel.ViewCuentaViewModel ()

Definición en la línea 57 del archivo ViewCuentaViewModel.cs.

3.46.3. Funciones

CargarDatos()

void BetaProyecto.ViewModels.ViewCuentaViewModel.CargarDatos () [private]

Recupera y sincroniza la información del perfil del usuario desde los datos globales para su edición en la interfaz.

Este método actúa como un mapeador entre GlobalData.Instance y las propiedades vinculadas de la vista. Realiza conversiones de tipos necesarias, como la transformación de DateTime a DateTimeOffset para el selector de fecha, y traduce el estado booleano de privacidad a un índice numérico compatible con los controles de selección de la UI.

Definición en la línea 78 del archivo ViewCuentaViewModel.cs.

GuardarCambios()

async Task BetaProyecto.ViewModels.ViewCuentaViewModel.GuardarCambios () [private]

Procesa y persiste de forma asíncrona las modificaciones realizadas en el perfil del usuario tanto en la base de datos como en el estado global.

Este método realiza una validación y transformación de datos antes de la persistencia:

1. Conversión: Transforma el objeto DateTimeOffset de la interfaz a DateTime y el índice de privacidad a un valor booleano.
2. Sincronización remota: Invoca al cliente de MongoDB para actualizar los documentos en la nube.
3. Actualización local: Si la operación remota es exitosa, sincroniza los nuevos valores en GlobalData.Instance para mantener la consistencia en la sesión actual.

Notifica el resultado de la operación al usuario mediante el servicio de diálogos y registra errores críticos en la consola de depuración.

Devuelve

Una tarea que representa la operación de guardado asíncrona.

Definición en la línea 111 del archivo ViewCuentaViewModel.cs.

3.46.4. Propiedades

BtnGuardar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCuentaViewModel.BtnGuardar [get]

Definición en la línea 54 del archivo ViewCuentaViewModel.cs.

BtnRefrescar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewCuentaViewModel.BtnRefrescar [get]

Definición en la línea 55 del archivo ViewCuentaViewModel.cs.

Email

string BetaProyecto.ViewModels.ViewCuentaViewModel.Email [get], [set]

Definición en la línea 24 del archivo ViewCuentaViewModel.cs.

FechaNacimiento

DateTimeOffset? BetaProyecto.ViewModels.ViewCuentaViewModel.FechaNacimiento [get], [set]

Definición en la línea 39 del archivo ViewCuentaViewModel.cs.

IndexPrivacidad

int BetaProyecto.ViewModels.ViewCuentaViewModel.IndexPrivacidad [get], [set]

Definición en la línea 47 del archivo ViewCuentaViewModel.cs.

NombreUsuario

string BetaProyecto.ViewModels.ViewCuentaViewModel.NombreUsuario [get], [set]

Definición en la línea 17 del archivo ViewCuentaViewModel.cs.

Pais

string BetaProyecto.ViewModels.ViewCuentaViewModel.Pais [get], [set]

Definición en la línea 31 del archivo ViewCuentaViewModel.cs.

3.47. Referencia de la clase BetaProyecto.ViewModels.ViewEditarCancionViewModel

3.47.1. Detalles

Definición en la línea 16 del archivo ViewEditarCancionViewModel.cs.

Métodos

- ViewEditarCancionViewModel (Canciones cancion, Action accionVolver)

Propiedades

- string TxtTitulo [get, set]
- ObservableCollection< string > ListaGeneros [get, set]
- string GeneroSeleccionado [get, set]
- ObservableCollection< string > ListaGenerosSeleccionados [get, set]
- string TxtBusqueda [get, set]
- ObservableCollection< Usuarios > ListaResultados [get, set]
- ObservableCollection< Usuarios > ListaArtistas [get, set]
- string RutaImagen [get, set]
- bool TieneImagen [get, set]
- Bitmap? ImagenPortada [get, set]
- bool EstaCargando [get, set]
- ReactiveCommand< Unit, Unit > BtnCancelar [get]
- ReactiveCommand< Unit, Unit > BtnGuardar [get]
- ReactiveCommand< Unit, Unit > BtnAgregarGenero [get]
- ReactiveCommand< string, Unit > BtnEliminarGenero [get]
- ReactiveCommand< Unit, Unit > BtnBuscarUsuarios [get]
- ReactiveCommand< Usuarios, Unit > BtnAgregarUsuario [get]
- ReactiveCommand< Usuarios, Unit > BtnEliminarUsuario [get]

3.47.2. Constructores

ViewEditarCancionViewModel()

BetaProyecto.ViewModels.ViewEditarCancionViewModel.ViewEditarCancionViewModel (
Canciones cancion,
Action accionVolver)

Definición en la línea 123 del archivo ViewEditarCancionViewModel.cs.

3.47.3. Funciones

AgregarGenero()

void BetaProyecto.ViewModels.ViewEditarCancionViewModel.AgregarGenero () [private]

Añade el género seleccionado actualmente a la lista de géneros asociados, validando que no esté vacío y que no se haya añadido previamente.

Este método gestiona la selección de etiquetas musicales mediante los siguientes pasos:

1. Validación de entrada: Verifica si GeneroSeleccionado contiene un valor válido y no nulo.
2. Control de duplicados: Comprueba si el género ya existe en ListaGenerosSeleccionados mediante una comparación insensible a mayúsculas.
3. Actualización: Si es un género nuevo, lo añade a la colección. En caso contrario, notifica al usuario a través de _dialogoService.

Al finalizar, restablece la propiedad GeneroSeleccionado a nulo para limpiar el selector de la interfaz.

Definición en la línea 292 del archivo ViewEditarCancionViewModel.cs.

AgregarUsuario()

void BetaProyecto.ViewModels.ViewEditarCancionViewModel.AgregarUsuario (
Usuarios usuario) [private]

Añade un usuario a la lista de artistas seleccionados, evitando duplicados y limpiando los resultados de búsqueda actuales.

Este método gestiona la selección de colaboradores mediante los siguientes pasos:

1. Validación de existencia: Verifica mediante el identificador único si el usuario ya se encuentra en ListaArtistas.
2. Actualización de colección: Si el usuario no es un duplicado, se añade a la lista de artistas vinculados.
3. Limpieza de interfaz: Independientemente del resultado, se restablece TxtBusqueda y se vacía ListaResultados para preparar una nueva consulta.

Parámetros

usuario	El objeto de tipo Usuarios que se desea vincular a la canción o lista.

Definición en la línea 363 del archivo ViewEditarCancionViewModel.cs.

BuscarUsuarios()

async void BetaProyecto.ViewModels.ViewEditarCancionViewModel.BuscarUsuarios () [private]

Realiza una búsqueda asíncrona de usuarios en la base de datos basándose en el texto introducido, filtrando aquellos que ya han sido seleccionados.

Este método gestiona la recuperación de perfiles mediante los siguientes pasos:

1. Validación de servicio: Verifica la disponibilidad del cliente de MongoDB a través de MongoClientSingleton.
2. Consulta con exclusión: Ejecuta la búsqueda utilizando TxtBusqueda y envía una lista de IDs de ListaArtistas para evitar resultados duplicados.
3. Actualización de interfaz: Si se obtienen resultados, inicializa ListaResultados con una nueva colección observable para refrescar la vista.

Definición en la línea 343 del archivo ViewEditarCancionViewModel.cs.

CargarColaboradoresOriginales()

async Task BetaProyecto.ViewModels.ViewEditarCancionViewModel.CargarColaboradoresOriginales () [private]

Recupera de forma asíncrona la información detallada de los colaboradores originales de la canción basándose en sus identificadores.

Este método gestiona la conversión de la lista de IDs almacenada en los metadatos de la canción a objetos de tipo Usuarios. Consulta la base de datos a través de MongoClientSingleton y, tras obtener los perfiles correspondientes, inicializa la propiedad ListaArtistas con una nueva colección observable para su representación en la interfaz.

Devuelve

Una tarea que representa la operación de carga asíncrona.

Definición en la línea 207 del archivo ViewEditarCancionViewModel.cs.

CargarGenerosDisponibles()

async Task BetaProyecto.ViewModels.ViewEditarCancionViewModel.CargarGenerosDisponibles () [private]

Recupera de forma asíncrona el catálogo completo de géneros musicales definidos en el sistema.

Este método establece conexión con la base de datos a través de MongoClientSingleton para obtener la lista maestra de nombres de géneros. Una vez recibidos, inicializa la propiedad ListaGeneros con una nueva colección observable, permitiendo que los selectores de la interfaz de usuario se pueblen dinámicamente con los valores actualizados.

Devuelve

Una tarea que representa la operación de carga asíncrona.

Definición en la línea 189 del archivo ViewEditarCancionViewModel.cs.

CargarImagenDesdeUrl()

async Task BetaProyecto.ViewModels.ViewEditarCancionViewModel.CargarImagenDesdeUrl (
string url) [private]

Descarga de forma asíncrona una imagen desde una dirección URL y la asigna al mapa de bits de la portada.

Este método gestiona la recuperación de recursos remotos mediante los siguientes pasos:

1. Petición HTTP: Utiliza un HttpClient para obtener el flujo de bytes de la imagen desde la red.
2. Procesamiento de Memoria: Transfiere los bytes a un System.IO.MemoryStream para su decodificación.
3. Asignación Visual: Inicializa la propiedad ImagenPortada con el nuevo Bitmap y actualiza el estado de TieneImagen.

El método incluye un bloque try-catch silencioso para asegurar que fallos en la red o URLs inválidas no interrumpan la ejecución de la aplicación.

Parámetros

url	La dirección URL completa de la imagen que se desea cargar.

Devuelve

Una tarea que representa la operación de carga asíncrona.

Definición en la línea 232 del archivo ViewEditarCancionViewModel.cs.

CargarImagenLocal()

void BetaProyecto.ViewModels.ViewEditarCancionViewModel.CargarImagenLocal (
string ruta) [private]

Intenta cargar y visualizar una imagen desde el almacenamiento local del sistema.

Este método gestiona la carga de recursos gráficos locales mediante los siguientes pasos:

1. Validación: Comprueba la existencia física del archivo en la ruta proporcionada.
2. Decodificación: Si el archivo existe, inicializa la propiedad ImagenPortada con un nuevo objeto Bitmap.
3. Control de Estado: Actualiza la propiedad booleana TieneImagen y notifica el cambio a la interfaz mediante RaisePropertyChanged.

El método captura cualquier excepción durante la lectura para evitar interrupciones en la ejecución, asegurando que el estado de la UI se mantenga consistente.

Parámetros

ruta	La ruta absoluta del archivo de imagen en el disco local.

Definición en la línea 265 del archivo ViewEditarCancionViewModel.cs.

EliminarGenero()

void BetaProyecto.ViewModels.ViewEditarCancionViewModel.EliminarGenero (
string genero) [private]

Elimina un género específico de la lista de géneros seleccionados para la canción.

Este método gestiona la edición de etiquetas musicales mediante los siguientes pasos:

1. Validación: Verifica si el género proporcionado existe dentro de la colección ListaGenerosSeleccionados.
2. Remoción: Si se encuentra la coincidencia, elimina el elemento de la lista.
3. Sincronización: La interfaz de usuario se actualiza automáticamente al ser una colección de tipo observable.

Parámetros

genero	El nombre del género que se desea remover de la selección actual.

Definición en la línea 324 del archivo ViewEditarCancionViewModel.cs.

EliminarUsuario()

void BetaProyecto.ViewModels.ViewEditarCancionViewModel.EliminarUsuario (
Usuarios usuario) [private]

Elimina un usuario de la lista de artistas seleccionados, validando que no sea el usuario que ha iniciado sesión.

Este método gestiona la remoción de colaboradores mediante los siguientes pasos:

1. Validación de identidad: Comprueba si el usuario a eliminar coincide con el ID del usuario actual en GlobalData.Instance.UserIdGD.
2. Restricción de seguridad: Si coinciden, se muestra una alerta de error mediante _dialogoService para impedir que el usuario se elimine a sí mismo.
3. Actualización: Si la validación es correcta y el usuario existe en la colección, se procede a removerlo de ListaArtistas.

Parámetros

usuario	El objeto de tipo Usuarios que se desea remover de la selección actual.

Definición en la línea 386 del archivo ViewEditarCancionViewModel.cs.

GuardarCambios()

async Task BetaProyecto.ViewModels.ViewEditarCancionViewModel.GuardarCambios () [private]

Procesa y persiste de forma asíncrona las modificaciones realizadas en una canción existente, gestionando la actualización de medios y metadatos.

Este método orquesta la actualización de la canción siguiendo un flujo transaccional:

1. Gestión de Imagen: Detecta si la ruta de la imagen es local o remota. Si es local, procede a subir el nuevo archivo mediante _storageService.
2. Sincronización remota: Envía los nuevos títulos, IDs de colaboradores y géneros al cliente de Mongo↩ DB para actualizar el registro físico.
3. Actualización de estado local: Si la persistencia es exitosa, sincroniza los cambios en el objeto _cancionOriginal para asegurar la consistencia visual al regresar a la vista anterior.

Durante el proceso, controla la propiedad EstaCargando para feedback visual y gestiona posibles excepciones mediante el servicio de diálogos.

Devuelve

Una tarea que representa la operación de guardado asíncrona.

Definición en la línea 413 del archivo ViewEditarCancionViewModel.cs.

3.47.4. Propiedades

BtnAgregarGenero

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnAgregarGenero [get]

Definición en la línea 115 del archivo ViewEditarCancionViewModel.cs.

BtnAgregarUsuario

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnAgregarUsuario [get]

Definición en la línea 118 del archivo ViewEditarCancionViewModel.cs.

BtnBuscarUsuarios

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnBuscarUsuarios [get]

Definición en la línea 117 del archivo ViewEditarCancionViewModel.cs.

BtnCancelar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnCancelar [get]

Definición en la línea 113 del archivo ViewEditarCancionViewModel.cs.

BtnEliminarGenero

ReactiveCommand<string, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnEliminarGenero [get]

Definición en la línea 116 del archivo ViewEditarCancionViewModel.cs.

BtnEliminarUsuario

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnEliminarUsuario [get]

Definición en la línea 119 del archivo ViewEditarCancionViewModel.cs.

BtnGuardar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarCancionViewModel.BtnGuardar [get]

Definición en la línea 114 del archivo ViewEditarCancionViewModel.cs.

EstaCargando

bool BetaProyecto.ViewModels.ViewEditarCancionViewModel.EstaCargando [get], [set]

Definición en la línea 106 del archivo ViewEditarCancionViewModel.cs.

GeneroSeleccionado

string BetaProyecto.ViewModels.ViewEditarCancionViewModel.GeneroSeleccionado [get], [set]

Definición en la línea 44 del archivo ViewEditarCancionViewModel.cs.

ImagenPortada

Bitmap? BetaProyecto.ViewModels.ViewEditarCancionViewModel.ImagenPortada [get], [set]

Definición en la línea 98 del archivo ViewEditarCancionViewModel.cs.

ListaArtistas

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewEditarCancionViewModel.ListaArtistas [get], [set]

Definición en la línea 72 del archivo ViewEditarCancionViewModel.cs.

ListaGeneros

ObservableCollection<string> BetaProyecto.ViewModels.ViewEditarCancionViewModel.ListaGeneros [get], [set]

Definición en la línea 37 del archivo ViewEditarCancionViewModel.cs.

ListaGenerosSeleccionados

ObservableCollection<string> BetaProyecto.ViewModels.ViewEditarCancionViewModel.ListaGenerosSeleccionados [get], [set]

Definición en la línea 51 del archivo ViewEditarCancionViewModel.cs.

ListaResultados

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewEditarCancionViewModel.ListaResultados [get], [set]

Definición en la línea 65 del archivo ViewEditarCancionViewModel.cs.

RutaImagen

string BetaProyecto.ViewModels.ViewEditarCancionViewModel.RutaImagen [get], [set]

Definición en la línea 79 del archivo ViewEditarCancionViewModel.cs.

TieneImagen

bool BetaProyecto.ViewModels.ViewEditarCancionViewModel.TieneImagen [get], [set]

Definición en la línea 91 del archivo ViewEditarCancionViewModel.cs.

TxtBusqueda

string BetaProyecto.ViewModels.ViewEditarCancionViewModel.TxtBusqueda [get], [set]

Definición en la línea 58 del archivo ViewEditarCancionViewModel.cs.

TxtTitulo

string BetaProyecto.ViewModels.ViewEditarCancionViewModel.TxtTitulo [get], [set]

Definición en la línea 30 del archivo ViewEditarCancionViewModel.cs.

3.48. Referencia de la clase BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel

3.48.1. Detalles

Definición en la línea 16 del archivo ViewEditarListaPersonalizadaViewModel.cs.

Métodos

- ViewEditarListaPersonalizadaViewModel (ListaPersonalizada playlist, Action accionVolver)

Propiedades

- string TxtNombre [get, set]
- string TxtDescripcion [get, set]
- string RutaImagen [get, set]
- bool TieneImagen [get, set]
- Bitmap? ImagenPortada [get, set]
- string TxtBusqueda [get, set]
- ObservableCollection< Canciones > ListaResultados [get, set]
- ObservableCollection< Canciones > ListaCancionesSeleccionadas [get, set]
- bool EstaCargando [get, set]
- ReactiveCommand< Unit, Unit > BtnAtras [get]
- ReactiveCommand< Unit, Unit > BtnGuardar [get]
- ReactiveCommand< Unit, Unit > BtnBuscarCanciones [get]
- ReactiveCommand< Canciones, Unit > BtnAgregarCancion [get]
- ReactiveCommand< Canciones, Unit > BtnEliminarCancion [get]

3.48.2. Constructores

ViewEditarListaPersonalizadaViewModel()

BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.ViewEditarListaPersonalizadaViewModel (
ListaPersonalizada playlist,
Action accionVolver)

Definición en la línea 108 del archivo ViewEditarListaPersonalizadaViewModel.cs.

3.48.3. Funciones

AgregarCancion()

void BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.AgregarCancion (
Canciones cancion) [private]

Incorpora una canción específica a la lista de selección actual y limpia el estado de búsqueda.

Este método gestiona la selección de pistas musicales mediante los siguientes pasos:

1. Validación de unicidad: Verifica que la canción no haya sido agregada previamente comparando su identificador único.
2. Transferencia de estado: Añade la canción a ListaCancionesSeleccionadas y la remueve simultáneamente de la lista de resultados de búsqueda para evitar duplicidad visual.
3. Reinicio de filtros: Restablece la cadena de búsqueda TxtBusqueda para facilitar una nueva consulta.

Parámetros

cancion	El objeto de tipo Canciones que se desea añadir a la lista o playlist.

Definición en la línea 258 del archivo ViewEditarListaPersonalizadaViewModel.cs.

BuscarCanciones()

async void BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.BuscarCanciones () [private]

Realiza una búsqueda asíncrona de canciones en la base de datos y filtra aquellas que ya han sido seleccionadas para la lista actual.

Este método gestiona el filtrado dinámico de contenido mediante los siguientes pasos:

1. Consulta remota: Solicita al cliente de MongoDB las canciones que coincidan con el término almacenado en TxtBusqueda.
2. Filtrado local: Aplica una operación LINQ para excluir de los resultados cualquier canción cuyo identificador ya se encuentre en ListaCancionesSeleccionadas.
3. Actualización de UI: Inicializa la propiedad ListaResultados con una nueva colección observable, permitiendo que la interfaz muestre únicamente las opciones elegibles.

Definición en la línea 234 del archivo ViewEditarListaPersonalizadaViewModel.cs.

CargarImagenDesdeUrl()

async Task BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.CargarImagenDesdeUrl (
string url) [private]

Descarga de forma asíncrona una imagen desde una dirección URL y la asigna al mapa de bits de la portada.

Este método gestiona la recuperación de recursos remotos mediante los siguientes pasos:

1. Petición HTTP: Utiliza un HttpClient para obtener el flujo de bytes de la imagen desde la red.
2. Procesamiento de Memoria: Transfiere los bytes a un System.IO.MemoryStream para su decodificación.
3. Asignación Visual: Inicializa la propiedad ImagenPortada con el nuevo Bitmap y actualiza el estado de TieneImagen.

En caso de error en la red o formato inválido, se captura la excepción y se notifica al usuario mediante _dialogoService.

Parámetros

url	La dirección URL completa de la imagen que se desea cargar.

Devuelve

Una tarea que representa la operación de carga asíncrona.

Definición en la línea 171 del archivo ViewEditarListaPersonalizadaViewModel.cs.

CargarImagenLocal()

void BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.CargarImagenLocal (
string ruta) [private]

Intenta cargar y asignar una imagen desde el almacenamiento local del sistema de archivos.

Este método gestiona la carga de recursos gráficos locales mediante los siguientes pasos:

1. Validación de ruta: Verifica la existencia física del archivo mediante System.IO.File.Exists.
2. Instanciación: Si el archivo es válido, crea un nuevo objeto Bitmap y lo asigna a ImagenPortada.
3. Control de estado: Actualiza la propiedad booleana TieneImagen para reflejar el éxito o fallo de la operación en la interfaz.

El bloque try-catch asegura que errores de formato o permisos de lectura no interrumpan la ejecución del programa.

Parámetros

ruta	La ruta absoluta en el disco local donde se encuentra el archivo de imagen.

Definición en la línea 203 del archivo ViewEditarListaPersonalizadaViewModel.cs.

EliminarCancion()

void BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.EliminarCancion (
Canciones cancion) [private]

Remueve una canción específica de la colección de pistas seleccionadas para la lista de reproducción.

Este método gestiona la edición de la lista mediante los siguientes pasos:

1. Identificación: Localiza la instancia del objeto Canciones dentro de la colección ListaCancionesSeleccionadas.
2. Remoción: Elimina el elemento de la lista, lo cual desencadena automáticamente la actualización de la interfaz de usuario al ser una colección observable.

Parámetros

cancion	El objeto de tipo Canciones que se desea retirar de la selección actual.

Definición en la línea 278 del archivo ViewEditarListaPersonalizadaViewModel.cs.

GuardarCambios()

async Task BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.GuardarCambios () [private]

Procesa y persiste de forma asíncrona las modificaciones realizadas en una lista de reproducción existente, incluyendo la gestión de medios y la estructura de pistas.

Este método orquesta la actualización de la playlist mediante el siguiente flujo de trabajo:

1. Sincronización de Imagen: Evalúa si la ruta de la portada es local o remota. En caso de ser local, sube el archivo a la nube mediante _storageService para obtener una URL persistente.
2. Preparación de Metadatos: Extrae y proyecta los identificadores únicos de la colección ListaCancionesSeleccionadas.
3. Persistencia en BD: Invoca al cliente de MongoDB para actualizar el nombre, descripción, lista de IDs y URL de portada en el documento correspondiente.
4. Finalización: Tras el éxito, libera el estado de carga y ejecuta la acción de retorno a la vista anterior.

En caso de error, se notifica al usuario mediante el servicio de diálogos y se registra la excepción para depuración.

Devuelve

Una tarea que representa la operación de guardado asíncrona.

Definición en la línea 297 del archivo ViewEditarListaPersonalizadaViewModel.cs.

3.48.4. Propiedades

BtnAgregarCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.BtnAgregarCancion [get]

Definición en la línea 104 del archivo ViewEditarListaPersonalizadaViewModel.cs.

BtnAtras

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.BtnAtras [get]

Definición en la línea 101 del archivo ViewEditarListaPersonalizadaViewModel.cs.

BtnBuscarCanciones

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.BtnBuscarCanciones [get]

Definición en la línea 103 del archivo ViewEditarListaPersonalizadaViewModel.cs.

BtnEliminarCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.BtnEliminarCancion [get]

Definición en la línea 105 del archivo ViewEditarListaPersonalizadaViewModel.cs.

BtnGuardar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.BtnGuardar [get]

Definición en la línea 102 del archivo ViewEditarListaPersonalizadaViewModel.cs.

EstaCargando

bool BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.EstaCargando [get], [set]

Definición en la línea 94 del archivo ViewEditarListaPersonalizadaViewModel.cs.

ImagenPortada

Bitmap? BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.ImagenPortada [get], [set]

Definición en la línea 65 del archivo ViewEditarListaPersonalizadaViewModel.cs.

ListaCancionesSeleccionadas

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.ListaCancionesSeleccionadas [get], [set]

Definición en la línea 86 del archivo ViewEditarListaPersonalizadaViewModel.cs.

ListaResultados

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.ListaResultados [get], [set]

Definición en la línea 79 del archivo ViewEditarListaPersonalizadaViewModel.cs.

RutaImagen

string BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.RutaImagen [get], [set]

Definición en la línea 45 del archivo ViewEditarListaPersonalizadaViewModel.cs.

TieneImagen

bool BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.TieneImagen [get], [set]

Definición en la línea 58 del archivo ViewEditarListaPersonalizadaViewModel.cs.

TxtBusqueda

string BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.TxtBusqueda [get], [set]

Definición en la línea 72 del archivo ViewEditarListaPersonalizadaViewModel.cs.

TxtDescripcion

string BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.TxtDescripcion [get], [set]

Definición en la línea 38 del archivo ViewEditarListaPersonalizadaViewModel.cs.

TxtNombre

string BetaProyecto.ViewModels.ViewEditarListaPersonalizadaViewModel.TxtNombre [get], [set]

Definición en la línea 30 del archivo ViewEditarListaPersonalizadaViewModel.cs.

3.49. Referencia de la clase BetaProyecto.ViewModels.ViewGestionarBDViewModel

3.49.1. Detalles

Definición en la línea 20 del archivo ViewGestionarBDViewModel.cs.

Métodos

- ViewGestionarBDViewModel ()

Propiedades

- ReactiveCommand< Unit, Unit > BtnRecargar [get]
- ReactiveCommand< Unit, Unit > BtnGuardarCambios [get]
- int IndiceTab [get, set]
- ObservableCollection< Usuarios > ListaUsuarios [get]
- ObservableCollection< string > RolesDisponibles [get]
- Usuarios NuevoUsuario [get, set]
- ReactiveCommand< Unit, Unit > BtnCrearUsuario [get]
- ReactiveCommand< Unit, Unit > BtnEliminarUsuario [get]
- Usuarios SelectedUsuario [get, set]
- ObservableCollection< Canciones > ListaCanciones [get]
- ObservableCollection< string > ListaGenerosCombox [get]
- Canciones NuevaCancion [get, set]
- string TxtBusquedaCrear [get, set]
- bool HayResultadosCrear [get, set]
- ObservableCollection< Usuarios > ListaResultadosCrear [get]
- ObservableCollection< Usuarios > ListaArtistasCrear [get]
- string GeneroSeleccionadoCrear [get, set]
- ObservableCollection< string > ListaGenerosSeleccionadosCrear [get]
- bool EsArchivoLocal [get, set]
- bool EsYoutube [get]
- ReactiveCommand< Unit, Unit > BtnBuscarUsuariosCrear [get]
- ReactiveCommand< Usuarios, Unit > BtnAgregarUsuarioCrear [get]
- ReactiveCommand< Usuarios, Unit > BtnEliminarUsuarioCrear [get]
- ReactiveCommand< Unit, Unit > BtnAgregarGeneroCrear [get]
- ReactiveCommand< string, Unit > BtnEliminarGeneroCrear [get]
- ReactiveCommand< Unit, Unit > BtnCrearCancion [get]
- ReactiveCommand< Unit, Unit > BtnEliminarCancion [get]
- Canciones SelectedCancion [get, set]
- string TxtBusquedaEditar [get, set]
- bool HayResultadosEditar [get, set]
- ObservableCollection< Usuarios > ListaResultadosEditar [get]
- ObservableCollection< Usuarios > ListaArtistasEditar [get]
- string GeneroSeleccionadoEditar [get, set]
- ObservableCollection< string > ListaGenerosSeleccionadosEditar [get]
- string TxtRutaArchivoEditar [get, set]
- string TxtUrlYoutubeEditar [get, set]
- bool EsArchivoLocalEditar [get, set]
- bool EsYoutubeEditar [get]
- ReactiveCommand< Unit, Unit > BtnBuscarUsuariosEditar [get]
- ReactiveCommand< Usuarios, Unit > BtnAgregarUsuarioEditar [get]

- ReactiveCommand< Usuarios, Unit > BtnEliminarUsuarioEditar [get]
- ReactiveCommand< Unit, Unit > BtnAgregarGeneroEditar [get]
- ReactiveCommand< string, Unit > BtnEliminarGeneroEditar [get]
- ObservableCollection< Generos > ListaGeneros [get]
- string NuevoGeneroTxt [get, set]
- ReactiveCommand< Unit, Unit > BtnCrearGenero [get]
- ReactiveCommand< Unit, Unit > BtnEliminarGenero [get]
- Generos SelectedGenero [get, set]
- ObservableCollection< ListaPersonalizada > ListaPlaylists [get]
- ListaPersonalizada NuevaPlaylist [get, set]
- string TxtBusquedaCancionCrear [get, set]
- bool HayResultadosCancionCrear [get, set]
- ObservableCollection< Canciones > ListaResultadosCancionesCrear [get]
- ObservableCollection< Canciones > ListaCancionesPlaylistCrear [get]
- ReactiveCommand< Unit, Unit > BtnBuscarCancionesCrear [get]
- ReactiveCommand< Canciones, Unit > BtnAgregarCancionPlaylistCrear [get]
- ReactiveCommand< Canciones, Unit > BtnEliminarCancionPlaylistCrear [get]
- ReactiveCommand< Unit, Unit > BtnCrearPlaylist [get]
- ReactiveCommand< Unit, Unit > BtnEliminarPlaylist [get]
- ListaPersonalizada SelectedPlaylist [get, set]
- string TxtBusquedaCancionEditar [get, set]
- bool HayResultadosCancionEditar [get, set]
- ObservableCollection< Canciones > ListaResultadosCancionesEditar [get]
- ObservableCollection< Canciones > ListaCancionesPlaylistEditar [get]
- ReactiveCommand< Unit, Unit > BtnBuscarCancionesEditar [get]
- ReactiveCommand< Canciones, Unit > BtnAgregarCancionPlaylistEditar [get]
- ReactiveCommand< Canciones, Unit > BtnEliminarCancionPlaylistEditar [get]
- ObservableCollection< Reportes > ListaReportes [get]
- List< string > EstadosReporte [get]
- ObservableCollection< Reportes > ListaTipoProblema [get]
- List< string > TipoProblema [get]
- Reportes NuevoReporte [get, set]
- ReactiveCommand< Unit, Unit > BtnCrearReporte [get]
- ReactiveCommand< Unit, Unit > BtnEliminarReporte [get]
- Reportes SelectedReporte [get, set]
- bool EstaCargando [get, set]
- bool NoEstaCargando [get]
- string MensajeCarga [get, set]

3.49.2. Constructores

ViewGestionarBDViewModel()

BetaProyecto.ViewModels.ViewGestionarBDViewModel.ViewGestionarBDViewModel ()

Definición en la línea 386 del archivo ViewGestionarBDViewModel.cs.

3.49.3. Funciones

AgregarCancionAPlaylist()

```
void BetaProyecto.ViewModels.ViewGestionarBDViewModel.AgregarCancionAPlaylist (
            Canciones c,
            ObservableCollection< Canciones > destino,
            Action limpiar)  [private]
```

Añade una canción a la colección de destino de la playlist y limpia el estado de búsqueda.

Definición en la línea 676 del archivo ViewGestionarBDViewModel.cs.

AgregarGenero()

```
void BetaProyecto.ViewModels.ViewGestionarBDViewModel.AgregarGenero (
            string genero,
            ObservableCollection< string > listaDestino,
            Action limpiarCombo)  [private]
```

Valida y añade un nuevo género a la lista de selección, evitando duplicados.

Definición en la línea 645 del archivo ViewGestionarBDViewModel.cs.

AgregarGeneroBD()

```
async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.AgregarGeneroBD ()  [private]
```

Inserta un nuevo género musical en la base de datos tras validar su inexistencia previa.

Definición en la línea 887 del archivo ViewGestionarBDViewModel.cs.

AgregarUsuario()

```
void BetaProyecto.ViewModels.ViewGestionarBDViewModel.AgregarUsuario (
            Usuarios usuario,
            ObservableCollection< Usuarios > listaDestino,
            Action limpiarUI)  [private]
```

Añade un usuario a la lista de destino y ejecuta una acción de limpieza en la interfaz.

Definición en la línea 630 del archivo ViewGestionarBDViewModel.cs.

BuscarCanciones()

```
void BetaProyecto.ViewModels.ViewGestionarBDViewModel.BuscarCanciones (
            string texto,
            ObservableCollection< Canciones > resultados,
            ObservableCollection< Canciones > yaSeleccionadas,
            Action< bool > setHayResultados)  [private]
```

Remueve un género de la colección especificada.

Definición en la línea 665 del archivo ViewGestionarBDViewModel.cs.

BuscarUsuarios()

void BetaProyecto.ViewModels.ViewGestionarBDViewModel.BuscarUsuarios (
string texto,
ObservableCollection< Usuarios > resultados,
ObservableCollection< Usuarios > yaSeleccionados,
Action< bool > setHayResultados) [private]

Filtra la lista global de usuarios basándose en un criterio de búsqueda y excluye a los que ya han sido seleccionados.

Parámetros

texto	Cadena de búsqueda.
resultados	Colección donde se volcarán los resultados.
yaSeleccionados	Colección de usuarios a excluir.
setHayResultados	Acción para actualizar el estado de visibilidad de resultados.

Definición en la línea 619 del archivo ViewGestionarBDViewModel.cs.

CargarDatosEditarCancion()

void BetaProyecto.ViewModels.ViewGestionarBDViewModel.CargarDatosEditarCancion () [private]

Prepara el formulario de edición de canciones cargando los metadatos y detectando el tipo de origen del audio (Local vs YouTube).

Definición en la línea 542 del archivo ViewGestionarBDViewModel.cs.

CargarDatosEditarPlaylist()

void BetaProyecto.ViewModels.ViewGestionarBDViewModel.CargarDatosEditarPlaylist () [private]

Mapea los identificadores de canciones de una playlist seleccionada a objetos completos para su edición en la lista de pistas.

Definición en la línea 591 del archivo ViewGestionarBDViewModel.cs.

CargarTodo()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.CargarTodo () [private]

Realiza una carga masiva inicial de usuarios, canciones, playlists, reportes y géneros desde MongoDB.

Definición en la línea 691 del archivo ViewGestionarBDViewModel.cs.

CrearCancionTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.CrearCancionTask () [private]

Gestiona la publicación de una nueva canción, procesando la subida de archivos multimedia y el cálculo de duraciones mediante APIs externas o análisis local.

Definición en la línea 789 del archivo ViewGestionarBDViewModel.cs.

CrearPlaylistTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.CrearPlaylistTask () [private]

Persiste una nueva playlist en MongoDB, gestionando la subida de la imagen de portada y la vinculación de IDs de canciones.

Definición en la línea 924 del archivo ViewGestionarBDViewModel.cs.

CrearReporteTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.CrearReporteTask () [private]

Registra un nuevo reporte de error o infracción en el sistema.

Definición en la línea 973 del archivo ViewGestionarBDViewModel.cs.

CrearUsuarioTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.CrearUsuarioTask () [private]

Orquesta el proceso de creación de un nuevo usuario, incluyendo validación de duplicados, carga de imagen a la nube y encriptación de credenciales.

Definición en la línea 722 del archivo ViewGestionarBDViewModel.cs.

EjecutarConCarga()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.EjecutarConCarga (
Func< Task > tarea,
string mensaje = "Procesando...") [private]

Wrapper para ejecutar tareas asíncronas controlando los estados de carga y visualización de mensajes para el usuario.

Parámetros

tarea	Función asíncrona a ejecutar.
mensaje	Mensaje de carga a mostrar en la interfaz.

Definición en la línea 1622 del archivo ViewGestionarBDViewModel.cs.

EliminarCancionTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarCancionTask () [private]

Ejecuta el proceso de borrado de una canción, eliminando el archivo físico de Cloudinary si aplica y actualizando los contadores de sus autores.

Definición en la línea 1493 del archivo ViewGestionarBDViewModel.cs.

EliminarGenero()

void BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarGenero (
string genero,
ObservableCollection< string > listaObjetivo) [private]

Remueve un género de la colección especificada.

Definición en la línea 656 del archivo ViewGestionarBDViewModel.cs.

EliminarGeneroTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarGeneroTask () [private]

Elimina un género musical del catálogo global de la aplicación.

Definición en la línea 1570 del archivo ViewGestionarBDViewModel.cs.

EliminarPlaylistTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarPlaylistTask () [private]

Remueve una playlist de la base de datos tras confirmación del usuario.

Definición en la línea 1545 del archivo ViewGestionarBDViewModel.cs.

EliminarReporteTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarReporteTask () [private]

Remueve el registro de un reporte del sistema.

Definición en la línea 1595 del archivo ViewGestionarBDViewModel.cs.

EliminarUsuario()

void BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarUsuario (
Usuarios usuario,
ObservableCollection< Usuarios > listaObjetivo) [private]

Remueve un usuario de la colección especificada.

Definición en la línea 637 del archivo ViewGestionarBDViewModel.cs.

EliminarUsuarioTask()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.EliminarUsuarioTask () [private]

Elimina permanentemente un usuario de la base de datos tras confirmar la acción y validar que no sea el usuario en sesión.

Definición en la línea 1461 del archivo ViewGestionarBDViewModel.cs.

GuardarSeleccionado()

async Task BetaProyecto.ViewModels.ViewGestionarBDViewModel.GuardarSeleccionado () [private]

Método central de edición que detecta la pestaña activa del panel (Usuario, Canción, Género, Playlist, Reporte) y sincroniza los cambios con MongoDB.

Definición en la línea 1016 del archivo ViewGestionarBDViewModel.cs.

ObtenerDuracionLocal()

int BetaProyecto.ViewModels.ViewGestionarBDViewModel.ObtenerDuracionLocal (
string rutaArchivo) [private]

Analiza un archivo multimedia local para extraer su duración exacta en segundos utilizando la librería TagLib.

Parámetros

rutaArchivo	Ruta física del archivo en el disco.

Devuelve

Segundos de duración (0 en caso de fallo).

Definición en la línea 1646 del archivo ViewGestionarBDViewModel.cs.

ResetearBorradores()

void BetaProyecto.ViewModels.ViewGestionarBDViewModel.ResetearBorradores () [private]

Reinicializa todos los objetos de borrador y limpia las listas temporales utilizadas en los formularios de creación.

Definición en la línea 514 del archivo ViewGestionarBDViewModel.cs.

3.49.4. Propiedades

BtnAgregarCancionPlaylistCrear

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnAgregarCancion↩
PlaylistCrear [get]

Definición en la línea 276 del archivo ViewGestionarBDViewModel.cs.

BtnAgregarCancionPlaylistEditar

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnAgregarCancion↩
PlaylistEditar [get]

Definición en la línea 317 del archivo ViewGestionarBDViewModel.cs.

BtnAgregarGeneroCrear

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnAgregarGeneroCrear [get]

Definición en la línea 129 del archivo ViewGestionarBDViewModel.cs.

BtnAgregarGeneroEditar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnAgregarGeneroEditar [get]

Definición en la línea 209 del archivo ViewGestionarBDViewModel.cs.

BtnAgregarUsuarioCrear

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnAgregarUsuarioCrear
[get]

Definición en la línea 125 del archivo ViewGestionarBDViewModel.cs.

BtnAgregarUsuarioEditar

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnAgregarUsuarioEditar
[get]

Definición en la línea 205 del archivo ViewGestionarBDViewModel.cs.

BtnBuscarCancionesCrear

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnBuscarCancionesCrear [get]

Definición en la línea 275 del archivo ViewGestionarBDViewModel.cs.

BtnBuscarCancionesEditar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnBuscarCancionesEditar [get]

Definición en la línea 316 del archivo ViewGestionarBDViewModel.cs.

BtnBuscarUsuariosCrear

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnBuscarUsuariosCrear [get]

Definición en la línea 124 del archivo ViewGestionarBDViewModel.cs.

BtnBuscarUsuariosEditar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnBuscarUsuariosEditar [get]

Definición en la línea 204 del archivo ViewGestionarBDViewModel.cs.

BtnCrearCancion

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnCrearCancion [get]

Definición en la línea 133 del archivo ViewGestionarBDViewModel.cs.

BtnCrearGenero

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnCrearGenero [get]

Definición en la línea 227 del archivo ViewGestionarBDViewModel.cs.

BtnCrearPlaylist

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnCrearPlaylist [get]

Definición en la línea 280 del archivo ViewGestionarBDViewModel.cs.

BtnCrearReporte

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnCrearReporte [get]

Definición en la línea 351 del archivo ViewGestionarBDViewModel.cs.

BtnCrearUsuario

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnCrearUsuario [get]

Definición en la línea 56 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarCancion

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarCancion [get]

Definición en la línea 136 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarCancionPlaylistCrear

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarCancion↩PlaylistCrear [get]

Definición en la línea 277 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarCancionPlaylistEditar

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarCancion↩PlaylistEditar [get]

Definición en la línea 318 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarGenero

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarGenero [get]

Definición en la línea 230 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarGeneroCrear

ReactiveCommand<string, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarGeneroCrear [get]

Definición en la línea 130 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarGeneroEditar

ReactiveCommand<string, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarGeneroEditar [get]

Definición en la línea 210 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarPlaylist

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarPlaylist [get]

Definición en la línea 283 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarReporte

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarReporte [get]

Definición en la línea 354 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarUsuario

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarUsuario [get]

Definición en la línea 57 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarUsuarioCrear

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarUsuarioCrear [get]

Definición en la línea 126 del archivo ViewGestionarBDViewModel.cs.

BtnEliminarUsuarioEditar

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnEliminarUsuarioEditar [get]

Definición en la línea 206 del archivo ViewGestionarBDViewModel.cs.

BtnGuardarCambios

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnGuardarCambios [get]

Definición en la línea 29 del archivo ViewGestionarBDViewModel.cs.

BtnRecargar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarBDViewModel.BtnRecargar [get]

Definición en la línea 28 del archivo ViewGestionarBDViewModel.cs.

EsArchivoLocal

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.EsArchivoLocal [get], [set]

Definición en la línea 112 del archivo ViewGestionarBDViewModel.cs.

EsArchivoLocalEditar

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.EsArchivoLocalEditar [get], [set]

Definición en la línea 192 del archivo ViewGestionarBDViewModel.cs.

EstaCargando

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.EstaCargando [get], [set]

Definición en la línea 365 del archivo ViewGestionarBDViewModel.cs.

EstadosReporte

List<string> BetaProyecto.ViewModels.ViewGestionarBDViewModel.EstadosReporte [get]

Valor inicial:

```
= new List<string>
        {
            "Pendiente",
            "Investigando",
            "Finalizado"
        }
```

Definición en la línea 327 del archivo ViewGestionarBDViewModel.cs.

EsYoutube

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.EsYoutube [get]

Definición en la línea 121 del archivo ViewGestionarBDViewModel.cs.

EsYoutubeEditar

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.EsYoutubeEditar [get]

Definición en la línea 201 del archivo ViewGestionarBDViewModel.cs.

GeneroSeleccionadoCrear

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.GeneroSeleccionadoCrear [get], [set]

Definición en la línea 103 del archivo ViewGestionarBDViewModel.cs.

GeneroSeleccionadoEditar

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.GeneroSeleccionadoEditar [get], [set]

Definición en la línea 169 del archivo ViewGestionarBDViewModel.cs.

HayResultadosCancionCrear

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.HayResultadosCancionCrear [get], [set]

Definición en la línea 265 del archivo ViewGestionarBDViewModel.cs.

HayResultadosCancionEditar

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.HayResultadosCancionEditar [get], [set]

Definición en la línea 306 del archivo ViewGestionarBDViewModel.cs.

HayResultadosCrear

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.HayResultadosCrear [get], [set]

Definición en la línea 93 del archivo ViewGestionarBDViewModel.cs.

HayResultadosEditar

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.HayResultadosEditar [get], [set]

Definición en la línea 159 del archivo ViewGestionarBDViewModel.cs.

IndiceTab

int BetaProyecto.ViewModels.ViewGestionarBDViewModel.IndiceTab [get], [set]

Definición en la línea 34 del archivo ViewGestionarBDViewModel.cs.

ListaArtistasCrear

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaArtistasCrear [get]

Definición en la línea 99 del archivo ViewGestionarBDViewModel.cs.

ListaArtistasEditar

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaArtistasEditar [get]

Definición en la línea 165 del archivo ViewGestionarBDViewModel.cs.

ListaCanciones

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaCanciones [get]

Definición en la línea 73 del archivo ViewGestionarBDViewModel.cs.

ListaCancionesPlaylistCrear

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaCancionesPlaylistCrear [get]

Definición en la línea 272 del archivo ViewGestionarBDViewModel.cs.

ListaCancionesPlaylistEditar

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaCancionesPlaylistEditar [get]

Definición en la línea 313 del archivo ViewGestionarBDViewModel.cs.

ListaGeneros

ObservableCollection<Generos> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaGeneros [get]

Definición en la línea 217 del archivo ViewGestionarBDViewModel.cs.

ListaGenerosCombox

ObservableCollection<string> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaGenerosCombox [get]

Definición en la línea 74 del archivo ViewGestionarBDViewModel.cs.

ListaGenerosSeleccionadosCrear

ObservableCollection<string> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaGenerosSeleccionadosCrear [get]

Definición en la línea 108 del archivo ViewGestionarBDViewModel.cs.

ListaGenerosSeleccionadosEditar

ObservableCollection<string> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaGenerosSeleccionadosEditar [get]

Definición en la línea 174 del archivo ViewGestionarBDViewModel.cs.

ListaPlaylists

ObservableCollection<ListaPersonalizada> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaPlaylists [get]

Definición en la línea 245 del archivo ViewGestionarBDViewModel.cs.

ListaReportes

ObservableCollection<Reportes> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaReportes [get]

Definición en la línea 326 del archivo ViewGestionarBDViewModel.cs.

ListaResultadosCancionesCrear

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaResultadosCanciones↩
Crear [get]

Definición en la línea 271 del archivo ViewGestionarBDViewModel.cs.

ListaResultadosCancionesEditar

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaResultadosCanciones↩
Editar [get]

Definición en la línea 312 del archivo ViewGestionarBDViewModel.cs.

ListaResultadosCrear

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaResultadosCrear [get]

Definición en la línea 98 del archivo ViewGestionarBDViewModel.cs.

ListaResultadosEditar

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaResultadosEditar [get]

Definición en la línea 164 del archivo ViewGestionarBDViewModel.cs.

ListaTipoProblema

ObservableCollection<Reportes> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaTipoProblema [get]

Definición en la línea 333 del archivo ViewGestionarBDViewModel.cs.

ListaUsuarios

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewGestionarBDViewModel.ListaUsuarios [get]

Definición en la línea 45 del archivo ViewGestionarBDViewModel.cs.

MensajeCarga

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.MensajeCarga [get], [set]

Definición en la línea 377 del archivo ViewGestionarBDViewModel.cs.

NoEstaCargando

bool BetaProyecto.ViewModels.ViewGestionarBDViewModel.NoEstaCargando [get]

Definición en la línea 374 del archivo ViewGestionarBDViewModel.cs.

NuevaCancion

Canciones BetaProyecto.ViewModels.ViewGestionarBDViewModel.NuevaCancion [get], [set]

Definición en la línea 78 del archivo ViewGestionarBDViewModel.cs.

NuevaPlaylist

ListaPersonalizada BetaProyecto.ViewModels.ViewGestionarBDViewModel.NuevaPlaylist [get], [set]

Definición en la línea 249 del archivo ViewGestionarBDViewModel.cs.

NuevoGeneroTxt

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.NuevoGeneroTxt [get], [set]

Definición en la línea 221 del archivo ViewGestionarBDViewModel.cs.

NuevoReporte

Reportes BetaProyecto.ViewModels.ViewGestionarBDViewModel.NuevoReporte [get], [set]

Definición en la línea 345 del archivo ViewGestionarBDViewModel.cs.

NuevoUsuario

Usuarios BetaProyecto.ViewModels.ViewGestionarBDViewModel.NuevoUsuario [get], [set]

Definición en la línea 50 del archivo ViewGestionarBDViewModel.cs.

RolesDisponibles

ObservableCollection<string> BetaProyecto.ViewModels.ViewGestionarBDViewModel.RolesDisponibles [get]

Definición en la línea 46 del archivo ViewGestionarBDViewModel.cs.

SelectedCancion

Canciones BetaProyecto.ViewModels.ViewGestionarBDViewModel.SelectedCancion [get], [set]

Definición en la línea 140 del archivo ViewGestionarBDViewModel.cs.

SelectedGenero

Generos BetaProyecto.ViewModels.ViewGestionarBDViewModel.SelectedGenero [get], [set]

Definición en la línea 234 del archivo ViewGestionarBDViewModel.cs.

SelectedPlaylist

ListaPersonalizada BetaProyecto.ViewModels.ViewGestionarBDViewModel.SelectedPlaylist [get], [set]

Definición en la línea 287 del archivo ViewGestionarBDViewModel.cs.

SelectedReporte

Reportes BetaProyecto.ViewModels.ViewGestionarBDViewModel.SelectedReporte [get], [set]

Definición en la línea 358 del archivo ViewGestionarBDViewModel.cs.

SelectedUsuario

Usuarios BetaProyecto.ViewModels.ViewGestionarBDViewModel.SelectedUsuario [get], [set]

Definición en la línea 61 del archivo ViewGestionarBDViewModel.cs.

TipoProblema

List<string> BetaProyecto.ViewModels.ViewGestionarBDViewModel.TipoProblema [get]

Valor inicial:

```
= new List<string>
      {
         "Copyright / Derechos de autor",
         "Contenido ofensivo o inapropiado",
         "Audio defectuoso o silencio",
         "Spam / Información falsa",
         "Otro"
      }
```

Definición en la línea 334 del archivo ViewGestionarBDViewModel.cs.

TxtBusquedaCancionCrear

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.TxtBusquedaCancionCrear [get], [set]

Definición en la línea 257 del archivo ViewGestionarBDViewModel.cs.

TxtBusquedaCancionEditar

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.TxtBusquedaCancionEditar [get], [set]

Definición en la línea 299 del archivo ViewGestionarBDViewModel.cs.

TxtBusquedaCrear

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.TxtBusquedaCrear [get], [set]

Definición en la línea 86 del archivo ViewGestionarBDViewModel.cs.

TxtBusquedaEditar

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.TxtBusquedaEditar [get], [set]

Definición en la línea 152 del archivo ViewGestionarBDViewModel.cs.

TxtRutaArchivoEditar

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.TxtRutaArchivoEditar [get], [set]

Definición en la línea 178 del archivo ViewGestionarBDViewModel.cs.

TxtUrlYoutubeEditar

string BetaProyecto.ViewModels.ViewGestionarBDViewModel.TxtUrlYoutubeEditar [get], [set]

Definición en la línea 185 del archivo ViewGestionarBDViewModel.cs.

3.50. Referencia de la clase BetaProyecto.ViewModels.ViewGestionarCuentaViewModel

3.50.1. Detalles

Definición en la línea 13 del archivo ViewGestionarCuentaViewModel.cs.

Métodos

- ViewGestionarCuentaViewModel ()

Propiedades

- Action< ListaPersonalizada >? SolicitudIrAEditarPlaylist [get, set]
- Action< Canciones >? SolicitudIrAEditarCanciones [get, set]
- ObservableCollection< Canciones > MisCanciones [get, set]
- ObservableCollection< ListaPersonalizada > MisPlaylists [get, set]
- ReactiveCommand< Canciones, Unit > BtnEditarCancion [get]
- ReactiveCommand< Canciones, Unit > BtnEliminarCancion [get]
- ReactiveCommand< ListaPersonalizada, Unit > BtnEditarPlaylist [get]
- ReactiveCommand< ListaPersonalizada, Unit > BtnEliminarPlaylist [get]
- ReactiveCommand< Unit, Unit > BtnRefrescar [get]

3.50.2. Constructores

ViewGestionarCuentaViewModel()

BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.ViewGestionarCuentaViewModel ()

Definición en la línea 48 del archivo ViewGestionarCuentaViewModel.cs.

3.50.3. Funciones

CargarContenidoUsuario()

async Task BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.CargarContenidoUsuario () [private]

Recupera y carga de forma asíncrona el catálogo de canciones y listas de reproducción creadas por el usuario actual.

Este método gestiona la carga de contenido personal en dos fases:

1. Consulta paralela: Lanza simultáneamente las peticiones a MongoDB para obtener las canciones por autor y las playlists por creador utilizando el ID de GlobalData.Instance.UserIdGD.
2. Sincronización: Utiliza Task.WhenAll para optimizar el tiempo de respuesta y, una vez recibidos los datos, inicializa las colecciones MisCanciones y MisPlaylists.

Esto asegura que la interfaz de usuario se actualice con todo el contenido propio del usuario de una sola vez.

Devuelve

Una tarea que representa la operación de carga asíncrona.

Definición en la línea 188 del archivo ViewGestionarCuentaViewModel.cs.

EliminarCancion()

async Task BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.EliminarCancion (
Canciones cancion) [private]

Gestiona el proceso integral de eliminación de una canción, incluyendo la limpieza de recursos en la nube y la actualización de la base de datos.

Este método ejecuta un flujo de borrado seguro mediante los siguientes pasos:

1. Confirmación: Solicita permiso al usuario mediante _dialogoService para evitar eliminaciones accidentales.
2. Limpieza de Almacenamiento: Identifica si el archivo reside en Cloudinary y lo elimina físicamente mediante _storageService.
3. Persistencia y Métricas: Remueve el registro en MongoDB y decrementa el contador de canciones publicadas del usuario.
4. Actualización de UI: Remueve la instancia de la colección local MisCanciones para reflejar el cambio instantáneamente.

Parámetros

cancion	El objeto Canciones que se desea eliminar definitivamente del sistema.

Devuelve

Una tarea que representa la operación de eliminación asíncrona.

Definición en la línea 102 del archivo ViewGestionarCuentaViewModel.cs.

EliminarPlaylist()

async Task BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.EliminarPlaylist (
ListaPersonalizada playlist) [private]

Gestiona el proceso de eliminación de una lista de reproducción personalizada de la base de datos y de la interfaz de usuario.

Este método ejecuta un flujo de borrado seguro estructurado en los siguientes pasos:

1. Confirmación: Solicita una validación explícita al usuario a través de _dialogoService para prevenir eliminaciones accidentales.
2. Persistencia: Invoca al cliente de MongoDB para eliminar el registro físico de la lista mediante su identificador único.
3. Actualización de UI: Si la operación en la base de datos es exitosa, remueve la instancia de la colección MisPlaylists para refrescar la vista inmediatamente.

Notifica al usuario el resultado de la operación mediante mensajes de alerta traducidos.

Parámetros

playlist	El objeto ListaPersonalizada que se desea eliminar definitivamente del sistema.

Devuelve

Una tarea que representa la operación de eliminación asíncrona.

Definición en la línea 156 del archivo ViewGestionarCuentaViewModel.cs.

3.50.4. Propiedades

BtnEditarCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.BtnEditarCancion [get]

Definición en la línea 39 del archivo ViewGestionarCuentaViewModel.cs.

BtnEditarPlaylist

ReactiveCommand<ListaPersonalizada, Unit> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.BtnEditar↩ Playlist [get]

Definición en la línea 42 del archivo ViewGestionarCuentaViewModel.cs.

BtnEliminarCancion

ReactiveCommand<Canciones, Unit> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.BtnEliminarCancion [get]

Definición en la línea 40 del archivo ViewGestionarCuentaViewModel.cs.

BtnEliminarPlaylist

ReactiveCommand<ListaPersonalizada, Unit> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.BtnEliminar↩ Playlist [get]

Definición en la línea 43 del archivo ViewGestionarCuentaViewModel.cs.

BtnRefrescar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.BtnRefrescar [get]

Definición en la línea 45 del archivo ViewGestionarCuentaViewModel.cs.

MisCanciones

ObservableCollection<Canciones> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.MisCanciones [get], [set]

Definición en la línea 25 del archivo ViewGestionarCuentaViewModel.cs.

MisPlaylists

ObservableCollection<ListaPersonalizada> BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.MisPlaylists [get], [set]

Definición en la línea 32 del archivo ViewGestionarCuentaViewModel.cs.

SolicitudIrAEditarCanciones

Action<Canciones>? BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.SolicitudIrAEditarCanciones [get], [set]

Definición en la línea 21 del archivo ViewGestionarCuentaViewModel.cs.

SolicitudIrAEditarPlaylist

Action<ListaPersonalizada>? BetaProyecto.ViewModels.ViewGestionarCuentaViewModel.SolicitudIrAEditarPlaylist [get], [set]

Definición en la línea 20 del archivo ViewGestionarCuentaViewModel.cs.

3.51. Referencia de la clase BetaProyecto.ViewModels.ViewGestionarReportesViewModel

3.51.1. Detalles

Definición en la línea 12 del archivo ViewGestionarReportesViewModel.cs.

Métodos

- ViewGestionarReportesViewModel ()

Propiedades

- ObservableCollection< Reportes > ListaPendientes [get]
- ObservableCollection< Reportes > ListaInvestigando [get]
- ObservableCollection< Reportes > ListaFinalizados [get]
- ObservableCollection< string > OpcionesEstado [get]
- Reportes ReporteSeleccionado [get, set]
- Reportes SelectedPendiente [get, set]
- Reportes SelectedInvestigando [get, set]
- Reportes SelectedFinalizado [get, set]
- string EstadoEdit [get, set]
- string ResolucionEdit [get, set]
- ReactiveCommand< Unit, Unit > BtnRefrescar [get]
- ReactiveCommand< Unit, Unit > BtnGuardar [get]

3.51.2. Constructores

ViewGestionarReportesViewModel()

BetaProyecto.ViewModels.ViewGestionarReportesViewModel.ViewGestionarReportesViewModel ()

Definición en la línea 109 del archivo ViewGestionarReportesViewModel.cs.

3.51.3. Funciones

CargarDatos()

async Task BetaProyecto.ViewModels.ViewGestionarReportesViewModel.CargarDatos () [private]

Recupera todos los reportes de la base de datos y los clasifica en colecciones independientes según su estado actual.

Este método realiza una limpieza integral de las listas y selecciones actuales para evitar duplicidad visual. Posteriormente, consulta MongoDB y distribuye cada reporte en las categorías de ”Pendiente”, ”Investigando” o ”Finalizado” basándose en el valor de su propiedad Estado, facilitando la organización por columnas en la interfaz.

Devuelve

Una tarea que representa la operación de carga y clasificación asíncrona.

Definición en la línea 132 del archivo ViewGestionarReportesViewModel.cs.

GuardarCambios()

async Task BetaProyecto.ViewModels.ViewGestionarReportesViewModel.GuardarCambios () [private]

Persiste de forma asíncrona las modificaciones realizadas en el estado y la resolución del reporte seleccionado.

Este método sincroniza los cambios con la base de datos MongoDB mediante los siguientes pasos:

1. Validación: Verifica que exista una instancia válida en ReporteSeleccionado.
2. Sincronización: Envía los nuevos valores de EstadoEdit y ResolucionEdit al servidor.
3. Refresco: Si la operación es exitosa, notifica al usuario y reejecuta CargarDatos para reorganizar los reportes en sus respectivas columnas visuales.

Devuelve

Una tarea que representa la operación de actualización asíncrona.

Definición en la línea 170 del archivo ViewGestionarReportesViewModel.cs.

3.51.4. Propiedades

BtnGuardar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarReportesViewModel.BtnGuardar [get]

Definición en la línea 107 del archivo ViewGestionarReportesViewModel.cs.

BtnRefrescar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewGestionarReportesViewModel.BtnRefrescar [get]

Definición en la línea 106 del archivo ViewGestionarReportesViewModel.cs.

EstadoEdit

string BetaProyecto.ViewModels.ViewGestionarReportesViewModel.EstadoEdit [get], [set]

Definición en la línea 92 del archivo ViewGestionarReportesViewModel.cs.

ListaFinalizados

ObservableCollection<Reportes> BetaProyecto.ViewModels.ViewGestionarReportesViewModel.ListaFinalizados [get]

Definición en la línea 20 del archivo ViewGestionarReportesViewModel.cs.

ListaInvestigando

ObservableCollection<Reportes> BetaProyecto.ViewModels.ViewGestionarReportesViewModel.ListaInvestigando [get]

Definición en la línea 19 del archivo ViewGestionarReportesViewModel.cs.

ListaPendientes

ObservableCollection<Reportes> BetaProyecto.ViewModels.ViewGestionarReportesViewModel.ListaPendientes [get]

Definición en la línea 18 del archivo ViewGestionarReportesViewModel.cs.

OpcionesEstado

ObservableCollection<string> BetaProyecto.ViewModels.ViewGestionarReportesViewModel.OpcionesEstado [get]

Valor inicial:

```
= new()
      { "Pendiente", "Investigando", "Finalizado" }
```

Definición en la línea 21 del archivo ViewGestionarReportesViewModel.cs.

ReporteSeleccionado

Reportes BetaProyecto.ViewModels.ViewGestionarReportesViewModel.ReporteSeleccionado [get], [set]

Definición en la línea 26 del archivo ViewGestionarReportesViewModel.cs.

ResolucionEdit

string BetaProyecto.ViewModels.ViewGestionarReportesViewModel.ResolucionEdit [get], [set]

Definición en la línea 99 del archivo ViewGestionarReportesViewModel.cs.

SelectedFinalizado

Reportes BetaProyecto.ViewModels.ViewGestionarReportesViewModel.SelectedFinalizado [get], [set]

Definición en la línea 76 del archivo ViewGestionarReportesViewModel.cs.

SelectedInvestigando

Reportes BetaProyecto.ViewModels.ViewGestionarReportesViewModel.SelectedInvestigando [get], [set]

Definición en la línea 59 del archivo ViewGestionarReportesViewModel.cs.

SelectedPendiente

Reportes BetaProyecto.ViewModels.ViewGestionarReportesViewModel.SelectedPendiente [get], [set]

Definición en la línea 42 del archivo ViewGestionarReportesViewModel.cs.

3.52. Referencia de la clase BetaProyecto.ViewModels.ViewListaPersonalizadaViewModel

3.52.1. Detalles

Definición en la línea 8 del archivo ViewListaPersonalizadaViewModel.cs.

Métodos

- ViewListaPersonalizadaViewModel (ListaPersonalizada playlist, Action accionVolver)

Propiedades

- ListaPersonalizada Playlist [get]
- ReactiveCommand< Unit, Unit > BtnVolver [get]
- int CantidadCanciones [get]

3.52.2. Constructores

ViewListaPersonalizadaViewModel()

```
BetaProyecto.ViewModels.ViewListaPersonalizadaViewModel.ViewListaPersonalizadaViewModel (
        ListaPersonalizada playlist,
        Action accionVolver)
```

Definición en la línea 18 del archivo ViewListaPersonalizadaViewModel.cs.

3.52.3. Propiedades

BtnVolver

```
ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewListaPersonalizadaViewModel.BtnVolver  [get]
```

Definición en la línea 13 del archivo ViewListaPersonalizadaViewModel.cs.

CantidadCanciones

```
int BetaProyecto.ViewModels.ViewListaPersonalizadaViewModel.CantidadCanciones  [get]
```

Definición en la línea 16 del archivo ViewListaPersonalizadaViewModel.cs.

Playlist

```
ListaPersonalizada BetaProyecto.ViewModels.ViewListaPersonalizadaViewModel.Playlist  [get]
```

Definición en la línea 10 del archivo ViewListaPersonalizadaViewModel.cs.

3.53. Referencia de la clase BetaProyecto.ViewModels.ViewModelBase

3.53.1. Detalles

Definición en la línea 8 del archivo ViewModelBase.cs.

3.54. Referencia de la clase BetaProyecto.ViewModels.ViewPerfilViewModel

3.54.1. Detalles

Definición en la línea 12 del archivo ViewPerfilViewModel.cs.

Métodos

- ViewPerfilViewModel ()

Propiedades

- string NombreUsuario [get, set]
- string ImagenPerfil [get, set]
- ObservableCollection< ListaUsuarios > Secciones [get, set]
- ReactiveCommand< Unit, Unit > BtnRefrescar [get]

3.54.2. Constructores

ViewPerfilViewModel()

BetaProyecto.ViewModels.ViewPerfilViewModel.ViewPerfilViewModel ()

Definición en la línea 30 del archivo ViewPerfilViewModel.cs.

3.54.3. Funciones

CargarDatos()

async Task BetaProyecto.ViewModels.ViewPerfilViewModel.CargarDatos () [private]

Recupera y organiza de forma asíncrona la información del perfil, artistas sugeridos y usuarios seguidos.

Este método gestiona la carga de la red social del usuario mediante los siguientes pasos:

1. Inicialización: Carga la identidad básica (nombre y foto) desde GlobalData.Instance.
2. Carga Paralela: Ejecuta simultáneamente las peticiones a MongoDB para obtener los perfiles seguidos y el catálogo global de usuarios mediante Task.WhenAll.
3. Categorización: Estructura los resultados en secciones diferenciadas ("Descubre Artistas" y "↩ Siguiendo") utilizando claves de traducción para los encabezados.
4. Asignación: Actualiza la propiedad Secciones, lo que dispara la actualización de los controles agrupados en la interfaz.

Cualquier fallo durante la consulta se registra en la consola de depuración para evitar el colapso de la vista.

Devuelve

Una tarea que representa la operación de carga y estructuración asíncrona.

Definición en la línea 54 del archivo ViewPerfilViewModel.cs.

3.54.4. Propiedades

BtnRefrescar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewPerfilViewModel.BtnRefrescar [get]

Definición en la línea 28 del archivo ViewPerfilViewModel.cs.

ImagenPerfil

string BetaProyecto.ViewModels.ViewPerfilViewModel.ImagenPerfil [get], [set]

Definición en la línea 19 del archivo ViewPerfilViewModel.cs.

NombreUsuario

string BetaProyecto.ViewModels.ViewPerfilViewModel.NombreUsuario [get], [set]

Definición en la línea 16 del archivo ViewPerfilViewModel.cs.

Secciones

ObservableCollection<ListaUsuarios> BetaProyecto.ViewModels.ViewPerfilViewModel.Secciones [get], [set]

Definición en la línea 22 del archivo ViewPerfilViewModel.cs.

3.55. Referencia de la clase BetaProyecto.ViewModels.ViewPublicarCancionViewModel

3.55.1. Detalles

Definición en la línea 15 del archivo ViewPublicarCancionViewModel.cs.

Métodos

- ViewPublicarCancionViewModel (Action accionVolver)

Propiedades

- string TxtTitulo [get, set]
- ObservableCollection< string > ListaGeneros [get, set]
- string GeneroSeleccionado [get, set]
- ObservableCollection< string > ListaGenerosSeleccionados [get, set]
- string TxtBusqueda [get, set]
- ObservableCollection< Usuarios > ListaResultados [get, set]
- ObservableCollection< Usuarios > ListaArtistas [get, set]
- string RutaImagen [get, set]
- bool TieneImagen [get]
- Bitmap? ImagenPortada [get, set]
- bool EsYoutube [get, set]
- bool EsArchivo [get]
- string LinkYoutube [get, set]
- string RutaMp3 [get, set]
- bool EstaCargando [get, set]
- ReactiveCommand< Unit, Unit > BtnVolverAtras [get]
- ReactiveCommand< Unit, Unit > BtnPublicar [get]
- ReactiveCommand< Unit, Unit > BtnBuscarUsuarios [get]
- ReactiveCommand< Usuarios, Unit > BtnAgregarUsuario [get]
- ReactiveCommand< Usuarios, Unit > BtnEliminarUsuario [get]
- ReactiveCommand< Unit, Unit > BtnAgregarGenero [get]
- ReactiveCommand< string, Unit > BtnEliminarGenero [get]

3.55.2. Constructores

ViewPublicarCancionViewModel()

BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ViewPublicarCancionViewModel (
Action accionVolver)

Definición en la línea 151 del archivo ViewPublicarCancionViewModel.cs.

3.55.3. Funciones

AgregarGenero()

void BetaProyecto.ViewModels.ViewPublicarCancionViewModel.AgregarGenero () [private]

Añade el género seleccionado actualmente a la lista de géneros asociados, validando que no esté vacío y que no se haya añadido previamente.

El método verifica si GeneroSeleccionado contiene un valor válido. Si el género ya existe en ListaGenerosSeleccionados (comparación insensible a mayúsculas), se muestra una alerta de error a través de _dialogoService. En cualquier caso, tras el intento de adición, se restablece la propiedad GeneroSeleccionado a nulo para limpiar la selección de la interfaz.

Definición en la línea 214 del archivo ViewPublicarCancionViewModel.cs.

AgregarUsuario()

void BetaProyecto.ViewModels.ViewPublicarCancionViewModel.AgregarUsuario (
Usuarios usuario) [private]

Añade un usuario a la lista de artistas seleccionados, evitando duplicados y limpiando los resultados de búsqueda actuales.

El método verifica mediante el ID si el usuario ya se encuentra en ListaArtistas. Tras la validación, independientemente de si se añadió o no, se restablece TxtBusqueda y se vacía ListaResultados para limpiar la interfaz de búsqueda.

Parámetros

usuario	El objeto de tipo Usuarios que se desea vincular o añadir.

Definición en la línea 259 del archivo ViewPublicarCancionViewModel.cs.

BuscarUsuarios()

async void BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BuscarUsuarios () [private]

Realiza una búsqueda asíncrona de usuarios en la base de datos basada en el texto introducido, filtrando aquellos que ya han sido seleccionados.

Este método utiliza MongoClientSingleton para consultar usuarios cuyo nombre coincida con TxtBusqueda. Para evitar duplicados, se envían los IDs de la ListaArtistas actual como lista de exclusión. Si se encuentran resultados, se actualiza ListaResultados; de lo contrario, se limpia. En caso de fallo en la conexión, se muestra una alerta mediante _dialogoService.

Definición en la línea 300 del archivo ViewPublicarCancionViewModel.cs.

CargarGeneros()

async Task BetaProyecto.ViewModels.ViewPublicarCancionViewModel.CargarGeneros () [private]

Carga la lista de géneros disponibles desde la base de datos y los asigna a la propiedad ListaGeneros.

Este método recupera todos los nombres de géneros registrados en MongoDB mediante el cliente singleton. Si la conexión es exitosa, inicializa ListaGeneros; de lo contrario, registra el error en el flujo de depuración del sistema.

Devuelve

Una tarea que representa la operación asíncrona.

Definición en la línea 356 del archivo ViewPublicarCancionViewModel.cs.

CargarImagenLocal()

void BetaProyecto.ViewModels.ViewPublicarCancionViewModel.CargarImagenLocal (
string ruta) [private]

Carga una imagen desde una ruta local y la asigna a la propiedad ImagenPortada.

Intenta crear un objeto Bitmap a partir de la ruta proporcionada. Si el archivo no existe o ocurre un error durante la lectura, se asigna null a ImagenPortada para evitar fallos visuales. Finalmente, notifica el cambio de la propiedad TieneImagen para actualizar la UI.

Parámetros

ruta	La ruta del sistema de archivos donde se encuentra la imagen.

Definición en la línea 328 del archivo ViewPublicarCancionViewModel.cs.

EliminarGenero()

void BetaProyecto.ViewModels.ViewPublicarCancionViewModel.EliminarGenero (
string genero) [private]

Elimina un género específico de la lista de géneros seleccionados para la canción.

Este método verifica si el género proporcionado existe dentro de ListaGenerosSeleccionados. Si se encuentra, lo elimina, lo que actualiza automáticamente cualquier control de la interfaz vinculado a esta colección.

Parámetros

genero	El nombre del género que se desea remover de la selección actual.

Definición en la línea 243 del archivo ViewPublicarCancionViewModel.cs.

EliminarUsuario()

void BetaProyecto.ViewModels.ViewPublicarCancionViewModel.EliminarUsuario (
Usuarios usuario) [private]

Elimina un usuario de la lista de artistas seleccionados, validando que no sea el usuario que ha iniciado sesión.

El método comprueba si el usuario a eliminar coincide con el ID del usuario actual en GlobalData.↩ Instance.UserIdGD. Si coinciden, se muestra una alerta de error mediante _dialogoService para impedir que un usuario se elimine a sí mismo de una lista. Si la validación es correcta, procede a removerlo de ListaArtistas.

Parámetros

usuario	El objeto de tipo Usuarios que se desea remover de la selección.

Definición en la línea 279 del archivo ViewPublicarCancionViewModel.cs.

ObtenerDuracionMp3()

int BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ObtenerDuracionMp3 (
string rutaArchivo) [private]

Obtiene la duración de un archivo MP3 en segundos utilizando la biblioteca TagLib#.

Accede a las propiedades del archivo en disco para extraer su duración total. Si el archivo no existe o se produce una excepción al intentar leer los metadatos de audio, el error se captura y el método devuelve 0 segundos para no interrumpir el flujo.

Parámetros

rutaArchivo	La ruta completa del archivo de audio local.

Devuelve

La duración total en segundos.

Definición en la línea 378 del archivo ViewPublicarCancionViewModel.cs.

PublicarCancion()

async Task BetaProyecto.ViewModels.ViewPublicarCancionViewModel.PublicarCancion () [private]

Realiza el proceso completo de publicación de una canción, integrando subida de archivos, obtención de datos y persistencia.

Este método orquesta un flujo complejo dividido en cuatro fases principales:

1. Subida de imagen: Sube la portada seleccionada a la nube.
2. Gestión de audio: Sube el archivo MP3 o procesa el enlace de YouTube para obtener la duración y URL final.
3. Creación de modelo: Construye el objeto Canciones con autores y géneros seleccionados.
4. Persistencia: Guarda la canción en la BD y actualiza el contador de canciones del usuario.

Durante la ejecución, se controla la propiedad EstaCargando para feedback visual en la UI.

Devuelve

Una tarea que representa la operación de publicación asíncrona.

Definición en la línea 408 del archivo ViewPublicarCancionViewModel.cs.

3.55.4. Propiedades

BtnAgregarGenero

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnAgregarGenero [get]

Definición en la línea 147 del archivo ViewPublicarCancionViewModel.cs.

BtnAgregarUsuario

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnAgregarUsuario [get]

Definición en la línea 145 del archivo ViewPublicarCancionViewModel.cs.

BtnBuscarUsuarios

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnBuscarUsuarios [get]

Definición en la línea 144 del archivo ViewPublicarCancionViewModel.cs.

BtnEliminarGenero

ReactiveCommand<string, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnEliminarGenero [get]

Definición en la línea 148 del archivo ViewPublicarCancionViewModel.cs.

BtnEliminarUsuario

ReactiveCommand<Usuarios, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnEliminarUsuario [get]

Definición en la línea 146 del archivo ViewPublicarCancionViewModel.cs.

BtnPublicar

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnPublicar [get]

Definición en la línea 143 del archivo ViewPublicarCancionViewModel.cs.

BtnVolverAtras

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.BtnVolverAtras [get]

Definición en la línea 142 del archivo ViewPublicarCancionViewModel.cs.

EsArchivo

bool BetaProyecto.ViewModels.ViewPublicarCancionViewModel.EsArchivo [get]

Definición en la línea 109 del archivo ViewPublicarCancionViewModel.cs.

EstaCargando

bool BetaProyecto.ViewModels.ViewPublicarCancionViewModel.EstaCargando [get], [set]

Definición en la línea 135 del archivo ViewPublicarCancionViewModel.cs.

EsYoutube

bool BetaProyecto.ViewModels.ViewPublicarCancionViewModel.EsYoutube [get], [set]

Definición en la línea 100 del archivo ViewPublicarCancionViewModel.cs.

GeneroSeleccionado

string BetaProyecto.ViewModels.ViewPublicarCancionViewModel.GeneroSeleccionado [get], [set]

Definición en la línea 42 del archivo ViewPublicarCancionViewModel.cs.

ImagenPortada

Bitmap? BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ImagenPortada [get], [set]

Definición en la línea 92 del archivo ViewPublicarCancionViewModel.cs.

LinkYoutube

string BetaProyecto.ViewModels.ViewPublicarCancionViewModel.LinkYoutube [get], [set]

Definición en la línea 112 del archivo ViewPublicarCancionViewModel.cs.

ListaArtistas

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ListaArtistas [get], [set]

Definición en la línea 72 del archivo ViewPublicarCancionViewModel.cs.

ListaGeneros

ObservableCollection<string> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ListaGeneros [get], [set]

Definición en la línea 35 del archivo ViewPublicarCancionViewModel.cs.

ListaGenerosSeleccionados

ObservableCollection<string> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ListaGenerosSeleccionados [get], [set]

Definición en la línea 50 del archivo ViewPublicarCancionViewModel.cs.

ListaResultados

ObservableCollection<Usuarios> BetaProyecto.ViewModels.ViewPublicarCancionViewModel.ListaResultados [get], [set]

Definición en la línea 65 del archivo ViewPublicarCancionViewModel.cs.

RutaImagen

string BetaProyecto.ViewModels.ViewPublicarCancionViewModel.RutaImagen [get], [set]

Definición en la línea 80 del archivo ViewPublicarCancionViewModel.cs.

RutaMp3

string BetaProyecto.ViewModels.ViewPublicarCancionViewModel.RutaMp3 [get], [set]

Definición en la línea 119 del archivo ViewPublicarCancionViewModel.cs.

TieneImagen

bool BetaProyecto.ViewModels.ViewPublicarCancionViewModel.TieneImagen [get]

Definición en la línea 89 del archivo ViewPublicarCancionViewModel.cs.

TxtBusqueda

string BetaProyecto.ViewModels.ViewPublicarCancionViewModel.TxtBusqueda [get], [set]

Definición en la línea 58 del archivo ViewPublicarCancionViewModel.cs.

TxtTitulo

string BetaProyecto.ViewModels.ViewPublicarCancionViewModel.TxtTitulo [get], [set]

Definición en la línea 27 del archivo ViewPublicarCancionViewModel.cs.

3.56. Referencia de la clase BetaProyecto.ViewModels.ViewSobreNosotrosViewModel

3.56.1. Detalles

Definición en la línea 9 del archivo ViewSobreNosotrosViewModel.cs.

Métodos

- ViewSobreNosotrosViewModel ()

Propiedades

- Action? VolverAtras [get, set]
- ReactiveCommand< Unit, Unit > btnVolverAtras [get]
- ReactiveCommand< Unit, Unit > BtnAbrirGitHub [get]

3.56.2. Constructores

ViewSobreNosotrosViewModel()

BetaProyecto.ViewModels.ViewSobreNosotrosViewModel.ViewSobreNosotrosViewModel ()

Definición en la línea 14 del archivo ViewSobreNosotrosViewModel.cs.

3.56.3. Propiedades

BtnAbrirGitHub

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewSobreNosotrosViewModel.BtnAbrirGitHub [get]

Definición en la línea 13 del archivo ViewSobreNosotrosViewModel.cs.

btnVolverAtras

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewSobreNosotrosViewModel.btnVolverAtras [get]

Definición en la línea 12 del archivo ViewSobreNosotrosViewModel.cs.

VolverAtras

Action? BetaProyecto.ViewModels.ViewSobreNosotrosViewModel.VolverAtras [get], [set]

Implementa BetaProyecto.ViewModels.INavegable.

Definición en la línea 11 del archivo ViewSobreNosotrosViewModel.cs.

3.57. Referencia de la clase BetaProyecto.ViewModels.ViewUsuariosViewModel

3.57.1. Detalles

Definición en la línea 15 del archivo ViewUsuariosViewModel.cs.

Métodos

- ViewUsuariosViewModel (string idUsuario, Action accionVolver)

Propiedades

- Usuarios Usuario [get, set]
- List< Canciones > CancionesSubidas [get, set]
- List< ListaPersonalizada > PlaylistsCreadas [get, set]
- string TxtMensajeTimer [get, set]
- string TxtVariableTimer [get, set]
- bool EsSeguido [get, set]
- string TextoBotonSeguir [get, set]
- string ColorBotonSeguir [get, set]
- string FechaNacimientoFormateada [get]
- int CantidadCanciones [get]
- ReactiveCommand< Unit, Unit > BtnVolver [get]
- ReactiveCommand< Unit, Unit > BtnSeguir [get]

3.57.2. Constructores

ViewUsuariosViewModel()

```
BetaProyecto.ViewModels.ViewUsuariosViewModel.ViewUsuariosViewModel (
            string idUsuario,
            Action accionVolver)
```

Definición en la línea 105 del archivo ViewUsuariosViewModel.cs.

3.57.3. Funciones

ActualizarBtnSeguir()

```
void BetaProyecto.ViewModels.ViewUsuariosViewModel.ActualizarBtnSeguir ()  [private]
```

Actualiza el indicador de estado de seguimiento según si el usuario cargado está presente en los seguidores globales lista.

Este método establece el valor de la propiedad EsSeguido para reflejar si el actualmente el usuario cargado está siendo seguido. Debe llamarse cada vez que la lista de seguidores o el usuario cargado cambie a asegúrese de que el estado de seguimiento siga siendo preciso.

Definición en la línea 177 del archivo ViewUsuariosViewModel.cs.

AlterarSeguimiento()

async Task BetaProyecto.ViewModels.ViewUsuariosViewModel.AlterarSeguimiento () [private]

Cambia el estado de seguimiento del usuario cargado actualmente para el usuario activo. Si el usuario activo ya está siguiendo al usuario cargado, este método dejará de seguir; de lo contrario, iniciará un seguimiento.

Si el usuario activo intenta seguirse a sí mismo, se muestra una alerta y no hay acción se toma. El método actualiza tanto el estado de seguimiento como la lista local de seguidores al éxito.

Devuelve

Devuelve una tarea que representa la operación asíncrona.

Definición en la línea 136 del archivo ViewUsuariosViewModel.cs.

CargarListasDetalladas()

async Task BetaProyecto.ViewModels.ViewUsuariosViewModel.CargarListasDetalladas (
string idUser) [private]

Carga de forma asíncrona las listas detalladas de canciones y listas de reproducción creadas por el usuario especificado y actualiza el propiedades correspondientes en el hilo de la interfaz.

Este método recupera las canciones y listas de reproducción del usuario desde la fuente de datos y las actualizaciones las propiedades vinculadas a la interfaz de usuario. Las actualizaciones se envían al hilo de la interfaz para garantizar la seguridad del hilo al modificarlo elementos de la interfaz de usuario.

Parámetros

idUser	El identificador único del usuario cuyas canciones cargadas y listas de reproducción creadas se deben cargar. No puede ser nulo.

Devuelve

Devuelve una tarea que representa la operación de carga asíncrona.

Definición en la línea 286 del archivo ViewUsuariosViewModel.cs.

CargarUsuario()

async Task BetaProyecto.ViewModels.ViewUsuariosViewModel.CargarUsuario () [private]

Carga asincrónicamente los datos de usuario del identificador de usuario seleccionado actualmente y actualiza el relacionado propiedades.

Este método recupera la información del usuario de la fuente de datos basada en el usuario actual identificador y actualiza el usuario vinculado y las propiedades calculadas relacionadas en el hilo de la interfaz. Destinado a uso interno dentro del modelo de vista para asegurar la consistencia de la interfaz de usuario después de cambios en los datos del usuario.

Devuelve

Devuelve una tarea que representa la operación de carga asíncrona.

Definición en la línea 257 del archivo ViewUsuariosViewModel.cs.

IniciarHiloActualizacion()

void BetaProyecto.ViewModels.ViewUsuariosViewModel.IniciarHiloActualizacion (
CancellationToken token) [private]

Inicia un ciclo de actualización en segundo plano que actualiza periódicamente el usuario y las listas relacionadas hasta que se cancele solicitado.

El bucle de actualización se ejecuta de forma asíncrona y actualiza los elementos de la interfaz de usuario para reflejar el estado actual. estado de actualización. El método no bloquea el hilo de llamada. Para detener el proceso de actualización, indica cancelación a través del token proporcionado.

Parámetros

token	Un token de cancelación que se puede usar para solicitar la finalización del ciclo de actualización.

Definición en la línea 198 del archivo ViewUsuariosViewModel.cs.

3.57.4. Propiedades

BtnSeguir

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewUsuariosViewModel.BtnSeguir [get]

Definición en la línea 99 del archivo ViewUsuariosViewModel.cs.

BtnVolver

ReactiveCommand<Unit, Unit> BetaProyecto.ViewModels.ViewUsuariosViewModel.BtnVolver [get]

Definición en la línea 98 del archivo ViewUsuariosViewModel.cs.

CancionesSubidas

List<Canciones> BetaProyecto.ViewModels.ViewUsuariosViewModel.CancionesSubidas [get], [set]

Definición en la línea 34 del archivo ViewUsuariosViewModel.cs.

CantidadCanciones

int BetaProyecto.ViewModels.ViewUsuariosViewModel.CantidadCanciones [get]

Definición en la línea 94 del archivo ViewUsuariosViewModel.cs.

ColorBotonSeguir

string BetaProyecto.ViewModels.ViewUsuariosViewModel.ColorBotonSeguir [get], [set]

Definición en la línea 85 del archivo ViewUsuariosViewModel.cs.

EsSeguido

bool BetaProyecto.ViewModels.ViewUsuariosViewModel.EsSeguido [get], [set]

Definición en la línea 65 del archivo ViewUsuariosViewModel.cs.

FechaNacimientoFormateada

string BetaProyecto.ViewModels.ViewUsuariosViewModel.FechaNacimientoFormateada [get]

Definición en la línea 92 del archivo ViewUsuariosViewModel.cs.

PlaylistsCreadas

List<ListaPersonalizada> BetaProyecto.ViewModels.ViewUsuariosViewModel.PlaylistsCreadas [get], [set]

Definición en la línea 41 del archivo ViewUsuariosViewModel.cs.

TextoBotonSeguir

string BetaProyecto.ViewModels.ViewUsuariosViewModel.TextoBotonSeguir [get], [set]

Definición en la línea 78 del archivo ViewUsuariosViewModel.cs.

TxtMensajeTimer

string BetaProyecto.ViewModels.ViewUsuariosViewModel.TxtMensajeTimer [get], [set]

Definición en la línea 50 del archivo ViewUsuariosViewModel.cs.

TxtVariableTimer

string BetaProyecto.ViewModels.ViewUsuariosViewModel.TxtVariableTimer [get], [set]

Definición en la línea 58 del archivo ViewUsuariosViewModel.cs.

Usuario

Usuarios BetaProyecto.ViewModels.ViewUsuariosViewModel.Usuario [get], [set]

Definición en la línea 26 del archivo ViewUsuariosViewModel.cs.

4. Documentación de archivos

4.1. Referencia del archivo BetaProyecto.API/Controllers/MusicController.cs

4.2. MusicController.cs

Ir a la documentación de este archivo.

```
00001 using Microsoft.AspNetCore.Mvc;
00002 using System.Diagnostics;
00003 using System.Text.Json.Nodes;
00004 using System.Text;
00005
00006 namespace BetaProyecto.API.Controllers
00007 {
00008     [ApiController]
00009     [Route("api/[controller]")]
00010     public class MusicController : ControllerBase
00011     {
00012         private readonly string _ytDlpPath;
00013         private readonly string _cookiesPath;
00014
00015         public MusicController()
00016         {
00017             // Definimos rutas
00018             string appDir = AppContext.BaseDirectory;
00019
00020             // Carpeta segura en AppData (donde tenemos permisos de escritura)
00021             string userDir = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
    "MusicSearchApi");
00022             if (!Directory.Exists(userDir)) Directory.CreateDirectory(userDir);
00023
00024             // Rutas finales en la carpeta segura
00025             _ytDlpPath = Path.Combine(userDir, "yt-dlp.exe");
00026             _cookiesPath = Path.Combine(userDir, "cookies.txt");
00027
00028             // Inicializamos el entorno instalando yt-dlp y preparando cookies
00029             InicializarEntorno(appDir, userDir);
00030
00031             // Comprobamos actualizaciones de yt-dlp al iniciar la API
00032             ActualizarYtDlp();
00033         }
00034         /// <summary>
00035         /// Configura el entorno de ejecución local, asegurando la presencia de las dependencias binarias y sanitizando
    archivos de configuración.
00036         /// </summary>
00037         /// <remarks>
00038         /// Este método de preparación realiza las siguientes operaciones críticas:
00039         /// <list type="number">
00040         /// <item><b>Despliegue de Binarios:</b> Verifica la existencia de <c>yt-dlp.exe</c> en la ruta de ejecución.
    Si no está presente o el archivo está dañado (0 bytes), realiza una copia desde el directorio de instalación.</item>
00041         /// <item><b>Sanitización de Cookies:</b> Procesa el archivo <c>cookies.txt</c> para eliminar la marca de
    orden de bytes (BOM). Esto es indispensable ya que <c>yt-dlp</c> requiere una codificación UTF-8 pura para validar
    sesiones de usuario.</item>
00042         /// <item><b>Normalización de Rutas:</b> Centraliza los archivos operativos en carpetas de datos de usuario
    para evitar problemas de permisos de escritura.</item>
00043         /// </list>
00044         /// Cualquier error durante el acceso a archivos o escritura de disco se captura y se registra en la consola para
    facilitar el diagnóstico.
00045         /// </remarks>
00046         /// <param name="appDir">Directorio raíz donde se encuentran los archivos originales de la aplicación.</param>
00047         /// <param name="userDir">Directorio de datos de usuario (AppData) donde se desplegará el entorno de
    trabajo.</param>
00048         private void InicializarEntorno(string appDir, string userDir)
00049         {
00050             try
00051             {
00052                 // Instalar YT-DLP
00053                 // Solo copiamos si no existe o pesa 0 bytes (por seguridad)
00054                 if (!System.IO.File.Exists(_ytDlpPath) || new FileInfo(_ytDlpPath).Length == 0)
00055                 {
00056                     string origenExe = Path.Combine(appDir, "yt-dlp.exe");
00057                     if (System.IO.File.Exists(origenExe))
00058                     {
00059                         Console.WriteLine($"[API] Instalando yt-dlp en: {_ytDlpPath}");
00060                         System.IO.File.Copy(origenExe, _ytDlpPath, true);
00061                     }
00062                 }
00063
00064                 // Limpieza de cookies (SANITIZAR BOM)
00065                 // Leemos el cookies.txt original y lo guardamos SIN la marca invisible(BOM) (\ufeff)
```

```
00066             // Esto es crucial porque yt-dlp no reconoce las cookies si el archivo tiene BOM, lo que causa errores de autenticación.
00067             // El bom es una marca que algunos editores de texto agregan al inicio de los archivos para indicar que están codificados en UTF-8, pero yt-dlp no lo maneja bien.
00068             string origenCookies = Path.Combine(appDir, "cookies.txt");
00069             if (System.IO.File.Exists(origenCookies))
00070             {
00071                 // Leemos el texto (esto se traga el BOM automáticamente)
00072                 string contenido = System.IO.File.ReadAllText(origenCookies);
00073 
00074                 // Lo guardamos forzando UTF8 SIN BOM (new UTF8Encoding(false))
00075                 // Esto crea un archivo "cookies.txt" perfecto para yt-dlp en la carpeta AppData
00076                 System.IO.File.WriteAllText(_cookiesPath, contenido, new UTF8Encoding(false));
00077 
00078                 Console.WriteLine($"[API] Cookies saneadas y copiadas a: {_cookiesPath}");
00079             }
00080             else
00081             {
00082                 Console.WriteLine("[API] No se encontró cookies.txt original.");
00083             }
00084         }
00085         catch (Exception ex)
00086         {
00087             Console.WriteLine($"[API] Error inicializando entorno: {ex.Message}");
00088         }
00089     }
00090     /// <summary>
00091     /// Ejecuta el comando de auto-actualización del binario <c>yt-dlp</c> de forma silenciosa.
00092     /// </summary>
00093     /// <remarks>
00094     /// Dado que las plataformas de video cambian sus algoritmos frecuentemente, este método asegura que la herramienta
00095     /// de extracción esté en su versión más reciente mediante los siguientes pasos:
00096     /// <list type="number">
00097     /// <item><b>Validación:</b> Comprueba la existencia del ejecutable antes de intentar la actualización.</item>
00098     /// <item><b>Ejecución en segundo plano:</b> Inicia un proceso externo con el argumento <c>--update</c> configurado para no mostrar ventanas (<c>CreateNoWindow</c>).</item>
00099     /// <item><b>Redirección:</b> Captura las salidas del proceso para evitar bloqueos y espera su finalización síncrona.</item>
00100     /// <item><b>Resiliencia:</b> El bloque <c>catch</c> ignora silenciosamente fallos (como falta de internet o bloqueos de firewall) para permitir que la aplicación principal siga funcionando incluso si la actualización falla.</item>
00101     /// </list>
00102     /// </remarks>
00103     private void ActualizarYtDlp()
00104     {
00105         try
00106         {
00107             if (!System.IO.File.Exists(_ytDlpPath)) return;
00108             Console.WriteLine("[API] Buscando actualizaciones de yt-dlp...");
00109 
00110             var psi = new ProcessStartInfo
00111             {
00112                 FileName = _ytDlpPath,
00113                 Arguments = "--update",
00114                 RedirectStandardOutput = true,
00115                 RedirectStandardError = true,
00116                 UseShellExecute = false,
00117                 CreateNoWindow = true
00118             };
00119 
00120             using var process = Process.Start(psi);
00121             process.WaitForExit();
00122             Console.WriteLine("[API] Actualización completada.");
00123         }
00124         catch { /* Ignorar errores de red */ }
00125     }
00126 
00127 
00128     /// <summary>
00129     /// Procesa una solicitud HTTP GET para extraer la URL de streaming directo y los metadatos de un video de YouTube.
00130     /// </summary>
00131     /// <remarks>
00132     /// El flujo de ejecución de este endpoint es el siguiente:
00133     /// <list type="number">
00134     /// <item><b>Validación:</b> Comprueba que la URL recibida no sea nula y que el binario <c>yt-dlp</c> esté disponible en el servidor.</item>
00135     /// <item><b>Preparación de Argumentos:</b> Configura <c>yt-dlp</c> con los parámetros <c>--dump-json</c> (para obtener datos en lugar de descargar) y <c>--cookies</c> (usando la versión sanitizada para evitar bloqueos).</item>
00136     /// <item><b>Ejecución de Proceso:</b> Inicia un proceso externo de forma asíncrona, capturando la salida estándar codificada en UTF-8 para evitar errores con caracteres especiales.</item>
00137     /// <item><b>Análisis de Datos:</b> Parsea la salida JSON generada por la herramienta para extraer el enlace directo (<c>url</c>) y la duración exacta en segundos (<c>duration</c>).</item>
00138     /// </list>
00139     /// </remarks>
00140     /// <param name="url">La dirección URL del video de YouTube proporcionada como parámetro de consulta (query
```

```
    string).</param>
00141       /// <returns>
00142       /// Un objeto JSON que contiene la URL de streaming directo y la duración;
00143       /// o un código de error (400 o 500) con el detalle del fallo.
00144       /// </returns>
00145       [HttpGet("stream")]
00146       public async Task<IActionResult> GetStreamUrl([FromQuery] string url)
00147       {
00148           // Validamos que la URL no llegue vacía o con espacios
00149           if (string.IsNullOrWhiteSpace(url)) return BadRequest("Falta la URL");
00150
00151           try
00152           {
00153               // Verificamos físicamente si el ejecutable yt-dlp está en la carpeta del servidor
00154               if (!System.IO.File.Exists(_ytDlpPath)) return StatusCode(500, "FATAL: yt-dlp no instalado.");
00155
00156               // Preparamos los comandos para el ejecutable:
00157               // --dump-json: No descarga, solo devuelve datos técnicos.
00158               // --no-playlist: Ignora listas, solo procesa el video del enlace.
00159               // -f bestaudio: Busca el enlace con la mejor calidad de sonido disponible.
00160               string argumentos = $"--dump-json --no-playlist -f bestaudio \"{url}\"";
00161
00162               // Si tenemos el archivo de cookies (ya saneado), lo añadimos para evitar bloqueos de YouTube
00163               if (System.IO.File.Exists(_cookiesPath))
00164               {
00165                   argumentos += $" --cookies \"{_cookiesPath}\"";
00166               }
00167
00168               // Configuramos cómo se va a lanzar el proceso externo (yt-dlp.exe)
00169               var psi = new ProcessStartInfo
00170               {
00171                   FileName = _ytDlpPath,           // Ruta del .exe
00172                   Arguments = argumentos,          // Los comandos de arriba
00173                   RedirectStandardOutput = true,   // Permite que la API lea lo que el programa escribe
00174                   RedirectStandardError = true,    // Permite leer errores si los hay
00175                   UseShellExecute = false,         // Obligatorio para redirigir flujos de datos
00176                   CreateNoWindow = true,           // No abre la ventana negra de consola
00177                   StandardOutputEncoding = Encoding.UTF8 // Asegura que tildes y caracteres raros se lean bien
00178               };
00179               //Iniciamos el proceso
00180               using var process = new Process { StartInfo = psi };
00181               process.Start();
00182
00183               //Leemos de forma asíncrona toda la respuesta JSON que genera yt-dlp
00184               string jsonOutput = await process.StandardOutput.ReadToEndAsync();
00185
00186               //Esperamos a que el programa termine de cerrarse
00187               await process.WaitForExitAsync();
00188
00189               //Si el programa falló (ExitCode != 0) o no devolvió nada, avisamos del error
00190               if (process.ExitCode != 0 || string.IsNullOrWhiteSpace(jsonOutput))
00191               {
00192                   return StatusCode(500, "Error obteniendo audio. Revisa cookies o bloqueo regional.");
00193               }
00194
00195               // Convertimos el texto JSON en un objeto manipulable en C#
00196               var nodo = JsonNode.Parse(jsonOutput);
00197
00198               // Devolvemos un objeto limpio con solo los dos datos que le importan al cliente:
00199               // La URL directa del flujo de audio y la duración en segundos.
00200               return Ok(new
00201               {
00202                   url = nodo["url"]?.ToString(),
00203                   duracion = nodo["duration"]?.GetValue<int>() ?? 0
00204               });
00205           }
00206           catch (Exception ex)
00207           {
00208               Console.WriteLine($"[API CRITICAL] {ex.Message}");
00209               return StatusCode(500, ex.Message);
00210           }
00211       }
00212   }
00213 }
```

4.3. Referencia del archivo BetaProyecto.API/Controllers/StorageController.cs

4.4. StorageController.cs

Ir a la documentación de este archivo.

```
00001 using CloudinaryDotNet;
00002 using CloudinaryDotNet.Actions;
00003 using Microsoft.AspNetCore.Mvc;
00004 using Newtonsoft.Json.Linq;
00005
00006 namespace BetaProyecto.API.Controllers
00007 {
00008     [Route("api/[controller]")]
00009     [ApiController]
00010     public class StorageController : ControllerBase
00011     {
00012         // CLAVES PARA IMGBB (FOTOS)
00013         private const string ImgbbApiKey = "718db7c8748de9625904739b3e6a4265";
00014
00015         // CLAVES PARA CLOUDINARY (AUDIO)
00016         private const string CloudName = "dyyi9sb9v";
00017         private const string ApiKey = "926712452194393";
00018         private const string ApiSecret = "VCgQV20lLbnA5DO9I9NGMdKvjII";
00019
00020         private readonly Cloudinary _cloudinary;
00021
00022         public StorageController()
00023         {
00024             // Creamos un objeto 'Account' empaquetando nuestras 3 claves.
00025             var account = new Account(CloudName, ApiKey, ApiSecret);
00026             // Creamos la conexión real con Cloudinary usando esa cuenta.
00027             _cloudinary = new Cloudinary(account);
00028             // Obligamos a que la conexión use HTTPS (seguridad, candado verde).
00029             _cloudinary.Api.Secure = true;
00030         }
00031         /// <summary>
00032         /// Recibe un archivo de imagen, lo procesa en memoria y lo carga en el servicio externo ImgBB.
00033         /// </summary>
00034         /// <remarks>
00035         /// Este endpoint actúa como un puente (proxy) entre la aplicación cliente y ImgBB:
00036         /// <list type="number">
00037         /// <item><b>Validación:</b> Asegura que el archivo no sea nulo.</item>
00038         /// <item><b>Conversión:</b> Transforma la imagen binaria a una cadena Base64 (texto), que es el formato
      requerido por la API de ImgBB.</item>
00039         /// <item><b>Transmisión:</b> Realiza una petición POST segura enviando la API Key y el contenido.</item>
00040         /// <item><b>Resolución:</b> Extrae la URL directa de la respuesta JSON para que la App pueda guardarla en
      su base de datos.</item>
00041         /// </list>
00042         /// </remarks>
00043         /// <param name="archivo">El archivo de imagen enviado desde el formulario (IFormFile).</param>
00044         /// <returns>Un objeto JSON con la URL pública de la imagen alojada.</returns>
00045         // ENDPOINT 1: SUBIR IMAGEN -> Va a ImgBB
00046         [HttpPost("subir-imagen")] //ruta --> api/Storage/subir-imagen
00047         public async Task<IActionResult> SubirImagen(IFormFile archivo)
00048         {
00049             // Verificación de seguridad: ¿El archivo existe y tiene datos?
00050             if (archivo == null || archivo.Length == 0) return BadRequest("No hay imagen");
00051
00052             try
00053             {
00054                 // 'using': Crea un cliente HTTP (un navegador web invisible) y lo destruye al terminar para ahorrar RAM.
00055                 using (var client = new HttpClient())
00056                 {
00057                     // ImgBB necesita Base64
00058                     // Variable para guardar la imagen convertida a texto.
00059                     string base64Image;
00060
00061                     // Abrimos un flujo de memoria temporal en la RAM.
00062                     using (var ms = new MemoryStream())
00063                     {
00064                         // Copiamos el archivo que llegó de internet(nuestra aplicación) a la memoria RAM.
00065                         await archivo.CopyToAsync(ms);
00066                         //Convertimos los bytes de la imagen a una cadena de texto Base64.
00067                         base64Image = Convert.ToBase64String(ms.ToArray());
00068                     }
00069                     // Preparamos el "formulario" virtual para enviar.
00070                     var content = new FormUrlEncodedContent(new[]
00071                     {
00072                         // Campo 1: La clave API.
00073                         new KeyValuePair<string, string>("key", ImgbbApiKey),
00074                         // Campo 2: La imagen convertida a texto.
00075                         new KeyValuePair<string, string>("image", base64Image)
00076                     });
00077
00078                     // Enviamos la petición POST a la URL de ImgBB con el formulario.
00079                     var response = await client.PostAsync("https://api.imgbb.com/1/upload", content);
00080                     // Leemos la respuesta del servidor (que viene en texto JSON).
00081                     var jsonString = await response.Content.ReadAsStringAsync();
00082
00083                     // Si ImgBB dice que algo salió mal (código 400 o 500)...
00084                     if (!response.IsSuccessStatusCode)
00085                         return StatusCode(500, "Error ImgBB: " + jsonString);
```

```
00086
00087                   // Analizamos el texto JSON para convertirlo en objeto.
00088                   var json = JObject.Parse(jsonString);
00089
00090                   // Buscamos dentro del JSON: objeto 'data' -> propiedad 'url'.
00091                   string urlFinal = json["data"]["url"].ToString();
00092
00093                   // Devolvemos un código 200 OK con la URL lista para usar.
00094                   return Ok(new { url = urlFinal });
00095               }
00096           }
00097           catch (Exception ex)
00098           {
00099               return StatusCode(500, "Error subiendo imagen: " + ex.Message);
00100           }
00101       }
00102       /// <summary>
00103       /// Procesa un archivo multimedia de audio y lo carga de forma asíncrona en el servicio de almacenamiento de
      Cloudinary.
00104       /// </summary>
00105       /// <remarks>
00106       /// El flujo de este endpoint incluye:
00107       /// <list type="number">
00108       /// <item><b>Validación:</b> Comprobación de integridad del archivo recibido.</item>
00109       /// <item><b>Tratamiento de Stream:</b> Apertura del archivo como flujo de datos para evitar la carga total en
      RAM.</item>
00110       /// <item><b>Categorización:</b> Uso de parámetros de video (requeridos por Cloudinary para archivos de
      audio) y asignación de carpeta destino.</item>
00111       /// <item><b>Persistencia:</b> Obtención de una URL segura (HTTPS) para su almacenamiento en la base de
      datos.</item>
00112       /// </list>
00113       /// </remarks>
00114       /// <param name="archivo">El archivo de audio (mp3, wav, etc.) enviado a través de la petición HTTP.</param>
00115       /// <returns>Un objeto JSON con la URL segura del recurso alojado o un mensaje de error detallado.</returns>
00116       // ENDPOINT 2: SUBIR AUDIO -> Va a Cloudinary
00117       [HttpPost("subir-audio")]
00118       public async Task<IActionResult> SubirAudio(IFormFile archivo)
00119       {
00120           // Verificación: ¿El archivo existe?
00121           if (archivo == null || archivo.Length == 0) return BadRequest("No hay audio");
00122
00123           try
00124           {
00125               // Abrimos el archivo como un flujo de lectura (Stream).
00126               using (var stream = archivo.OpenReadStream())
00127               {
00128                   // CONFIGURACIÓN CLAVE:
00129                   // Usamos 'VideoUploadParams' porque Cloudinary trata el audio como video.
00130                   var uploadParams = new VideoUploadParams()
00131                   {
00132                       // Adjuntamos el archivo (Nombre + Flujo de datos).
00133                       File = new FileDescription(archivo.FileName, stream),
00134                       // Carpeta en Cloudinary
00135                       Folder = "musicsearch_audios"
00136                   };
00137
00138                   // Llamamos a la librería para que suba el archivo.
00139                   // Esto hace todo el proceso de conexión y subida por nosotros.
00140                   var uploadResult = await _cloudinary.UploadAsync(uploadParams);
00141
00142                   // Verificamos si la librería reportó algún error interno.
00143                   if (uploadResult.Error != null)
00144                       return StatusCode(500, "Error Cloudinary: " + uploadResult.Error.Message);
00145
00146                   // Si todo fue bien, devolvemos la 'SecureUrl' (la dirección HTTPS del audio).
00147                   return Ok(new { url = uploadResult.SecureUrl.ToString() });
00148               }
00149           }
00150           catch (Exception ex)
00151           {
00152               return StatusCode(500, "Error subiendo audio: " + ex.Message);
00153           }
00154       }
00155       /// <summary>
00156       /// Solicita la eliminación permanente de un recurso multimedia alojado en Cloudinary a partir de su URL pública.
00157       /// </summary>
00158       /// <remarks>
00159       /// Este endpoint implementa una lógica de filtrado y limpieza:
00160       /// <list type="number">
00161       /// <item><b>Discriminación de dominio:</b> Solo procesa eliminaciones si la URL pertenece a Cloudinary,
      ignorando otros proveedores (como ImgBB) para evitar errores de API.</item>
00162       /// <item><b>Extracción de Identificador:</b> Procesa la URL para obtener el <c>PublicId</c>, que es la
      clave única que Cloudinary necesita para localizar el archivo.</item>
00163       /// <item><b>Borrado por tipo de recurso:</b> Define específicamente el <c>ResourceType.Video</c> (usado
      para audio en tu configuración) para asegurar que el motor de búsqueda de Cloudinary encuentre el objeto.</item>
00164       /// </list>
00165       /// </remarks>
```

```
00166         /// <param name="url">La dirección URL completa del archivo que se desea eliminar.</param>
00167         /// <returns>Un mensaje de confirmación de éxito o un error detallado si la operación falla.</returns>
00168         [HttpDelete("eliminar")]
00169         public async Task<IActionResult> EliminarArchivo([FromQuery] string url)
00170         {
00171             if (string.IsNullOrEmpty(url)) return BadRequest("URL vacía");
00172 
00173             // Solo borramos cosas de Cloudinary
00174             if (!url.Contains("cloudinary.com"))
00175                 return Ok(new { mensaje = "Ignorado (No es Cloudinary)" });
00176 
00177             try
00178             {
00179                 // Sacamos el ID oculto en la URL
00180                 string publicId = ObtenerPublicId(url);
00181                 if (string.IsNullOrEmpty(publicId)) return BadRequest("URL no válida");
00182 
00183                 // Intentamos borrar como si fuera un video
00184                 var paramsVid = new DeletionParams(publicId) { ResourceType = ResourceType.Video };
00185                 var result = await _cloudinary.DestroyAsync(paramsVid);
00186 
00187                 if (result.Result == "ok") return Ok(new { mensaje = "Eliminado" });
00188                 else return StatusCode(500, "Fallo al borrar: " + result.Result);
00189             }
00190             catch (Exception ex)
00191             {
00192                 return StatusCode(500, "Error API: " + ex.Message);
00193             }
00194         }
00195         // Metodos Helpers
00196         /// <summary>
00197         /// Analiza una URL de Cloudinary y extrae el identificador único (Public ID) necesario para operaciones de gestión.
00198         /// </summary>
00199         /// <remarks>
00200         /// El método realiza un "parseo" quirúrgico de la URL siguiendo este algoritmo:
00201         /// <list type="number">
00202         /// <item><b>Localización:</b> Busca el segmento <c>/upload/</c> que separa la configuración del servidor de los datos del archivo.</item>
00203         /// <item><b>Limpieza de Versión:</b> Omite el componente de versión (ej. <c>v17397... </c>) que Cloudinary genera automáticamente.</item>
00204         /// <item><b>Extracción de Carpeta y Nombre:</b> Captura la ruta interna y el nombre del archivo.</item>
00205         /// <item><b>Remoción de Extensión:</b> Elimina el sufijo del formato (ej. <c>.mp3</c>) para obtener el ID limpio que requiere la API de borrado.</item>
00206         /// </list>
00207         /// </remarks>
00208         /// <param name="url">La dirección URL completa del recurso alojado en Cloudinary.</param>
00209         /// <returns>El Public ID del recurso (incluyendo carpetas) o <c>null</c> si el formato de la URL es inválido.</returns>
00210         private string ObtenerPublicId(string url)
00211         {
00212             try
00213             {
00214                 var uri = new Uri(url);
00215                 string path = uri.AbsolutePath;
00216                 // Ejemplo: /dyyi9sb9v/video/upload/v1234/musicsearch_audios/cancion.mp3
00217 
00218                 int indexUpload = path.IndexOf("/upload/");
00219                 if (indexUpload == -1) return null;
00220 
00221                 // Cortamos todo hasta después de /upload/
00222                 string resto = path.Substring(indexUpload + 8);
00223 
00224                 // Saltamos la versión (v12345/...)
00225                 int indexSlash = resto.IndexOf('/');
00226                 string idConExt = resto.Substring(indexSlash + 1);
00227 
00228                 // Quitamos la extensión (.mp3 o .jpg)
00229                 int indexPunto = idConExt.LastIndexOf('.');
00230                 return idConExt.Substring(0, indexPunto);
00231             }
00232             catch { return null; }
00233         }
00234     }
00235 }
```

4.5. Referencia del archivo BetaProyecto.API/obj/Debug/net9.0/BetaProyecto.API.AssemblyInfo.cs

4.6. BetaProyecto.API.AssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     Este código fue generado por una herramienta.
00004 //     Versión de runtime:4.0.30319.42000
00005 //
00006 //     Los cambios en este archivo podrían causar un comportamiento incorrecto y se perderán si
00007 //     se vuelve a generar el código.
00008 // </auto-generated>
00009 //------------------------------------------------------------------------------
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("BetaProyecto.API")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly:
      System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+a97b917dd7c4543f45c421f681ed1e79f37b051e")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("BetaProyecto.API")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("BetaProyecto.API")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generado por la clase WriteCodeFragment de MSBuild.
00023
```

4.7. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/BetaProyecto.API.AssemblyInfo.cs

4.8. BetaProyecto.API.AssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //
00005 //     Changes to this file may cause incorrect behavior and will be lost if
00006 //     the code is regenerated.
00007 // </auto-generated>
00008 //------------------------------------------------------------------------------
00009
00010 using System;
00011 using System.Reflection;
00012
00013 [assembly: System.Reflection.AssemblyCompanyAttribute("BetaProyecto.API")]
00014 [assembly: System.Reflection.AssemblyConfigurationAttribute("Release")]
00015 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00016 [assembly:
      System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+a97b917dd7c4543f45c421f681ed1e79f37b051e")]
00017 [assembly: System.Reflection.AssemblyProductAttribute("BetaProyecto.API")]
00018 [assembly: System.Reflection.AssemblyTitleAttribute("BetaProyecto.API")]
00019 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00020
00021 // Generated by the MSBuild WriteCodeFragment class.
00022
```

4.9. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/win-x64/BetaProyecto.API.AssemblyInfo.cs

4.10. BetaProyecto.API.AssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //
00005 //     Changes to this file may cause incorrect behavior and will be lost if
00006 //     the code is regenerated.
00007 // </auto-generated>
00008 //------------------------------------------------------------------------------
00009
00010 using System;
```

```
00011 using System.Reflection;
00012
00013 [assembly: System.Reflection.AssemblyCompanyAttribute("BetaProyecto.API")]
00014 [assembly: System.Reflection.AssemblyConfigurationAttribute("Release")]
00015 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00016 [assembly:
      System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+a97b917dd7c4543f45c421f681ed1e79f37b051e")]
00017 [assembly: System.Reflection.AssemblyProductAttribute("BetaProyecto.API")]
00018 [assembly: System.Reflection.AssemblyTitleAttribute("BetaProyecto.API")]
00019 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00020
00021 // Generado por la clase WriteCodeFragment de MSBuild.
00022
```

4.11. Referencia del archivo BetaProyecto.API/obj/Debug/net9.0/BetaProyecto.API.GlobalUsings.g.cs

4.12. BetaProyecto.API.GlobalUsings.g.cs

Ir a la documentación de este archivo.

```
00001 // <auto-generated/>
00002 global using Microsoft.AspNetCore.Builder;
00003 global using Microsoft.AspNetCore.Hosting;
00004 global using Microsoft.AspNetCore.Http;
00005 global using Microsoft.AspNetCore.Routing;
00006 global using Microsoft.Extensions.Configuration;
00007 global using Microsoft.Extensions.DependencyInjection;
00008 global using Microsoft.Extensions.Hosting;
00009 global using Microsoft.Extensions.Logging;
00010 global using System;
00011 global using System.Collections.Generic;
00012 global using System.IO;
00013 global using System.Linq;
00014 global using System.Net.Http;
00015 global using System.Net.Http.Json;
00016 global using System.Threading;
00017 global using System.Threading.Tasks;
```

4.13. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/BetaProyecto.API.GlobalUsings.g.cs

4.14. BetaProyecto.API.GlobalUsings.g.cs

Ir a la documentación de este archivo.

```
00001 // <auto-generated/>
00002 global using Microsoft.AspNetCore.Builder;
00003 global using Microsoft.AspNetCore.Hosting;
00004 global using Microsoft.AspNetCore.Http;
00005 global using Microsoft.AspNetCore.Routing;
00006 global using Microsoft.Extensions.Configuration;
00007 global using Microsoft.Extensions.DependencyInjection;
00008 global using Microsoft.Extensions.Hosting;
00009 global using Microsoft.Extensions.Logging;
00010 global using System;
00011 global using System.Collections.Generic;
00012 global using System.IO;
00013 global using System.Linq;
00014 global using System.Net.Http;
00015 global using System.Net.Http.Json;
00016 global using System.Threading;
00017 global using System.Threading.Tasks;
```

4.15. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/win-x64/BetaProyecto.API.GlobalUsings.g.cs

4.16. BetaProyecto.API.GlobalUsings.g.cs

Ir a la documentación de este archivo.

```
00001 // <auto-generated/>
00002 global using Microsoft.AspNetCore.Builder;
00003 global using Microsoft.AspNetCore.Hosting;
00004 global using Microsoft.AspNetCore.Http;
00005 global using Microsoft.AspNetCore.Routing;
00006 global using Microsoft.Extensions.Configuration;
00007 global using Microsoft.Extensions.DependencyInjection;
00008 global using Microsoft.Extensions.Hosting;
00009 global using Microsoft.Extensions.Logging;
00010 global using System;
00011 global using System.Collections.Generic;
00012 global using System.IO;
00013 global using System.Linq;
00014 global using System.Net.Http;
00015 global using System.Net.Http.Json;
00016 global using System.Threading;
00017 global using System.Threading.Tasks;
```

4.17. Referencia del archivo BetaProyecto.API/obj/Debug/net9.0/BetaProyecto.API.MvcApplicationPartsAssemblyInfo.cs

4.18. BetaProyecto.API.MvcApplicationPartsAssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     Este código fue generado por una herramienta.
00004 //     Versión de runtime:4.0.30319.42000
00005 //
00006 //     Los cambios en este archivo podrían causar un comportamiento incorrecto y se perderán si
00007 //     se vuelve a generar el código.
00008 // </auto-generated>
00009 //------------------------------------------------------------------------------
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly:
      Microsoft.AspNetCore.Mvc.ApplicationParts.ApplicationPartAttribute("Swashbuckle.AspNetCore.SwaggerGen")]
00015
00016 // Generado por la clase WriteCodeFragment de MSBuild.
00017
```

4.19. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/win-x64/BetaProyecto.API.MvcApplicationPartsAssemblyInfo.cs

4.20. BetaProyecto.API.MvcApplicationPartsAssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //
00005 //     Changes to this file may cause incorrect behavior and will be lost if
00006 //     the code is regenerated.
00007 // </auto-generated>
00008 //------------------------------------------------------------------------------
00009
00010 using System;
00011 using System.Reflection;
00012
00013 [assembly:
      Microsoft.AspNetCore.Mvc.ApplicationParts.ApplicationPartAttribute("Swashbuckle.AspNetCore.SwaggerGen")]
00014
00015 // Generado por la clase WriteCodeFragment de MSBuild.
00016
```

4.21. Referencia del archivo BetaProyecto/App.axaml.cs

4.22. App.axaml.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia;
00002 using Avalonia.Controls.ApplicationLifetimes;
00003 using Avalonia.Markup.Xaml;
00004 using BetaProyecto.ViewModels;
00005 using BetaProyecto.Views.MarcoApp;
00006 using System;
00007 using System.Diagnostics;
00008 using System.IO;
00009 using System.Threading.Tasks;
00010
00011 namespace BetaProyecto
00012 {
00013     public partial class App : Application
00014     {
00015         private Process? _apiProcess;
00016
00017         public override void Initialize()
00018         {
00019             AvaloniaXamlLoader.Load(this);
00020         }
00021
00022         public override void OnFrameworkInitializationCompleted()
00023         {
00024             if (ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)
00025             {
00026                 // Primero inicializamos la ventana que vamos a usar como si fuera un marco
00027                 desktop.MainWindow = new MarcoApp
00028                 {
00029                     DataContext = new MarcoAppViewModel(), //Le conectamos su ViewModel
00030                 };
00031
00032                 // Configuramos el cierre de la API para que solo se ejecute cuando el usuario cierre la app
00033                 desktop.Exit += (sender, args) => CerrarApi();
00034
00035                 // Arrancamos la API en segundo plano
00036                 Task.Run(() =>
00037                 {
00038                     //Buscar si ya existe el proceso (Puerto 7500 ocupado) para que no crashée al arrancar la API si ya esta
      ocupado
00039                     try
00040                     {
00041                         var zombies = Process.GetProcessesByName("BetaProyecto.API");
00042
00043                         if (zombies.Length > 0) // Si hay procesos previos, los matamos para liberar el puerto
00044                         {
00045                             Debug.WriteLine($"[App] Detectada API previa ({zombies.Length} procesos). Matando para liberar
      puerto 7500...");
00046                             foreach (var proc in zombies)
00047                             {
00048                                 proc.Kill();
00049                                 proc.WaitForExit(); // Esperamos a que Windows libere el puerto
00050                             }
00051                             Debug.WriteLine("[App] Limpieza completada.");
00052                         }
00053                     }
00054                     catch (Exception ex)
00055                     {
00056                         Debug.WriteLine($"[App] Error al intentar matar zombies: {ex.Message}");
00057                     }
00058
00059                     // Iniciamos API una vez que comprobamos si tenemos los puertos limpios
00060                     IniciarApiNormal();
00061                 });
00062             }
00063
00064             base.OnFrameworkInitializationCompleted();
00065         }
00066
00067         private void IniciarApiNormal()
00068         {
00069             try
00070             {
00071                 // Lógica de búsqueda de rutas (Desarrollo vs Producción(App real))
00072                 string baseDir = AppDomain.CurrentDomain.BaseDirectory;
00073                 string apiPath = "";
00074
00075                 // Rutas posibles (En desarrollo suele estar en la carpeta del proyecto API, en producción(App real) puede
      estar dentro de una subcarpeta "API" o directamente en el mismo nivel que el ejecutable)
```

```
00076             string rutaDev = Path.GetFullPath(Path.Combine(baseDir,
      @"..\..\..\..\BetaProyecto.API\bin\Debug\net9.0\BetaProyecto.API.exe"));
00077             string rutaProdSubcarpeta = Path.Combine(baseDir, "API", "BetaProyecto.API.exe");
00078
00079             //Buscamos si existen las rutas para ver si estamos en desarrollo o producción(App real)
00080             if (File.Exists(rutaDev))
00081             {
00082                 apiPath = rutaDev;
00083             }
00084             else if (File.Exists(rutaProdSubcarpeta))
00085             {
00086                 apiPath = rutaProdSubcarpeta;
00087             }
00088
00089             if (!string.IsNullOrEmpty(apiPath)) // Por seguridad, solo intentamos arrancar si encontramos el .exe para
      evitar errores
00090             {
00091                 // Si ya hay un proceso nuestro vivo, no arrancamos otro
00092                 if (_apiProcess != null && !_apiProcess.HasExited) return;
00093
00094                 var startInfo = new ProcessStartInfo
00095                 {
00096                     FileName = apiPath, // La ruta al .exe de la API
00097                     UseShellExecute = false, // Para poder controlar los detalles de la ventana
00098                     CreateNoWindow = false, // Crea la ventana interna para que luego podamos ocultarla
00099                     WindowStyle = ProcessWindowStyle.Hidden, // Ponemos la ventana oculta.
00100                    WorkingDirectory = Path.GetDirectoryName(apiPath) // Para que encuentre pueda encontrar la API los
      archivos appsettings.json y cookies.txt
00101                 };
00102
00103                 _apiProcess = Process.Start(startInfo); // Arrancamos la API
00104                 Debug.WriteLine($"[App]  API arrancada (PID: {_apiProcess?.Id})");
00105             }
00106             else
00107             {
00108                 Debug.WriteLine("[App]  No encuentro la API. ¿Compilada?");
00109             }
00110         }
00111         catch (Exception ex)
00112         {
00113             Debug.WriteLine($"[App]  Error arranque API: {ex.Message}");
00114         }
00115     }
00116
00117     private void CerrarApi()
00118     {
00119         try
00120         {
00121             // Matar nuestro proceso hijo (La API)
00122             if (_apiProcess != null && !_apiProcess.HasExited)
00123             {
00124                 _apiProcess.Kill();
00125                 _apiProcess = null;
00126                 Debug.WriteLine("[App] API cerrada correctamente.");
00127             }
00128
00129             // Barrido de seguridad por si acaso quedó algún zombie suelto de un crash anterior
00130             var zombies = Process.GetProcessesByName("BetaProyecto.API");
00131             foreach (var z in zombies)
00132             {
00133                 try
00134                 {
00135                     z.Kill();
00136                 }
00137                 catch (Exception ex)
00138                 {
00139                     Debug.WriteLine($"[App] Error al matar proceso zombie: {ex.Message}");
00140                 }
00141             }
00142         }
00143         catch (Exception ex)
00144         {
00145             Debug.WriteLine($"[App] Error al cerrar API: {ex.Message}");
00146         }
00147     }
00148 }
00149 }
```

4.23. Referencia del archivo BetaProyecto/Helpers/ControladorDiccionarios.cs

4.24. ControladorDiccionarios.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia;
00002 using Avalonia.Controls;
00003 using Avalonia.Markup.Xaml.Styling;
00004 using System;
00005 using System.Linq;
00006
00007 namespace BetaProyecto.Helpers
00008 {
00009     public static class ControladorDiccionarios
00010     {
00011         // Carga inicial
00012         /// <summary>
00013         /// Establece el entorno visual y regional de la aplicación al iniciar.
00014         /// </summary>
00015         /// <remarks>
00016         /// Este método centraliza la carga de preferencias del usuario, invocando secuencialmente
00017         /// las funciones de localización (idioma), estilo (tema oscuro/claro) y tipografía.
00018         /// </remarks>
00019         /// <param name="tema">Nombre del recurso de estilo a aplicar (ej. "Dark" o "Light").</param>
00020         /// <param name="idioma">Código de cultura o nombre del archivo de traducción.</param>
00021         /// <param name="fuente">Nombre de la familia tipográfica global de la interfaz.</param>
00022         public static void CargarConfiguracionInicial(string tema, string idioma, string fuente)
00023         {
00024             AplicarIdioma(idioma);
00025             AplicarTema(tema);
00026             AplicarFuente(fuente);
00027         }
00028
00029         // Métodos publicos
00030
00031         /// <summary>
00032         /// Cambia dinámicamente el aspecto visual de la aplicación cargando y aplicando un diccionario de recursos de tema
      específico.
00033         /// </summary>
00034         /// <remarks>
00035         /// Este método gestiona el motor de estilos mediante los siguientes pasos:
00036         /// <list type="number">
00037         /// <item><b>Validación:</b> Si el parámetro <paramref name="tema"/> es nulo o vacío, se establece
      "ModoClaro" como valor predeterminado por seguridad.</item>
00038         /// <item><b>Construcción de URI:</b> Genera una ruta de recurso de Avalonia (URI) apuntando a los archivos
      <c>.axaml</c> de la carpeta <c>Assets/Interfaces/</c>.</item>
00039         /// <item><b>Inyección de Estilos:</b> Delega en <see cref="ReemplazarRecurso"/> para sustituir el
      diccionario actual identificado con el alias "Interfaces" por el nuevo tema cargado.</item>
00040         /// </list>
00041         /// </remarks>
00042         /// <param name="tema">El nombre del tema que se desea aplicar (ej. "ModoOscuro", "ModoClaro"). Este debe
      coincidir con el nombre del archivo .axaml en los activos.</param>
00043         public static void AplicarTema(string tema)
00044         {
00045             if (string.IsNullOrEmpty(tema)) tema = "ModoClaro";
00046             string uri = $"avares://BetaProyecto/Assets/Interfaces/{tema}.axaml";
00047             ReemplazarRecurso(uri, "Interfaces");
00048         }
00049         /// <summary>
00050         /// Cambia dinámicamente el idioma de la interfaz de usuario cargando el diccionario de recursos de traducción
      correspondiente.
00051         /// </summary>
00052         /// <remarks>
00053         /// Este método orquestra la localización de la aplicación mediante los siguientes pasos:
00054         /// <list type="number">
00055         /// <item><b>Normalización:</b> Si el parámetro <paramref name="idioma"/> no está definido, se establece
      "Spanish" como idioma base predeterminado.</item>
00056         /// <item><b>Construcción de Ruta:</b> Genera una URI de recurso de Avalonia que apunta a los diccionarios
      de idiomas situados en la carpeta <c>Assets/Language/</c>.</item>
00057         /// <item><b>Actualización de Recursos:</b> Invoca a <see cref="ReemplazarRecurso"/> para sustituir el
      diccionario identificado con la clave "Language", desencadenando la actualización inmediata de todas las etiquetas
      vinculadas mediante el convertidor de localización.</item>
00058         /// </list>
00059         /// </remarks>
00060         /// <param name="idioma">El nombre del archivo de idioma (sin extensión) que se desea aplicar (ej. "Spanish",
      "English").</param>
00061         public static void AplicarIdioma(string idioma)
00062         {
00063             if (string.IsNullOrEmpty(idioma)) idioma = "Spanish";
00064             string uri = $"avares://BetaProyecto/Assets/Language/{idioma}.axaml";
00065             ReemplazarRecurso(uri, "Language");
00066         }
00067         /// <summary>
00068         /// Cambia dinámicamente la familia tipográfica global de la aplicación cargando el diccionario de estilos
      correspondiente.
00069         /// </summary>
00070         /// <remarks>
00071         /// Este método gestiona la identidad visual a través de las fuentes mediante los siguientes pasos:
00072         /// <list type="number">
00073         /// <item><b>Asignación por Defecto:</b> Si el parámetro <paramref name="fuente"/> es nulo o vacío, se
      utiliza "Lexend" como tipografía base del sistema.</item>
00074         /// <item><b>Normalización de Nombre:</b> Verifica si el nombre de la fuente incluye el prefijo "Fuente". Si no
```

```
es así, lo concatena para coincidir con la nomenclatura de los archivos <c>.axaml</c> de estilos.</item>
00075         /// <item><b>Construcción de URI:</b> Genera la ruta de acceso al recurso dentro de la carpeta
<c>Assets/Styles/</c>.</item>
00076         /// <item><b>Aplicación de Estilo:</b> Invoca a <see cref="ReemplazarRecurso"/> para sustituir el diccionario
con el alias "Styles", actualizando la fuente en toda la interfaz de usuario en tiempo de ejecución.</item>
00077         /// </list>
00078         /// </remarks>
00079         /// <param name="fuente">El nombre de la fuente o del archivo de estilo (ej. "Lexend", "Roboto",
"FuenteInter").</param>
00080         public static void AplicarFuente(string fuente)
00081         {
00082             if (string.IsNullOrEmpty(fuente)) fuente = "Lexend";
00083 
00084             // Ajuste de nombre por si viene sin prefijo
00085             string nombreArchivo = fuente.StartsWith("Fuente") ? fuente : "Fuente" + fuente;
00086             string uri = $"avares://BetaProyecto/Assets/Styles/{nombreArchivo}.axaml";
00087 
00088             ReemplazarRecurso(uri, "Styles");
00089         }
00090 
00091         // Motor de reemplazo de recursos
00092         /// <summary>
00093         /// Localiza y sustituye un diccionario de recursos específico dentro de la colección global de la aplicación.
00094         /// </summary>
00095         /// <remarks>
00096         /// Este método es el motor de la personalización dinámica y opera mediante los siguientes pasos:
00097         /// <list type="number">
00098         /// <item><b>Acceso Global:</b> Obtiene la instancia actual de la aplicación y accede a su colección <see
cref="ResourceDictionary.MergedDictionaries"/>.</item>
00099         /// <item><b>Identificación:</b> Escanea los diccionarios cargados buscando un objeto <see
cref="ResourceInclude"/> cuya propiedad <c>Source</c> contenga la cadena definida en <paramref
name="carpetaIdentificadora"/> (ej. "Language", "Interfaces", "Styles").</item>
00100         /// <item><b>Limpieza:</b> Si se encuentra un recurso previo del mismo tipo, se elimina de la colección para
evitar conflictos de claves de recursos.</item>
00101         /// <item><b>Inyección:</b> Instancia un nuevo <see cref="ResourceInclude"/> con la <paramref
name="uriNueva"/> y lo añade a la colección global.</item>
00102         /// </list>
00103         /// Si la URI es inválida o el archivo no existe, el error se captura en el bloque <c>catch</c> para mantener la
estabilidad de la interfaz de usuario.
00104         /// </remarks>
00105         /// <param name="uriNueva">La ruta absoluta del nuevo archivo AXAML que se desea cargar.</param>
00106         /// <param name="carpetaIdentificadora">La palabra clave (nombre de la subcarpeta) que identifica qué tipo de
recurso se está reemplazando.</param>
00107         private static void ReemplazarRecurso(string uriNueva, string carpetaIdentificadora)
00108         {
00109             var app = Application.Current;
00110             if (app == null) return;
00111 
00112             var diccionarios = app.Resources.MergedDictionaries;
00113 
00114             // Buscar y eliminar el viejo (buscando por la carpeta en la ruta)
00115             var recursoViejo = diccionarios.FirstOrDefault(d =>
00116                 d is ResourceInclude include &&
00117                 include.Source != null &&
00118                 include.Source.ToString().Contains(carpetaIdentificadora));
00119 
00120             if (recursoViejo != null)
00121             {
00122                 diccionarios.Remove(recursoViejo);
00123             }
00124 
00125             // Añadimos el nuevo recurso
00126             try
00127             {
00128                 var nuevoRecurso = new ResourceInclude(new Uri("avares://BetaProyecto/"))
00129                 {
00130                     Source = new Uri(uriNueva)
00131                 };
00132                 diccionarios.Add(nuevoRecurso);
00133             }
00134             catch (Exception ex)
00135             {
00136                 System.Diagnostics.Debug.WriteLine($"Error cargando RECURSO {uriNueva}: {ex.Message}");
00137             }
00138         }
00139     }
00140 }
```

4.25. Referencia del archivo BetaProyecto/Helpers/Encriptador.cs

4.26. Encriptador.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Controls;
00002 using System;
00003 using System.Collections.Generic;
00004 using System.IO;
00005 using System.Linq;
00006 using System.Security.Cryptography;
00007 using System.Text;
00008 using System.Threading.Tasks;
00009
00010 namespace BetaProyecto.Helpers
00011 {
00012     public static class Encriptador
00013     {
00014         // --- CLAVES PARA EL CIFRADO SIMÉTRICO(AES) ---
00015         // En criptografía simétrica, la MISMA clave sirve para cerrar y abrir.
00016         // _claveAes: La "llave" principal (16 bytes = 128 bits).
00017         // _ivAes: Vector de Inicialización (añade aleatoriedad al primer bloque para más seguridad).
00018         private static readonly byte[] _claveAes = Encoding.UTF8.GetBytes("1234567890123456");
00019         private static readonly byte[] _ivAes = Encoding.UTF8.GetBytes("mi_vector_inicio");
00020
00021         /// <summary>
00022         /// Aplica un algoritmo de hashing criptográfico SHA-256 a una cadena de texto para proteger información sensible.
00023         /// </summary>
00024         /// <remarks>
00025         /// Este proceso de seguridad transforma la contraseña mediante los siguientes pasos:
00026         /// <list type="number">
00027         /// <item><b>Codificación:</b> Convierte la cadena original en una secuencia de bytes utilizando el estándar
      UTF-8.</item>
00028         /// <item><b>Cifrado:</b> Utiliza una instancia de <see cref="SHA256"/> para calcular un resumen único
      (hash) de 256 bits.</item>
00029         /// <item><b>Representación:</b> Transforma los bytes resultantes en una cadena hexadecimal de longitud fija
      (64 caracteres) mediante un <see cref="StringBuilder"/>.</item>
00030         /// <item><b>Gestión de Memoria:</b> Emplea la sentencia <c>using</c> para garantizar la liberación
      inmediata de los recursos criptográficos en la memoria RAM.</item>
00031         /// </list>
00032         /// Nota: El hashing es una operación unidireccional; no es posible revertir el resultado para obtener la contraseña
      original.
00033         /// </remarks>
00034         /// <param name="password">La contraseña en texto plano que se desea anonimizar.</param>
00035         /// <returns>Una cadena de texto en formato hexadecimal que representa el hash único de la contraseña.</returns>
00036         public static string HashPassword(string password)
00037         {
00038             // Creamos un objeto Sha257
00039             // Usamos 'using' para que se limpie de la memoria RAM automáticamente al terminar
00040             using (SHA256 sha256 = SHA256.Create())
00041             {
00042                 // Convertimos de string a array de byte
00043                 byte[] bytesPassword = Encoding.UTF8.GetBytes(password);
00044
00045                 // Calculamos el hash
00046                 byte[] bytesDelHash = sha256.ComputeHash(bytesPassword);
00047
00048                 // Preparamos el StringBuilder para el foreach
00049                 StringBuilder builder = new StringBuilder();
00050
00051                 // Y convertimos los bytes del hash a hexadecimal
00052                 foreach (byte b in bytesDelHash)
00053                 {
00054                     builder.Append(b.ToString("x2"));
00055                 }
00056                 // Y devolvermos el stringbuilder como string para compararlo
00057                 return builder.ToString();
00058             }
00059         }
00060         // Encriptar
00061         // Recibe los bytes "limpios" de la canción y devuelve una "papilla" ilegible.
00062         /// <summary>
00063         /// Realiza un cifrado simétrico AES sobre una secuencia de bytes para proteger el contenido de archivos multimedia.
00064         /// </summary>
00065         /// <remarks>
00066         /// Este proceso de encriptación transforma los datos originales en un formato ilegible mediante los siguientes pasos
      técnicos:
00067         /// <list type="number">
00068         /// <item><b>Inicialización:</b> Se instancia el algoritmo <see cref="Aes"/> y se configuran las propiedades
      <c>Key</c> (clave secreta) e <c>IV</c> (vector de inicialización).</item>
00069         /// <item><b>Canalización (Streaming):</b> Se utiliza un <see cref="CryptoStream"/> como intermediario
      para procesar los datos a través de un transformador de cifrado.</item>
00070         /// <item><b>Escritura Segura:</b> Los bytes originales se escriben en el flujo de memoria, donde se aplican las
      operaciones matemáticas del estándar AES.</item>
00071         /// <item><b>Finalización:</b> Se ejecuta <c>FlushFinalBlock</c> para procesar los bytes restantes y
      garantizar la integridad del bloque cifrado.</item>
00072         /// </list>
00073         /// El resultado es un array de bytes que solo puede ser recuperado mediante el método de desencriptación
      correspondiente utilizando la misma clave e IV.
00074         /// </remarks>
00075         /// <param name="byteSinEncrip">El array de bytes original (en texto plano o formato multimedia crudo) que se
      desea proteger.</param>
```

```
00076       /// <returns>Un array de bytes cifrados mediante el estándar AES, listos para ser almacenados de forma segura en el almacenamiento persistente.</returns>
00077       public static byte[] EncriptarBytes(byte[] byteSinEncrip)
00078       {
00079           // Creamos el algoritmo AES (Estándar de cifrado simétrico)
00080           using (Aes aes = Aes.Create())
00081           {
00082               aes.Key = _claveAes; // Asignamos nuestra clave secreta
00083               aes.IV = _ivAes;     // Y el vector de inicio
00084
00085               // Preparamos un flujo en memoria para guardar el resultado
00086               using (var memoryStream = new MemoryStream())
00087               {
00088                   // CryptoStream es un "túnel" que cifra todo lo que pasa por él.
00089                   // CryptoStreamMode.Write: Lo usamos para ESCRIBIR datos y que salgan cifrados.
00090                   using (var cryptoStream = new CryptoStream(memoryStream, aes.CreateEncryptor(), CryptoStreamMode.Write))
00091                   {
00092                       // Escribimos los datos de la canción en el túnel
00093                       cryptoStream.Write(byteSinEncrip, 0, byteSinEncrip.Length);
00094
00095                       // "FlushFinalBlock" asegura que el último trocito de datos se escriba bien
00096                       cryptoStream.FlushFinalBlock();
00097
00098                       // Devolvemos el array de bytes cifrados (ilegibles)
00099                       return memoryStream.ToArray();
00100                   }
00101               }
00102           }
00103       }
00104
00105       // Desencriptar
00106       // Lee el archivo cifrado del disco y crea uno temporal limpio que VLC pueda entender.
00107       /// <summary>
00108       /// Lee un archivo cifrado del almacenamiento local, lo descifra utilizando el algoritmo AES y guarda el resultado en una ubicación temporal.
00109       /// </summary>
00110       /// <remarks>
00111       /// Este método es el pilar de la recuperación de datos protegidos y opera bajo los siguientes pasos:
00112       /// <list type="number">
00113       /// <item><b>Carga de Datos:</b> Recupera los bytes cifrados del disco mediante <see cref="File.ReadAllBytesAsync"/>.</item>
00114       /// <item><b>Configuración Criptográfica:</b> Reinstancia el motor <see cref="Aes"/> asegurando el uso de la misma clave y vector de inicialización (IV) empleados durante la encriptación.</item>
00115       /// <item><b>Procesamiento de Flujo:</b> Utiliza un <see cref="CryptoStream"/> en modo lectura (<see cref="CryptoStreamMode.Read"/>) que actúa como un filtro de transformación, convirtiendo los bytes cifrados en datos originales.</item>
00116       /// <item><b>Persistencia Temporal:</b> Vuelca el flujo descifrado en un nuevo archivo físico (normalmente un .mp3 temporal) para que sea accesible por los servicios de reproducción.</item>
00117       /// </list>
00118       /// Al utilizar flujos asíncronos, se garantiza que la interfaz de usuario no se bloquee durante el procesamiento de archivos de gran tamaño.
00119       /// </remarks>
00120       /// <param name="rutaEntrada">La ruta del archivo cifrado (habitualmente con extensión .enc) que se desea procesar.</param>
00121       /// <param name="rutaSalida">La ruta de destino donde se escribirá el archivo resultante ya descifrado.</param>
00122       /// <returns>Una tarea asíncrona que representa el proceso de lectura, descifrado y escritura.</returns>
00123       public static async Task DesencriptarArchivo(string rutaEntrada, string rutaSalida)
00124       {
00125           // Volvemos a crear AES con la MISMA CLAVE (imprescindible en simétrico)
00126           using (Aes aes = Aes.Create())
00127           {
00128               aes.Key = _claveAes;
00129               aes.IV = _ivAes;
00130
00131               // 1. Leemos el archivo cifrado del disco (son bytes "basura" para un humano)
00132               byte[] bytesCifrados = await File.ReadAllBytesAsync(rutaEntrada);
00133
00134               // 2. Preparamos los flujos (Streams) para procesar los datos
00135               using (var memoryStreamEntrada = new MemoryStream(bytesCifrados))
00136               {
00137                   using (var memoryStreamSalida = new MemoryStream())
00138                   {
00139                       // 3. Creamos el túnel de descifrado.
00140                       // CryptoStreamMode.Read: Al LEER del túnel, los datos salen limpios.
00141                       using (var cryptoStream = new CryptoStream(memoryStreamEntrada, aes.CreateDecryptor(), CryptoStreamMode.Read))
00142                       {
00143                           // Copiamos los datos a través del túnel hacia la salida
00144                           await cryptoStream.CopyToAsync(memoryStreamSalida);
00145                       }
00146
00147                       // 4. Guardamos el resultado "limpio" en el archivo de salida (temp)
00148                       await File.WriteAllBytesAsync(rutaSalida, memoryStreamSalida.ToArray());
00149                   }
00150               }
00151           }
```

```
00152         }
00153     }
00154 }
```

4.27. Referencia del archivo BetaProyecto/Helpers/TextoTraducidoConverter.cs

4.28. TextoTraducidoConverter.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia;
00002 using Avalonia.Data;
00003 using Avalonia.Data.Converters;
00004 using System;
00005 using System.Collections.Generic;
00006 using System.Globalization;
00007 using System.Linq;
00008 using System.Text;
00009 using System.Threading.Tasks;
00010
00011 namespace BetaProyecto.Helpers
00012 {
00013     // Este convertidor recibe una CLAVE (string) y devuelve el TEXTO traducido del diccionario
00014     public class TextoTraducidoConverter : IValueConverter
00015     {
00016         /// <summary>
00017         /// Traduce dinámicamente una clave de recurso en su valor correspondiente definido en los diccionarios de la
      aplicación.
00018         /// </summary>
00019         /// <remarks>
00020         /// Este convertidor facilita la internacionalización (i18n) en la capa de vista mediante los siguientes pasos:
00021         /// <list type="number">
00022         /// <item><b>Validación:</b> Verifica si el valor de entrada es una cadena de texto válida (la clave del
      recurso).</item>
00023         /// <item><b>Búsqueda:</b> Consulta el diccionario de recursos activo de <see cref="Application.Current"/>
      intentando localizar la clave.</item>
00024         /// <item><b>Resolución:</b> Si encuentra el recurso (ej. una traducción o una ruta de imagen), lo devuelve; de
      lo contrario, retorna la clave original como valor de respaldo (fallback).</item>
00025         /// </list>
00026         /// Es ideal para enlazar propiedades de texto en XAML que deben reaccionar a cambios de idioma en tiempo de
      ejecución.
00027         /// </remarks>
00028         /// <param name="value">La clave del recurso (string) que se desea localizar.</param>
00029         /// <param name="targetType">El tipo de la propiedad de destino.</param>
00030         /// <param name="parameter">Parámetro opcional (no utilizado).</param>
00031         /// <param name="culture">Información de cultura (no utilizada, se prioriza el diccionario activo).</param>
00032         /// <returns>El objeto localizado encontrado en los recursos o el texto original si no existe coincidencia.</returns>
00033         public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)
00034         {
00035             // Si el valor es nulo o no es string, no hacemos nada
00036             if (value is not string claveRecurso)
00037                 return value;
00038
00039             // Intentamos buscar la clave en los recursos de la App
00040             if (Application.Current != null &&
00041                 Application.Current.TryGetResource(claveRecurso, null, out var recursoEncontrado))
00042             {
00043                 return recursoEncontrado;
00044             }
00045
00046             // Si no se encuentra (o si es un texto normal), devolvemos el texto original
00047             return claveRecurso;
00048         }
00049         //ConvertBack no lo necesitamos porque solo es para mostrar texto, no para editarlo, así que devolvemos UnsetValue
      para indicar que no se puede convertir hacia atrás.
00050         public object? ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)
00051         {
00052             return BindingNotification.UnsetValue;
00053         }
00054     }
00055 }
```

4.29. Referencia del archivo BetaProyecto/Models/Canciones.cs

4.30. Canciones.cs

Ir a la documentación de este archivo.

```
00001 using MongoDB.Bson;
00002 using MongoDB.Bson.Serialization.Attributes;
00003 using System;
00004 using System.Collections.Generic;
00005 using System.Linq;
00006
00007 namespace BetaProyecto.Models
00008 {
00009     public class Canciones
00010     {
00011         [BsonId]
00012         [BsonRepresentation(BsonType.ObjectId)]
00013         public string Id { get; set; }
00014
00015         [BsonElement("titulo")]
00016         public string Titulo { get; set; }
00017
00018         [BsonElement("autores_ids")]
00019         [BsonRepresentation(BsonType.ObjectId)]
00020         public List<string> AutoresIds { get; set; }
00021
00022         // [BsonIgnore] le dice a mongo que ignore la propiedad
00023         [BsonIgnore]
00024         public string NombreArtista { get; set; } = "Artista Desconocido";
00025
00026         //La necesitamos para los botones flyout de la tarjetas
00027         [BsonIgnore]
00028         public List<string> ListaArtistasIndividuales
00029         {
00030             get
00031             {
00032                 if (string.IsNullOrEmpty(NombreArtista)) return new List<string>();
00033
00034                 // Cortamos por la coma y quitamos espacios
00035                 return NombreArtista.Split(',').Select(a => a.Trim()).ToList();
00036             }
00037         }
00038
00039         [BsonElement("imagen_portada_url")]
00040         public string ImagenPortadaUrl { get; set; }
00041
00042         [BsonElement("url_cancion")]
00043         public string UrlCancion { get; set; }
00044
00045         // OBJETOS ANIDADOS
00046         [BsonElement("datos")]
00047         public DatosCancion Datos { get; set; } = new DatosCancion();
00048
00049         [BsonElement("metricas")]
00050         public MetricasCancion Metricas { get; set; } = new MetricasCancion();
00051     }
00052
00053     public class DatosCancion
00054     {
00055         [BsonElement("duracion_segundos")]
00056         public int DuracionSegundos { get; set; }
00057
00058         [BsonElement("generos")]
00059         public List<string> Generos { get; set; } = new List<string>();
00060
00061         [BsonIgnore]
00062         public string GenerosTexto => string.Join(", ", Generos);
00063
00064         [BsonElement("fecha_lanzamiento")]
00065         public DateTime FechaLanzamiento { get; set; }
00066     }
00067
00068     public class MetricasCancion
00069     {
00070         [BsonElement("total_reproducciones")]
00071         public long TotalReproducciones { get; set; } = 0;
00072
00073         [BsonElement("total_megustas")]
00074         public long TotalMegustas { get; set; } = 0;
00075
00076         [BsonElement("puntuacion_tendencia")]
00077         public double PuntuacionTendencia { get; set; } = 0.0;
00078     }
00079
00080 }
```

4.31. Referencia del archivo BetaProyecto/Models/Generos.cs

4.32. Generos.cs

Ir a la documentación de este archivo.

```
00001 using MongoDB.Bson;
00002 using MongoDB.Bson.Serialization.Attributes;
00003 using System;
00004 using System.Collections.Generic;
00005 using System.Linq;
00006 using System.Text;
00007 using System.Threading.Tasks;
00008
00009 namespace BetaProyecto.Models
00010 {
00011     public class Generos
00012     {
00013         [BsonId]
00014         [BsonRepresentation(BsonType.ObjectId)]
00015         public string Id { get; set; }
00016
00017         [BsonElement("nombre")]
00018         public string Nombre { get; set; }
00019     }
00020 }
```

4.33. Referencia del archivo BetaProyecto/Models/ListaPersonalizada.cs

4.34. ListaPersonalizada.cs

Ir a la documentación de este archivo.

```
00001 using MongoDB.Bson;
00002 using MongoDB.Bson.Serialization.Attributes;
00003 using System;
00004 using System.Collections.Generic;
00005 using System.Linq;
00006 using System.Text;
00007 using System.Threading.Tasks;
00008
00009 namespace BetaProyecto.Models
00010 {
00011     public class ListaPersonalizada
00012     {
00013         [BsonId]
00014         [BsonRepresentation(BsonType.ObjectId)]
00015         public string Id { get; set; }
00016
00017         [BsonElement("nombre")]
00018         public string Nombre { get; set; }
00019
00020         [BsonElement("descripcion")]
00021         public string Descripcion { get; set; }
00022
00023         [BsonElement("urlportada")]
00024         public string UrlPortada { get; set; }
00025
00026         // Aquí guardamos solo los IDs de las canciones
00027         [BsonElement("listacanciones")]
00028         [BsonRepresentation(BsonType.ObjectId)]
00029         public List<string> IdsCanciones { get; set; } = new List<string>();
00030
00031         [BsonElement("id_usuario")]
00032         [BsonRepresentation(BsonType.ObjectId)]
00033         public string IdUsuario { get; set; }
00034
00035         // Esta propiedad NO se guarda en BD, la rellenamos nosotros después
00036         [BsonIgnore]
00037         public List<Canciones> CancionesCompletas { get; set; } = new List<Canciones>();
00038     }
00039 }
```

4.35. Referencia del archivo BetaProyecto/Models/ListaUsuarios.cs

4.36. ListaUsuarios.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Collections.ObjectModel;
00004 using System.Linq;
00005 using System.Text;
00006 using System.Threading.Tasks;
00007
00008 namespace BetaProyecto.Models
00009 {
00010     public class ListaUsuarios
00011     {
00012         public string Titulo { get; set; }
00013         public ObservableCollection<Usuarios> Lista { get; set; }
00014
00015         public ListaUsuarios(string titulo, ObservableCollection<Usuarios> lista)
00016         {
00017             Titulo = titulo;
00018             Lista = lista;
00019         }
00020     }
00021 }
```

4.37. Referencia del archivo BetaProyecto/Models/Reportes.cs

4.38. Reportes.cs

Ir a la documentación de este archivo.

```
00001 using MongoDB.Bson;
00002 using MongoDB.Bson.Serialization.Attributes;
00003 using System;
00004 using System.Collections.Generic;
00005 using System.Linq;
00006 using System.Text;
00007 using System.Threading.Tasks;
00008
00009 namespace BetaProyecto.Models
00010 {
00011     public class Reportes
00012     {
00013         [BsonId]
00014         [BsonRepresentation(BsonType.ObjectId)]
00015         public string Id { get; set; }
00016
00017         [BsonElement("tipo_problema")]
00018         public string TipoProblema { get; set; }
00019
00020         [BsonElement("descripcion")]
00021         public string Descripcion { get; set; }
00022
00023         [BsonElement("estado")]
00024         public string Estado { get; set; } = "Pendiente";
00025
00026         [BsonElement("referencias")]
00027         public ReferenciasReporte Referencias { get; set; }
00028
00029         [BsonElement("fecha_creacion")]
00030         public DateTime FechaCreacion { get; set; } = DateTime.UtcNow;
00031
00032         [BsonElement("resolucion")]
00033         public string Resolucion { get; set; } = "";
00034         //Campos calculados en la aplicación, no se almacenan en la BD
00035         [BsonIgnore]
00036         public string NombreReportante { get; set; } = "Usuario Desconocido";
00037
00038         [BsonIgnore]
00039         public string TituloCancionReportada { get; set; } = "Canción Desconocida";
00040
00041         // Propiedad para cambiar el color de la tarjeta según estado
00042         [BsonIgnore]
00043         public string ColorEstado => Estado switch
00044         {
```

```
00045             "Pendiente" => "#FF5252",      // Rojo
00046             "Investigando" => "#FFB300",   // Naranja
00047             "Finalizado" => "#4CAF50",     // Verde
00048             _ => "Gray"
00049         };
00050     }
00051
00052     public class ReferenciasReporte
00053     {
00054         [BsonElement("usuario_reportante_id")]
00055         public string UsuarioReportanteId { get; set; }
00056
00057         [BsonElement("cancion_reportada_id")]
00058         public string CancionReportadaId { get; set; }
00059     }
00060 }
```

4.39. Referencia del archivo BetaProyecto/Models/Roles.cs

4.40. Roles.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Linq;
00004 using System.Text;
00005 using System.Threading.Tasks;
00006
00007 namespace BetaProyecto.Models
00008 {
00009     public class Roles
00010     {
00011         // Definimos los nombres exactos tal cual salen en tu Base de Datos
00012         public const string SuperAdmin = "SuperAdmin";
00013         public const string Admin = "Admin";
00014         public const string Usuario = "Usuario";
00015     }
00016 }
```

4.41. Referencia del archivo BetaProyecto/Models/TarjetasCanciones.cs

4.42. TarjetasCanciones.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Controls;
00002 using System;
00003 using System.Collections.Generic;
00004 using System.Collections.ObjectModel;
00005 using System.Linq;
00006 using System.Text;
00007 using System.Threading.Tasks;
00008
00009 namespace BetaProyecto.Models
00010 {
00011     public class TarjetasCanciones
00012     {
00013         public string TituloSeccion { get; set; }
00014         public ObservableCollection<Canciones> ListaCanciones { get; set; }
00015
00016         public TarjetasCanciones(string titulo,ObservableCollection<Canciones> canciones)
00017         {
00018             TituloSeccion = titulo;
00019             ListaCanciones = canciones;
00020         }
00021     }
00022 }
```

4.43. Referencia del archivo BetaProyecto/Models/TarjetasListas.cs

4.44. TarjetasListas.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using System.Collections.Generic;
00003 using System.Collections.ObjectModel;
00004 using System.Linq;
00005 using System.Text;
00006 using System.Threading.Tasks;
00007
00008 namespace BetaProyecto.Models
00009 {
00010     public class TarjetasListas
00011     {
00012         public string TituloSeccion { get; set; }
00013         public ObservableCollection<ListaPersonalizada> Listas { get; set; }
00014
00015         public TarjetasListas(string titulo, ObservableCollection<ListaPersonalizada> listas)
00016         {
00017             TituloSeccion = titulo;
00018             Listas = listas;
00019         }
00020     }
00021 }
```

4.45. Referencia del archivo BetaProyecto/Models/Usuarios.cs

4.46. Usuarios.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using System.Collections.Generic;
00003 using MongoDB.Bson;
00004 using MongoDB.Bson.Serialization.Attributes;
00005
00006 namespace BetaProyecto.Models
00007 {
00008     public class Usuarios
00009     {
00010         // 1. EL ID ES OBLIGATORIO (Mapea el _id de Mongo a string)
00011         [BsonId]
00012         [BsonRepresentation(BsonType.ObjectId)]
00013         public string Id { get; set; }
00014
00015         [BsonElement("username")]
00016         public string Username { get; set; }
00017
00018         [BsonElement("email")]
00019         public string Email { get; set; }
00020
00021         [BsonElement("password")]
00022         public string Password { get; set; }
00023
00024         [BsonElement("rol")]
00025         public string Rol { get; set; }
00026
00027         // 2. OBJETOS ANIDADOS (Aquí está el truco)
00028         // En lugar de poner las propiedades sueltas, creamos propiedades de tipo clase
00029
00030         [BsonElement("perfil")]
00031         public PerfilUsuario Perfil { get; set; } = new PerfilUsuario();
00032
00033         [BsonElement("estadisticas")]
00034         public EstadisticasUsuario Estadisticas { get; set; } = new EstadisticasUsuario();
00035
00036         [BsonElement("listas")]
00037         public ListasUsuario Listas { get; set; } = new ListasUsuario();
00038
00039         [BsonElement("configuracion")]
00040         public ConfiguracionUser Configuracion { get; set; } = new ConfiguracionUser();
00041
00042         [BsonElement("fecha_registro")]
00043         public DateTime FechaRegistro { get; set; }
00044     }
```

```
00045
00046     // SUBCLASE 1: PERFIL
00047     public class PerfilUsuario
00048     {
00049         [BsonElement("imagen_url")]
00050         public string ImagenUrl { get; set; }
00051
00052         [BsonElement("fecha_nacimiento")]
00053         public DateTime FechaNacimiento { get; set; } // DateTime se lleva mejor con Mongo que DateOnly
00054
00055         [BsonElement("es_privada")]
00056         public bool EsPrivada { get; set; }
00057         [BsonElement("pais")]
00058         public string Pais { get; set; }
00059     }
00060
00061     // SUBCLASE 2: ESTADISTICAS
00062     public class EstadisticasUsuario
00063     {
00064         [BsonElement("n_canciones_subidas")]
00065         public int NumCancionesSubidas { get; set; }
00066     }
00067
00068     // SUBCLASE 3: LISTAS
00069     public class ListasUsuario
00070     {
00071         // Convertimos los ObjectId de la lista a strings automáticamente
00072         [BsonElement("seguidores")]
00073         [BsonRepresentation(BsonType.ObjectId)]
00074         public List<string> Seguidores { get; set; } = new List<string>();
00075
00076         [BsonElement("favoritos")]
00077         [BsonRepresentation(BsonType.ObjectId)]
00078         public List<string> Favoritos { get; set; } = new List<string>();
00079     }
00080     // SUBCLASE 4: CONFIGURACION
00081     public class ConfiguracionUser
00082     {
00083         [BsonElement("tema")]
00084         public string DiccionarioTema { get; set; } = "ModoClaro";
00085
00086         [BsonElement("idioma")]
00087         public string DiccionarioIdioma { get; set; } = "Español";
00088
00089         [BsonElement("fuente")]
00090         public string DiccionarioFuente { get; set; } = "Lexend";
00091     }
00092 }
```

4.47. Referencia del archivo BetaProyecto.API/obj/Debug/net9.0/.NETCoreApp,Version=v9.0.AssemblyAttributes.cs

4.48. .NETCoreApp,Version=v9.0.AssemblyAttributes.cs

Ir a la documentación de este archivo.

```
00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0",
      FrameworkDisplayName = ".NET 9.0")]
```

4.49. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/.NETCoreApp,Version=v9.0.AssemblyAttributes.cs

4.50. .NETCoreApp,Version=v9.0.AssemblyAttributes.cs

Ir a la documentación de este archivo.

```
00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0",
      FrameworkDisplayName = ".NET 9.0")]
```

4.51. Referencia del archivo BetaProyecto.API/obj/Release/net9.0/win-x64/.NETCoreApp,Version=v9.0.AssemblyAttributes.cs

4.52. .NETCoreApp,Version=v9.0.AssemblyAttributes.cs

Ir a la documentación de este archivo.

```
00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0",
      FrameworkDisplayName = ".NET 9.0")]
```

4.53. Referencia del archivo BetaProyecto/obj/Debug/net9.0/.NETCoreApp,Version=v9.0.AssemblyAttributes.cs

4.54. .NETCoreApp,Version=v9.0.AssemblyAttributes.cs

Ir a la documentación de este archivo.

```
00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0",
      FrameworkDisplayName = ".NET 9.0")]
```

4.55. Referencia del archivo BetaProyecto/obj/Release/net9.0/.NETCoreApp,Version=v9.0.AssemblyAttributes.cs

4.56. .NETCoreApp,Version=v9.0.AssemblyAttributes.cs

Ir a la documentación de este archivo.

```
00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0",
      FrameworkDisplayName = ".NET 9.0")]
```

4.57. Referencia del archivo BetaProyecto/obj/Release/net9.0/win-x64/.NETCoreApp,Version=v9.0.AssemblyAttributes.cs

4.58. .NETCoreApp,Version=v9.0.AssemblyAttributes.cs

Ir a la documentación de este archivo.

```
00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v9.0",
      FrameworkDisplayName = ".NET 9.0")]
```

4.59. Referencia del archivo BetaProyecto/obj/Debug/net9.0/BetaProyecto.AssemblyInfo.cs

4.60. BetaProyecto.AssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     Este código fue generado por una herramienta.
00004 //     Versión de runtime:4.0.30319.42000
00005 //
00006 //     Los cambios en este archivo podrían causar un comportamiento incorrecto y se perderán si
00007 //     se vuelve a generar el código.
00008 // </auto-generated>
00009 //------------------------------------------------------------------------------
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("BetaProyecto")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly:
      System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+5bb222bb92830db378fc5efbc481f79bb7ef988c")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("BetaProyecto")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("BetaProyecto")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generado por la clase WriteCodeFragment de MSBuild.
00023
```

4.61. Referencia del archivo BetaProyecto/obj/Release/net9.0/BetaProyecto.AssemblyInfo.cs

4.62. BetaProyecto.AssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //
00005 //     Changes to this file may cause incorrect behavior and will be lost if
00006 //     the code is regenerated.
00007 // </auto-generated>
00008 //------------------------------------------------------------------------------
00009
00010 using System;
00011 using System.Reflection;
00012
00013 [assembly: System.Reflection.AssemblyCompanyAttribute("BetaProyecto")]
00014 [assembly: System.Reflection.AssemblyConfigurationAttribute("Release")]
00015 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00016 [assembly:
      System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+a97b917dd7c4543f45c421f681ed1e79f37b051e")]
00017 [assembly: System.Reflection.AssemblyProductAttribute("BetaProyecto")]
00018 [assembly: System.Reflection.AssemblyTitleAttribute("BetaProyecto")]
00019 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00020
00021 // Generated by the MSBuild WriteCodeFragment class.
00022
```

4.63. Referencia del archivo BetaProyecto/obj/Release/net9.0/win-x64/BetaProyecto.AssemblyInfo.cs

4.64. BetaProyecto.AssemblyInfo.cs

Ir a la documentación de este archivo.

```
00001 //------------------------------------------------------------------------------
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //
00005 //     Changes to this file may cause incorrect behavior and will be lost if
00006 //     the code is regenerated.
00007 // </auto-generated>
00008 //------------------------------------------------------------------------------
00009
00010 using System;
00011 using System.Reflection;
00012
00013 [assembly: System.Reflection.AssemblyCompanyAttribute("BetaProyecto")]
00014 [assembly: System.Reflection.AssemblyConfigurationAttribute("Release")]
00015 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00016 [assembly:
      System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+a97b917dd7c4543f45c421f681ed1e79f37b051e")]
00017 [assembly: System.Reflection.AssemblyProductAttribute("BetaProyecto")]
00018 [assembly: System.Reflection.AssemblyTitleAttribute("BetaProyecto")]
00019 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00020
00021 // Generado por la clase WriteCodeFragment de MSBuild.
00022
```

4.65. Referencia del archivo BetaProyecto.API/Program.cs

4.66. Program.cs

Ir a la documentación de este archivo.

```
00001 var builder = WebApplication.CreateBuilder(args);
00002
00003 // Add services to the container.
00004
00005 builder.Services.AddControllers();
00006 // Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi
00007 //builder.Services.AddOpenApi();
00008
00009 builder.Services.AddEndpointsApiExplorer();
00010
00011 builder.Services.AddSwaggerGen();
00012 // Configurar los puertos para HTTPS y HTTP
00013 builder.WebHost.UseUrls("https://localhost:7500", "http://localhost:5151");
00014
00015 var app = builder.Build();
00016
00017 app.UseSwagger();
00018
00019 app.UseSwaggerUI();
00020
00021 // Configure the HTTP request pipeline.
00022 if (app.Environment.IsDevelopment())
00023 {
00024     //app.MapOpenApi();
00025 }
00026
00027 app.UseHttpsRedirection();
00028
00029 app.UseAuthorization();
00030
00031 app.MapControllers();
00032
00033 app.Run();
```

4.67. Referencia del archivo BetaProyecto/Program.cs

4.68. Program.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia;
00002 using Avalonia.ReactiveUI;
00003 using LibVLCSharp.Shared;
00004 using System;
00005 using System.Diagnostics;
```

```
00006 using System.IO;
00007
00008 namespace BetaProyecto
00009 {
00010     internal sealed class Program
00011     {
00012         // Initialization code. Don't use any Avalonia, third-party APIs or any
00013         // SynchronizationContext-reliant code before AppMain is called: things aren't initialized
00014         // yet and stuff might break.
00015         [STAThread]
00016         public static void Main(string[] args)
00017         {
00018             // Primero , inicializamos LibVLCSharp
00019             try
00020             {
00021                 Core.Initialize();
00022             }
00023             catch (Exception ex)
00024             {
00025                 Debug.WriteLine($"Error inicializando VLC: {ex.Message}");
00026             }
00027
00028
00029             // Arrancamos la aplicación Avalonia
00030             BuildAvaloniaApp()
00031                 .StartWithClassicDesktopLifetime(args);
00032         }
00033
00034         // Avalonia configuration, don't remove; also used by visual designer.
00035         public static AppBuilder BuildAvaloniaApp()
00036             => AppBuilder.Configure<App>()
00037                 .UsePlatformDetect()
00038                 .WithInterFont()
00039                 .LogToTrace()
00040                 .UseReactiveUI();
00041     }
00042 }
```

4.69. Referencia del archivo BetaProyecto/Services/AudioService.cs

4.70. AudioService.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using System.IO;
00003 using System.Linq;
00004 using System.Net.Http;
00005 using System.Text.Json.Nodes;
00006 using System.Threading.Tasks;
00007
00008
00009 namespace BetaProyecto.Services
00010 {
00011     public class AudioService
00012     {
00013         // URL de nuestra API
00014         private const string API_URL = "https://localhost:7500/api/Music/stream";
00015
00016         private readonly HttpClient _httpClient;
00017         public class InfoCancionNube
00018         {
00019             public string Url { get; set; }
00020             public int DuracionSegundos { get; set; }
00021         }
00022
00023         public AudioService()
00024         {
00025             // Esto le dice a la App: "No te quejes si el certificado del servidor es 'raro' o de desarrollo".
00026             // Sin esto evitamos que falle la conexión HTTPS
00027             var handler = new HttpClientHandler();
00028             handler.ServerCertificateCustomValidationCallback = (message, cert, chain, errors) => true;
00029             _httpClient = new HttpClient(handler);
00030         }
00031         /// <summary>
00032         /// Solicita de forma asíncrona la extracción y el análisis de metadatos de un recurso de YouTube a través de una
    API externa.
00033         /// </summary>
00034         /// <remarks>
00035         /// Este método gestiona la comunicación con el microservicio de streaming mediante los siguientes pasos:
00036         /// <list type="number">
```

```
00037         /// <item><b>Construcción:</b> Genera una URI de consulta codificando la <paramref name="urlYoutube"/> como parámetro.</item>
00038         /// <item><b>Petición:</b> Realiza una llamada GET utilizando el <c>HttpClient</c> configurado.</item>
00039         /// <item><b>Procesamiento:</b> Si la respuesta es exitosa, deserializa el cuerpo JSON para extraer la URL del flujo de audio y la duración total en segundos.</item>
00040         /// </list>
00041         /// En caso de fallo en la red o error en el servidor, captura la excepción y devuelve <c>null</c> para evitar cierres inesperados.
00042         /// </remarks>
00043         /// <param name="urlYoutube">La dirección URL completa del video de YouTube que se desea procesar.</param>
00044         /// <returns>
00045         /// Un objeto <see cref="InfoCancionNube"/> con la URL de streaming y la duración;
00046         /// devuelve <c>null</c> si la API no responde correctamente o el recurso no es válido.
00047         /// </returns>
00048         public async Task<InfoCancionNube> ObtenerMp3(string urlYoutube)
00049         {
00050             try
00051             {
00052                 // Construimos la petición: https://localhost:7500/api/Music/stream?url=...
00053                 string urlPeticion = $"{API_URL}?url={urlYoutube}";
00054 
00055                 // Llamamos al servidor
00056                 var respuesta = await _httpClient.GetAsync(urlPeticion);
00057 
00058                 if (respuesta.IsSuccessStatusCode)
00059                 {
00060                     // Leemos la respuesta (que es un texto JSON)
00061                     string jsonString = await respuesta.Content.ReadAsStringAsync();
00062 
00063                     // Sacamos el valor de "url"
00064                     var nodoJson = JsonNode.Parse(jsonString);
00065 
00066                     return new InfoCancionNube
00067                     {
00068                         Url = nodoJson["url"]?.ToString(),
00069                         // Si 'duracion' viene nulo, ponemos 0
00070                         DuracionSegundos = nodoJson["duracion"] != null ? (int)nodoJson["duracion"] : 0
00071                     };
00072                 }
00073             }
00074             catch (Exception ex)
00075             {
00076                 System.Diagnostics.Debug.WriteLine("Error llamando a la API: " + ex.Message);
00077             }
00078 
00079             return null;
00080         }
00081         /// <summary>
00082         /// Gestiona la descarga, protección mediante cifrado y recuperación de archivos de audio en el almacenamiento local.
00083         /// </summary>
00084         /// <remarks>
00085         /// Este método implementa un sistema de seguridad y caché distribuido en dos niveles:
00086         /// <list type="number">
00087         /// <item><b>Caché persistente (.enc):</b> Si el archivo no existe, se descarga de internet, se cifra mediante AES y se guarda en la carpeta de datos locales de la aplicación.</item>
00088         /// <item><b>Caché temporal (.mp3):</b> Para permitir la reproducción en el motor de audio (VLC), el archivo se desencripta "on-the-fly" hacia una ruta temporal volátil.</item>
00089         /// <item><b>Optimización:</b> Si el archivo ya ha sido procesado previamente, evita la descarga redundante y prioriza la recuperación desde el disco.</item>
00090         /// </list>
00091         /// En caso de error crítico durante el cifrado o acceso a disco, el método retorna la URL original como mecanismo de contingencia.
00092         /// </remarks>
00093         /// <param name="url">La dirección URL de origen del flujo de audio.</param>
00094         /// <param name="idCancion">Identificador único de la canción, utilizado para nombrar los archivos en el almacenamiento físico.</param>
00095         /// <returns>
00096         /// Una cadena con la ruta local al archivo MP3 desencriptado listo para su reproducción,
00097         /// o la URL original si ocurre una excepción.
00098         /// </returns>
00099         public async Task<string> ObtenerRutaAudioSegura(string url, string idCancion)
00100         {
00101             try
00102             {
00103                 // Definimos rutas
00104                 // carpetaStorage: Aquí guardaremos los archivos .enc (CIFRADOS). Es persistente.
00105                 string carpetaStorage = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "BetaProyecto", "EncryptedStorage");
00106                 // carpetaTemp: Aquí guardaremos temporalmente el .mp3 (DESCIFRADO) para VLC.
00107                 string carpetaTemp = Path.Combine(Path.GetTempPath(), "BetaProyectoMusicTemp");
00108                 // Nos aseguramos de que las carpetas existan
00109                 Directory.CreateDirectory(carpetaStorage);
00110                 Directory.CreateDirectory(carpetaTemp);
00111 
00112                 // Rutas finales de los archivos
00113                 string rutaCifrada = Path.Combine(carpetaStorage, $"{idCancion}.enc");
```

```
00114                 string rutaPlay = Path.Combine(carpetaTemp, $"{idCancion}.mp3");
00115
00116                 // COMPROBAR CACHÉ (¿Ya la hemos descargado antes?)
00117                 if (File.Exists(rutaCifrada))
00118                 {
00119                     System.Diagnostics.Debug.WriteLine($"[CACHE] Canción cifrada encontrada en disco.");
00120
00121                     // Si ya tenemos el temporal listo, lo usamos directo para ir rápido
00122                     if (File.Exists(rutaPlay)) return rutaPlay;
00123
00124                     // Si no está el temporal, usamos el Encriptador para "abrir el candado"
00125                     // Lee el .enc -> Usa la clave AES -> Escribe el .mp3
00126                     await Helpers.Encriptador.DesencriptarArchivo(rutaCifrada, rutaPlay);
00127
00128                     return rutaPlay;
00129                 }
00130
00131                 // DESCARGAR Y PROTEGER (Si no la tenemos)
00132                 System.Diagnostics.Debug.WriteLine($"[RED] Descargando audio...");
00133
00134                 // Descargamos los bytes crudos de internet
00135                 var datosAudio = await _httpClient.GetByteArrayAsync(url);
00136
00137                 System.Diagnostics.Debug.WriteLine($"[SEGURIDAD] Cifrando con AES...");
00138
00139                 // Usamos el Encriptador para "cerrar el candado" antes de guardar
00140                 byte[] datosCifrados = Helpers.Encriptador.EncriptarBytes(datosAudio);
00141
00142                 // Guardamos la versión CIFRADA (Esta es la que se queda en el disco duro)
00143                 await File.WriteAllBytesAsync(rutaCifrada, datosCifrados);
00144
00145                 // Guardamos la versión DESCIFRADA temporalmente (Solo para que VLC la reproduzca ahora)
00146                 await File.WriteAllBytesAsync(rutaPlay, datosAudio);
00147
00148                 return rutaPlay;
00149             }
00150             catch (Exception ex)
00151             {
00152                 System.Diagnostics.Debug.WriteLine($"[ERROR] Fallo en descarga segura: {ex.Message}");
00153                 // Si algo falla, devolvemos la URL original para intentar streaming directo que lo mas seguro es que falle,
00154                 // pero al menos no se crasea la app.
00155                 return url;
00156             }
00157         }
00158     }
00159 }
```

4.71. Referencia del archivo BetaProyecto/Services/DialogoService.cs

4.72. DialogoService.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia;
00002 using Avalonia.Controls.ApplicationLifetimes;
00003 using BetaProyecto.Views.WindowsRising;
00004 using System.Threading.Tasks;
00005
00006 namespace BetaProyecto.Services
00007 {
00008     // Esta clase implemente el servicio de diálogo
00009     public class DialogoService : IDialogoService
00010     {
00011         public void MostrarAlerta(string mensaje)
00012         {
00013             // La lógica visual de Avalonia se queda encapsulada aquí
00014             if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)
00015             {
00016                 var aviso = new VentanaAviso(mensaje);
00017                 aviso.ShowDialog(desktop.MainWindow);
00018             }
00019         }
00020         public async Task<bool> Preguntar(string titulo, string mensaje, string textoSi, string textoNo)
00021         {
00022             if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)
00023             {
00024                 // Creamos la ventana pasando todos los textos personalizables
00025                 var ventana = new VentanaConfirmacion(titulo, mensaje, textoSi, textoNo);
00026
00027                 // Esperamos el resultado
00028                 var resultado = await ventana.ShowDialog<bool>(desktop.MainWindow);
```

```
00029
00030             return resultado;
00031         }
00032         return false;
00033     }
00034 }
00035 }
```

4.73. Referencia del archivo BetaProyecto/Services/IDialogoService.cs

4.74. IDialogoService.cs

Ir a la documentación de este archivo.

```
00001
00002 using System.Threading.Tasks;
00003
00004 namespace BetaProyecto.Services
00005 {
00006     public interface IDialogoService
00007
00008     {   // Solo definimos la interfaz, sin detalles de implementación ni referencias
00009         void MostrarAlerta(string mensaje);
00010         Task<bool> Preguntar(string titulo, string mensaje, string textoSi, string textoNo);
00011     }
00012 }
```

4.75. Referencia del archivo BetaProyecto/Services/MongoAtlas.cs

4.76. MongoAtlas.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Helpers;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Singleton;
00004 using MongoDB.Bson;
00005 using MongoDB.Driver;
00006 using System;
00007 using System.Collections.Generic;
00008 using System.Linq;
00009 using System.Threading.Tasks;
00010
00011 namespace BetaProyecto.Services
00012 {
00013     public class MongoAtlas
00014     {
00015
00016         ////////////////////////////////
00017         // Variables de la base de datos
00018         ////////////////////////////////
00019         // Las declaramos aquí fuera para que no se "mueran" al terminar la función
00020         private MongoClient _client;
00021
00022         // Esta propiedad pública permitirá que los ViewModels puedan pedir datos a la BD
00023         public IMongoDatabase? Database{ get; private set; }
00024         public MongoAtlas()
00025         {
00026
00027         }
00028         // Método para conectar a la base de datos en la nube
00029         // Esto permite que la conexion siga viva mientras estamos en la aplicación
00030         public async Task<bool> Conectar()
00031         {
00032             if (Database != null)
00033             {
00034                 // Ya existe una conexion
00035                 return true;
00036             }
00037             else
00038             {
00039                 const string connectionUri =
    "mongodb+srv://admin:12345@appaudiolibcluster.6orokhr.mongodb.net/?appName=AppAudioLibCluster";
00040                 try
00041                 {
```

```
00042                 // 1. Usamos la configuración automática
00043                 var settings = MongoClientSettings.FromConnectionString(connectionUri);
00044
00045                 settings.ServerApi = new ServerApi(ServerApiVersion.V1);
00046
00047                 // Creamos el cliente y nos conectamos al server
00048                 _client = new MongoClient(settings);
00049                 Database = _client.GetDatabase("AppAudioLibDB");
00050
00051                 // Enviamos un ping para confirmar una conexión exitosa
00052                 var result = await Database.RunCommandAsync<BsonDocument>(new BsonDocument("ping", 1));
00053                 System.Diagnostics.Debug.WriteLine("Conexión exitosa a MongoDB");
00054                 return true;
00055             }
00056             catch (Exception ex)
00057             {
00058                 System.Diagnostics.Debug.WriteLine("****DEBUG****" + ex);
00059                 return false;
00060             }
00061         }
00062
00063     }
00064     public async Task<Usuarios> LoginUsuario(string username, string password)
00065     {
00066         try
00067         {
00068             // Conectamos con la colección que queremos.
00069             var coleccionUsuarios= Database.GetCollection<Usuarios>("usuarios");
00070
00071             string passwordHash = Encriptador.HashPassword(password);
00072
00073             // Buscamos un usuario que tenga ESE username Y ESA contraseña
00074             var usuario = await coleccionUsuarios
00075                                 .Find(u => u.Username == username && u.Password == passwordHash)
00076                                 .FirstOrDefaultAsync();
00077
00078             return usuario;
00079         }
00080         catch (Exception ex)
00081         {
00082             Console.WriteLine("Error en Login: " + ex.Message);
00083             return null;
00084         }
00085     }
00086
00087     #region Metodos Select / Obtener Datos
00088     ////////////////////////////////////////
00089     ////METODOS SELECT O OBTENER DE DATOS////
00090     ////////////////////////////////////////
00091     #region Para obtener canciones
00092     public async Task<List<Canciones» ObtenerCancionesFavoritos()
00093     {
00094         var listafavoritos = GlobalData.Instance.FavoritosGD;
00095         //Comprobarmos si en la lista de favoritos hay algo
00096         if (listafavoritos.Equals(null) || listafavoritos.Count == 0 )
00097         {
00098             System.Diagnostics.Debug.WriteLine("No se encontraron datos");
00099             return new List<Canciones>();
00100         }
00101
00102         if(!await Conectar()){
00103             System.Diagnostics.Debug.WriteLine("Error de conexión");
00104             return new List<Canciones>();
00105
00106         }
00107
00108         try
00109         {
00110             //Apuntamos a la tablas de queremos de la base de datos
00111             var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00112
00113             //Creamo un filtro que dice "Buscame todas las canciones que tengan el ID en esta lista"
00114             var filtro = Builders<Canciones>.Filter.In(c => c.Id, listafavoritos);
00115
00116             var listaObtenida = await coleccionCanciones.Find(filtro).ToListAsync();
00117
00118             await RellenarNombresDeArtistas(listaObtenida);
00119
00120             return listaObtenida;
00121
00122         }
00123         catch (Exception ex)
00124         {
00125             System.Diagnostics.Debug.WriteLine("Error recuperando canciones favoritas: " + ex.Message);
00126             return new List<Canciones>();
00127         }
00128
```

```
00129       }//ObtenerCancionesFavoritos
00130       public async Task<List<Canciones» ObtenerCacionesNovedades()
00131       {
00132           if (!await Conectar())
00133           {
00134               System.Diagnostics.Debug.WriteLine("Error de conexión");
00135               return new List<Canciones>();
00136           }
00137
00138           try
00139           {
00140               var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00141
00142               // Traemos todas las canciones
00143               var listaCanciones = await coleccionCanciones.Find(_ => true)
00144                                                            .SortByDescending(c => c.Id)
00145                                                            .Limit(10)
00146                                                            .ToListAsync();
00147               await RellenarNombresDeArtistas(listaCanciones);
00148
00149               return listaCanciones;
00150           }
00151           catch (Exception ex)
00152           {
00153               System.Diagnostics.Debug.WriteLine("Error obteniendo canciones: " + ex.Message);
00154               return new List<Canciones>();
00155           }
00156       }
00157
00158       public async Task<List<Canciones» ObtenerCancionesPorGenero(string genero)
00159       {
00160           if (!await Conectar())
00161           {
00162               System.Diagnostics.Debug.WriteLine("Error de conexión");
00163               return new List<Canciones>();
00164
00165           }
00166
00167           try
00168           {
00169               var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00170
00171               // Hacemos el Find buscado por género
00172               var listaCanciones = await coleccionCanciones.Find(c => c.Datos.Generos.Contains(genero))
00173                                                            .Limit(15)
00174                                                            .ToListAsync();
00175
00176               await RellenarNombresDeArtistas(listaCanciones);
00177
00178               return listaCanciones;
00179           }
00180           catch (Exception ex)
00181           {
00182               System.Diagnostics.Debug.WriteLine("Error obteniendo canciones: " + ex.Message);
00183               return new List<Canciones>();
00184           }
00185       }
00186
00187       //Obtener todas las canciones
00188       public async Task<List<Canciones» ObtenerCanciones()
00189       {
00190           if (!await Conectar())
00191           {
00192               System.Diagnostics.Debug.WriteLine("Error de conexión");
00193               return new List<Canciones>();
00194
00195           }
00196
00197           try
00198           {
00199               // Apuntamos a la colección
00200               var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00201               var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00202
00203               // Traemos todas las canciones
00204               var listaCanciones = await coleccionCanciones.Find(_ => true).ToListAsync();
00205
00206               await RellenarNombresDeArtistas(listaCanciones);
00207
00208               return listaCanciones;
00209           }
00210           catch (Exception ex)
00211           {
00212               System.Diagnostics.Debug.WriteLine("Error obteniendo canciones: " + ex.Message);
00213               return new List<Canciones>();
00214           }
00215       }
```

```
00216         public async Task<List<Canciones» ObtenerCancionesPorListaIds(List<string> ids)
00217         {
00218             if (!await Conectar())
00219             {
00220                 System.Diagnostics.Debug.WriteLine("Error de conexión");
00221                 return new List<Canciones>();
00222             }
00223 
00224             var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00225 
00226             // Filtro "IN": Dame todas las canciones cuyo ID esté en esta lista
00227             var filtro = Builders<Canciones>.Filter.In(c => c.Id, ids);
00228 
00229             var listacanciones = await coleccionCanciones.Find(filtro).ToListAsync();
00230 
00231             await RellenarNombresDeArtistas(listacanciones);
00232 
00233             return listacanciones;
00234         }
00235 
00236         public async Task<List<Canciones» ObtenerCancionesPorBusqueda(string textoBusqueda)
00237         {
00238             if (!await Conectar())
00239             {
00240                 System.Diagnostics.Debug.WriteLine("Error de conexión");
00241                 return new List<Canciones>();
00242             }
00243             try
00244             {
00245                 // Apuntamos a la colección.
00246                 var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00247 
00248                 // Hacemos el Find buscado por nombre (IGNORA LAS MAYUSCULAS Y MINUSCULAS poniendo la "i"
    activamos esto)
00249                 var filtro = Builders<Canciones>.Filter.Regex(c => c.Titulo, new BsonRegularExpression(textoBusqueda, "i"));
00250 
00251                 var listaCanciones = await coleccionCanciones.Find(filtro).ToListAsync();
00252 
00253                 await RellenarNombresDeArtistas(listaCanciones);
00254 
00255                 return listaCanciones;
00256             }
00257             catch (Exception ex)
00258             {
00259                 System.Diagnostics.Debug.WriteLine("Error obteniendo canciones: " + ex.Message);
00260                 return new List<Canciones>();
00261             }
00262         }
00263 
00264         public async Task<List<Canciones» ObtenerCancionesPorAutor(string idAutor)
00265         {
00266             if (!await Conectar())
00267             {
00268                 System.Diagnostics.Debug.WriteLine("Error de conexión");
00269                 return new List<Canciones>();
00270             }
00271 
00272             try
00273             {
00274                 var coleccion = Database.GetCollection<Canciones>("canciones");
00275 
00276                 // 'AnyEq' sirve para buscar un valor dentro de un array en Mongo
00277                 var filtro = Builders<Canciones>.Filter.AnyEq(c => c.AutoresIds, idAutor);
00278 
00279                 var lista = await coleccion.Find(filtro).ToListAsync();
00280 
00281                 await RellenarNombresDeArtistas(lista);
00282 
00283                 return lista;
00284             }
00285             catch (Exception ex)
00286             {
00287                 System.Diagnostics.Debug.WriteLine("Error obteniendo canciones por autor: " + ex.Message);
00288                 return new List<Canciones>();
00289             }
00290         }
00291 
00292         #endregion
00293 
00294         #region Para obtener usuarios
00295         public async Task<List<Usuarios» ObtenerTodosLosUsuarios()
00296         {
00297             if (!await Conectar())
00298             {
00299                 System.Diagnostics.Debug.WriteLine("Error de conexión");
00300                 return new List<Usuarios>();
00301             }
```

```
00302           try
00303           {
00304               var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00305
00306               // Traemos la lista (limitada a 20 o 50 para no sobrecargar si hay millones)
00307               var listaUsuarios = await coleccionUsuarios.Find(_ => true).Limit(50).ToListAsync();
00308
00309               return listaUsuarios;
00310           }
00311           catch (Exception ex)
00312           {
00313               System.Diagnostics.Debug.WriteLine("Error al obtener usuarios: " + ex.Message);
00314               return new List<Usuarios>();
00315           }
00316
00317       }
00318
00319       public async Task<List<Usuarios» ObtenerUsuariosPorBusqueda(string textoBusqueda, List<string> idsExcluidos)
00320       {
00321           if (!await Conectar())
00322           {
00323               System.Diagnostics.Debug.WriteLine("Error de conexión");
00324               return new List<Usuarios>();
00325           }
00326           try
00327           {
00328               var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00329
00330               // Filtro básico: Buscar por nombre (ignorando mayúsculas/minúsculas)
00331               var filtroBusqueda = Builders<Usuarios>.Filter.Regex(c => c.Username, new
    BsonRegularExpression(textoBusqueda, "i"));
00332
00333              // Filtro de exclusión: Si nos pasan IDs, le decimos a Mongo "Busca X, PERO que el ID NO ESTÉ en esta lista"
00334               // 'Nin' significa "Not In" (No está en...)
00335               var filtroExclusion = Builders<Usuarios>.Filter.Nin(u => u.Id, idsExcluidos);
00336
00337               //Declaramos un filtro
00338               FilterDefinition<Usuarios> filtroFinal;
00339
00340               // Combinamos ambos filtros con un AND
00341               filtroFinal = Builders<Usuarios>.Filter.And(filtroBusqueda, filtroExclusion);
00342
00343               // Ejecutamos la consulta con el filtro combinado
00344               var listaUsuarios = await coleccionUsuarios.Find(filtroFinal).ToListAsync();
00345
00346               return listaUsuarios;
00347           }
00348           catch (Exception ex)
00349           {
00350               System.Diagnostics.Debug.WriteLine("Error obteniendo canciones: " + ex.Message);
00351               return new List<Usuarios>();
00352           }
00353       }
00354
00355       public async Task<List<Usuarios» ObtenerUsuariosPorListaIds(List<string> listaIds)
00356       {
00357           // 1. Si la lista está vacía ni nos molestamos en buscar en la base de datos
00358           if (listaIds == null || listaIds.Count == 0)
00359           {
00360               return new List<Usuarios>();
00361           }
00362
00363           if (!await Conectar())
00364           {
00365               return new List<Usuarios>();
00366           }
00367
00368           try
00369           {
00370               var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00371
00372               // Busca cualquier usuario cuyo Id esté DENTRO de la lista 'listaIds'
00373               var filtro = Builders<Usuarios>.Filter.In(u => u.Id, listaIds);
00374               var listaUsuarios = await coleccionUsuarios.Find(filtro).ToListAsync();
00375
00376               return listaUsuarios;
00377           }
00378           catch (Exception ex)
00379           {
00380               System.Diagnostics.Debug.WriteLine("Error al obtener lista de usuarios por IDs: " + ex.Message);
00381               return new List<Usuarios>();
00382           }
00383       }
00384       #endregion
00385
00386       #region Para obtener listaspersonalizas
00387
```

```
00388         //Obtener todas las listas
00389         public async Task<List<ListaPersonalizada» ObtenerListasReproduccion()
00390         {
00391             if (!await Conectar())
00392             {
00393                 System.Diagnostics.Debug.WriteLine("Error de conexión");
00394                 return new List<ListaPersonalizada>();
00395             }
00396             try
00397             {
00398                 var coleccionListas = Database.GetCollection<ListaPersonalizada>("listapersonalizada");
00399                 var listas = await coleccionListas.Find(_ => true).ToListAsync();
00400 
00401                 // Rellenamos las canciones reales para cada lista pra poder reporducirlas o mostrarlas depende el caso
00402                 foreach (var lista in listas)
00403                 {
00404                     if (lista.IdsCanciones.Count > 0)
00405                     {
00406                         lista.CancionesCompletas = await ObtenerCancionesPorListaIds(lista.IdsCanciones);
00407                     }
00408                 }
00409 
00410                 return listas;
00411             }
00412             catch (Exception ex)
00413             {
00414                 System.Diagnostics.Debug.WriteLine("Error obteniendo playlists: " + ex.Message);
00415                 return new List<ListaPersonalizada>();
00416             }
00417         }
00418 
00419         public async Task<List<ListaPersonalizada» ObtenerPlaylistsPorCreador(string idUsuario)
00420         {
00421             if (!await Conectar())
00422             {
00423                 System.Diagnostics.Debug.WriteLine("Error de conexión");
00424                 return new List<ListaPersonalizada>();
00425             }
00426 
00427             try
00428             {
00429                 var coleccion = Database.GetCollection<ListaPersonalizada>("listapersonalizada");
00430 
00431                 // 'AnyEq' sirve para buscar un valor dentro de un campo normal en este caso IdUsuario en Mongo
00432                 var filtro = Builders<ListaPersonalizada>.Filter.Eq(l => l.IdUsuario, idUsuario);
00433 
00434                 var listas = await coleccion.Find(filtro).ToListAsync();
00435 
00436                 foreach (var lista in listas)
00437                 {
00438                     if (lista.IdsCanciones.Count > 0)
00439                     {
00440                         lista.CancionesCompletas = await ObtenerCancionesPorListaIds(lista.IdsCanciones);
00441                     }
00442                 }
00443 
00444                 return listas;
00445             }
00446             catch (Exception ex)
00447             {
00448                 System.Diagnostics.Debug.WriteLine("Error obteniendo playlists por creador: " + ex.Message);
00449                 return new List<ListaPersonalizada>();
00450             }
00451         }
00452         #endregion
00453 
00454 
00455         /// <summary>
00456         /// Se ocupa de traer todos los reportes de la base de datos
00457         /// </summary>
00458         /// <returns></returns>
00459         public async Task<List<Reportes» ObtenerReportes()
00460         {
00461             if (!await Conectar())
00462             {
00463                 return new List<Reportes>();
00464             }
00465 
00466             try
00467             {
00468                 var coleccion = Database.GetCollection<Reportes>("reportes");
00469 
00470                 // Los traemos ordenados por fecha (los más nuevos primero)
00471                 var lista = await coleccion.Find(_ => true)
00472                                            .SortByDescending(r => r.FechaCreacion)
00473                                            .ToListAsync();
00474 
```

```
00475             // RELLENAMOS LOS NOMBRES REALES (Usuario y Canción)
00476             var colUsuarios = Database.GetCollection<Usuarios>("usuarios");
00477             var colCanciones = Database.GetCollection<Canciones>("canciones");
00478 
00479             foreach (var reporte in lista)
00480             {
00481 
00482                 var usuario = await colUsuarios.Find(u => u.Id == reporte.Referencias.UsuarioReportanteId)
00483                                                .Project(u => new { u.Username })
00484                                                .FirstOrDefaultAsync();
00485 
00486                 reporte.NombreReportante = usuario != null ? usuario.Username : "Usuario Eliminado";
00487 
00488 
00489                 var cancion = await colCanciones.Find(c => c.Id == reporte.Referencias.CancionReportadaId)
00490                                                 .Project(c => new { c.Titulo })
00491                                                 .FirstOrDefaultAsync();
00492 
00493                 reporte.TituloCancionReportada = cancion != null ? cancion.Titulo : "Canción Eliminada";
00494 
00495             }
00496 
00497             return lista;
00498         }
00499         catch (Exception ex)
00500         {
00501             System.Diagnostics.Debug.WriteLine("Error obteniendo reportes: " + ex.Message);
00502             return new List<Reportes>();
00503         }
00504     }
00505 
00506     /// <summary>
00507     /// Obtenemos la lista de nombres de géneros de la base de datos
00508     /// </summary>
00509     /// <returns></returns>
00510     public async Task<List<string» ObtenerNombresGeneros()
00511     {
00512         if (!await Conectar())
00513         {
00514             return new List<string>();
00515         }
00516         try
00517         {
00518             var coleccion = Database.GetCollection<Generos>("generos");
00519 
00520             var listaGeneros = await coleccion.Find(_ => true)
00521                                               .SortBy(g => g.Nombre)
00522                                               .ToListAsync();
00523 
00524             return listaGeneros.Select(g => g.Nombre).ToList();
00525         }
00526         catch (Exception ex)
00527         {
00528             System.Diagnostics.Debug.WriteLine("Error obteniendo géneros: " + ex.Message);
00529             return new List<string>();
00530         }
00531     }
00532 
00533     /// <summary>
00534     /// Obtenemos los objetos completos de géneros de la base de datos
00535     /// </summary>
00536     /// <returns></returns>
00537     public async Task<List<Generos» ObtenerGenerosCompletos()
00538     {
00539         if (!await Conectar())
00540         {
00541             return new List<Generos>();
00542         }
00543 
00544         try
00545         {
00546             var coleccion = Database.GetCollection<Generos>("generos");
00547             return await coleccion.Find(_ => true).ToListAsync();
00548         }
00549         catch (Exception ex)
00550         {
00551             System.Diagnostics.Debug.WriteLine("Error obteniendo géneros: " + ex.Message);
00552             return new List<Generos>();
00553         }
00554     }
00555 
00556     /// <summary>
00557     /// Obtiene una mezcla de canciones por género (veteranos + indies) partir del puntaje de tendencia
00558     /// </summary>
00559     /// <param name="genero"></param>
00560     /// <returns></returns>
00561     public async Task<List<Canciones» ObtenerMixPorGenero(string genero)
```

```
00562       {
00563           if (!await Conectar())
00564           {
00565               return new List<Canciones>();
00566           }
00567
00568           try
00569           {
00570               var coleccion = Database.GetCollection<Canciones>("canciones");
00571
00572               // Filtro para buscar canciones del género solicitado
00573               var filtroGenero = Builders<Canciones>.Filter.Eq("datos.generos", genero);
00574
00575               // LOS VETERANOS
00576               // Estos son los que mantienen al usuario enganchado
00577               var veteranos = await coleccion.Find(filtroGenero)
00578                                       .SortByDescending(c => c.Metricas.PuntuacionTendencia)
00579                                       .Limit(10)
00580                                       .ToListAsync();
00581
00582               // LOS INDIES
00583               // Estos son los que nos intersa darles visibilidad
00584               var indies = await coleccion.Find(filtroGenero)
00585                                       .SortBy(c => c.Metricas.TotalReproducciones)
00586                                       .Limit(10)
00587                                       .ToListAsync();
00588
00589               // LA MEZCLA (Veteranos + Indies)
00590               var mezcla = new List<Canciones>();
00591               mezcla.AddRange(veteranos);
00592
00593              // Añadimos los indies (evitando que si una canción es Top y tiene pocas visitas salga repetida, aunque es raro)
00594               foreach (var indie in indies)
00595               {
00596                   if (!mezcla.Any(c => c.Id == indie.Id))
00597                   {
00598                       mezcla.Add(indie);
00599                   }
00600               }
00601
00602               // 4. BARAJAMOS EL RESULTADO FINAL
00603               var random = new Random();
00604               var listaFinal = mezcla.OrderBy(x => random.Next()).ToList();
00605
00606               // Rellenamos nombres de artistas
00607               await RellenarNombresDeArtistas(listaFinal);
00608
00609               return listaFinal;
00610           }
00611           catch (Exception ex)
00612           {
00613               System.Diagnostics.Debug.WriteLine("Error en Mix Por Género: " + ex.Message);
00614               return new List<Canciones>();
00615           }
00616       }
00617       #endregion
00618
00619       #region Metodos Inserts
00620       ////////////////////////////////////////
00621       ////////////METODOS INSERTS/////////////
00622       ////////////////////////////////////////
00623
00624       /// <summary>
00625       /// Se ocupa de añadir una canción a la lista de favoritos del usuario en la base de datos
00626       /// </summary>
00627       /// <param name="idUsuario"></param>
00628       /// <param name="idCancion"></param>
00629       /// <returns></returns>
00630       public async Task<bool> AgregarAFavorito(string idUsuario, string idCancion)
00631       {
00632           if (!await Conectar())
00633           {
00634               return false;
00635           }
00636           try
00637           {
00638               var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00639
00640               //Configuramos el filtro para encontrar el usuario a actualizar
00641               var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idUsuario);
00642
00643              //Y ahora buscamos la lista de favoritos que queremos actualizar y le decimos el ID de la canción que tiene que
     añadir
00644               var update = Builders<Usuarios>.Update.AddToSet("listas.favoritos", ObjectId.Parse(idCancion));
00645
00646               //Ejecutamos la actualización
00647               var documentosActualizados = await coleccionUsuarios.UpdateOneAsync(filtro, update);
```

```
00648
00649               var seActualizo = documentosActualizados.ModifiedCount > 0;
00650
00651               return seActualizo;
00652           }
00653           catch (Exception ex)
00654           {
00655               System.Diagnostics.Debug.WriteLine("Error Añadir a Fav: " + ex);
00656               return false;
00657           }
00658       }
00659
00660       /// <summary>
00661       /// Se ocupa de añadir un usuario a la lista de seguidores del usuario en la base de datos
00662       /// </summary>
00663       /// <param name="idUsuario"></param>
00664       /// <param name="idUsuarioASeguir"></param>
00665       /// <returns></returns>
00666       public async Task<bool> SeguirUsuario(string idUsuario, string idUsuarioASeguir)
00667       {
00668           if (!await Conectar())
00669           {
00670               return false;
00671           }
00672
00673           try
00674           {
00675               var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00676
00677               var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idUsuario);
00678
00679               var update = Builders<Usuarios>.Update.AddToSet("listas.seguidores", ObjectId.Parse(idUsuarioASeguir));
00680
00681               var documentosActualizados = await coleccionUsuarios.UpdateOneAsync(filtro, update);
00682
00683               var seActualizo = documentosActualizados.ModifiedCount > 0;
00684
00685               return seActualizo;
00686           }
00687           catch (Exception ex)
00688           {
00689               System.Diagnostics.Debug.WriteLine("Error al seguir: " + ex);
00690               return false;
00691           }
00692       }
00693
00694       /// <summary>
00695       /// Se ocupa de publicar una nueva canción en la base de datos
00696       /// </summary>
00697       /// <param name="nuevaCancion"></param>
00698       /// <returns></returns>
00699       public async Task<bool> PublicarCancion(Canciones nuevaCancion)
00700       {
00701           if (!await Conectar())
00702           {
00703               return false;
00704           }
00705
00706           try
00707           {
00708               var coleccionCanciones = Database.GetCollection<Canciones>("canciones");
00709               await coleccionCanciones.InsertOneAsync(nuevaCancion);
00710               return true;
00711           }
00712           catch (Exception ex)
00713           {
00714               System.Diagnostics.Debug.WriteLine("Error al publicar canción: " + ex.Message);
00715               return false;
00716           }
00717       }
00718       /// <summary>
00719       /// Se ocupa de crear una nueva listapersonalizada en la base de datos
00720       /// </summary>
00721       /// <param name="nuevaLista"></param>
00722       /// <returns></returns>
00723       public async Task<bool> CrearListaReproduccion(ListaPersonalizada nuevaLista)
00724       {
00725           if (!await Conectar())
00726           {
00727               return false;
00728           }
00729
00730           try
00731           {
00732               var coleccion = Database.GetCollection<ListaPersonalizada>("listapersonalizada");
00733               await coleccion.InsertOneAsync(nuevaLista);
00734               return true;
```

```
00735           }
00736           catch (Exception ex)
00737           {
00738               System.Diagnostics.Debug.WriteLine("Error creando lista: " + ex.Message);
00739               return false;
00740           }
00741       }
00742
00743       /// <summary>
00744       /// Se ocupa de enviar un nuevo reporte a la base de datos
00745       /// </summary>
00746       /// <param name="nuevoReporte"></param>
00747       /// <returns></returns>
00748       public async Task<bool> EnviarReporte(Reportes nuevoReporte)
00749       {
00750           if (!await Conectar())
00751           {
00752               return false;
00753           }
00754
00755           try
00756           {
00757               var coleccion = Database.GetCollection<Reportes>("reportes");
00758               await coleccion.InsertOneAsync(nuevoReporte);
00759               return true;
00760           }
00761           catch (Exception ex)
00762           {
00763               System.Diagnostics.Debug.WriteLine("Error al enviar reporte: " + ex.Message);
00764               return false;
00765           }
00766       }
00767
00768       /// <summary>
00769       /// Se ocupa de crear un nuevo género en la base de datos
00770       /// </summary>
00771       /// <param name="nuevoGenero"></param>
00772       /// <returns></returns>
00773       public async Task<bool> CrearGenero(string nuevoGenero)
00774       {
00775           if (!await Conectar())
00776           {
00777               return false;
00778           }
00779           try
00780           {
00781               var coleccion = Database.GetCollection<Generos>("generos");
00782
00783               // Comprobamos si hay algun género con el mismo nombre (ignorando mayúsculas/minúsculas)
00784               var filtro = Builders<Generos>.Filter.Regex(g => g.Nombre, new
    MongoDB.Bson.BsonRegularExpression($"^{nuevoGenero}$", "i"));
00785
00786               var existe = await coleccion.Find(filtro).AnyAsync();
00787
00788               if (existe)
00789               {
00790                   return false;
00791               }
00792
00793               // Si no existe, lo creamos
00794               var nuevo = new Generos { Nombre = nuevoGenero };
00795               await coleccion.InsertOneAsync(nuevo);
00796               return true;
00797           }
00798           catch (Exception ex)
00799           {
00800               System.Diagnostics.Debug.WriteLine("Error al insetar un nuevo género: " + ex.Message);
00801               return false;
00802           }
00803       }
00804
00805       /// <summary>
00806       /// Se ocupa de crear un nuevo usuario en la base de datos
00807       /// </summary>
00808       /// <param name="nuevoUsuario"></param>
00809       /// <returns></returns>
00810       public async Task<bool> CrearUsuario(Usuarios nuevoUsuario)
00811       {
00812           if (!await Conectar())
00813           {
00814               return false;
00815           }
00816           try
00817           {
00818               var coleccion = Database.GetCollection<Usuarios>("usuarios");
00819
00820               //Verificamos que no exista ya un usuario con ese email
```

```
00821             var existe = await coleccion.Find(u => u.Email == nuevoUsuario.Email).AnyAsync();
00822             if (existe) return false;
00823 
00824             await coleccion.InsertOneAsync(nuevoUsuario);
00825             return true;
00826         }
00827         catch (Exception ex)
00828         {
00829             System.Diagnostics.Debug.WriteLine("Error registrando usuario: " + ex.Message);
00830             return false;
00831         }
00832     }
00833     #endregion
00834 
00835     #region Metodos Deletes
00836     ////////////////////////////////////////
00837     ////////////METODOS DELETE//////////////
00838     ////////////////////////////////////////
00839 
00840     /// <summary>
00841     ///     Se ocupa de eliminar una canción de la lista de favoritos del usuario en la base de datos
00842     /// </summary>
00843     /// <param name="idUsuario"></param>
00844     /// <param name="idCancion"></param>
00845     /// <returns></returns>
00846     public async Task<bool> EliminarDeFavorito(string idUsuario, string idCancion)
00847     {
00848         if (!await Conectar())
00849         {
00850             return false;
00851         }
00852         try
00853         {
00854             var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00855             // Configuramos el filtro para encontrar el usuario a actualizar
00856             var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idUsuario);
00857 
00858             //Y ahora buscamos la lista de favoritos que queremos actualizar y le decimos el ID de la canción que tiene que
    añadir
00859             var update = Builders<Usuarios>.Update.Pull("listas.favoritos", ObjectId.Parse(idCancion));
00860 
00861             //Ejecutamos la actualización
00862             var documentosActualizados = await coleccionUsuarios.UpdateOneAsync(filtro, update);
00863 
00864             var seActualizo = documentosActualizados.ModifiedCount > 0;
00865 
00866             return seActualizo;
00867         }
00868         catch (Exception ex)
00869         {
00870             System.Diagnostics.Debug.WriteLine("Error Eliminar de Fav: " + ex);
00871             return false;
00872         }
00873     }
00874     /// <summary>
00875     ///     Se ocupa de eliminar un usuario de la lista de seguidores del usuario en la base de datos
00876     /// </summary>
00877     /// <param name="idUsuario"></param>
00878     /// <param name="idUsuarioADejar"></param>
00879     /// <returns></returns>
00880     public async Task<bool> DejarDeSeguirUsuario(string idUsuario, string idUsuarioADejar)
00881     {
00882         if (!await Conectar())
00883         {
00884             return false;
00885         }
00886         try
00887         {
00888             var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
00889 
00890             var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idUsuario);
00891 
00892             var update = Builders<Usuarios>.Update.Pull("listas.seguidores", ObjectId.Parse(idUsuarioADejar));
00893 
00894             var documentosActualizados = await coleccionUsuarios.UpdateOneAsync(filtro, update);
00895 
00896             var seActualizo = documentosActualizados.ModifiedCount > 0;
00897 
00898             return seActualizo;
00899         }
00900         catch (Exception ex)
00901         {
00902             System.Diagnostics.Debug.WriteLine("Error al dejar de seguir: " + ex);
00903             return false;
00904         }
00905     }
00906 
```

```
00907         /// <summary>
00908         ///     Se ocupa de eliminar una canción de la base de datos por id
00909         /// </summary>
00910         /// <param name="idCancion"></param>
00911         /// <returns></returns>
00912         public async Task<bool> EliminarCancionPorId(string idCancion)
00913         {
00914             if(!await Conectar())
00915             {
00916                 return false;
00917             }
00918             //Limpiamos de listas de reproducciones para evitar errores de referencias
00919             var colListas = Database.GetCollection<ListaPersonalizada>("listapersonalizada");
00920 
00921             //Buscamos las listas donde se encuntras las canciones
00922             var filtroListas = Builders<ListaPersonalizada>.Filter.AnyEq(l => l.IdsCanciones, idCancion);
00923 
00924             //Sacamos el id de las listras que hemos encontrado
00925             var updateListas = Builders<ListaPersonalizada>.Update.Pull(l => l.IdsCanciones, idCancion);
00926 
00927             //Ejecutamos la actualizacion
00928             await colListas.UpdateManyAsync(filtroListas, updateListas);
00929 
00930 
00931             // Limpiamos la cancion de las litas de favoritos para evitar errores de referencias
00932             var colUsuarios = Database.GetCollection<Usuarios>("usuarios");
00933 
00934             var updateUsuarios = Builders<Usuarios>.Update.Pull(u => u.Listas.Favoritos, idCancion);
00935             var filtroUsuarios = Builders<Usuarios>.Filter.AnyEq(u => u.Listas.Favoritos, idCancion);
00936 
00937             await colUsuarios.UpdateManyAsync(filtroUsuarios, updateUsuarios);
00938             var coleccionCanciones= Database.GetCollection<Canciones>("canciones");
00939 
00940             var filtro = Builders<Canciones>.Filter.Eq(c => c.Id, idCancion);
00941 
00942             var documentosEliminados = await coleccionCanciones.DeleteOneAsync(filtro);
00943 
00944             var seElimino = documentosEliminados.DeletedCount > 0;
00945 
00946             return seElimino;
00947         }
00948         /// <summary>
00949         ///     Eliminamos la listapersonalizada de la base de datos por id
00950         /// </summary>
00951         /// <param name="idPlaylist"></param>
00952         /// <returns></returns>
00953         public async Task<bool> EliminarPlaylistPorId(string idPlaylist)
00954         {
00955             if (!await Conectar())
00956             {
00957                 return false;
00958             }
00959             var coleccionLista= Database.GetCollection<ListaPersonalizada>("listapersonalizada");
00960 
00961             var filtro = Builders<ListaPersonalizada>.Filter.Eq(l => l.Id, idPlaylist);
00962 
00963             var documentosEliminados = await coleccionLista.DeleteOneAsync(filtro);
00964 
00965             var seElimino = documentosEliminados.DeletedCount > 0;
00966 
00967             return seElimino;
00968         }
00969         /// <summary>
00970         ///     Eliminamos el genero de la base de datos
00971         /// </summary>
00972         /// <param name="generoAEliminar"></param>
00973         /// <returns></returns>
00974         public async Task<bool> EliminarGenero(Generos generoAEliminar)
00975         {
00976             if (!await Conectar())
00977             {
00978                 return false;
00979             }
00980             try
00981 
00982             {
00983                 var colCanciones = Database.GetCollection<Canciones>("canciones");
00984                 var colGeneros = Database.GetCollection<Generos>("generos");
00985 
00986                 // 1. VERIFICAR USO
00987                 // Buscamos si existe alguna canción que contenga este nombre de género
00988                 var filtroEnUso = Builders<Canciones>.Filter.AnyEq(c => c.Datos.Generos, generoAEliminar.Nombre);
00989 
00990                 var estaEnUso = await colCanciones.Find(filtroEnUso).AnyAsync();
00991 
00992                 if (estaEnUso)
00993                 {
```

```
00994                 System.Diagnostics.Debug.WriteLine($"[PROTECCIÓN] '{generoAEliminar.Nombre}' está en uso.");
00995                 return false;
00996             }
00997
00998             var result = await Database.GetCollection<Generos>("generos").DeleteOneAsync(g => g.Id ==
      generoAEliminar.Id);
00999
01000             var EsEliminado = result.DeletedCount > 0;
01001
01002             return EsEliminado;
01003         }
01004         catch (Exception ex)
01005         {
01006             System.Diagnostics.Debug.WriteLine("Error al insetar un nuevo género: " + ex.Message);
01007             return false;
01008         }
01009     }
01010     /// <summary>
01011     ///     Eliminamos al usuarios de la base de datos
01012     /// </summary>
01013     /// <param name="idUsuario"></param>
01014     /// <returns></returns>
01015     public async Task<bool> EliminarUsuario(string idUsuario)
01016     {
01017         if (!await Conectar())
01018         {
01019             return false;
01020         }
01021         try {
01022             var colUsuarios = Database.GetCollection<Usuarios>("usuarios");
01023             var colCanciones = Database.GetCollection<Canciones>("canciones");
01024             var colListas = Database.GetCollection<ListaPersonalizada>("listapersonalizada");
01025
01026             // Limpieamos el usuario de listas
01027             var updateCanciones = Builders<Canciones>.Update.Pull(c => c.AutoresIds, idUsuario);
01028             var filtroCanciones = Builders<Canciones>.Filter.AnyEq(c => c.AutoresIds, idUsuario);
01029
01030             await colCanciones.UpdateManyAsync(filtroCanciones, updateCanciones);
01031
01032             //Ahora borramos las canciones que no tiene autor ya que borramos este usuario de todas
01033
01034             var filtroHuerfanas = Builders<Canciones>.Filter.Size(c => c.AutoresIds, 0);
01035             await colCanciones.DeleteManyAsync(filtroHuerfanas);
01036
01037             // Limpiamos la usuario de las listas de seguidos de otros usaurios
01038             var updateSeguidores = Builders<Usuarios>.Update.Pull(u => u.Listas.Seguidores, idUsuario);
01039             var filtroSeguidores = Builders<Usuarios>.Filter.AnyEq(u => u.Listas.Seguidores, idUsuario);
01040
01041             await colUsuarios.UpdateManyAsync(filtroSeguidores, updateSeguidores);
01042
01043             // 3. BORRAR SUS PLAYLISTS
01044             await colListas.DeleteManyAsync(l => l.IdUsuario == idUsuario);
01045
01046             // 1. Borrar el usuario
01047            var resUser = await Database.GetCollection<Usuarios>("usuarios").DeleteOneAsync(u => u.Id == idUsuario);
01048
01049             return resUser.DeletedCount > 0;
01050         }
01051         catch (Exception ex)
01052         {
01053             System.Diagnostics.Debug.WriteLine("Error al insetar un nuevo género: " + ex.Message);
01054             return false;
01055         }
01056     }
01057     /// <summary>
01058     ///     Elimina un reporte del la base de datos
01059     /// </summary>
01060     /// <param name="idReporte"></param>
01061     /// <returns></returns>
01062     public async Task<bool> EliminarReporte(string idReporte)
01063     {
01064         if (!await Conectar())
01065         {
01066             return false;
01067         }
01068         try
01069         {
01070             var coleccion = Database.GetCollection<Reportes>("reportes");
01071             var resultado = await coleccion.DeleteOneAsync(r => r.Id == idReporte);
01072             return resultado.DeletedCount > 0;
01073         }
01074         catch (Exception ex)
01075         {
01076             System.Diagnostics.Debug.WriteLine("Error al eliminar reporte: " + ex.Message);
01077             return false;
01078         }
01079     }
```

```
01080         #endregion
01081
01082         #region Metodos Updates
01083         ////////////////////////////////////////
01084         ////////////METODOS UPDATES///////////////
01085         ////////////////////////////////////////
01086
01087         /// <summary>
01088         ///     Actualizamos el usuario logeado mediantes comprobaciones con el singleton
01089         /// </summary>
01090         /// <param name="idUsuario"></param>
01091         /// <param name="nuevoNombre"></param>
01092         /// <param name="nuevoEmail"></param>
01093         /// <param name="nuevoPais"></param>
01094         /// <param name="nuevaFecha"></param>
01095         /// <param name="esPrivada"></param>
01096         /// <returns></returns>
01097         public async Task<bool> ActualizarPerfilUsuario(string idUsuario, string nuevoNombre, string nuevoEmail, string
      nuevoPais, DateTime nuevaFecha, bool esPrivada)
01098         {
01099             //Preparamos el contructor y una lista con para guardar los cambios
01100             var builder = Builders<Usuarios>.Update;
01101             var listaCambios = new List<UpdateDefinition<Usuarios»();
01102
01103             // Comparamos con lo que tenemos en memoria (GlobalData) para ver si cambió
01104
01105             // Si el algo ha cambiado, lo añadimos a la lista de cambios
01106             if (nuevoNombre != GlobalData.Instance.UsernameGD)
01107                 listaCambios.Add(builder.Set(u => u.Username, nuevoNombre));
01108
01109             if (nuevoEmail != GlobalData.Instance.EmailGD)
01110                 listaCambios.Add(builder.Set(u => u.Email, nuevoEmail));
01111
01112             if (nuevoPais != GlobalData.Instance.PaisGD)
01113                 listaCambios.Add(builder.Set(u => u.Perfil.Pais, nuevoPais));
01114
01115             if (nuevaFecha.Date != GlobalData.Instance.FechaNacimientoGD.Date)
01116                 listaCambios.Add(builder.Set(u => u.Perfil.FechaNacimiento, nuevaFecha));
01117
01118             if (esPrivada != GlobalData.Instance.Es_PrivadaGD)
01119                 listaCambios.Add(builder.Set(u => u.Perfil.EsPrivada, esPrivada));
01120
01121             // Si no hay cambios no llamamos al método
01122             if (listaCambios.Count == 0)
01123             {
01124                 System.Diagnostics.Debug.WriteLine("No se detectaron cambios. No se envió nada a Mongo.");
01125                 return false;
01126             }
01127
01128             // Ejecutamos actualización
01129             if (!await Conectar())
01130             {
01131                 return false;
01132             }
01133
01134             var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idUsuario);
01135
01136             // Combinamos todas las pequeñas actualizaciones en una sola
01137             var updateFinal = builder.Combine(listaCambios);
01138
01139             var resultado = await Database.GetCollection<Usuarios>("usuarios").UpdateOneAsync(filtro, updateFinal);
01140
01141             var EsActualizado = resultado.MatchedCount > 0;
01142
01143             return EsActualizado;
01144         }
01145
01146         /// <summary>
01147         ///     Actualiza un usaurio de la base de datos
01148         /// </summary>
01149         /// <param name="id"></param>
01150         /// <param name="nombre"></param>
01151         /// <param name="email"></param>
01152         /// <param name="password"></param>
01153         /// <param name="rol"></param>
01154         /// <param name="pais"></param>
01155         /// <param name="imagenUrl"></param>
01156         /// <param name="fecha"></param>
01157         /// <param name="esPrivada"></param>
01158         /// <returns></returns>
01159
01160         public async Task<bool> ActualizarUsuario(string id, string nombre, string email, string password, string rol, string
      pais, string imagenUrl, DateTime fecha, bool esPrivada)
01161         {
01162             if (!await Conectar())
01163             {
01164                 return false;
```

```
01165           }
01166           try
01167           {
01168               var builder = Builders<Usuarios>.Update;
01169               var listaCambios = new List<UpdateDefinition<Usuarios»();
01170
01171               listaCambios.Add(builder.Set(u => u.Username, nombre));
01172               listaCambios.Add(builder.Set(u => u.Email, email));
01173               listaCambios.Add(builder.Set(u => u.Password, password));
01174               listaCambios.Add(builder.Set(u => u.Rol, rol));
01175               listaCambios.Add(builder.Set(u => u.Perfil.Pais, pais));
01176               listaCambios.Add(builder.Set(u => u.Perfil.ImagenUrl, imagenUrl));
01177               listaCambios.Add(builder.Set(u => u.Perfil.FechaNacimiento, fecha));
01178               listaCambios.Add(builder.Set(u => u.Perfil.EsPrivada, esPrivada));
01179
01180               var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, id);
01181
01182               var updateFinal = builder.Combine(listaCambios);
01183
01184               var resultado = await Database.GetCollection<Usuarios>("usuarios").UpdateOneAsync(filtro, updateFinal);
01185
01186               var EsActualizado = resultado.MatchedCount > 0;
01187
01188               return EsActualizado;
01189           }
01190           catch (Exception ex)
01191           {
01192               System.Diagnostics.Debug.WriteLine("Error actualizando usuario desde admin: " + ex.Message);
01193               return false;
01194           }
01195       }
01196
01197       public async Task<bool> ActualizarConfiguracionUsuario(string idUsuario, ConfiguracionUser nuevaConfig)
01198       {
01199           // Preparamos el constructor de actualizaciones
01200           var builder = Builders<Usuarios>.Update;
01201           var listaCambios = new List<UpdateDefinition<Usuarios»();
01202
01203           if (nuevaConfig.DiccionarioTema != GlobalData.Instance.DiccionarioTemaGD)
01204           {
01205               listaCambios.Add(builder.Set(u => u.Configuracion.DiccionarioTema, nuevaConfig.DiccionarioTema));
01206           }
01207           if (nuevaConfig.DiccionarioIdioma != GlobalData.Instance.DiccionarioIdiomaGD)
01208           {
01209               listaCambios.Add(builder.Set(u => u.Configuracion.DiccionarioIdioma, nuevaConfig.DiccionarioIdioma));
01210           }
01211           if (nuevaConfig.DiccionarioFuente != GlobalData.Instance.DiccionarioFuenteGD)
01212           {
01213               listaCambios.Add(builder.Set(u => u.Configuracion.DiccionarioFuente, nuevaConfig.DiccionarioFuente));
01214           }
01215
01216           if (listaCambios.Count == 0)
01217           {
01218               return true;
01219           }
01220
01221           if (!await Conectar())
01222           {
01223               return false;
01224           }
01225
01226           try
01227           {
01228               var updateFinal = builder.Combine(listaCambios);
01229
01230               var resultado = await Database.GetCollection<Usuarios>("usuarios")
01231                                       .UpdateOneAsync(u => u.Id == idUsuario, updateFinal);
01232
01233               return resultado.MatchedCount > 0;
01234           }
01235           catch (Exception ex)
01236           {
01237               System.Diagnostics.Debug.WriteLine("Error actualizando config: " + ex.Message);
01238               return false;
01239           }
01240       }
01241       public async Task<bool> ActualizarPlaylist(string nuevoNombre, string nuevaDesc, List<string> nuevasCanciones,
    string nuevaPortada, ListaPersonalizada original)
01242       {
01243           try
01244           {
01245               //Preparamos el contructor y una lista con para guardar los cambios
01246               var builder = Builders<ListaPersonalizada>.Update;
01247               var listaCambios = new List<UpdateDefinition<ListaPersonalizada»();
01248
01249               // Comparamos lo que han cambiado
01250
```

```
01251               if (nuevoNombre != original.Nombre)
01252               {
01253                   listaCambios.Add(builder.Set(p => p.Nombre, nuevoNombre));
01254               }
01255
01256               if (nuevaDesc != original.Descripcion)
01257               {
01258                   listaCambios.Add(builder.Set(p => p.Descripcion, nuevaDesc));
01259               }
01260
01261               if (nuevaPortada != original.UrlPortada)
01262               {
01263                   listaCambios.Add(builder.Set(p => p.UrlPortada, nuevaPortada));
01264               }
01265
01266               // SequenceEqual comprueba si tienen los mismos elementos en el mismo orden.
01267               if (nuevasCanciones != null && !nuevasCanciones.SequenceEqual(original.IdsCanciones ?? new List<string>()))
01268               {
01269                   listaCambios.Add(builder.Set(p => p.IdsCanciones, nuevasCanciones));
01270               }
01271
01272               // Si no hay cambios no llamamos al método
01273               if (listaCambios.Count == 0)
01274               {
01275                   return true;
01276               }
01277
01278               // Ejecutamos actualización
01279               if (!await Conectar())
01280               {
01281                   return false;
01282               }
01283
01284               var filtro = Builders<ListaPersonalizada>.Filter.Eq(p => p.Id, original.Id);
01285               var updateFinal = builder.Combine(listaCambios);
01286
01287               var resultado = await
      Database.GetCollection<ListaPersonalizada>("listapersonalizada").UpdateOneAsync(filtro, updateFinal);
01288
01289               return resultado.MatchedCount > 0;
01290           }
01291           catch (Exception ex)
01292           {
01293               System.Diagnostics.Debug.WriteLine("[ERROR MONGO] Al actualizar playlist: " + ex.Message);
01294               return false;
01295           }
01296       }
01297       public async Task<bool> ActualizarCancion(string nuevoTitulo, string nuevaPortada, List<string> nuevosAutores,
      List<string> nuevosGeneros, Canciones original)
01298       {
01299           try
01300           {
01301               var builder = Builders<Canciones>.Update;
01302               var listaCambios = new List<UpdateDefinition<Canciones»();
01303
01304               // Comparamos lo que han cambiado
01305
01306               if (nuevoTitulo != original.Titulo)
01307               {
01308                   listaCambios.Add(builder.Set(c => c.Titulo, nuevoTitulo));
01309               }
01310
01311               if (nuevaPortada != original.ImagenPortadaUrl)
01312               {
01313                   listaCambios.Add(builder.Set(c => c.ImagenPortadaUrl, nuevaPortada));
01314               }
01315
01316               // Usamos SequenceEqual para ver si la lista de IDs es idéntica
01317               if (nuevosAutores != null && !nuevosAutores.SequenceEqual(original.AutoresIds))
01318               {
01319                   listaCambios.Add(builder.Set(c => c.AutoresIds, nuevosAutores));
01320               }
01321
01322               var generosOriginales = original.Datos?.Generos ?? new List<string>();
01323
01324               if (nuevosGeneros != null && !nuevosGeneros.SequenceEqual(generosOriginales))
01325               {
01326                   listaCambios.Add(builder.Set(c => c.Datos.Generos, nuevosGeneros));
01327               }
01328
01329               // Si no hay cambios salimos del método
01330               if (listaCambios.Count == 0)
01331               {
01332                   return true;
01333               }
01334
01335               // Ejecutamos actualización
```

```
01336             if (!await Conectar())
01337             {
01338                 return false;
01339             }
01340 
01341             var filtro = Builders<Canciones>.Filter.Eq(c => c.Id, original.Id);
01342 
01343             var updateFinal = builder.Combine(listaCambios);
01344 
01345             var resultado = await Database.GetCollection<Canciones>("canciones")
01346                                           .UpdateOneAsync(filtro, updateFinal);
01347 
01348             var EsActualizado = resultado.MatchedCount > 0;
01349 
01350             return EsActualizado;
01351         }
01352         catch (Exception ex)
01353         {
01354             System.Diagnostics.Debug.WriteLine("[ERROR MONGO] ActualizarCancion: " + ex.Message);
01355             return false;
01356         }
01357     }
01358 
01359     public async Task<bool> ActualizarEstadoReporte(string nuevoEstado, string nuevaResolucion, Reportes original)
01360     {
01361         if (!await Conectar())
01362         {
01363             return true;
01364         }
01365         try
01366         {
01367             var builder = Builders<Reportes>.Update;
01368             var listaCambios = new List<UpdateDefinition<Reportes»();
01369 
01370             // -- Estado --
01371             if (nuevoEstado != original.Estado)
01372             {
01373                 listaCambios.Add(builder.Set(r => r.Estado, nuevoEstado));
01374             }
01375 
01376             // -- Resolución (Notas del admin) --
01377 
01378             if ((nuevaResolucion ?? "") != (original.Resolucion ?? ""))
01379             {
01380                 listaCambios.Add(builder.Set(r => r.Resolucion, nuevaResolucion));
01381             }
01382 
01383             // Si no hay cambios salimos del método
01384             if (listaCambios.Count == 0)
01385             {
01386                 return true;
01387             }
01388 
01389             // Ejecutamos actualización
01390             if (!await Conectar())
01391             {
01392                 return false;
01393             }
01394             var coleccion = Database.GetCollection<Reportes>("reportes");
01395 
01396             var filtro = Builders<Reportes>.Filter.Eq(r => r.Id, original.Id);
01397 
01398             var updateFinal = builder.Combine(listaCambios);
01399 
01400             var resultado = await coleccion.UpdateOneAsync(filtro, updateFinal);
01401 
01402             var EsActualizado = resultado.MatchedCount > 0;
01403 
01404             return EsActualizado;
01405         }
01406         catch
01407         {
01408             return false;
01409         }
01410     }
01411 
01412     public async Task IncrementarMetricaCancion(string idCancion, string campo, int cantidad)
01413     {
01414         if (!await Conectar())
01415         {
01416             return;
01417         }
01418 
01419         try
01420         {
01421             var coleccion = Database.GetCollection<Canciones>("canciones");
01422             var filtro = Builders<Canciones>.Filter.Eq(c => c.Id, idCancion);
```

```
01423
01424             // Usamos Inc (Increment) que es atómico y eficiente
01425             var update = Builders<Canciones>.Update.Inc(campo, cantidad);
01426
01427             await coleccion.UpdateOneAsync(filtro, update);
01428
01429             _ = ActualizarTendencia(idCancion);
01430
01431
01432         }
01433         catch (Exception ex)
01434         {
01435             System.Diagnostics.Debug.WriteLine($"Error actualizando métrica {campo}: {ex.Message}");
01436         }
01437     }
01438
01439     private async Task ActualizarTendencia(string idCancion)
01440     {
01441         try
01442         {
01443             var coleccion = Database.GetCollection<Canciones>("canciones");
01444             var cancion = await coleccion.Find(c => c.Id == idCancion).FirstOrDefaultAsync();
01445
01446             if (cancion == null) return;
01447
01448             long visitas = cancion.Metricas.TotalReproducciones;
01449             long likes = cancion.Metricas.TotalMegustas;
01450
01451             DateTime fechaLanzamiento = cancion.Datos.FechaLanzamiento == DateTime.MinValue
01452                                         ? DateTime.Now
01453                                         : cancion.Datos.FechaLanzamiento;
01454
01455             // --- FÓRMULA DE GRAVEDAD / TENDENCIA ---
01456
01457             //Calculamos los dias de vida de la canción
01458             double diasDeVida = (DateTime.Now - fechaLanzamiento).TotalDays;
01459
01460             //Por si la fecha de lanzamiento esta mal puesta
01461             if (diasDeVida < 0) diasDeVida = 0;
01462
01463             // El Cálculo: (Popularidad) / (Tiempo)^Gravedad
01464             // - Los Likes valen el DOBLE que una visita normal.
01465             // - Sumamos +1 a los días para evitar dividir por cero el primer día.
01466             // - Elevamos a 1.4 para que la puntuación baje rápido con el tiempo (necesita muchas visitas para mantenerse).
01467             double rawScore = (visitas + (likes * 2)) / Math.Pow(diasDeVida + 1, 1.4);
01468
01469             // NORMALIZACIÓN 0 - 100 (Escala Logarítmica)
01470             // - Math.Max(1, ...) asegura que el logaritmo nunca sea negativo o error
01471             // - Multiplicamos por 18: Con aprox 350.000 de rawScore llegas al 100.
01472             double scoreFinal = Math.Log10(Math.Max(1, rawScore)) * 18;
01473
01474             // Si pasa de 100, se queda en 100
01475             if (scoreFinal > 100) scoreFinal = 100;
01476
01477             // REDONDEO (2 decimales)
01478             scoreFinal = Math.Round(scoreFinal, 2);
01479
01480             // Actualizamos Mongo
01481             var update = Builders<Canciones>.Update.Set(c => c.Metricas.PuntuacionTendencia, scoreFinal);
01482             await coleccion.UpdateOneAsync(c => c.Id == idCancion, update);
01483         }
01484         catch (Exception ex)
01485         {
01486             System.Diagnostics.Debug.WriteLine("Error recalcular tendencia: " + ex.Message);
01487         }
01488     }
01489
01490     public async Task IncrementarContadorCancionesUsuario(string idUsuario, int cantidad)
01491     {
01492         if (!await Conectar())
01493         {
01494             return;
01495         }
01496
01497         try
01498         {
01499             var coleccion = Database.GetCollection<Usuarios>("usuarios");
01500             var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idUsuario);
01501
01502             // Si es 1, suma. Si es -1, resta.
01503             var update = Builders<Usuarios>.Update.Inc("estadisticas.n_canciones_subidas", cantidad);
01504
01505             await coleccion.UpdateOneAsync(filtro, update);
01506         }
01507         catch (Exception ex)
01508         {
01509             System.Diagnostics.Debug.WriteLine("Error actualizando contador de usuario: " + ex.Message);
```

```
01510         }
01511     }
01512     public async Task<bool> ActualizarGenero(string id, string nuevoNombre)
01513     {
01514         if (!await Conectar())
01515         {
01516             return false;
01517         }
01518         try
01519         {
01520             var coleccion = Database.GetCollection<Generos>("generos");
01521 
01522             // VALIDACIÓN: ¿Existe OTRO género con ese nombre?
01523             // Buscamos: (Nombre IGUAL al nuevo) Y (Id DIFERENTE al mío)
01524             var filtroDuplicado = Builders<Generos>.Filter.And(
01525                 Builders<Generos>.Filter.Regex(g => g.Nombre, new
    MongoDB.Bson.BsonRegularExpression($"^{nuevoNombre}$", "i")),
01526                 Builders<Generos>.Filter.Ne(g => g.Id, id) // Excluye mi id
01527             );
01528 
01529             var existeOtro = await coleccion.Find(filtroDuplicado).AnyAsync();
01530             if (existeOtro)
01531             {
01532                 return false; // Nombre ocupado por otro.
01533             }
01534 
01535             // Actualizamos
01536             var filtro = Builders<Generos>.Filter.Eq(g => g.Id, id);
01537             var update = Builders<Generos>.Update.Set(g => g.Nombre, nuevoNombre);
01538 
01539 
01540             var resultado = await coleccion.UpdateOneAsync(filtro, update);
01541 
01542             var EsActualizado = resultado.MatchedCount > 0;
01543 
01544             return EsActualizado;
01545         }
01546         catch (Exception ex)
01547         {
01548             System.Diagnostics.Debug.WriteLine("Error actualizando género: " + ex.Message);
01549             return false;
01550         }
01551     }
01552     #endregion
01553 
01554     #region Metodos Helpers
01555     ////////////////////////////////////
01556     ////////////METODOS HELPERS////////////
01557     ////////////////////////////////////
01558     private async Task RellenarNombresDeArtistas(List<Canciones> listaCanciones)
01559     {
01560         // Necesitamos acceso a la colección de usuarios para buscar los nombres
01561         var coleccionUsuarios = Database.GetCollection<Usuarios>("usuarios");
01562 
01563         foreach (var cancion in listaCanciones)
01564         {
01565             // Solo entramos si hay autores en la lista de IDs
01566             if (cancion.AutoresIds != null && cancion.AutoresIds.Count > 0)
01567             {
01568                 List<string> autoresUsername = new List<string>();
01569 
01570                 foreach (var idAutor in cancion.AutoresIds)
01571                 {
01572                     var filtro = Builders<Usuarios>.Filter.Eq(u => u.Id, idAutor);
01573 
01574                     // LA PROYECCIÓN
01575                     // En lugar de traer todo, hacemos: u => new Usuarios { ... }
01576                     // Esto crea un objeto Usuarios "vacío" y rellena SOLO el Username.
01577                     // El resto de campos (Email, Password...) serán null.
01578                     var usuario = await coleccionUsuarios.Find(filtro)
01579                                         .Project(u => new Usuarios { Username = u.Username })
01580                                         .FirstOrDefaultAsync();
01581 
01582                     if (usuario != null)
01583                     {
01584                         autoresUsername.Add(usuario.Username);
01585                     }
01586                 }
01587 
01588                 // Unimos los nombres (Ej: "Fito, Estopa")
01589                 cancion.NombreArtista = string.Join(", ", autoresUsername);
01590             }
01591             else
01592             {
01593                 cancion.NombreArtista = "Artista Desconocido";
01594             }
01595         }
```

```
01596         }
01597         #endregion
01598
01599     }
01600 }
```

4.77. Referencia del archivo BetaProyecto/Services/StorageService.cs

4.78. StorageService.cs

Ir a la documentación de este archivo.

```
00001 using Newtonsoft.Json.Linq;
00002 using System;
00003 using System.IO;
00004 using System.Net.Http;
00005 using System.Net.Http.Headers;
00006 using System.Threading.Tasks;
00007
00008 namespace BetaProyecto.Services
00009 {
00010     public class StorageService
00011     {
00012         // Url de nuestra API
00013         private const string ApiUrl = "https://localhost:7500/api/Storage";
00014
00015
00016         /// <summary>
00017         /// Carga un archivo de imagen desde el sistema de archivos local hacia el servidor de almacenamiento en la nube.
00018         /// </summary>
00019         /// <remarks>
00020         /// Este método actúa como un envoltorio especializado que invoca la lógica genérica de comunicación con la API
00021         /// utilizando el endpoint específico para el procesamiento de imágenes. Es ideal para la gestión de
00022         /// avatares de usuario, portadas de álbumes y miniaturas de playlists.
00023         /// </remarks>
00024         /// <param name="rutaArchivoEnTuPc">La ruta absoluta del archivo de imagen en el almacenamiento
    local.</param>
00025         /// <returns>
00026         /// Una tarea que representa la operación asíncrona. El valor devuelto contiene la URL pública
00027         /// generada por el servicio de almacenamiento (Cloudinary) si la carga fue exitosa.
00028         /// </returns>
00029         public async Task<string> SubirImagen(string rutaArchivoEnTuPc)
00030         {
00031             return await EnviarA_Api(rutaArchivoEnTuPc, "subir-imagen");
00032         }
00033
00034         /// <summary>
00035         /// Carga un archivo de audio desde el almacenamiento local hacia el servidor de distribución en la nube.
00036         /// </summary>
00037         /// <remarks>
00038         /// Este método encapsula la lógica de transferencia de archivos multimedia utilizando el endpoint dedicado
00039         /// para audio. El proceso sigue el siguiente flujo:
00040         /// <list type="number">
00041         /// <item><b>Empaquetado:</b> Invoca al método core <c>EnviarA_Api</c> para convertir el archivo físico
    en un flujo de datos (Stream).</item>
00042         /// <item><b>Transmisión:</b> Envía el recurso mediante una petición POST multipart/form-data al
    microservicio de almacenamiento.</item>
00043         /// <item><b>Respuesta:</b> Retorna la URL segura (HTTPS) proporcionada por Cloudinary, necesaria para la
    persistencia en la base de datos.</item>
00044         /// </list>
00045         /// </remarks>
00046         /// <param name="rutaArchivoEnTuPc">La ruta absoluta del archivo de audio (ej. .mp3, .wav) en el disco
    local.</param>
00047         /// <returns>
00048         /// Una tarea que contiene la URL pública del archivo alojado si la operación tiene éxito;
00049         /// de lo contrario, devuelve una cadena vacía o nula.
00050         /// </returns>
00051         public async Task<string> SubirCancion(string rutaArchivoEnTuPc)
00052         {
00053             return await EnviarA_Api(rutaArchivoEnTuPc, "subir-audio");
00054         }
00055
00056         // --- MÉTODO PARA ELIMINAR ---
00057         /// <summary>
00058         /// Solicita de forma asíncrona la eliminación de un recurso almacenado en la nube a través de la API de
    almacenamiento.
00059         /// </summary>
00060         /// <remarks>
00061         /// Este método gestiona la baja de archivos (imágenes o audio) siguiendo este flujo:
00062         /// <list type="number">
00063         /// <item><b>Validación:</b> Verifica que la URL proporcionada no sea nula o vacía.</item>
```

```
00064       /// <item><b>Seguridad:</b> Configura un <see cref="HttpClientHandler"/> para omitir la validación de certificados SSL, permitiendo peticiones a entornos <c>localhost</c> con certificados autofirmados.</item>
00065       /// <item><b>Petición:</b> Ejecuta un verbo HTTP <c>DELETE</c> enviando la URL del archivo como parámetro de consulta.</item>
00066       /// <item><b>Confirmación:</b> Evalúa la respuesta de la API para confirmar si el archivo fue removido con éxito del proveedor externo (Cloudinary).</item>
00067       /// </list>
00068       /// </remarks>
00069       /// <param name="urlCompleta">La dirección URL absoluta del recurso que se desea eliminar del almacenamiento remoto.</param>
00070       /// <returns>
00071       /// Una tarea que devuelve <c>true</c> si el servidor confirma la eliminación (Success Status Code);
00072       /// de lo contrario, <c>false</c> si el recurso no existe o hubo un fallo en la comunicación.
00073       /// </returns>
00074       public async Task<bool> EliminarArchivo(string urlCompleta)
00075       {
00076           // Validación básica: Si no hay URL, no hay nada que borrar.
00077           if (string.IsNullOrEmpty(urlCompleta)) return false;
00078
00079           try
00080           {
00081               // Configuración para aceptar certificados locales
00082               var handler = new HttpClientHandler();
00083               handler.ServerCertificateCustomValidationCallback = (message, cert, chain, errors) => true;
00084
00085               using (var client = new HttpClient(handler))
00086               {
00087                   // LA LLAMADA (DELETE)
00088                   // Construimos la URL así: https://localhost:7500/api/Storage/eliminar?url=https://
00089                   var response = await client.DeleteAsync($"{ApiUrl}/eliminar?url={urlCompleta}");
00090
00091                   // RESULTADO
00092                   // Devuelve true si la API respondió 200 OK (Borrado exitoso)
00093                   // Devuelve false si falló (400, 500, etc.)
00094                   return response.IsSuccessStatusCode;
00095               }
00096           }
00097           catch (Exception ex)
00098           {
00099               // Si hay error de conexión, solo lo registramos y devolvemos false
00100               System.Diagnostics.Debug.WriteLine("[StorageService] Error al eliminar: " + ex.Message);
00101               return false;
00102           }
00103       }
00104
00105       // Motor interno para enviar archivos a la API, reutilizado por ambos métodos públicos (SubirImagen y SubirCancion).
00106       /// <summary>
00107       /// Realiza la carga física de un archivo hacia la API de almacenamiento mediante una petición POST de tipo multipart/form-data.
00108       /// </summary>
00109       /// <remarks>
00110       /// Este método núcleo (core) orquestra la transferencia de archivos multimedia siguiendo este flujo:
00111       /// <list type="number">
00112       /// <item><b>Verificación local:</b> Valida la existencia del archivo en el sistema de archivos del cliente antes de iniciar la conexión.</item>
00113       /// <item><b>Configuración de Seguridad:</b> Implementa un bypass de validación SSL para permitir el desarrollo en entornos locales con certificados no firmados.</item>
00114       /// <item><b>Empaquetado (Multipart):</b> Abre el archivo como un flujo de datos (<see cref="StreamContent"/>) y lo encapsula en un contenedor compatible con formularios web.</item>
00115       /// <item><b>Transmisión y Respuesta:</b> Ejecuta la petición asíncrona hacia el <paramref name="endpoint"/> especificado y procesa la respuesta JSON para extraer la URL persistente generada por el servidor.</item>
00116       /// </list>
00117       /// </remarks>
00118       /// <param name="ruta">La ruta absoluta del archivo en el disco duro local.</param>
00119       /// <param name="endpoint">El segmento final de la URL de la API (ej. "subir-imagen" o "subir-audio").</param>
00120       /// <returns>La URL pública del archivo cargado en el servidor de almacenamiento.</returns>
00121       /// <exception cref="FileNotFoundException">Se lanza si la ruta especificada no apunta a un archivo válido.</exception>
00122       /// <exception cref="Exception">Encapsula errores de conectividad, rechazos de la API (códigos 4xx o 5xx) o fallos en el análisis del JSON.</exception>
00123       private async Task<string> EnviarA__Api(string ruta, string endpoint)
00124       {
00125           // Comprobamos si existe el archivo
00126           if (!File.Exists(ruta)) throw new FileNotFoundException("¡No encuentro el archivo en tu PC!");
00127
00128           try
00129           {
00130               // Como estamos en desarrollo, el certificado de seguridad de 'localhost' no es oficial.
00131               // Esta línea le dice al código: "Confía en el servidor".
00132               var handler = new HttpClientHandler();
00133               handler.ServerCertificateCustomValidationCallback = (message, cert, chain, errors) => true;
00134
00135               // Creamos el cliente
00136               using (var client = new HttpClient(handler))
```

```
00137             {
00138                 // Preparamos el paquete
00139                 using (var content = new MultipartFormDataContent())
00140                 {
00141                     // Abrimos el archivo de tu disco duro para leerlo.
00142                     var fileStream = File.OpenRead(ruta);
00143 
00144                     // Convertimos el archivo en un flujo de datos (StreamContent) para enviarlo.
00145                     var streamContent = new StreamContent(fileStream);
00146 
00147                     // Le ponemos una etiqueta para decirle que un archivo de datos
00148                     streamContent.Headers.ContentType = MediaTypeHeaderValue.Parse("multipart/form-data");
00149 
00150                     // Metemos el archivo en el sobre.
00151                     // "archivo" -> Es el nombre EXACTO que espera nuestra API
00152                     content.Add(streamContent, "archivo", Path.GetFileName(ruta));
00153 
00154                     // El envío (POST)
00155                     // Este sale hacia: https://localhost:7500/api/Storage/subir-imagen (o audio)
00156                     var response = await client.PostAsync($"{ApiUrl}/{endpoint}", content);
00157 
00158                     //Comprueba si fue bien. Si falla, se lanza HttpRequestException.
00159                     response.EnsureSuccessStatusCode();
00160 
00161                     // Si llegamos aquí, es que todo fue bien (Código 200)
00162                     var jsonString = await response.Content.ReadAsStringAsync();
00163                     var json = JObject.Parse(jsonString);
00164                     return json["url"].ToString();
00165                 }
00166             }
00167         }
00168         catch (HttpRequestException httpEx)
00169         {
00170             // Aquí atrapamos errores específicos de la web (404, 500...)
00171             throw new Exception($"La API rechazó el archivo: {httpEx.Message}");
00172         }
00173         catch (Exception ex)
00174         {
00175             // Si explota la conexión (servidor apagado, sin internet, etc.)
00176             throw new Exception($"Error conectando con la API: {ex.Message}");
00177         }
00178     }
00179 }
00180 }
```

4.79. Referencia del archivo BetaProyecto/Singleton/GlobalData.cs

4.80. GlobalData.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using System.Collections.Generic;
00003 using BetaProyecto.Models;
00004 
00005 namespace BetaProyecto.Singleton
00006 {
00007     public class GlobalData
00008     {
00009         private static GlobalData _instance;
00010         public static GlobalData Instance => _instance ??= new GlobalData();
00011 
00012         // Variables globales
00013         public string UserIdGD { get; set; }
00014         public string UsernameGD { get; set; }
00015         public string EmailGD { get; set; }
00016         public string PasswordGD { get; set; }
00017         public string RolGD { get; set; }
00018         public string UrlFotoPerfilGD { get; set; }
00019         public DateTime FechaNacimientoGD { get; set; }
00020         public bool Es_PrivadaGD { get; set; }
00021         public string PaisGD { get; set; }
00022         public int Num_canciones_subidasGD { get; set; }
00023         public List<string> SeguidoresGD { get; set; }
00024         public List<string> FavoritosGD { get; set; }
00025         public string DiccionarioTemaGD { get; set; }
00026         public string DiccionarioIdiomaGD { get; set; }
00027         public string DiccionarioFuenteGD { get; set; }
00028         public DateTime Fecha_registroGD { get; set; }
00029 
00030         // Método para cargar datos del usuario en las variables globales
```

```
00031         /// <summary>
00032         /// Sincroniza y mapea la información completa de un objeto <see cref="Usuarios"/> hacia las propiedades globales de la sesión actual.
00033         /// </summary>
00034         /// <remarks>
00035         /// Este método actúa como un adaptador que distribuye los datos del usuario autenticado en diferentes categorías:
00036         /// <list type="bullet">
00037         /// <item><b>Datos de Identidad:</b> Mapea ID, nombre, correo y rol directamente desde la raíz del objeto.</item>
00038         /// <item><b>Perfil y Preferencias:</b> Extrae información geográfica, imagen de perfil y estado de privacidad.</item>
00039         /// <item><b>Actividad y Social:</b> Inicializa contadores de estadísticas y asegura que las listas de seguidores y favoritos no sean nulas.</item>
00040         /// <item><b>Configuración de Entorno:</b> Carga los diccionarios de tema, idioma y fuente, aplicando valores por defecto si no existen preferencias guardadas.</item>
00041         /// </list>
00042         /// Se utiliza principalmente durante el inicio de sesión o tras una actualización exitosa del perfil del usuario para mantener la consistencia en toda la aplicación.
00043         /// </remarks>
00044         /// <param name="user">El objeto <see cref="Usuarios"/> recuperado de la base de datos que contiene la información maestra.</param>
00045         public void SetUserData(Usuarios user)
00046         {
00047             if (user != null)
00048             {
00049                 // Datos que están en la raíz
00050                 this.UserIdGD = user.Id;
00051                 this.UsernameGD = user.Username;
00052                 this.EmailGD = user.Email;
00053                 this.PasswordGD = user.Password;
00054                 this.RolGD = user.Rol;
00055                 this.Fecha_registroGD = user.FechaRegistro;
00056 
00057                 // Datos dentro de "Perfil"
00058                 if (user.Perfil != null)
00059                 {
00060                     this.UrlFotoPerfilGD = user.Perfil.ImagenUrl;
00061                     this.FechaNacimientoGD = user.Perfil.FechaNacimiento;
00062                     this.Es_PrivadaGD = user.Perfil.EsPrivada;
00063                     this.PaisGD = user.Perfil.Pais;
00064                 }
00065 
00066                 // Datos dentro de "Estadisticas"
00067                 if (user.Estadisticas != null)
00068                 {
00069                     this.Num_canciones_subidasGD = user.Estadisticas.NumCancionesSubidas;
00070                 }
00071                 else
00072                 {
00073                     this.Num_canciones_subidasGD = 0;
00074                 }
00075 
00076                 // Datos dentro de "Listas"
00077                 if (user.Listas != null)
00078                 {
00079                     this.SeguidoresGD = user.Listas.Seguidores ?? new List<string>(); ;
00080                     this.FavoritosGD = user.Listas.Favoritos ?? new List<string>(); ;
00081                 }
00082                 else
00083                 {
00084                     // Inicializamos listas vacías para evitar errores luego
00085                     this.SeguidoresGD = new List<string>();
00086                     this.FavoritosGD = new List<string>();
00087                 }
00088                 if (user.Configuracion != null)
00089                 {
00090                     this.DiccionarioTemaGD = user.Configuracion.DiccionarioTema;
00091                     this.DiccionarioIdiomaGD = user.Configuracion.DiccionarioIdioma;
00092                     this.DiccionarioFuenteGD = user.Configuracion.DiccionarioFuente;
00093                 }
00094                 else
00095                 {
00096                     // Valores por defecto para evitar errores
00097                     this.DiccionarioTemaGD = "ModoClaro";
00098                     this.DiccionarioIdiomaGD = "Spanish";
00099                     this.DiccionarioFuenteGD = "Lexend";
00100                 }
00101             }
00102         }
00103 
00104 
00105         // Para limpiar los datos
00106         public void ClearUserData()
00107         {
00108             this.UserIdGD = string.Empty;
00109             this.UsernameGD = string.Empty;
00110             this.EmailGD = string.Empty;
```

```
00111             this.PasswordGD = string.Empty;
00112             this.RolGD = string.Empty;
00113             this.UrlFotoPerfilGD = string.Empty;
00114             this.FechaNacimientoGD = DateTime.MinValue;
00115             this.Es_PrivadaGD = false;
00116             this.PaisGD = string.Empty;
00117             this.Num_canciones_subidasGD = 0;
00118             this.SeguidoresGD = new List<string>();
00119             this.FavoritosGD = new List<string>();
00120             this.DiccionarioTemaGD = string.Empty;
00121             this.DiccionarioIdiomaGD = string.Empty;
00122             this.DiccionarioFuenteGD = string.Empty;
00123         }
00124         // Para generar un objeto Usuarios completo a partir de las variables globales
00125         /// <summary>
00126         /// Reconstruye y devuelve un objeto de tipo <see cref="Usuarios"/> integrando todas las propiedades almacenadas
    en la sesión global.
00127         /// </summary>
00128         /// <remarks>
00129         /// Este método realiza una operación de ensamblado para convertir las propiedades planas de <see
    cref="GlobalData"/> en una estructura jerárquica compleja.
00130         /// Es fundamental para operaciones de persistencia, permitiendo que otros servicios (como el cliente de base de
    datos) reciban una entidad completa
00131         /// con sus objetos anidados de <see cref="PerfilUsuario"/>, <see cref="EstadisticasUsuario"/>, <see
    cref="ListasUsuario"/> y <see cref="ConfiguracionUser"/>.
00132         /// </remarks>
00133         /// <returns>
00134         /// Una nueva instancia de <see cref="Usuarios"/> que refleja el estado actual de la sesión del usuario,
00135         /// incluyendo sus preferencias de configuración y listas sociales.
00136         /// </returns>
00137         public Usuarios GetUsuarioObject()
00138         {
00139             // Reconstruimos el objeto completo
00140             var usuarioCompleto = new Usuarios
00141             {
00142                 Id = this.UserIdGD,
00143                 Username = this.UsernameGD,
00144                 Email = this.EmailGD,
00145                 Password = this.PasswordGD,
00146                 Rol = this.RolGD,
00147                 FechaRegistro = this.Fecha_registroGD,
00148 
00149                 // Reconstruimos el Perfil
00150                 Perfil = new PerfilUsuario
00151                 {
00152                     ImagenUrl = this.UrlFotoPerfilGD,
00153                     FechaNacimiento = this.FechaNacimientoGD,
00154                     EsPrivada = this.Es_PrivadaGD,
00155                     Pais = this.PaisGD
00156                 },
00157 
00158                 // Reconstruimos Estadísticas
00159                 Estadisticas = new EstadisticasUsuario
00160                 {
00161                     NumCancionesSubidas = this.Num_canciones_subidasGD
00162                 },
00163 
00164                 // Reconstruimos Listas
00165                 Listas = new ListasUsuario
00166                 {
00167                     Seguidores = this.SeguidoresGD ?? new List<string>(),
00168                     Favoritos = this.FavoritosGD ?? new List<string>()
00169                 },
00170                 Configuracion = new ConfiguracionUser{
00171                     DiccionarioTema = this.DiccionarioTemaGD ?? "ModoClaro",
00172                     DiccionarioIdioma = this.DiccionarioIdiomaGD ?? "Spanish",
00173                     DiccionarioFuente = this.DiccionarioFuenteGD ?? "Lexend"
00174                 }
00175             };
00176 
00177             return usuarioCompleto;
00178         }
00179         // Constructor
00180         private GlobalData() { }
00181     }
00182 }
```

4.81. Referencia del archivo BetaProyecto/Singleton/MongoClientSingleton.cs

4.82. MongoClientSingleton.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Services;
00002 using System;
00003 using System.Collections.Generic;
00004 using System.Linq;
00005 using System.Text;
00006 using System.Threading.Tasks;
00007
00008 namespace BetaProyecto.Singleton
00009 {
00010     public class MongoClientSingleton
00011     {
00012         //Singleton
00013         private static MongoClientSingleton _instance;
00014         public static MongoClientSingleton Instance => _instance ??= new MongoClientSingleton();
00015
00016         // Objeto guardado
00017         public MongoAtlas Cliente { get; private set; }
00018
00019         // Constructor
00020         private MongoClientSingleton(){
00021
00022             Cliente = new MongoAtlas();
00023         }
00024     }
00025 }
```

4.83. Referencia del archivo BetaProyecto/ViewLocator.cs

4.84. ViewLocator.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002 using Avalonia.Controls;
00003 using Avalonia.Controls.Templates;
00004 using BetaProyecto.ViewModels;
00005
00006 namespace BetaProyecto
00007 {
00008     public class ViewLocator : IDataTemplate
00009     {
00010
00011         public Control? Build(object? param)
00012         {
00013             if (param is null)
00014                 return null;
00015
00016             var name = param.GetType().FullName!.Replace("ViewModel", "View", StringComparison.Ordinal);
00017             var type = Type.GetType(name);
00018
00019             if (type != null)
00020             {
00021                 return (Control)Activator.CreateInstance(type)!;
00022             }
00023
00024             return new TextBlock { Text = "Not Found: " + name };
00025         }
00026
00027         public bool Match(object? data)
00028         {
00029             return data is ViewModelBase;
00030         }
00031     }
00032 }
```

4.85. Referencia del archivo BetaProyecto/ViewModels/CentralTabControlViewModel.cs

4.86. CentralTabControlViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Singleton;
```

```
00003 using ReactiveUI;
00004 using System;
00005 using System.Collections.Generic;
00006 using System.Diagnostics;
00007 using System.Reactive;
00008
00009 namespace BetaProyecto.ViewModels
00010 {
00011     public class CentralTabControlViewModel : ViewModelBase
00012     {
00013         // Sub-ViewModels
00014         public TabItemInicioViewModel InicioVM { get; }
00015         public TabItemBuscadorViewModel BuscadorVM { get; }
00016         public TabItemPopularesViewModel PopularesVM { get; }
00017
00018         // Actions globales (Navegación y Reproductor)
00019         public Action? IrAPerfil { get; set; }
00020         public Action? IrACuenta { get; set; }
00021         public Action? IrAGestionarCuenta { get; set; }
00022         public Action? IrAConfig { get; set; }
00023         public Action? IrASobreNosotros { get; set; }
00024         public Action? IrAAyuda { get; set; }
00025         public Action? IrAPublicarCancion { get; set; }
00026         public Action? IrACrearPlaylist { get; set; }
00027         public Action<Canciones>? IrADetallesCancion { get; set; }
00028         public Action<string>? IrAVerArtista { get; set; }
00029         public Action<Canciones>? IrACrearReporte {  get; set; }
00030         public Action<ListaPersonalizada>? IrADetallesPlaylist { get; set; }
00031         public Action<Canciones, List<Canciones» SolicitudCancion { get; set; }
00032
00033         //Comandos reactive
00034         public ReactiveCommand<Unit, Unit> BtnPerfil { get; }
00035         public ReactiveCommand<Unit, Unit> BtnCuenta { get; }
00036         public ReactiveCommand<Unit, Unit> BtnGestionarCuenta { get; }
00037         public ReactiveCommand<Unit, Unit> BtnConfiguracion { get; }
00038         public ReactiveCommand<Unit, Unit> BtnSobreNosotros { get; }
00039         public ReactiveCommand<Unit, Unit> BtnAyuda { get; }
00040         public ReactiveCommand<Unit,Unit> BtnPublicarCancion { get; }
00041         public ReactiveCommand<Unit, Unit> BtnCrearPlaylist { get; set; }
00042
00043         // Este comando esta aqui por es que que va a usar cada de los Sub-ViewModels
00044         public ReactiveCommand<Canciones, Unit> BtnReproducir { get; }
00045
00046         // Bindings
00047         private string _imagenPerfil;
00048         public string ImagenPerfil
00049         {
00050             get => _imagenPerfil;
00051             set => this.RaiseAndSetIfChanged(ref _imagenPerfil, value);
00052         }
00053
00054         public CentralTabControlViewModel()
00055         {
00056             // Preparamos los Sub-ViewModels
00057             InicioVM = new TabItemInicioViewModel();
00058
00059            // Configuramos las acciones que el InicioVM puede solicitar a la vista principal(Puente con MarcoAppViewModel)
00060             InicioVM.EnviarReproduccion = (cancion, lista) =>
00061             {
00062                 SolicitudCancion?.Invoke(cancion, lista);
00063             };
00064
00065             InicioVM.SolicitudVerDetalles = (cancion) =>
00066             {
00067                 IrADetallesCancion?.Invoke(cancion);
00068             };
00069
00070             InicioVM.SolicitudVerArtista = (idUsuario) =>
00071             {
00072                 IrAVerArtista?.Invoke(idUsuario);
00073             };
00074
00075             InicioVM.SolicitudCrearReporte = (cancion) =>
00076             {
00077                 IrACrearReporte?.Invoke(cancion);
00078             };
00079             InicioVM.SolicitudVerDetallasPlaylist = (playlist) =>
00080             {
00081                 IrADetallesPlaylist?.Invoke(playlist);
00082             };
00083
00084             BuscadorVM = new TabItemBuscadorViewModel();
00085             PopularesVM = new TabItemPopularesViewModel();
00086
00087             ImagenPerfil = GlobalData.Instance.UrlFotoPerfilGD;
00088
00089             // Configuramos los comandos reactive
```

```
00090           //Menu contextual imagen del perfil
00091           BtnPerfil = ReactiveCommand.Create(() => {
00092               Debug.WriteLine("Pulsado Perfil");
00093               IrAPerfil?.Invoke();
00094           });
00095           BtnCuenta = ReactiveCommand.Create(() => {
00096               Debug.WriteLine("Pulsado Cuenta");
00097               IrACuenta?.Invoke();
00098           });
00099           BtnGestionarCuenta = ReactiveCommand.Create(() => {
00100               Debug.WriteLine("Pulsado Gestionar Contenido");
00101               IrAGestionarCuenta?.Invoke();
00102           });
00103           BtnConfiguracion = ReactiveCommand.Create(() => {
00104               Debug.WriteLine("Pulsado Configuración");
00105               IrAConfig?.Invoke();
00106           });
00107           BtnSobreNosotros = ReactiveCommand.Create(() => {
00108               Debug.WriteLine("Pulsado Sobre nosotros");
00109               IrASobreNosotros?.Invoke();
00110           });
00111           BtnAyuda = ReactiveCommand.Create(() => {
00112               Debug.WriteLine("Pulsado Ayuda");
00113               IrAAyuda?.Invoke();
00114           });
00115           //Menu contextual de +
00116           BtnPublicarCancion = ReactiveCommand.Create(() =>
00117           {
00118               Debug.WriteLine("Pulsado Publicar Canción");
00119               IrAPublicarCancion?.Invoke();
00120           });
00121           BtnCrearPlaylist = ReactiveCommand.Create(() =>
00122           {
00123               Debug.WriteLine("Pulsado Crear Lista");
00124               IrACrearPlaylist?.Invoke();
00125           });
00126
00127
00128           //Reproducción individual
00129           BtnReproducir = ReactiveCommand.Create<Canciones>((cancion) =>
00130           {
00131               if (cancion != null)
00132               {
00133                   Debug.WriteLine($"[PLAY] Solicitando reproducir: {cancion.Titulo}");
00134                   SolicitudCancion?.Invoke(cancion,null);
00135               }
00136           });
00137       }
00138   }
00139 }
```

4.87. Referencia del archivo BetaProyecto/ViewModels/INavegable.cs

4.88. INavegable.cs

Ir a la documentación de este archivo.

```
00001 using System;
00002
00003 namespace BetaProyecto.ViewModels
00004 {
00005     public interface INavegable
00006     {
00007         //Actions
00008         Action VolverAtras { get; set; }
00009     }
00010 }
```

4.89. Referencia del archivo BetaProyecto/ViewModels/LoginViewModel.cs

4.90. LoginViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Helpers;
```

```
00002 using BetaProyecto.Services;
00003 using BetaProyecto.Singleton;
00004 using ReactiveUI;
00005 using System;
00006 using System.Reactive;
00007 using System.Threading.Tasks;
00008
00009 namespace BetaProyecto.ViewModels
00010 {
00011     public class LoginViewModel : ViewModelBase
00012     {
00013         //Servicios
00014         private readonly IDialogoService _dialogoService;
00015
00016         // Action para cambiar el contenido del marco
00017         public Action? AlCompletarLogin { get; set; }
00018         public Action? IrARegistarUser { get; set; }
00019
00020         // Bidings
00021         private string _userText = "";
00022         public string TxtUsuario
00023         {
00024             get => _userText;
00025             set => this.RaiseAndSetIfChanged(ref _userText, value);
00026         }
00027
00028         private string _passText = "";
00029         public string TxtPass
00030         {
00031             get => _passText;
00032             set => this.RaiseAndSetIfChanged(ref _passText, value);
00033         }
00034
00035
00036         // Comandos Reactive
00037         public ReactiveCommand<Unit, Unit> Login { get; }
00038         public ReactiveCommand<Unit, Unit> BtnRegistarUser { get; }
00039
00040         // Este constructor vacío es solo para el diseñador de Avalonia funcione
00041         public LoginViewModel() : this(null!)
00042         {
00043         }
00044
00045         public LoginViewModel(IDialogoService dialogoService)
00046         {
00047             _dialogoService = dialogoService;
00048
00049             // Validamos que ambos campos tengan texto para habilitar el botón de login
00050             var validacionLogin = this.WhenAnyValue(
00051                 x => x.TxtUsuario,
00052                 x => x.TxtPass,
00053                 (usuario, pass) => !string.IsNullOrWhiteSpace(usuario) && !string.IsNullOrWhiteSpace(pass)
00054             );
00055
00056             // Inicializamos el comando apuntando a la función asíncrona para evitar que la interfaz se congele
00057             Login = ReactiveCommand.CreateFromTask(IntentarLogin, validacionLogin);
00058             BtnRegistarUser = ReactiveCommand.Create(() =>
00059             {
00060                 IrARegistarUser?.Invoke();
00061             });
00062         }
00063         /// <summary>
00064         /// Intenta iniciar sesión al usuario conectándose a la base de datos y validando las credenciales proporcionadas.
00065         /// </summary>
00066         /// <remarks>Si el inicio de sesión tiene éxito, se cargan los datos del usuario y la configuración, y la vista es
00067         /// notificado para proceder. Si el inicio de sesión falla debido a credenciales incorrectas o problemas de conexión,
      una alerta es
00068         /// se muestra para informar al usuario. Este método no devuelve un resultado; realiza efectos secundarios como
00069         /// actualización del estado global y visualización de alertas. </remarks>
00070         /// <returns>Devuelve una tarea que representa la operación de inicio de sesión asíncrono. La tarea se completa
      cuando el intento de inicio de sesión tiene
00071         /// terminado, independientemente del éxito o el fracaso. </returns>
00072         private async Task IntentarLogin()
00073         {
00074             // Conectamos a la base de datos
00075             bool conectado = await MongoClientSingleton.Instance.Cliente.Conectar();
00076
00077             if (conectado)
00078             {
00079                 //Si hay conexión procedemos a verificar si existe el usuario y si es correcta la contraseña
00080                 var usuario = await MongoClientSingleton.Instance.Cliente.LoginUsuario(TxtUsuario, TxtPass);
00081
00082                 // Comprobamos si ha encontrado el usuario
00083                 if (usuario != null)
00084                 {
00085                     // Guardamos en el Singleton
00086                     GlobalData.Instance.SetUserData(usuario);
```

```
00087
00088                 // Cargamos los diccionarios de configuración del usuario
00089
00090                 ControladorDiccionarios.CargarConfiguracionInicial(
00091                     GlobalData.Instance.DiccionarioTemaGD,
00092                     GlobalData.Instance.DiccionarioIdiomaGD,
00093                     GlobalData.Instance.DiccionarioFuenteGD
00094                 );
00095
00096                 // Avisamos a la vista para que cambie de pantalla
00097                 AlCompletarLogin?.Invoke();
00098
00099                 _dialogoService.MostrarAlerta("Se a conectado correctamente con el usuario " +
    GlobalData.Instance.UsernameGD.ToString());
00100             }
00101             else
00102             {
00103                 // Conectó bien, pero usuario/pass están mal
00104                 _dialogoService.MostrarAlerta("Usuario o contraseña incorrectos. Inténtelo de nuevo.");
00105             }
00106         }
00107         else
00108         {
00109             // No hay internet o la BD está caída
00110             _dialogoService.MostrarAlerta("No se ha podido conectar con el servidor. Compruebe su conexión a Internet e
    inténtelo de nuevo más tarde.");
00111         }
00112     }
00113 }
00114 }
```

4.91. Referencia del archivo BetaProyecto/ViewModels/MarcoAppViewModel.cs

4.92. MarcoAppViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia;
00002 using Avalonia.Controls.ApplicationLifetimes;
00003 using Avalonia.Threading;
00004 using BetaProyecto.Models;
00005 using BetaProyecto.Services;
00006 using BetaProyecto.Singleton;
00007 using LibVLCSharp.Shared;
00008 using ReactiveUI;
00009 using System;
00010 using System.Collections.Generic;
00011 using System.Diagnostics;
00012 using System.Linq;
00013 using System.Reactive;
00014 using System.Threading.Tasks;
00015
00016 namespace BetaProyecto.ViewModels
00017 {
00018     public class MarcoAppViewModel : ViewModelBase
00019     {
00020         //Sercivio de Audio
00021         private readonly AudioService _audioService;
00022         private readonly LibVLC _libVLC;
00023         private readonly MediaPlayer _mediaPlayer;
00024
00025         // Variable para recordar cuál es el archivo temporal actual que está sonando
00026         private string _rutaTemporalActual = "";
00027
00028         //Reloj
00029         private readonly DispatcherTimer _timer;
00030
00031         //Servicio de dialogos para ventanas de aviso
00032         private readonly IDialogoService _dialogoService;
00033
00034         //Cache de los viewmodel para no crearlos cada vez cambiamos de vista
00035         private LoginViewModel _loginVM;
00036         private CentralTabControlViewModel _centralTabVM;
00037         private PanelUsuarioViewModel _panelUsuarioVM;
00038         private ViewSobreNosotrosViewModel _sobreNosotrosVM;
00039         private ViewAyudaViewModel _ayudaVM;
00040
00041         //Canción actual
00042         private Canciones _cancionActual;
00043         private List<Canciones> _colaReproduccion;
00044         private int _indiceCancionActual;
```

```
00045
00046         //Propiedades para boton aleatorio
00047         private bool _btnaleatorioActivo = false;
00048         private Random _random = new Random();
00049         private List<Canciones> _historialAleatorio = new List<Canciones>();
00050         private int _indiceHistorialModoAleatorio = -1;
00051
00052         //Propiedades para Binding del controlador de música, la informaciόm de la canción nombre, artista, imagen)
00053         private string _nombreCancion;
00054         public string NombreCancionActual
00055         {
00056             get => _nombreCancion;
00057             set => this.RaiseAndSetIfChanged(ref _nombreCancion, value);
00058         }
00059
00060         private string _nombreArtista = "";
00061         public string NombreArtistaActual
00062         {
00063             get => _nombreArtista;
00064             set => this.RaiseAndSetIfChanged(ref _nombreArtista, value);
00065         }
00066
00067         private string _imagenCancion = "https://i.ibb.co/v6CJTMX2/Icono-Musica.jpg";
00068         public string ImagenCancionActual
00069         {
00070             get => _imagenCancion;
00071             set => this.RaiseAndSetIfChanged(ref _imagenCancion, value);
00072         }
00073         //Propiedades para Binding del controlador de música iconos botones (play/pause, next, back, aleatorio, favorito)
00074         private string _iconoPlayPause;
00075         public string IconoPlayPause
00076         {
00077             get => _iconoPlayPause;
00078             set => this.RaiseAndSetIfChanged(ref _iconoPlayPause, value);
00079         }
00080
00081         private string _iconoNext;
00082         public string IconoNext
00083         {
00084             get => _iconoNext;
00085             set => this.RaiseAndSetIfChanged(ref _iconoNext, value);
00086         }
00087
00088         private string _iconoBack;
00089         public string IconoBack
00090         {
00091             get => _iconoBack;
00092             set => this.RaiseAndSetIfChanged(ref _iconoBack, value);
00093         }
00094
00095         private string _iconAleatorio;
00096         public string IconoAleatorio
00097         {
00098             get => _iconAleatorio;
00099             set => this.RaiseAndSetIfChanged(ref _iconAleatorio, value);
00100         }
00101
00102         private string _iconLike;
00103         public string IconoLike
00104         {
00105             get => _iconLike;
00106             set => this.RaiseAndSetIfChanged(ref _iconLike, value);
00107         }
00108
00109         //Propiedad para Binding del tiempo de la canción
00110         private string _tiempoActualCancion = "--:--";
00111         public string TiempoActualCancion
00112         {
00113             get => _tiempoActualCancion;
00114             set => this.RaiseAndSetIfChanged(ref _tiempoActualCancion, value);
00115         }
00116
00117         private string _tiempoTotalCancion = "--:--";
00118         public string TiempoTotalCancion
00119         {
00120             get => _tiempoTotalCancion;
00121             set => this.RaiseAndSetIfChanged(ref _tiempoTotalCancion, value);
00122         }
00123         //Propiedad para Binding del slider vincualado a la canción
00124
00125         private double _valorSliderCancion = 0;
00126         public double ValorSliderCancion
00127         {
00128             get => _valorSliderCancion;
00129             set => this.RaiseAndSetIfChanged(ref _valorSliderCancion, value);
00130         }
00131
```

```
00132         //Propiedad para Binding del slider de volumen
00133
00134         private double _valorSliderVolumen = 100;
00135         public double ValorSliderVolumen
00136         {
00137             get => _valorSliderVolumen;
00138             set
00139             {
00140                 this.RaiseAndSetIfChanged(ref _valorSliderVolumen, value);
00141                 //Actualiza volumen del MediaPlayer cada vez que cambia de valor
00142                 if (_mediaPlayer != null)
00143                 {
00144                     _mediaPlayer.Volume = (int)value;
00145                 }
00146             }
00147         }
00148
00149         //Navegación
00150         private ViewModelBase _vistaActual;
00151         public ViewModelBase VistaActual
00152         {
00153             get => _vistaActual;
00154             set => this.RaiseAndSetIfChanged(ref _vistaActual, value);
00155         }
00156         // Propidades para hacer POPUPS
00157         //Este contiene el ViewModel
00158         private ViewModelBase _popupActual;
00159         public ViewModelBase PopupActual
00160         {
00161             get => _popupActual;
00162             set
00163             {
00164                 this.RaiseAndSetIfChanged(ref _popupActual, value);
00165                 // Calculamos si el popup es visible automáticamente
00166                 PopupVisible = value != null;
00167             }
00168         }
00169         //Este controla la visibilidad del popup
00170         private bool _popupVisible;
00171         public bool PopupVisible
00172         {
00173             get => _popupVisible;
00174             set => this.RaiseAndSetIfChanged(ref _popupVisible, value);
00175         }
00176
00177
00178         // Propiedad para mostrar u ocultar el controlador de musica
00179         private bool _barraVisible;
00180         public bool BarraVisible
00181         {
00182             get => _barraVisible;
00183             set => this.RaiseAndSetIfChanged(ref _barraVisible, value);
00184         }
00185
00186         //Comandos ReactiveUI para los botones del controlador de música
00187         public ReactiveCommand<Unit, Unit> BtnPlayPauseCommand { get; }
00188         public ReactiveCommand<Unit, Unit> BtnFavCommand { get; }
00189         public ReactiveCommand<Unit, Unit> BtnNextCommand { get; }
00190         public ReactiveCommand<Unit, Unit> BtnBackCommand { get; }
00191         public ReactiveCommand<Unit, Unit> BtnAleatorioCommand { get; }
00192
00193         //Constructor
00194         public MarcoAppViewModel()
00195         {
00196             //Inicializamos Servicios
00197             _dialogoService = new DialogoService();
00198             _libVLC = new LibVLC();
00199             _mediaPlayer = new MediaPlayer(_libVLC);
00200             _audioService = new AudioService();
00201
00202             //Configuramos el timer (Se actualiza al segundo)
00203             _timer = new DispatcherTimer();
00204             _timer.Interval = TimeSpan.FromSeconds(1);
00205             _timer.Tick += Timer_Tick;
00206
00207             //Inicializamos iconos del reproductor con recursos del diccionario url del propio proyecto que van a
    /Assets/Imagenes/)
00208             IconoPlayPause = "Img_Play";
00209             IconoNext = "Img_Next_Disabled";
00210             IconoBack = "Img_Back_Disabled";
00211             IconoAleatorio = "Img_Aleatorio_Disabled";
00212             IconoLike = "Img_Like_OFF";
00213
00214             //Configuramos vista inicial (Login)
00215
00216             // Creamos el login
00217             _loginVM = new LoginViewModel(_dialogoService);
```

```
00218
00219           //Actions
00220
00221           //Action AlCompletarLogin: Cuando se pulse el botón de completar login, se ejecutará este código si el usuario es
    correcto
00222           _loginVM.AlCompletarLogin = () =>
00223           {
00224               IrAlCentralTabControl();
00225               BarraVisible = true;
00226           };
00227           //Action IrARegistarUser: Cuando se pulse el botón de registrar usuario
00228           _loginVM.IrARegistarUser = () =>
00229           {
00230               IrACrearUsuario();
00231           };
00232
00233           // Inicializamos los comandos
00234
00235           //- Comando btnPlayPause
00236           BtnPlayPauseCommand = ReactiveCommand.Create(AccionarPlayPause);
00237           //- Comando Favorito
00238           BtnFavCommand = ReactiveCommand.CreateFromTask(async () =>
00239           {
00240               if (_cancionActual != null)
00241                   await AlterarFavorito(_cancionActual);
00242           });
00243           //- Comando Next
00244           BtnNextCommand = ReactiveCommand.Create(NextCancion);
00245           //- Comando Back
00246           BtnBackCommand = ReactiveCommand.Create(BackCancion);
00247           //- Comando Aleatorio
00248           BtnAleatorioCommand = ReactiveCommand.Create(AlternarAleatorio);
00249
00250           // Asignamos la vista inicial
00251           VistaActual = _loginVM;
00252           // Al principio estamos en Login, así que ocultamos la barra
00253           BarraVisible = false;
00254       }
00255
00256       #region Método para cargar las vistas
00257       /// <summary>
00258       /// Muestra la vista de creación del usuario como una ventana emergente, lo que permite al usuario crear una nueva
    cuenta.
00259       /// </summary>
00260       /// <remarks>
00261       /// Este método reemplaza la ventana emergente actual con la vista de creación del usuario. Para cerrar el
00262       /// popup y volver al estado anterior, usar la acción de retroalimentación proporcionada dentro de la creación del
    usuario
00263       /// </remarks>
00264       public void IrACrearUsuario()
00265       {
00266           var viewCrearUsuarioVM = new ViewCrearUsuarioViewModel(
00267               accionVolver: () =>
00268               {
00269                   PopupActual = null;
00270               }
00271           );
00272           PopupActual = viewCrearUsuarioVM;
00273       }
00274       /// <summary>
00275       /// Navega a la vista de control de pestaña central, inicializándola si es necesario y configurándola como la vista
    actual.
00276       /// </summary>
00277       /// <remarks>Si la vista de control de pestaña central no se ha creado, este método la inicializa y
00278       /// configura sus acciones de navegación. Las llamadas posteriores reutilizarán la instancia de vista existente. Este
    método también
00279       /// actualiza los iconos de la interfaz como parte del proceso de navegación.</remarks>
00280       public void IrAlCentralTabControl()
00281       {
00282           if (_centralTabVM == null)
00283           {
00284               _centralTabVM = new CentralTabControlViewModel();
00285
00286               // Configuramos la navegación interna del TabControl
00287               _centralTabVM.IrAPerfil = () => IrAPanelUsuario(0);
00288               _centralTabVM.IrACuenta = () => IrAPanelUsuario(1);
00289               _centralTabVM.IrAGestionarCuenta = () => IrAPanelUsuario(2);
00290               _centralTabVM.IrAConfig = () => IrAPanelUsuario(3);
00291               _centralTabVM.IrASobreNosotros = () => IrASobreNosotros();
00292               _centralTabVM.IrAAyuda = () => IrAAyuda();
00293               _centralTabVM.IrAPublicarCancion = () => IrAPublicarCancion();
00294               _centralTabVM.IrACrearPlaylist = () => IrACrearPlaylist();
00295               _centralTabVM.IrADetallesCancion = (cancion) => IrADetallesCancion(cancion);
00296               _centralTabVM.IrAVerArtista = (idUsuario) => IrAVerArtista(idUsuario);
00297               _centralTabVM.IrACrearReporte = (cancion)=> IrACrearReporte(cancion);
00298               _centralTabVM.IrADetallesPlaylist = (playlist) => IrADetallesPlaylist(playlist);
00299               // Configurar reproducción
```

```
00300                 _centralTabVM.SolicitudCancion = (cancion, lista) =>
00301                 {
00302                     ReproducirCancion(cancion, lista);
00303                 };
00304             }
00305             RefrescarIconos();
00306             VistaActual = _centralTabVM;
00307         }
00308         /// <summary>
00309         /// Navega al panel de usuario y muestra la pestaña especificada.
00310         /// </summary>
00311         /// <remarks> Si el panel de usuario no se ha creado, este método lo inicializa y configura
00312         /// acciones relacionadas. Si el panel ya existe, actualiza la pestaña mostrada. La navegación oculta los principales
00313         /// barra mientras el panel de usuario está activo.</remarks>
00314         /// <param name="pestania">El índice de la pestaña que se mostrará en el panel de usuario. Debe ser un índice de
    pestañas válido y compatible con el panel.</param>
00315         public void IrAPanelUsuario(int pestania)
00316         {
00317             // Comprobamos si el panel de usuario ya existe o no
00318             if (_panelUsuarioVM == null)
00319             {
00320                 _panelUsuarioVM = new PanelUsuarioViewModel(pestania);
00321             }
00322             else
00323             {
00324                 // Si ya existe, solo actualizamos la pestaña que queremos ver
00325                 _panelUsuarioVM.IndiceTab = pestania;
00326             }
00327 
00328             //Conexiones de los Actions desde el PanelUsuario (Editar canción, editar playlist, volver atrás, logout, salir,
    refrescar)
00329             _panelUsuarioVM.IrAEditarCancion = (cancion) => IrAEditarCancion(cancion);
00330             _panelUsuarioVM.IrAEditarPlaylist = (playlist) => IrAEditarPlaylist(playlist);
00331             _panelUsuarioVM.VolverAtras = () =>
00332             {
00333                 VistaActual = _centralTabVM; // Volvemos a la vista anterior (TabControl)
00334                 BarraVisible = true;
00335             };
00336 
00337             _panelUsuarioVM.AccionLogout = CerrarSesion;
00338             _panelUsuarioVM.AccionSalir = CerrarAplicacion;
00339             _panelUsuarioVM.AccionRefrescarDesdePadre = RefrescarIconos;
00340 
00341 
00342             // Al entrar ocultamos la barra
00343             BarraVisible = false;
00344             // Y mostramos el panel de usuario
00345             VistaActual = _panelUsuarioVM;
00346         }
00347         /// <summary>
00348         /// Navega a la vista 'Sobre Nosotros', creándola y configurándola si no existe.
00349         /// </summary>
00350         /// <remarks>Si no se ha creado la vista 'Sobre Nosotros', este método la inicializa y
00351         /// lo establece como la vista actual. Las llamadas posteriores reutilizarán la instancia de vista existente. La barra de
    música está oculta
00352         /// mientras esta vista está activa. </remarks>
00353         public void IrASobreNosotros()
00354         {
00355             // Si no existe, la creamos y configuramos
00356             if (_sobreNosotrosVM == null)
00357             {
00358                 _sobreNosotrosVM = new ViewSobreNosotrosViewModel();
00359                 ActivarVolverAtras(_sobreNosotrosVM); // Le pasamos la función de "Volver"
00360             }
00361 
00362             // Si ya existe, simplemente la mostramos
00363             VistaActual = _sobreNosotrosVM;
00364             BarraVisible = false; // Ocultamos la barra de música
00365         }
00366         /// <summary>
00367         /// Muestra la vista de ayuda en la aplicación, creándola y configurándola si no existe.
00368         /// </summary>
00369         /// <remarks>Cuando se invoca, este método cambia la vista actual a la vista de ayuda y oculta el
00370         /// barra de aplicaciones. Si la vista de ayuda no se ha creado previamente, se inicializa y configura antes
00371         /// se está mostrando. </remarks>
00372         public void IrAAyuda()
00373         {
00374             // Si no existe, la creamos y configuramos
00375             if (_ayudaVM == null)
00376             {
00377                 _ayudaVM = new ViewAyudaViewModel();
00378                 ActivarVolverAtras(_ayudaVM); // Le pasamos la función de "Volver"
00379             }
00380 
00381             // Si ya existe, simplemente la mostramos
00382             VistaActual = _ayudaVM;
00383             BarraVisible = false;
```

```
00384         }
00385         /// <summary>
00386         /// Navega a la vista para publicar una nueva canción y actualiza el estado actual de la vista en consecuencia.
00387         /// </summary>
00388         /// <remarks>Este método reemplaza la vista actual con la vista de publicación de canciones y oculta los
00389         /// barra de navegación. Cuando el usuario regresa desde la vista de publicación, se restaura la vista original del
      control de pestañas.
00390         /// y la barra de navegación vuelve a ser visible. </remarks>
00391         private void IrAPublicarCancion()
00392         {
00393             // Creamos una nueva instancia del ViewModel de publicar canción cada vez para asegurarnos de que se reinicia el
      formulario
00394             var publicarCancionVM = new ViewPublicarCancionViewModel(
00395                 accionVolver: () =>
00396                 {
00397                     // AL VOLVER: Restauramos el TabControl como vista actual
00398                     VistaActual = _centralTabVM;
00399                     BarraVisible = true;
00400                 }
00401             );
00402             // Cambiamos la vista visible
00403             BarraVisible = false;
00404             VistaActual = publicarCancionVM;
00405 
00406         }
00407         /// <summary>
00408         /// Navega a la vista para crear una nueva lista de reproducción personalizada y restablece el formulario de creación
      de la lista de reproducción.
00409         /// </summary>
00410         /// <remarks>Este método reemplaza la vista actual con la vista de creación de listas de reproducción y oculta los
00411         /// barra de navegación. Cuando el usuario regresa desde la vista de creación de listas de reproducción, la vista
      original y la barra de navegación
00412         /// se restauran. </remarks>
00413         private void IrACrearPlaylist()
00414         {
00415             // Creamos una nueva instancia del ViewModel de publicar canción cada vez para asegurarnos de que se reinicia el
      formulario
00416             var crearplaylistVM = new ViewCrearListaPersonalizadaViewModel(
00417                 accionVolver: () =>
00418                 {
00419                     // AL VOLVER: Restauramos el TabControl como vista actual
00420                     VistaActual = _centralTabVM;
00421                     BarraVisible = true;
00422                 }
00423             );
00424             // Cambiamos la vista
00425             BarraVisible = false;
00426             VistaActual = crearplaylistVM;
00427 
00428         }
00429         /// <summary>
00430         /// Muestra la vista de detalles para la canción especificada y establece acciones relacionadas como reproducir,
      marcar como favorita y devolver.
00431         /// </summary>
00432         /// <param name="cancion">La canción para la que se muestran los detalles. No puede ser nula. </param>
00433         private void IrADetallesCancion(Canciones cancion)
00434         {
00435             var viewCancionesVM = new ViewCancionesViewModel(
00436                 cancion: cancion,
00437                 accionVolver: () =>
00438                 {
00439                     PopupActual = null;
00440                 },
00441               //Le pasamos las funciones de reproducción y favorito mediantes Actions para que se puedan usar en esa ventana
00442                 accionReproducir: (cancion) => ReproducirCancion(cancion, null),
00443                 accionLike: async (cancion) => await AlterarFavorito(cancion)
00444             );
00445             //Cambiamos la vista
00446             PopupActual = viewCancionesVM;
00447         }
00448         /// <summary>
00449         /// Muestra los detalles del usuario especificado en una vista emergente.
00450         /// </summary>
00451         /// <param name="idUsuario">El identificador único del usuario cuyos detalles se deben mostrar. No puede ser
      nulo o vacío. </param>
00452         private void IrAVerArtista(string idUsuario)
00453         {
00454             var viewUsuarioVM = new ViewUsuariosViewModel(
00455                 idUsuario: idUsuario,
00456                 accionVolver: () =>
00457                 {
00458                     PopupActual = null;
00459                 }
00460             );
00461             //Cambiamos la vista
00462             PopupActual = viewUsuarioVM;
00463         }
```

```
00464         /// <summary>
00465         /// Muestra la vista de creación del informe para la canción especificada.
00466         /// </summary>
00467         /// <param name="cancion">La canción para la que se mostrará la vista de creación del informe. No puede ser nula. </param>
00468         private void IrACrearReporte(Canciones cancion)
00469         {
00470             var viewCrearReporteVM = new ViewCrearReporteViewModel(
00471                 cancion: cancion,
00472                 accionVolver: () =>
00473                 {
00474                     PopupActual = null;
00475                 }
00476             );
00477             //Cambiamos la vista
00478             PopupActual = viewCrearReporteVM;
00479         }
00480         private void IrADetallesPlaylist(ListaPersonalizada playlist)
00481         {
00482             var viewListaPersonalizadaVM = new ViewListaPersonalizadaViewModel(
00483                 playlist: playlist,
00484                 accionVolver: () =>
00485                 {
00486                     PopupActual = null;
00487                 }
00488             );
00489             //Cambiamos la vista
00490             PopupActual = viewListaPersonalizadaVM;
00491         }
00492         /// <summary>
00493         /// Muestra la vista de edición de canciones para la canción especificada.
00494         /// </summary>
00495         /// <param name="cancion">La canción a editar. No puede ser nula. </param>
00496         private void IrAEditarCancion(Canciones cancion)
00497         {
00498             var viewEditarCancionVM = new ViewEditarCancionViewModel(
00499                 cancion: cancion,
00500                 accionVolver: () =>
00501                 {
00502                     PopupActual = null;
00503                 }
00504             );
00505             //Cambiamos la vista
00506             PopupActual = viewEditarCancionVM;
00507         }
00508         /// <summary>
00509         /// Muestra la vista de edición de la lista de reproducción personalizada especificada, permitiendo al usuario modificar sus detalles.
00510         /// </summary>
00511         /// <param name="playlist">La lista de reproducción personalizada que se va a editar. No puede ser nula. </param>
00512         private void IrAEditarPlaylist(ListaPersonalizada playlist)
00513         {
00514             var viewEditarListaPersonalizadaVM = new ViewEditarListaPersonalizadaViewModel(
00515                 playlist: playlist,
00516                 accionVolver: () =>
00517                 {
00518                     PopupActual = null;
00519                 }
00520             );
00521             //Cambiamos la vista
00522             PopupActual = viewEditarListaPersonalizadaVM;
00523         }
00524
00525
00526         // (Usamos una interfaz para ahorramos sobrecargar el metodo solo tendremos que añadir la interfaz)
00527         /// <summary>
00528         /// Asigna una llamada de retorno al modelo de vista especificado que permite la navegación de regreso a la vista principal.
00529         /// </summary>
00530         /// <remarks>Utilice este método para proporcionar un comportamiento de retorno estándar para los modelos de vista que admiten
00531         /// navegación. Después de invocar el callback asignado, la barra de navegación principal se vuelve visible. </remarks>
00532         /// <param name="vm">El modelo de vista que implementa la interfaz INavegable. El método establece su acción VolverAtras para navegar
00533         /// de vuelta al control central de pestañas. </param>
00534         public void ActivarVolverAtras(INavegable vm)
00535         {
00536             vm.VolverAtras = () =>
00537             {
00538                 IrAlCentralTabControl();
00539                 BarraVisible = true;
00540             };
00541         }
00542         #endregion
00543
```

```
00544       #region Métodos para reproducción de música
00545       /// <summary>
00546       /// Inicia la reproducción de la canción especificada, opcionalmente usando una lista de reproducción proporcionada como cola de reproducción.
00547       /// </summary>
00548       /// <remarks>Si se proporciona una lista de reproducción, la reproducción comenzará con la canción especificada dentro de esa lista.
00549       /// lista. De lo contrario, la reproducción se limita a la canción individual proporcionada. La cola de reproducción y el índice de canciones actual
00550       /// se actualizan en consecuencia. </remarks>
00551       /// <param name="cancion">La canción a reproducir. No puede ser nula. </param>
00552       /// <param name="listaOrigen">Una lista opcional de canciones para usar como cola de reproducción. Si es nula o vacía, solo la canción especificada será
00553       /// jugado. </param>
00554       public void ReproducirCancion(Canciones cancion, List<Canciones>? listaOrigen = null)
00555       {
00556           //Gestion de la cola de reproducción
00557           if (listaOrigen != null && listaOrigen.Count > 0)//Si se proporciona una lista de reproducción válida, la usamos como cola
00558           {
00559               _colaReproduccion = listaOrigen;
00560               _indiceCancionActual = _colaReproduccion.IndexOf(cancion);
00561           }
00562           else// Si no se proporciona una lista, creamos una cola artificial con solo la canción actual
00563           {
00564               _colaReproduccion = new List<Canciones> {cancion};
00565               _indiceCancionActual = 0;
00566           }
00567
00568
00569           // Actualizamos los iconos
00570           ActualizarIconoAleatorio(cancion);
00571           ActualizarIconoNextBack();
00572
00573           // Finalmente, cargamos y reproducimos la canción
00574           CargarYReproducir(cancion);
00575       }
00576       /// <summary>
00577       /// Carga la canción especificada y comienza la reproducción, actualizando la interfaz del reproductor y el estado relacionado en consecuencia.
00578       /// </summary>
00579       /// <remarks>Este método actualiza la información actual de la canción, los controles de reproducción y el usuario
00580       /// elementos de interfaz para reflejar la canción cargada. Determina la fuente de audio apropiada en función del
00581       /// URL de la canción, que admite enlaces a archivos en la nube tanto de YouTube como directos. Si la canción está marcada como favorita, el
00582       /// el icono like se actualiza. La reproducción se inicia de forma asíncrona y las métricas de reproducción de canciones se incrementan en el
00583       /// fondo. Si no se puede cargar el audio, se muestra una alerta al usuario. Este método es asincrónico.
00584       /// pero devuelve nulo; las excepciones se capturan y registran internamente. </remarks>
00585       /// <param name="cancion">La canción a cargar y reproducir. No debe ser nula y debe contener metadatos válidos y una URL reproducible.</param>
00586       public async void CargarYReproducir(Canciones cancion)
00587       {
00588
00589           LimpiarArchivoTemporal();
00590
00591           System.Diagnostics.Debug.WriteLine("3. [RECIBIDO] MarcoApp ha recibido la canción: " + cancion.Titulo);
00592           _cancionActual = cancion;
00593           try
00594           {
00595               //Mostrar la barra y actualizar textos (Binding)
00596               NombreCancionActual = cancion.Titulo;
00597               NombreArtistaActual = cancion.NombreArtista;
00598               ImagenCancionActual = cancion.ImagenPortadaUrl;
00599               //Ajustamos inferdaz de reproductor de música
00600               IconoPlayPause = "Img_Pause";
00601               ValorSliderCancion = 0;
00602               TiempoActualCancion = "--:--";
00603               TiempoTotalCancion= "--:--";
00604
00605               var listaFavoritos = GlobalData.Instance.FavoritosGD;
00606
00607               // Comprobamos si la lista existe y si contiene el ID de la canción
00608               if (listaFavoritos != null && listaFavoritos.Contains(cancion.Id))
00609               {
00610                   IconoLike = "Img_Like_ON";
00611               }
00612               else
00613               {
00614                   IconoLike = "Img_Like_OFF";
00615               }
00616
00617               System.Diagnostics.Debug.WriteLine($"Buscando audio para: {cancion.Titulo}...");
00618
00619               string urlStream = "";
00620               // CASO 1: Es un video de YouTube
00621               if (cancion.UrlCancion.Contains("youtube.com") || cancion.UrlCancion.Contains("youtu.be"))
```

```
00622             {
00623                 System.Diagnostics.Debug.WriteLine("Detectado enlace de YouTube. Solicitando stream a la API...");
00624 
00625                 // Llamamos al nuevo método (que devuelve InfoCancionNube)
00626                 var infoAudio = await _audioService.ObtenerMp3(cancion.UrlCancion);
00627 
00628                 if (infoAudio != null && !string.IsNullOrEmpty(infoAudio.Url))
00629                 {
00630                     urlStream = infoAudio.Url; // Usamos la URL "traducida" temporal
00631                 }
00632             }
00633             // CASO 2: Es un archivo directo de nuestra nube [CLOUDNARY]
00634             else
00635             {
00636                 System.Diagnostics.Debug.WriteLine("Modo Archivo: Solicitando acceso seguro...");
00637 
00638                 // Llamamos al servicio. Él se encarga de:
00639                 // 1. Descargar si no existe.
00640                 // 2. Cifrar en AES para guardar en disco (Storage).
00641                 // 3. Descifrar a un temporal para que VLC lo lea ahora.
00642 
00643                 urlStream = await _audioService.ObtenerRutaAudioSegura(cancion.UrlCancion, cancion.Id);
00644 
00645                 // Guarda la ruta para luego borrarla
00646                 if (urlStream.Contains("BetaProyectoMusicTemp"))
00647                 {
00648                     System.Diagnostics.Debug.WriteLine($"[RUTAS] Carpeta temporal: {urlStream}");
00649                     _rutaTemporalActual = urlStream;
00650                 }
00651 
00652             }
00653 
00654             if (!string.IsNullOrEmpty(urlStream))
00655             {
00656                 // Ahora que tenemos la URL/Ruta del .mp3 , la pasamos a VLC para reproducirla
00657                 var media = new Media(_libVLC, new Uri(urlStream));
00658                 _mediaPlayer.Play(media);
00659                 _timer.Start();
00660                 System.Diagnostics.Debug.WriteLine("Reproduciendo...");
00661                 // Lanzamos la tarea en segundo plano para no bloquear la música
00662                 _ = MongoClientSingleton.Instance.Cliente.IncrementarMetricaCancion(cancion.Id,
    "metricas.total_reproducciones", 1);
00663                 System.Diagnostics.Debug.WriteLine("Visualización añadida");
00664 
00665             }
00666             else
00667             {
00668                 System.Diagnostics.Debug.WriteLine("Error: La API no devolvió nada.");
00669                 _dialogoService.MostrarAlerta("No se pudo cargar el audio de esta canción.");
00670                 IconoPlayPause = "Img_Play";
00671             }
00672         }
00673         catch (Exception ex)
00674         {
00675             System.Diagnostics.Debug.WriteLine("Error al reproducir: " + ex.Message);
00676         }
00677     }
00678     /// <summary>
00679     /// Cambia la reproducción del reproductor multimedia entre los estados de reproducción y pausa.
00680     /// </summary>
00681     /// <remarks>Si el reproductor multimedia está reproduciéndose, este método detiene la reproducción y actualiza el
00682     /// reproducir/pausa el icono en consecuencia. Si el reproductor multimedia se detiene, este método inicia la
    reproducción y actualiza el icono,
00683     /// y inicia el temporizador asociado. Este método no genera excepciones y asume que el reproductor multimedia y
00684     /// los temporizadores están correctamente inicializados. </remarks>
00685     private void AccionarPlayPause()
00686     {
00687         if (_mediaPlayer.IsPlaying)
00688         {
00689             _mediaPlayer.Pause();
00690             IconoPlayPause = "Img_Play"; // Volvemos a mostrar Play
00691         }
00692         else
00693         {
00694             _mediaPlayer.Play();
00695             IconoPlayPause = "Img_Pause"; // Mostramos Pause
00696             _timer.Start();
00697         }
00698     }
00699     /// <summary>
00700     /// Cambia el modo de reproducción aleatoria para la lista de reproducción actual. Cuando está activado, el orden de
    reproducción se aleatoriza;
00701     /// cuando está desactivado, la reproducción se reanuda en el orden original de la canción actual.
00702     /// </summary>
00703     /// <remarks>Este método no tiene efecto si la lista de reproducción es nula o contiene uno o menos elementos.
00704     /// Cuando se activa el modo aleatorio, la canción actual se añade al historial de reproducción aleatoria para
    permitir su retorno
```

```
00705       /// a ella. Cuando se desactiva, la reproducción continúa desde la posición actual de la canción en la lista de
    reproducción original
00706       /// orden. </remarks>
00707       private void AlternarAleatorio()
00708       {
00709           // Comprobamos que la cola de reproducción es válida y tiene suficientes canciones para justificar el modo aleatorio
00710           if (_colaReproduccion == null || _colaReproduccion.Count <= 1)
00711           {
00712               return;
00713           }
00714           //Intercambiamos el estado del botón
00715           _btnaleatorioActivo = !_btnaleatorioActivo;
00716
00717           if (_btnaleatorioActivo)
00718           {
00719               // --- ACTIVAR ---
00720               System.Diagnostics.Debug.WriteLine("[ALEATORIO] ON");
00721               IconoAleatorio = "Img_Aleatorio_ON";
00722
00723               // Iniciamos el historial con la canción actual para poder volver a ella
00724               _historialAleatorio.Clear();
00725               if (_cancionActual != null)
00726               {
00727                   _historialAleatorio.Add(_cancionActual);
00728                   _indiceHistorialModoAleatorio = 0;
00729               }
00730           }
00731           else
00732           {
00733               // --- DESACTIVAR ---
00734               System.Diagnostics.Debug.WriteLine("[ALEATORIO] OFF");
00735               IconoAleatorio = "Img_Aleatorio_OFF";
00736
00737               // Al salir del modo aleatorio, buscamos la canción actual en la lista original
00738               // para seguir el orden normal desde ahí.
00739               if (_colaReproduccion != null && _cancionActual != null)
00740               {
00741                   _indiceCancionActual = _colaReproduccion.IndexOf(_cancionActual);
00742               }
00743           }
00744           ActualizarIconoNextBack();
00745       }
00746       /// <summary>
00747       /// Añade o elimina la canción especificada de la lista de favoritos del usuario, actualizando el estado favorito
00748       /// en consecuencia.
00749       /// </summary>
00750       /// <remarks>Si la canción ya está en la lista de favoritos, se elimina; de lo contrario, se añade.
00751       /// El método también actualiza el icono de like y ajusta la métrica de like count de la canción. No se realiza ninguna
    acción si el
00752       /// la canción proporcionada es nula. </remarks>
00753       /// <param name="cancion">La canción a añadir o quitar de la lista de favoritos. Si es nula, no se realiza la
    operación. </param>
00754       /// <returns>Devuelve una tarea que representa la operación asíncrona. </returns>
00755       private async Task AlterarFavorito(Canciones cancion)
00756       {
00757           // Si la canción es nula, no hacemos nada por seguridad
00758           if (cancion == null)
00759           {
00760               return;
00761           }
00762           // Obtenemos la lista de favoritos del usuario y el ID del usuario para realizar las operaciones necesarias
00763           var listaFavoritos = GlobalData.Instance.FavoritosGD;
00764           var idUsuario = GlobalData.Instance.UserIdGD;
00765           var idCancion = cancion.Id;
00766           // Comprobamos si la canción ya está en favoritos
00767           if (listaFavoritos.Contains(idCancion))
00768           {
00769               // Quitar de favoritos
00770               listaFavoritos.Remove(idCancion);
00771               if (_cancionActual != null && _cancionActual.Id == idCancion)
00772                   IconoLike = "Img_Like_OFF";
00773
00774               await MongoClientSingleton.Instance.Cliente.EliminarDeFavorito(idUsuario, idCancion);
00775               System.Diagnostics.Debug.WriteLine($"Canción {idCancion} eliminada de favoritos del usuario {idUsuario}.");
00776
00777               // 2. Restamos 1 al contador de la canción
00778               await MongoClientSingleton.Instance.Cliente.IncrementarMetricaCancion(idCancion,
    "metricas.total_megustas", -1);
00779               System.Diagnostics.Debug.WriteLine("Like quitado (-1)");
00780           }
00781           else// La canción no estaba en favoritos, la añadimos
00782           {
00783               // Añadir a favoritos
00784               listaFavoritos.Add(idCancion);
00785               if (_cancionActual != null && _cancionActual.Id == idCancion)
00786                   IconoLike = "Img_Like_ON";
00787
```

```
00788                 await MongoClientSingleton.Instance.Cliente.AgregarAFavorito(idUsuario, idCancion);
00789                 System.Diagnostics.Debug.WriteLine($"Canción {idCancion} añadida a favoritos del usuario {idUsuario}.");
00790
00791                 // 2. Sumamos 1 al contador de la canción
00792                 await MongoClientSingleton.Instance.Cliente.IncrementarMetricaCancion(idCancion,
      "metricas.total_megustas", 1);
00793                 System.Diagnostics.Debug.WriteLine("Like añadido (+1)");
00794             }
00795         }
00796         /// <summary>
00797         /// Avanza a la reproducción de la siguiente canción en la lista de reproducción, manejando tanto el modo secuencial
      como el de mezcla.
00798         /// </summary>
00799         /// <remarks>En el modo de mezcla, este método selecciona aleatoriamente la siguiente canción entre las que aún no
      están disponibles.
00800         /// jugado, manteniendo un historial para evitar repeticiones hasta que se hayan reproducido todas las canciones. En
      el modo secuencial,
00801         /// avanza a la siguiente canción en orden si está disponible. Si todas las canciones se han reproducido en modo de
      mezcla, el historial
00802         /// se restablece excepto para la canción actual, y la reproducción continúa. Este método no tiene efecto si la lista de
      reproducción es
00803         /// vacío. </remarks>
00804         private void NextCancion()
00805         {
00806             // Comprobamos que la cola de reproducción es válida y tiene canciones
00807             if (_colaReproduccion != null && _colaReproduccion.Count > 0)
00808             {
00809                 if (_btnaleatorioActivo) // --- MODO ALEATORIO ---
00810                 {
00811                     // Comprobamos si el usuario a usado el Back antes de avanzar, si es así, avanzamos por el historial
00812                     if (_indiceHistorialModoAleatorio < _historialAleatorio.Count - 1)
00813                     {
00814                         _indiceHistorialModoAleatorio++;
00815                         var cancionHistorial = _historialAleatorio[_indiceHistorialModoAleatorio];
00816                         ActualizarIconoNextBack();
00817                         CargarYReproducir(cancionHistorial);
00818                     }
00819                     else // Si el usuario no ha usado el Back o ya está al final del historial, generamos elegimos una canción
      aleatoria
00820                     {
00821                         // Buscamos qué canciones de la lista original NO están en el historial todavía
00822                         // Usamos LINQ: "Dame las canciones de la cola EXCEPTO las del historial"
00823                         var cancionesPendientes = _colaReproduccion.Except(_historialAleatorio).ToList();
00824
00825                         // Comprobamos si quedan canciones por sonar
00826                         if (cancionesPendientes.Count > 0)
00827                         {
00828                             // Elegimos una al azar de las que FALTAN
00829                             int indiceRandom = _random.Next(0, cancionesPendientes.Count);
00830                             var nuevaCancion = cancionesPendientes[indiceRandom];
00831
00832                             // Guardamos en historial y avanzamos
00833                             _historialAleatorio.Add(nuevaCancion);
00834                             _indiceHistorialModoAleatorio++;
00835                             ActualizarIconoNextBack();
00836                             CargarYReproducir(nuevaCancion);
00837                         }
00838                         else // Si no quedan canciones por sonar, significa que el usuario ya ha escuchado toda la lista en modo
      aleatorio
00839                         {
00840                             // Reiniciamos el historial (salvo la canción actual)
00841                             var cancionActualTemp = _cancionActual;
00842                             _historialAleatorio.Clear();
00843                             _historialAleatorio.Add(cancionActualTemp);
00844                             _indiceHistorialModoAleatorio = 0;
00845
00846                             // Volvemos a llamar a esta función para que ahora sí encuentre pendientes
00847                             NextCancion();
00848                         }
00849                     }
00850                 }
00851                 else // --- MODO NORMAL ---
00852                 {
00853                     // Solo avanzamos si no es la última
00854                     if (_indiceCancionActual < _colaReproduccion.Count - 1)
00855                     {
00856                         _indiceCancionActual++;
00857                         ActualizarIconoNextBack();
00858                         CargarYReproducir(_colaReproduccion[_indiceCancionActual]);
00859                     }
00860                 }
00861             }
00862         }
00863         /// <summary>
00864         /// Mueve la reproducción a la pista anterior en la lista de reproducción o al historial de reproducción, dependiendo
      de la reproducción actual
00865         /// modo.
```

```
00866         /// </summary>
00867         /// <remarks>En el modo de mezcla, este método navega hacia atrás a través del historial de reproducción si
00868         /// posible. En el modo normal, se mueve a la pista anterior en la lista de reproducción a menos que ya esté en la
      primera pista.
00869         /// No se toma ninguna medida si no hay pistas en la lista de reproducción o si ya está al principio del historial o
00870         /// lista de reproducción. </remarks>
00871         private void BackCancion()
00872         {
00873             if (_colaReproduccion != null && _colaReproduccion.Count > 0)
00874             {
00875                 // --- MODO ALEATORIO ---
00876                 if (_btnaleatorioActivo)
00877                 {
00878                     // Solo retrocedemos si el puntero no está en el principio del historial
00879                     if (_indiceHistorialModoAleatorio > 0)
00880                     {
00881                         _indiceHistorialModoAleatorio--;
00882                         var cancionAnterior = _historialAleatorio[_indiceHistorialModoAleatorio];
00883                         ActualizarIconoNextBack();
00884                         CargarYReproducir(cancionAnterior);
00885                     }
00886                 }
00887                 else // --- MODO NORMAL ---
00888                 {
00889                     // Solo retrocedemos si no es la primera
00890                     if (_indiceCancionActual > 0)
00891                     {
00892                         _indiceCancionActual--;
00893                         ActualizarIconoNextBack();
00894                         CargarYReproducir(_colaReproduccion[_indiceCancionActual]);
00895                     }
00896                 }
00897             }
00898         }
00899         /// <summary>
00900         /// Actualiza los iconos de los botones de navegación Siguiente y Atrás en función del modo y la posición de
      reproducción actuales
00901         /// en la lista de reproducción.
00902         /// </summary>
00903         /// <remarks>Este método habilita o deshabilita los iconos de los botones Siguiente y Atrás, dependiendo de si
00904         /// la lista de reproducción está en modo de mezcla y la posición actual de la canción. En el modo de mezcla, el
      botón Siguiente permanece
00905         /// activado, mientras que el botón Atrás solo está activado si hay un historial de reproducción. En modo normal, los
      botones están
00906         /// activado o desactivado según si la canción actual es la primera o la última en la lista de reproducción. </remarks>
00907         private void ActualizarIconoNextBack()
00908         {
00909             // CASO 1: No hay lista o la lista está vacía o solo tiene 1 canción
00910             if (_colaReproduccion == null || _colaReproduccion.Count <= 1)
00911             {
00912                 IconoBack = "Img_Back_Disabled";
00913                 IconoNext = "Img_Next_Disabled";
00914                 return; // Salimos, no hay nada más que calcular
00915             }
00916 
00917             if(_btnaleatorioActivo)// MODO ALEATORIO
00918             {
00919                 // En Aleatorio, NEXT siempre está activo (es infinito)
00920                 IconoNext = "Img_Next";
00921 
00922                 // BACK depende del historial
00923                 if (_indiceHistorialModoAleatorio > 0)
00924                     IconoBack = "Img_Back";
00925                 else
00926                     IconoBack = "Img_Back_Disabled";
00927             }
00928             else // MODO NORMAL
00929             {
00930                 // CASO 2: Botón ATRÁS (Back)
00931                 // Si el índice es 0 (primera canción), lo desactivamos. Si no, lo activamos.
00932                 if (_indiceCancionActual == 0)
00933                 {
00934                     IconoBack = "Img_Back_Disabled";
00935                 }
00936                 else
00937                 {
00938                     IconoBack = "Img_Back";
00939                 }
00940                 // CASO 3: Botón SIGUIENTE (Next)
00941                 // Si el índice es el último (Total - 1), lo desactivamos. Si no, lo activamos.
00942                 if (_indiceCancionActual == _colaReproduccion.Count - 1)
00943                 {
00944                     IconoNext = "Img_Next_Disabled";
00945                 }
00946                 else
00947                 {
00948                     IconoNext = "Img_Next";
```

```
00949                 }
00950             }
00951         }
00952         /// <summary>
00953         /// Actualiza el icono de reproducción aleatoria según la cola de reproducción actual y el estado del modo aleatorio.
00954         /// </summary>
00955         /// <remarks>Si la cola de reproducción contiene una o ninguna canción, el icono aleatorio está desactivado. Cuando el
00956         /// el modo aleatorio está activo y la cola cambia, el historial de reproducción aleatoria se restablece para comenzar desde el
00957         /// canción especificada. </remarks>
00958         /// <param name="cancionInicio">La canción que se usará como punto de partida en el historial de reproducción aleatoria cuando el modo aleatorio esté activo. </param>
00959         private void ActualizarIconoAleatorio(Canciones cancionInicio)
00960         {
00961             // CASO A: Lista insuficiente (1 o 0 canciones) -> SE DESHABILITA
00962             if (_colaReproduccion == null || _colaReproduccion.Count <= 1)
00963             {
00964                 IconoAleatorio = "Img_Aleatorio_Disabled";
00965             }
00966             // CASO B: Lista válida (> 1 canción) -> SE HABILITA
00967             else
00968             {
00969                 // Respetamos si el usuario ya lo tenía activado
00970                 if (_btnaleatorioActivo)
00971                 {
00972                     IconoAleatorio = "Img_Aleatorio_ON";
00973 
00974                     // IMPORTANTE: Como ha cambiado la lista, reiniciamos el historial
00975                     // para empezar de cero con la nueva playlist.
00976                     _historialAleatorio.Clear();
00977                     _historialAleatorio.Add(cancionInicio);
00978                     _indiceHistorialModoAleatorio = 0;
00979 
00980                     System.Diagnostics.Debug.WriteLine("[ALEATORIO] Lista cambiada. Historial reiniciado.");
00981                 }
00982                 else
00983                 {
00984                     IconoAleatorio = "Img_Aleatorio_OFF";
00985                 }
00986             }
00987         }
00988         /// <summary>
00989         /// Maneja las marcas de eventos del temporizador para actualizar el progreso de reproducción, la hora actual y la duración total del medio
00990         /// reproductor, o para avanzar a la siguiente pista cuando termina la reproducción.
00991         /// </summary>
00992         /// <remarks>Este método está destinado a ser utilizado como un manejador de eventos para un temporizador periódico.
00993         /// asociado con la reproducción de medios. Actualiza los elementos de la interfaz de usuario, como el deslizador de progreso y las pantallas de tiempo.
00994         /// respuesta al estado actual del reproductor multimedia. Si la reproducción ha terminado, avanza automáticamente a la
00995         /// siguiente pista o restablece la interfaz de usuario según sea apropiado. </remarks>
00996         /// <param name="sender">La fuente del evento, normalmente el temporizador que activó la marca. </param>
00997         /// <param name="e">Un objeto que contiene los datos del evento. </param>
00998         private void Timer_Tick(object? sender, EventArgs e)
00999         {
01000             // Solo actualizamos si VLC está reproduciendo y si ya sabe cuánto dura la canción
01001             if (_mediaPlayer.IsPlaying && _mediaPlayer.Length > 0)
01002             {
01003                 // Actualizar slider de progreso de la canción
01004 
01005                 // _mediaPlayer.Position va de 0.0 a 1.0 (es un porcentaje)
01006                 // Lo multiplicamos por 100 para que coincida con nuestro Slider (Maximum=100)
01007                 // Usamos Math.Min y Max para evitar errores raros de desbordamiento
01008                 double nuevoValor = _mediaPlayer.Position * 100;
01009                 ValorSliderCancion = Math.Clamp(nuevoValor, 0, 100);
01010 
01011                 // Actualizar tiempos de la canción
01012 
01013                 // Tiempo actual
01014                 var tiempoActual = TimeSpan.FromMilliseconds(_mediaPlayer.Time);
01015                 TiempoActualCancion = tiempoActual.ToString(@"mm\:ss");
01016 
01017                 // Tiempo total (Duración)
01018                 // Solo lo calculamos si VLC ya ha descargado los metadatos de la duración
01019                 var tiempoTotal = TimeSpan.FromMilliseconds(_mediaPlayer.Length);
01020                 TiempoTotalCancion = tiempoTotal.ToString(@"mm\:ss");
01021             }
01022             else
01023             {
01024                 // Si la canción ha terminado (llegó al final)
01025                 if (_mediaPlayer.State == VLCState.Ended)
01026                 {
01027                     // Paramos el timer un momento para que no se repita el evento
01028                     _timer.Stop();
```

```
01029
01030                   // Llamamos a la siguente canción automáticamente
01031                   NextCancion();
01032               }
01033               // Si la canción se detuvo o error
01034               else if (_mediaPlayer.State == VLCState.Stopped || _mediaPlayer.State == VLCState.Error)
01035               {
01036                   _timer.Stop();
01037                   IconoPlayPause = "Img_Play";
01038                   ValorSliderCancion = 0;
01039                   TiempoActualCancion = "00:00";
01040               }
01041           }
01042       }
01043       #endregion
01044
01045       #region Métodos helpers
01046
01047       /// <summary>
01048       /// Actualiza los iconos y las propiedades relacionadas para reflejar el tema de la aplicación actual.
01049       /// </summary>
01050       /// <remarks>Llame a este método después de que cambie el tema de la aplicación para asegurarse de que todos los iconos y
01051       /// las propiedades del deslizador se actualizan y notifican cualquier vinculación de datos de los cambios. </remarks>
01052       private void RefrescarIconos()
01053       {
01054           //Forzamos la actualización de los iconos para que se recarguen con el nuevo tema (Binding)
01055           this.RaisePropertyChanged(nameof(IconoPlayPause));
01056           this.RaisePropertyChanged(nameof(IconoNext));
01057           this.RaisePropertyChanged(nameof(IconoBack));
01058           this.RaisePropertyChanged(nameof(IconoAleatorio));
01059           this.RaisePropertyChanged(nameof(IconoLike));
01060           this.RaisePropertyChanged(nameof(ValorSliderCancion));
01061       }
01062
01063       /// <summary>
01064       /// Cierra la sesión actual del usuario y restablece el estado de la aplicación a la vista de inicio de sesión.
01065       /// </summary>
01066       /// <remarks>Este método detiene cualquier reproducción de medios activa, borra datos específicos del usuario y restablece la interfaz de usuario.
01067       /// elementos y elimina la información de usuario almacenada en caché para mayor seguridad. Después de la ejecución, la aplicación regresa al
01068       /// pantalla de inicio de sesión, asegurándose de que ningún dato de sesiones anteriores permanezca accesible. Este método debe ser llamado cuando un
01069       /// el usuario se desconecta o cuando una sesión debe finalizar de forma segura. </remarks>
01070       private void CerrarSesion()
01071       {
01072           // Parar música si esta sonando
01073           if (_mediaPlayer.IsPlaying) _mediaPlayer.Stop();
01074
01075           //Dejamos el reproductor
01076           _timer.Stop();
01077           IconoPlayPause = "Img_Play";
01078           ValorSliderCancion = 0;
01079           TiempoActualCancion = "--:--";
01080           TiempoTotalCancion = "--:--";
01081           NombreCancionActual = "";
01082           NombreArtistaActual = "";
01083           ImagenCancionActual = "https://i.ibb.co/v6CJTMX2/Icono-Musica.jpg";
01084
01085           // Limpiamos los datos globales
01086           GlobalData.Instance.ClearUserData();
01087           BarraVisible = false;
01088
01089           // Limpiamos todas las vistas que puedan contener datos del usuario
01090           _centralTabVM = null;
01091           _panelUsuarioVM = null;
01092
01093           // Limpiamos los campos del LoginVM existente para que no salgan rellenos
01094           _loginVM.TxtUsuario = "";
01095           _loginVM.TxtPass = "";
01096
01097           // Volvemos a la vista Login original
01098           VistaActual = _loginVM;
01099       }
01100       /// <summary>
01101       /// Realiza un cierre limpio de la aplicación, finalizando los procesos relacionados, liberando recursos, y
01102       /// cerrando la ventana de la aplicación.
01103       /// </summary>
01104       /// <remarks>Este método termina por la fuerza cualquier instancia en ejecución de 'BetaProyecto.API'
01105       /// proceso, descarte de recursos multimedia, detiene temporizadores internos y cierra la aplicación. Si el
01106       /// la aplicación se ejecuta con una vida útil de escritorio clásica, utiliza el mecanismo de apagado apropiado;
01107       /// de lo contrario, abandona el proceso. Utilice este método para asegurarse de que todos los recursos se liberan y la aplicación
01108       /// salidas limpias. </remarks>
01109       private void CerrarAplicacion()
01110       {
```

```
01111           try
01112           {
01113               // Buscamos cualquier proceso que se llame como tu API
01114               var procesosApi = Process.GetProcessesByName("BetaProyecto.API");
01115               foreach (var proc in procesosApi)
01116               {
01117                   System.Diagnostics.Debug.WriteLine($"Cerrando API: {proc.ProcessName}");
01118                   proc.Kill(); // Forzamos el cierre
01119               }
01120           }
01121           catch (Exception ex)
01122           {
01123               System.Diagnostics.Debug.WriteLine("Error al cerrar la API: " + ex.Message);
01124           }
01125
01126           // Limpiamos recursos multimedia
01127           if (_mediaPlayer != null)
01128           {
01129               _mediaPlayer.Stop();
01130               _mediaPlayer.Dispose();
01131           }
01132           LimpiarArchivoTemporal();
01133           if (_libVLC != null) _libVLC.Dispose();
01134
01135           _timer.Stop();
01136
01137           // Cerramos la aplicación
01138           if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)
01139           {
01140               desktop.Shutdown();
01141           }
01142           else
01143           {
01144               Environment.Exit(0);
01145           }
01146       }
01147
01148       /// <summary>
01149       /// Elimina el archivo temporal actual si existe y libera cualquier recurso multimedia asociado.
01150       /// </summary>
01151       /// <remarks>Si el archivo temporal está en uso o no se puede eliminar, el método suprime
01152       /// excepciones y registra un mensaje de depuración. Este método está destinado a ser llamado cuando los archivos
      multimedia temporales no están
01153       /// se necesita más tiempo para liberar espacio en disco y liberar atajos de archivos. </remarks>
01154       private void LimpiarArchivoTemporal()
01155       {
01156           try
01157           {
01158               // Si tenemos una ruta guardada y el archivo existe...
01159               if (!string.IsNullOrEmpty(_rutaTemporalActual) && System.IO.File.Exists(_rutaTemporalActual))
01160               {
01161                   if (_mediaPlayer.Media != null)
01162                   {
01163                       _mediaPlayer.Media.Dispose(); // Destruye el enlace al archivo
01164                       _mediaPlayer.Media = null;    // Limpia la propiedad del reproductor
01165                   }
01166
01167                   // Borramos el archivo temporal
01168                   System.IO.File.Delete(_rutaTemporalActual);
01169                   System.Diagnostics.Debug.WriteLine($"[SEGURIDAD] Archivo temporal eliminado:
      {_rutaTemporalActual}");
01170                   _rutaTemporalActual = "";
01171               }
01172           }
01173           catch (Exception ex)
01174           {
01175               // Si falla (por ejemplo, si VLC todavía lo tiene bloqueado), no pasa nada,
01176               // Windows limpia los temporales eventualmente.
01177               System.Diagnostics.Debug.WriteLine($"[AVISO] No se pudo borrar el temporal: {ex.Message}");
01178           }
01179       }
01180       #endregion
01181
01182   }
01183 }
```

4.93. Referencia del archivo BetaProyecto/ViewModels/PanelUsuarioViewModel.cs

4.94. PanelUsuarioViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Singleton;
00003 using ReactiveUI;
00004 using System;
00005
00006 namespace BetaProyecto.ViewModels
00007 {
00008     public class PanelUsuarioViewModel : ViewModelBase, INavegable
00009     {
00010         //Sub-ViewModels
00011         public ViewPerfilViewModel ViewPerfilVM { get; }
00012         public ViewCuentaViewModel ViewCuentaVM { get; }
00013         public ViewGestionarCuentaViewModel ViewGestionarCuentaVM { get; }
00014         public ViewConfiguracionViewModel ViewConfiguracionVM { get; }
00015         public ViewGestionarReportesViewModel ViewGestionarReportesVM { get; }
00016         public ViewGestionarBDViewModel ViewGestionarBDVM { get; }
00017
00018         //Actions
00019         public Action VolverAtras { get; set; }
00020         public Action AccionLogout { get; set; }
00021         public Action AccionSalir { get; set; }
00022         public Action<ListaPersonalizada> IrAEditarPlaylist { get; set;}
00023         public Action<Canciones> IrAEditarCancion { get; set; }
00024         public Action? AccionRefrescarDesdePadre { get; set; }
00025
00026         // 0 = Perfil, 1 = Cuenta, 2 = GestionarCuenta, 3 = Configuración
00027         private int _indiceTab;
00028         public int IndiceTab
00029         {
00030             get => _indiceTab;
00031             set => this.RaiseAndSetIfChanged(ref _indiceTab, value);
00032         }
00033
00034         // Solo para SuperAdmin
00035         private bool _puedeVerBD;
00036         public bool PuedeVerBD
00037         {
00038             get => _puedeVerBD;
00039             set => this.RaiseAndSetIfChanged(ref _puedeVerBD, value);
00040         }
00041
00042         // Para Admin y SuperAdmin
00043         private bool _puedeVerReportes;
00044         public bool PuedeVerReportes
00045         {
00046             get => _puedeVerReportes;
00047             set => this.RaiseAndSetIfChanged(ref _puedeVerReportes, value);
00048         }
00049         // Constructor
00050         // Le pasamos el índice inicial (por defecto 0 si no decimos nada) para la pestaña
00051         public PanelUsuarioViewModel(int tabInicial = 0)
00052         {
00053             IndiceTab = tabInicial;
00054
00055             //Inicializamos los sub-ViewModels
00056             ViewPerfilVM = new ViewPerfilViewModel();
00057             ViewCuentaVM = new ViewCuentaViewModel();
00058             ViewGestionarCuentaVM = new ViewGestionarCuentaViewModel();
00059
00060             ViewGestionarCuentaVM.SolicitudIrAEditarPlaylist = (playlist) =>
00061             {
00062                 IrAEditarPlaylist?.Invoke(playlist);
00063             };
00064
00065             ViewGestionarCuentaVM.SolicitudIrAEditarCanciones = (canciones) =>
00066             {
00067                 IrAEditarCancion?.Invoke(canciones);
00068             };
00069
00070             ViewConfiguracionVM = new ViewConfiguracionViewModel(
00071                         accionVolver: () => VolverAtras?.Invoke(),
00072                         accionLogout: () => AccionLogout?.Invoke(),
00073                         accionSalir: () => AccionSalir?.Invoke(),
00074                         accionRefrescar: () => AccionRefrescarDesdePadre?.Invoke()
00075             );
00076             ConfigurarPermisos();
00077
00078             ViewGestionarReportesVM = new ViewGestionarReportesViewModel();
00079             ViewGestionarBDVM = new ViewGestionarBDViewModel();
00080         }
00081         /// <summary>
00082         /// Evalúa y establece los privilegios de acceso del usuario a las funciones administrativas del panel basándose en su
      rol.
00083         /// </summary>
00084         /// <remarks>
00085         /// Este método consulta el rol actual desde <see cref="GlobalData.Instance.RolGD"/> y actualiza las propiedades
00086         /// de visibilidad de la interfaz. La gestión de base de datos se restringe exclusivamente al rol <see
```

```
    cref="Roles.SuperAdmin"/>,
00087       /// mientras que el acceso a reportes se habilita tanto para administradores como para superadministradores.
00088       /// </remarks>
00089       private void ConfigurarPermisos()
00090       {
00091           string rolActual = GlobalData.Instance.RolGD;
00092
00093           // Gestión de Base de Datos -> SOLO SuperAdmin
00094           PuedeVerBD = (rolActual == Roles.SuperAdmin);
00095
00096           // Gestión de Reportes -> SuperAdmin O Admin
00097           PuedeVerReportes = (rolActual == Roles.SuperAdmin || rolActual == Roles.Admin);
00098       }
00099   }
00100 }
```

4.95. Referencia del archivo BetaProyecto/ViewModels/TabItemBuscadorViewModel.cs

4.96. TabItemBuscadorViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Singleton;
00003 using ReactiveUI;
00004 using System.Collections.ObjectModel;
00005 using System.Reactive;
00006 using System.Threading.Tasks;
00007
00008 namespace BetaProyecto.ViewModels
00009 {
00010     public class TabItemBuscadorViewModel : ViewModelBase
00011     {
00012         //Bidings
00013         private string _txtBusqueda;
00014         public string TxtBusqueda
00015         {
00016             get => _txtBusqueda;
00017             set => this.RaiseAndSetIfChanged(ref _txtBusqueda, value);
00018         }
00019         private string _txtInfoResultado;
00020         public string TxtInfoResultado
00021         {
00022             get => _txtInfoResultado;
00023             set => this.RaiseAndSetIfChanged(ref _txtInfoResultado, value);
00024         }
00025
00026         // Propiedad extra para el número (para usar con Run en XAML)
00027         private string _txtContador;
00028         public string TxtContador
00029         {
00030             get => _txtContador;
00031             set => this.RaiseAndSetIfChanged(ref _txtContador, value);
00032         }
00033
00034         private ObservableCollection<Canciones> _listaBusqueda;
00035         public ObservableCollection<Canciones> ListaBusqueda
00036         {
00037             get => _listaBusqueda;
00038             set => this.RaiseAndSetIfChanged(ref _listaBusqueda, value);
00039         }
00040
00041         //Comandos Reactive
00042         public ReactiveCommand<Unit, Unit> BtnBuscar { get; }
00043
00044         public TabItemBuscadorViewModel()
00045         {
00046             //Inicialización de lista
00047             ListaBusqueda = new ObservableCollection<Canciones>();
00048
00049             // Validación para habilitar el botón de búsqueda solo cuando haya texto
00050             var validacionBuscar = this.WhenAnyValue(
00051                 x => x.TxtBusqueda,
00052                 (textoABuscar) => !string.IsNullOrWhiteSpace(textoABuscar)
00053             );
00054             //Configuramos comandos reactive
00055             BtnBuscar = ReactiveCommand.CreateFromTask(BuscarEnBD, validacionBuscar);
00056         }
00057
```

```
00058         /// <summary>
00059         /// Realiza una búsqueda asíncrona de canciones en la base de datos utilizando el texto de búsqueda actual y
      actualiza el
00060         /// resultados de búsqueda y propiedades de estado relacionadas.
00061         /// </summary>
00062         /// <remarks>Si la conexión a la base de datos no está disponible, el método actualiza las propiedades de estado
00063         /// para indicar un error de conexión. Los resultados de búsqueda y las propiedades de estado se actualizan en
      función de si hay alguno
00064         /// las canciones coincidentes se encuentran. </remarks>
00065         /// <returns>Devuelve una tarea que representa la operación de búsqueda asíncrona. </returns>
00066         private async Task BuscarEnBD()
00067         {
00068             TxtInfoResultado = "Bus_Msg_Buscando";
00069             TxtContador = "";
00070 
00071             if (MongoClientSingleton.Instance.Cliente != null)
00072             {
00073                 var listaResultadosBusqueda = await
      MongoClientSingleton.Instance.Cliente.ObtenerCancionesPorBusqueda(TxtBusqueda);
00074 
00075                 if (listaResultadosBusqueda != null && listaResultadosBusqueda.Count > 0)
00076                 {
00077                     ListaBusqueda = new ObservableCollection<Canciones>(listaResultadosBusqueda);
00078                     TxtInfoResultado = "Bus_Res_Encontrados";
00079                     TxtContador = $" {listaResultadosBusqueda.Count}";
00080                 }
00081                 else
00082                 {
00083                     ListaBusqueda.Clear();
00084                     TxtInfoResultado = "Bus_Res_SinResultados";
00085                     TxtContador = "";
00086                 }
00087             }
00088             else
00089             {
00090                 TxtInfoResultado = "Msg_Error_Conexion";
00091                 TxtContador = "";
00092             }
00093         }
00094     }
00095 }
```

4.97. Referencia del archivo BetaProyecto/ViewModels/TabItemInicioViewModel.cs

4.98. TabItemInicioViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Controls;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Services;
00004 using BetaProyecto.Singleton;
00005 using ReactiveUI;
00006 using System;
00007 using System.Collections.Generic;
00008 using System.Collections.ObjectModel;
00009 using System.Linq;
00010 using System.Reactive;
00011 using System.Threading.Tasks;
00012 
00013 namespace BetaProyecto.ViewModels
00014 {
00015     public class TabItemInicioViewModel : ViewModelBase
00016     {
00017         //Servicio de diálogo
00018         private readonly IDialogoService _dialogoService;
00019 
00020         //Actions
00021         public Action<Canciones, List<Canciones»? EnviarReproduccion { get; set; }
00022         public Action<Canciones>? SolicitudVerDetalles { get; set; }
00023         public Action<string>? SolicitudVerArtista { get; set; }
00024         public Action<Canciones>? SolicitudCrearReporte { get; set; }
00025         public Action<ListaPersonalizada>? SolicitudVerDetallasPlaylist { get; set; }
00026 
00027         //Comandos Reactive
00028         public ReactiveCommand<object, Unit> BtnReproducirDesdeTarjeta { get; }
00029         public ReactiveCommand<ListaPersonalizada, Unit> BtnReproducirPlaylist { get; }
00030         public ReactiveCommand<Unit, Unit> BtnRefrescar { get; }
00031 
00032         // Menú de los 3 puntos Canciones
00033         public ReactiveCommand<Canciones, Unit> BtnIrADetalleCancion { get; }
```

```
00034         public ReactiveCommand<object, Unit> BtnIrAArtista { get; }
00035         public ReactiveCommand<Canciones, Unit> BtnIrAReportar { get; }
00036
00037         //Menú de los 3 puntos Playlists
00038         public ReactiveCommand<ListaPersonalizada, Unit> BtnIrADetallesPlaylist { get; }
00039
00040         // Biding que contiene las listas (Novedades, Rock, etc.)
00041         private ObservableCollection<TarjetasCanciones> _tarjetas;
00042         public ObservableCollection<TarjetasCanciones> Tarjetas
00043         {
00044             get => _tarjetas;
00045             set => this.RaiseAndSetIfChanged(ref _tarjetas, value);
00046         }
00047
00048         private ObservableCollection<TarjetasListas> _playlists;
00049         public ObservableCollection<TarjetasListas> Playlists
00050         {
00051             get => _playlists;
00052             set => this.RaiseAndSetIfChanged(ref _playlists, value);
00053
00054         }
00055         private string _txtFv;
00056         public string TxtFav
00057         {
00058             get => _txtFv;
00059             set => this.RaiseAndSetIfChanged(ref _txtFv, value);
00060         }
00061
00062         public TabItemInicioViewModel()
00063         {
00064             // Inicializamos el servicios
00065             _dialogoService = new DialogoService();
00066
00067             // Configuramos los comandos reactive
00068             BtnReproducirDesdeTarjeta = ReactiveCommand.Create<object>(ReproducirDesdeBoton);
00069
00070             BtnReproducirPlaylist = ReactiveCommand.Create<ListaPersonalizada>(ReproducirPlaylist);
00071
00072             BtnIrADetalleCancion = ReactiveCommand.Create<Canciones>(cancion =>
00073             {
00074                 System.Diagnostics.Debug.WriteLine($"Solicitando detalles de: {cancion.Titulo}");
00075                 SolicitudVerDetalles?.Invoke(cancion);
00076             });
00077
00078             BtnIrAArtista = ReactiveCommand.Create<object>(IrAArtistaDesdeBoton);
00079
00080             BtnIrAReportar = ReactiveCommand.Create<Canciones>(cancion =>
00081             {
00082                 System.Diagnostics.Debug.WriteLine($"Creando reporte de: {cancion.Titulo} con el id {cancion.Id}");
00083                 SolicitudCrearReporte?.Invoke(cancion);
00084             });
00085             BtnIrADetallesPlaylist = ReactiveCommand.Create<ListaPersonalizada>(playlist =>
00086             {
00087                 System.Diagnostics.Debug.WriteLine($"Solicitando detalles de lista: {playlist.Nombre}");
00088                 SolicitudVerDetallasPlaylist?.Invoke(playlist);
00089
00090             });
00091             BtnRefrescar = ReactiveCommand.CreateFromTask(async () =>
00092             {
00093                 await CargarDatosCanciones();
00094             });
00095             // Ejecutamos la tarea en segundo plano para no bloquear la interfaz
00096             _ = CargarDatosCanciones();
00097         }
00098         /// <summary>
00099         /// Dirige la navegación a la vista de un artista cuando se activa mediante un botón asociado con el nombre del
      artista.
00100         /// </summary>
00101         /// <remarks>Este método se utiliza típicamente como un manejador de comandos para elementos de la interfaz de
      usuario que representan
00102         /// artistas dentro de un contexto de canción. Recupera el artista y la información de la canción relevante desde el
      botón
00103         /// jerarquía y plantea una solicitud para mostrar los detalles del artista. El parámetro debe estructurarse como
00104         /// descrito para que la navegación tenga éxito. </remarks>
00105         /// <param name="parametro">El parametro de comando, que se espera sea un botón cuyo DataContext contiene
      el nombre del artista y cuya etiqueta
00106         /// hace referencia al botón padre que contiene la información de la canción. </param>
00107         private void IrAArtistaDesdeBoton(object parametro)
00108         {
00109             // Recuperamos el objeto de donde viene el comando (El botón con el nombre del artista)
00110             // y lo casteamos a Button para sacarle la inforamación que necesitamos
00111             if (parametro is Button botonPequeño)
00112             {
00113                 // Recuperamos el Nombre del Artista (DataContext del botón (Que seria un Objeto Canciones))
00114                 var nombreArtista = botonPequeño.DataContext as string;
00115
00116                 // Recuperamos el botón "jefe" (el abre el menú contextual donde esta el nombre del artistas)
```

```
00117               // Que guardamos su referencia en el Tag del botón pequeño
00118               if (botonPequeño.Tag is Button botonJefe)
00119               {
00120                   //Ahora le decimos que se oculte par que no se quede abierto al pasar la vista artista
00121                   botonJefe.Flyout?.Hide();
00122
00123                   // Y recuperamos la canción (DataContext del botón jefe)
00124                   var cancion = botonJefe.DataContext as Canciones;
00125
00126                   // Aseguramos que no esten vacíos
00127                   if (!string.IsNullOrEmpty(nombreArtista) && cancion != null)
00128                   {
00129                       // Buscamos el índice del artista en la lista del artista que queremos mostrar
00130                       int indice = cancion.ListaArtistasIndividuales.IndexOf(nombreArtista);
00131                       // Con ese índice, buscamos el ID del artista en la lista de IDs (que es paralela a la de nombres)
00132                       if (indice >= 0 && cancion.AutoresIds != null && indice < cancion.AutoresIds.Count)
00133                       {
00134                           string idUsuario = cancion.AutoresIds[indice];//Sacamos el id del artista a mostrar
00135                         SolicitudVerArtista?.Invoke(idUsuario);// Enviamos la solicitud para mostrar la vista del artista con su
      ID
00136                       }
00137                   }
00138               }
00139           }
00140       }
00141       /// <summary>
00142       /// Maneja el comando de reproducción disparado desde un botón, iniciando la reproducción de la canción
      seleccionada y su
00143       /// colección asociada.
00144       /// </summary>
00145       /// <remarks>Este método se utiliza típicamente como un manejador de comandos para botones de reproducción en
      el usuario
00146       /// interfaz. El DataContext del botón debe hacer referencia a la canción que se va a reproducir, y su etiqueta debe
      hacer referencia al
00147       /// colección a la que pertenece la canción. Si falta alguno de los valores o es inválido, el método lo hace
00148       /// nada. </remarks>
00149       /// <param name="parametro">El parámetro de comando, que se espera sea un botón cuyo DataContext es una
      canción a reproducir y cuya etiqueta contiene
00150       /// la colección de canciones. No debe ser nula y debe ser un botón con valores válidos de DataContext y Tag.
      </param>
00151       private void ReproducirDesdeBoton(object parametro)
00152       {
00153           // Recuperamos el objeto de donde viene el comando (El botón de Play)
00154           // y lo casteamos a Button para sacar la información que necesitamos
00155           if (parametro is Button boton)
00156           {
00157               // Recuperamos la Canción a reproducir (DataContext del botón (Que sería el Objeto Canciones))
00158               var cancion = boton.DataContext as Canciones;
00159
00160               // Recuperamos la lista completa a la que pertenece esa canción (la lista de origen)
00161               // Que guardamos en el Tag del botón para saber el contexto (si viene de Populares, Buscador, etc.)
00162               // Viene como 'IEnumerable', así que la casteamos para poder trabajar con ella
00163               var coleccionOrigen = boton.Tag as IEnumerable<Canciones>;
00164
00165               // Aseguramos que hemos recuperado bien tanto la canción como su lista de origen
00166               if (cancion != null && coleccionOrigen != null)
00167               {
00168                   // Convertimos la colección a una Lista concreta para asegurarnos de pasar una copia exacta al reproductor
00169                   var listaExacta = coleccionOrigen.ToList();
00170
00171                   // Enviamos la solicitud de reproducción al padre (MarcoApp) pasando:
00172                   // 1. La canción concreta que se ha pulsado (para que empiece por ahí)
00173                   // 2. La lista completa de contexto (para que sepa qué poner cuando acabe esta)
00174                   EnviarReproduccion?.Invoke(cancion, listaExacta);
00175               }
00176           }
00177       }
00178       /// <summary>
00179       /// Inicia la reproducción de la lista de reproducción especificada, reproduciendo la primera canción y poniendo en
      fila las canciones restantes para
00180       /// reproducción.
00181       /// </summary>
00182       /// <remarks>Si la lista de reproducción es nula o no contiene canciones, se muestra una alerta para informar el
00183       /// usuario que la lista de reproducción está vacía. </remarks>
00184       /// <param name="playlist">La lista de reproducción a reproducir. No debe ser nula y debe contener al menos una
      canción. </param>
00185       private void ReproducirPlaylist(ListaPersonalizada playlist)
00186       {
00187           if (playlist != null && playlist.CancionesCompletas.Count > 0)
00188           {
00189               // La primera canción que sonará
00190               var primeraCancion = playlist.CancionesCompletas[0];
00191
00192               // La lista completa (cola de reproducción)
00193               var cola = playlist.CancionesCompletas;
00194
00195               // Enviamos al MarcoApp
```

```
00196                 EnviarReproduccion?.Invoke(primeraCancion, cola);
00197             }
00198             else
00199             {
00200                 // "Esta lista está vacía."
00201                 _dialogoService.MostrarAlerta("Msg_Error_PlaylistVacia");
00202             }
00203         }
00204         /// <summary>
00205         /// Carga y organiza de forma asíncrona los datos de canciones y listas de reproducción desde la base de datos, actualizando los correspondientes
00206         /// colecciones para la vinculación de datos.
00207         /// </summary>
00208         /// <remarks>Si la conexión a la base de datos no está disponible, se muestra una alerta y no hay datos
00209         /// cargado. Las canciones y listas de reproducción se agrupan en secciones como favoritas, nuevos lanzamientos, rock, personalizadas
00210         /// listas y listas de comunidades, asignadas a sus respectivas colecciones para la vinculación de la interfaz. </remarks>
00211         /// <returns>Devuelve una tarea que representa la operación de carga asíncrona. </returns>
00212         private async Task CargarDatosCanciones()
00213         {
00214             if (MongoClientSingleton.Instance.Cliente == null)
00215             {
00216                 // "Error de conexión a la base de datos"
00217                 _dialogoService.MostrarAlerta("Msg_Error_Conexion");
00218             }
00219             else
00220             {
00221                 var cliente = MongoClientSingleton.Instance.Cliente;
00222                 var miIdUsuario = GlobalData.Instance.UserIdGD;
00223 
00224                 // Lógica de carga
00225                 //-Listade canciones
00226                 var songsNovedades = cliente.ObtenerCacionesNovedades();
00227                 var songsFavoritos = cliente.ObtenerCancionesFavoritos();
00228                 var songsRock = cliente.ObtenerCancionesPorGenero("Rock");
00229                 var songsGeneral = cliente.ObtenerCanciones();
00230                 //-Playlist
00231                 var taskPlaylists = cliente.ObtenerListasReproduccion();
00232 
00233                 // Esperamos todo a la vez (Paralelismo)
00234                 await Task.WhenAll(songsNovedades, songsFavoritos, songsRock, songsGeneral, taskPlaylists);
00235 
00236                 // Creamos la lista de secciones
00237                 var listaTarjetas = new ObservableCollection<TarjetasCanciones>();
00238 
00239                 // --- PROCESAR CANCIONES (TarjetasCanciones) ---
00240                 // OJO: Estas claves "Sec_..." vienen de un Diccionario si posieramos el texto directamente,
00241                 // se podria direcamente esa palabra pero no cambiaria entre idiomas
00242 
00243                 if (songsFavoritos.Result.Count > 0)
00244                     listaTarjetas.Add(new TarjetasCanciones("Sec_Favoritos", new ObservableCollection<Canciones>(songsFavoritos.Result))); // "Favoritos "
00245 
00246                 listaTarjetas.Add(new TarjetasCanciones("Sec_Novedades", new ObservableCollection<Canciones>(songsNovedades.Result))); // "Novedades "
00247 
00248                 if (songsRock.Result.Count > 0)
00249                     listaTarjetas.Add(new TarjetasCanciones("Sec_Rock", new ObservableCollection<Canciones>(songsRock.Result))); // "Rock "
00250 
00251                 listaTarjetas.Add(new TarjetasCanciones("Sec_ParaTi", new ObservableCollection<Canciones>(songsGeneral.Result))); // "Para ti "
00252 
00253 
00254                 Tarjetas = listaTarjetas;// Asignamos la lista al Binding
00255 
00256                 // --- PROCESAR LISTAS (TarjetasListas) ---
00257                 // Aquí separamos las listas en secciones
00258                 var todasListas = taskPlaylists.Result;
00259                 var listaPlaylist = new ObservableCollection<TarjetasListas>();
00260 
00261                 // Mis Listas (Filtramos por mi ID)
00262                 var misListas = todasListas.Where(l => l.IdUsuario == miIdUsuario).ToList();
00263                 if (misListas.Count > 0)
00264                 {
00265                     listaPlaylist.Add(new TarjetasListas("Sec_MisListas", new ObservableCollection<ListaPersonalizada>(misListas))); // "Mis Listas  "
00266                 }
00267 
00268                 // Comunidad (El resto)
00269                 var otrasListas = todasListas.Where(l => l.IdUsuario != miIdUsuario).ToList();
00270                 if (otrasListas.Count > 0)
00271                 {
00272                     listaPlaylist.Add(new TarjetasListas("Sec_Comunidad", new ObservableCollection<ListaPersonalizada>(otrasListas))); // "De la Comunidad  "
00273                 }
```

```
00274
00275             Playlists = listaPlaylist;// Asignamos la lista al Binding
00276
00277         }
00278     }
00279 }
00280 }
```

4.99. Referencia del archivo BetaProyecto/ViewModels/TabItemPopularesViewModel.cs

4.100. TabItemPopularesViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Singleton;
00003 using ReactiveUI;
00004 using System.Collections.ObjectModel;
00005 using System.Threading.Tasks;
00006
00007 namespace BetaProyecto.ViewModels
00008 {
00009     public class TabItemPopularesViewModel : ViewModelBase
00010     {
00011
00012         //Binding
00013         private ObservableCollection<string> _listaGeneros;
00014         public ObservableCollection<string> ListaGeneros
00015         {
00016             get => _listaGeneros;
00017             set => this.RaiseAndSetIfChanged(ref _listaGeneros, value);
00018         }
00019
00020         private string _generoSeleccionado;
00021         public string GeneroSeleccionado
00022         {
00023             get => _generoSeleccionado;
00024             set
00025             {
00026                 this.RaiseAndSetIfChanged(ref _generoSeleccionado, value);
00027                 // Al cambiar el valor, buscamos automáticamente
00028                 if (!string.IsNullOrEmpty(value))
00029                     _ = CargarCancionesPorGenero(value);
00030             }
00031         }
00032
00033         private string _txtInfo;
00034         public string TxtInfo
00035         {
00036             get => _txtInfo;
00037             set => this.RaiseAndSetIfChanged(ref _txtInfo, value);
00038         }
00039
00040         private string _txtGeneroMostrado;
00041         public string TxtGeneroMostrado
00042         {
00043             get => _txtGeneroMostrado;
00044             set => this.RaiseAndSetIfChanged(ref _txtGeneroMostrado, value);
00045         }
00046
00047         //Lista de canciones
00048         private ObservableCollection<Canciones> _listaPopulares;
00049         public ObservableCollection<Canciones> ListaPopulares
00050         {
00051             get => _listaPopulares;
00052             set => this.RaiseAndSetIfChanged(ref _listaPopulares, value);
00053         }
00054
00055         public TabItemPopularesViewModel()
00056         {
00057             // Inicializamos listas
00058             ListaGeneros = new ObservableCollection<string>();
00059             ListaPopulares = new ObservableCollection<Canciones>();
00060
00061             // Dejamos esto vacío al inicio o con una clave de "Seleccione..."
00062             TxtInfo = "";
00063             TxtGeneroMostrado = "";
00064
00065             // Ejecutamos tarea en segndo plano para cargar géneros
```

```
00066             _ = CargarGeneros();
00067         }
00068
00069         /// <summary>
00070         /// Carga asincrónicamente la lista de canciones populares para el género especificado y actualiza la pantalla
      relacionada
00071         /// propiedades.
00072         /// </summary>
00073         /// <remarks>Si no se encuentran canciones para el género especificado, las propiedades de visualización se
      actualizan a
00074         /// indica que no se encontraron resultados. Si la conexión a la base de datos no está disponible, se establece un
      mensaje de error
00075         /// en su lugar. </remarks>
00076         /// <param name="genero">El nombre del género para el que se recuperarán las canciones populares. No puede ser
      nulo ni vacío. </param>
00077         /// <returns>Devuelve una tarea que representa la operación de carga asíncrona. </returns>
00078         private async Task CargarCancionesPorGenero(string genero)
00079         {
00080             // "Buscando..."
00081             TxtInfo = "Bus_Msg_Buscando";
00082             TxtGeneroMostrado = "";
00083             ListaPopulares.Clear();
00084
00085             if (MongoClientSingleton.Instance.Cliente != null)
00086             {
00087                 var resultados = await MongoClientSingleton.Instance.Cliente.ObtenerMixPorGenero(genero);
00088
00089                 if (resultados != null && resultados.Count > 0)
00090                 {
00091                     ListaPopulares = new ObservableCollection<Canciones>(resultados);
00092
00093                     // "Mostrando resultados de:" + " Rock"
00094                     TxtInfo = "Pop_Res_Mostrando";
00095                     TxtGeneroMostrado = $" {genero}";
00096                 }
00097                 else
00098                 {
00099                     // "No se encontraron coincidencias."
00100                     TxtInfo = "Bus_Res_SinResultados";
00101                     TxtGeneroMostrado = "";
00102                 }
00103             }
00104             else
00105             {
00106                 TxtInfo = "Msg_Error_Conexion";
00107             }
00108         }
00109
00110         /// <summary>
00111         /// Carga de forma asíncrona la lista de nombres de género desde la base de datos y actualiza la colección utilizada
      por la vista.
00112         /// </summary>
00113         /// <remarks>Si la conexión a la base de datos no está disponible, el método escribe un mensaje de error en el
00114         /// salida de depuración y no se actualiza la lista de géneros. </remarks>
00115         /// <returns>Devuelve una tarea que representa la operación de carga asíncrona. </returns>
00116         private async Task CargarGeneros()
00117         {
00118             if (MongoClientSingleton.Instance.Cliente != null)
00119             {
00120                 var listadeGeneros = await MongoClientSingleton.Instance.Cliente.ObtenerNombresGeneros();
00121                 ListaGeneros = new ObservableCollection<string>(listadeGeneros);
00122             }
00123             else
00124             {
00125                 System.Diagnostics.Debug.WriteLine("Error en la conexion de la base de datos");
00126             }
00127         }
00128     }
00129 }
```

4.101. Referencia del archivo BetaProyecto/ViewModels/VentanaAvisoViewModel.cs

4.102. VentanaAvisoViewModel.cs

Ir a la documentación de este archivo.

```
00001 using ReactiveUI;
00002 using System;
00003 using System.Reactive;
00004
00005 namespace BetaProyecto.ViewModels
```

```
00006 {
00007     public class VentanaAvisoViewModel : ViewModelBase
00008     {
00009         // Texto a mostrar
00010         public string Mensaje { get; }
00011
00012         // Comando para el botón
00013         public ReactiveCommand<Unit, Unit> BtnAceptar { get; }
00014
00015         // Recibimos el mensaje y una "Action" que es la función para cerrar la ventana física
00016         public VentanaAvisoViewModel(string mensaje, Action cerrarVentana)
00017         {
00018             Mensaje = mensaje;
00019
00020             // Al pulsar aceptar, ejecutamos la acción de cerrar
00021             BtnAceptar = ReactiveCommand.Create(() =>
00022             {
00023                 cerrarVentana();
00024             });
00025         }
00026     }
00027 }
```

4.103. Referencia del archivo BetaProyecto/ViewModels/VentanaConfirmacionViewModel.cs

4.104. VentanaConfirmacionViewModel.cs

Ir a la documentación de este archivo.

```
00001 using ReactiveUI;
00002 using System;
00003 using System.Reactive;
00004
00005 namespace BetaProyecto.ViewModels
00006 {
00007     public class VentanaConfirmacionViewModel : ViewModelBase
00008     {
00009         // Propiedades de texto
00010         public string TituloCabecera { get; }
00011         public string MensajeCuerpo { get; }
00012         public string TextoBotonSi { get; }
00013         public string TextoBotonNo { get; }
00014
00015         // Comandos Reactive
00016         public ReactiveCommand<Unit, Unit> BtnSi { get; }
00017         public ReactiveCommand<Unit, Unit> BtnNo { get; }
00018
00019         // La acción recibe un bool: true (Si) o false (No)
00020         public VentanaConfirmacionViewModel(string titulo, string mensaje, string textoSi, string textoNo, Action<bool>
    cerrarConResultado)
00021         {
00022             TituloCabecera = titulo;
00023             MensajeCuerpo = mensaje;
00024             TextoBotonSi = textoSi;
00025             TextoBotonNo = textoNo;
00026
00027             //Configuramos comandos reactive
00028             BtnSi = ReactiveCommand.Create(() => cerrarConResultado(true));
00029             BtnNo = ReactiveCommand.Create(() => cerrarConResultado(false));
00030         }
00031     }
00032 }
```

4.105. Referencia del archivo BetaProyecto/ViewModels/ViewAyudaViewModel.cs

4.106. ViewAyudaViewModel.cs

Ir a la documentación de este archivo.

```
00001 using ReactiveUI;
00002 using System;
00003 using System.Diagnostics;
00004 using System.Reactive;
```

```
00005
00006 namespace BetaProyecto.ViewModels
00007 {
00008     public class ViewAyudaViewModel : ViewModelBase, INavegable
00009     {
00010         //Actions
00011         public Action? VolverAtras { get; set; }
00012         //Comandos reactive
00013         public ReactiveCommand<Unit, Unit> btnVolverAtras { get; }
00014         public ViewAyudaViewModel() {
00015
00016             // Configuramos el comandos reactive
00017             btnVolverAtras = ReactiveCommand.Create(() =>
00018             {
00019                 Debug.WriteLine("Volviendo desde el Ayuda...");
00020                 VolverAtras?.Invoke();
00021             });
00022         }
00023     }
00024 }
```

4.107. Referencia del archivo BetaProyecto/ViewModels/ViewCancionesViewModel.cs

4.108. ViewCancionesViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Threading;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Singleton;
00004 using ReactiveUI;
00005 using System;
00006 using System.Collections.Generic;
00007 using System.Linq;
00008 using System.Reactive;
00009 using System.Threading;
00010 using System.Threading.Tasks;
00011
00012 namespace BetaProyecto.ViewModels
00013 {
00014     public class ViewCancionesViewModel : ViewModelBase
00015     {
00016         //Usamos reactive en cancion para que se actualice automaticamente con los hilos
00017         public Canciones _cancion;
00018         public Canciones Cancion
00019         {
00020             get => _cancion;
00021             set => this.RaiseAndSetIfChanged(ref _cancion, value);
00022         }
00023
00024         // Bindings
00025         private string _iconoLike;
00026         public string IconoLike
00027         {
00028             get => _iconoLike;
00029             set => this.RaiseAndSetIfChanged(ref _iconoLike, value);
00030         }
00031
00032         private string _txtMensajeTimer = "VisorCan_Timer_Consultando";
00033         public string TxtMensajeTimer
00034         {
00035             get => _txtMensajeTimer;
00036             set => this.RaiseAndSetIfChanged(ref _txtMensajeTimer, value);
00037         }
00038
00039         private string _txtVariableTimer = "";
00040         public string TxtVariableTimer
00041         {
00042             get => _txtVariableTimer;
00043             set => this.RaiseAndSetIfChanged(ref _txtVariableTimer, value);
00044         }
00045
00046         // Comandos Reactive
00047         public ReactiveCommand<Unit, Unit> BtnVolver { get; }
00048         public ReactiveCommand<Unit, Unit> BtnReproducir { get; }
00049         public ReactiveCommand<Unit, Unit> BtnLike { get; }
00050
00051         //Control de hilos
00052         private CancellationTokenSource _cancelToken;
00053
00054         // Propiedades formateas
```

```
00055         public string DuracionFormateada
00056         {
00057             get
00058             {
00059                 var tiempo = TimeSpan.FromSeconds(_cancion.Datos.DuracionSegundos);
00060                 return tiempo.ToString(@"m\:ss") + " min";
00061             }
00062         }
00063         public string FechaLanzamientoFormateada =>
00064             _cancion.Datos.FechaLanzamiento.ToString("dd MMMM, yyyy");
00065 
00066         public ViewCancionesViewModel(Canciones cancion, Action accionVolver, Action<Canciones>? accionReproducir,
      Action<Canciones> accionLike)
00067         {
00068             _cancion = cancion;
00069 
00070             ActualizarIconoLike();
00071 
00072             // Inicializamos el hilo
00073             _cancelToken = new CancellationTokenSource();
00074             IniciarHiloActualizacion(_cancelToken.Token);
00075 
00076             // Configuramos Comandos
00077             BtnVolver = ReactiveCommand.Create(() =>
00078             {
00079                 _cancelToken.Cancel();
00080                 accionVolver();
00081             });
00082 
00083             BtnReproducir = ReactiveCommand.Create(() =>
00084             {
00085                 // Al pulsar Play, avisamos al padre (MarcoApp) para que suene
00086                 accionReproducir?.Invoke(_cancion);
00087             });
00088 
00089             BtnLike = ReactiveCommand.CreateFromTask(async () =>
00090             {
00091                 accionLike(_cancion);
00092                 ActualizarIconoLike();
00093             });
00094         }
00095         /// <summary>
00096         /// Inicia un bucle de actualización en segundo plano que actualiza periódicamente la información actual de la
      canción hasta su cancelación
00097         /// es solicitado.
00098         /// </summary>
00099         /// <remarks>Este método ejecuta la lógica de actualización en un hilo en segundo plano y actualiza los elementos
      de la interfaz de usuario.
00100         /// usando el despachador. El ciclo de actualización continúa hasta que se indica el token de cancelación
      proporcionado. Intención
00101         /// para uso interno para mantener la interfaz de usuario sincronizada con los datos más recientes de las canciones.
      </remarks>
00102         /// <param name="token">Un token de cancelación que se puede usar para solicitar la finalización del ciclo de
      actualización. </param>
00103         private void IniciarHiloActualizacion(CancellationToken token)
00104         {
00105             Task.Run(async () =>
00106             {
00107                 try
00108                 {
00109                     while (!token.IsCancellationRequested)
00110                     {
00111                         // --- CUENTA ATRÁS VISUAL ---
00112                         // En lugar de esperar 3000ms de golpe, hacemos 3 pasos de 1000ms
00113                         for (int i = 5; i > 0; i--)
00114                         {
00115                             // Actualizamos el texto en la UI
00116                             await Dispatcher.UIThread.InvokeAsync(() =>
00117                             {
00118                                 // Aquí separamos: Clave + Valor
00119                                 TxtMensajeTimer = "VisorCan_Timer_Actualizando";
00120                                 TxtVariableTimer = $" {i} s";
00121                             });
00122 
00123                             // Esperamos 1 segundo
00124                             await Task.Delay(1000, token);
00125                         }
00126 
00127                         // Avisamos que estamos consultando
00128                         await Dispatcher.UIThread.InvokeAsync(() =>
00129                         {
00130                             TxtMensajeTimer = "VisorCan_Timer_Consultando";
00131                             TxtVariableTimer = ""; // Limpiamos el número
00132                         });
00133 
00134                         // --- CONSULTA A LA BASE DE DATOS ---
00135                         var listaUnaCancion = new List<string> { _cancion.Id };
```

```
00136                     var resultado = await
      MongoClientSingleton.Instance.Cliente.ObtenerCancionesPorListaIds(listaUnaCancion);
00137
00138                     var cancionActualizada = resultado.FirstOrDefault();
00139
00140                     if (cancionActualizada != null)
00141                     {
00142                         await Dispatcher.UIThread.InvokeAsync(() =>
00143                         {
00144                             // Actualizamos métricas
00145                             Cancion = cancionActualizada;
00146                             // Mensaje de éxito breve
00147                             TxtMensajeTimer = "VisorCan_Timer_Exito";
00148                             TxtVariableTimer = "";
00149                         });
00150                     }
00151                 }
00152             }
00153             catch (TaskCanceledException) { /* Hilo detenido */ }
00154         });
00155     }
00156     /// <summary>
00157     /// Actualiza el icono de like para indicar si la canción actual está marcada como favorita.
00158     /// </summary>
00159     /// <remarks>Este método establece el icono similar en función de la presencia del identificador de la canción actual
00160     /// en la lista global de favoritos. Debe llamarse siempre que haya cambiado el estado favorito de la canción
00161     /// para asegurarse de que el icono siga sincronizado con los favoritos del usuario. </remarks>
00162     private void ActualizarIconoLike()
00163     {
00164         var listaFavoritos = GlobalData.Instance.FavoritosGD;
00165
00166         if (listaFavoritos.Contains(_cancion.Id))// Si el ID de la canción actual está en favoritos
00167         {
00168             IconoLike = "Img_Like_ON";
00169         }
00170         else
00171         {
00172             // Si NO está -> Icono Normal
00173             IconoLike = "Img_Like_OFF";
00174         }
00175     }
00176 }
00177 }
```

4.109. Referencia del archivo BetaProyecto/ViewModels/ViewConfiguracionViewModel.cs

4.110. ViewConfiguracionViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Helpers;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Singleton;
00004 using ReactiveUI;
00005 using System;
00006 using System.Reactive;
00007
00008 namespace BetaProyecto.ViewModels
00009 {
00010     public class ViewConfiguracionViewModel : ViewModelBase
00011     {
00012         // Actions
00013         private readonly Action _accionRefrescarTema;
00014
00015         // Bindings
00016
00017         // FUENTE: 0=Lexend, 1=Carlito, 2=Arial, 3=Gloria, 4=OpenSans, 5=Roboto
00018         private int _indiceFuente;
00019         public int IndiceFuente
00020         {
00021             get => _indiceFuente;
00022             set
00023             {
00024                 this.RaiseAndSetIfChanged(ref _indiceFuente, value);
00025                 AplicarCambioFuente(value);
00026             }
00027         }
00028
00029         // IDIOMA: 0 = Español, 1 = Inglés
```

```
00030         private int _indiceIdioma;
00031         public int IndiceIdioma
00032         {
00033             get => _indiceIdioma;
00034             set
00035             {
00036                 this.RaiseAndSetIfChanged(ref _indiceIdioma, value);
00037                 AplicarCambioIdioma(value);
00038             }
00039         }
00040 
00041         // TEMA: True = Claro, False = Oscuro
00042         private bool _indiceTema = true;
00043         public bool IndiceTema
00044         {
00045             get => _indiceTema;
00046             set
00047             {
00048                 this.RaiseAndSetIfChanged(ref _indiceTema, value);
00049                 AplicarCambioTema(value);
00050                 _accionRefrescarTema?.Invoke();
00051             }
00052         }
00053 
00054         //Para cuando volvamos no veamos el radio button vacio si estamos en modo oscuro
00055         public bool IndiceTemaOscuro
00056         {
00057             get => !_indiceTema;
00058             set
00059             {
00060                 IndiceTema = !value;
00061             }
00062         }
00063 
00064         // Comandos Reactive
00065         public ReactiveCommand<Unit, Unit> BtnVolverAtras { get; }
00066         public ReactiveCommand<Unit, Unit> BtnCerrarSesion { get; }
00067         public ReactiveCommand<Unit, Unit> BtnSalirApp { get; }
00068 
00069         // Constructor
00070         public ViewConfiguracionViewModel(Action accionVolver, Action accionLogout, Action accionSalir, Action
    accionRefrescar)
00071         {
00072             _accionRefrescarTema = accionRefrescar;
00073 
00074             // Cargar el estado inicial basado en el GlobalData
00075             CargarEstadoInicial();
00076 
00077             // Configuramos comandos reactive
00078             BtnVolverAtras = ReactiveCommand.Create(() => accionVolver?.Invoke());
00079 
00080             BtnCerrarSesion = ReactiveCommand.Create(() =>
00081             {
00082                 GlobalData.Instance.ClearUserData();
00083                 accionLogout?.Invoke();
00084             });
00085 
00086             BtnSalirApp = ReactiveCommand.Create(() => accionSalir?.Invoke());
00087         }
00088 
00089         /// <summary>
00090         /// Sincroniza la interfaz de configuración con las preferencias del usuario almacenadas en los datos globales.
00091         /// </summary>
00092         /// <remarks>
00093         /// Este método recupera los valores actuales de tema, idioma y tipografía desde <see
    cref="GlobalData.Instance"/>.
00094         /// Posteriormente, traduce estas cadenas de texto a los índices correspondientes que utilizan los selectores de la vista
00095         /// y fuerza la notificación de cambio de propiedades mediante <c>RaisePropertyChanged</c> para que la UI
00096         /// refleje el estado real de la configuración.
00097         /// </remarks>
00098         private void CargarEstadoInicial()
00099         {
00100             // Usamos las variables de GlobalData para enseñar el estado actual de la configuración al usuario
00101             var tema = GlobalData.Instance.DiccionarioTemaGD ?? "ModoClaro";
00102             var idioma = GlobalData.Instance.DiccionarioIdiomaGD ?? "Spanish";
00103             var fuente = GlobalData.Instance.DiccionarioFuenteGD ?? "Lexend";
00104 
00105             // Sincronizar Tema
00106             _indiceTema = (tema == "ModoClaro");
00107             this.RaisePropertyChanged(nameof(IndiceTema));
00108             this.RaisePropertyChanged(nameof(IndiceTemaOscuro));
00109 
00110             // Sincronizar Idioma
00111             _indiceIdioma = (idioma == "English") ? 1 : 0;
00112             this.RaisePropertyChanged(nameof(IndiceIdioma));
00113 
00114             // Sincronizar Fuente
```

```
00115           _indiceFuente = fuente switch
00116           {
00117               "Lexend" => 0,
00118               "Carlito" => 1,
00119               "Arial" => 2,
00120               "GloriaHallelujah" => 3,
00121               "OpenSans" => 4,
00122               "Roboto" => 5,
00123               _ => 0
00124           };
00125           this.RaisePropertyChanged(nameof(IndiceFuente));
00126       }
00127       /// <summary>
00128       /// Ejecuta el cambio de apariencia visual de la aplicación entre modo claro y modo oscuro.
00129       /// </summary>
00130       /// <remarks>
00131       /// Este método gestiona el cambio de tema en tres niveles:
00132       /// <list type="number">
00133       /// <item><b>Visual:</b> Aplica el diccionario de recursos de forma instantánea mediante <see
    cref="ControladorDiccionarios"/>.</item>
00134       /// <item><b>Persistencia:</b> Si el tema es distinto al actual, sincroniza la preferencia en la base de datos
    MongoDB.</item>
00135       /// <item><b>Estado Global:</b> Actualiza la propiedad en <see cref="GlobalData.Instance"/> para mantener
    la consistencia en toda la sesión.</item>
00136       /// </list>
00137       /// </remarks>
00138       /// <param name="esClaro">Indica si se debe aplicar el "ModoClaro" (<c>true</c>) o el "ModoOscuro"
    (<c>false</c>).</param>
00139       private void AplicarCambioTema(bool esClaro)
00140       {
00141           string nuevoTema = esClaro ? "ModoClaro" : "ModoOscuro";
00142
00143           //Aplicamos el tema en la app
00144           ControladorDiccionarios.AplicarTema(nuevoTema);
00145
00146           // Comprobamos si realmente cambió respecto a GlobalData
00147           if (GlobalData.Instance.DiccionarioTemaGD != nuevoTema)
00148           {
00149               // Guardamos en Mongo
00150               GuardarConfiguracionEnMongo(nuevoTema, null, null);
00151
00152               // Actualizamos GlobalData
00153               GlobalData.Instance.DiccionarioTemaGD = nuevoTema;
00154           }
00155       }
00156       /// <summary>
00157       /// Ejecuta el cambio de idioma de la interfaz de usuario basándose en el índice seleccionado.
00158       /// </summary>
00159       /// <remarks>
00160       /// Este método gestiona la internacionalización en tres niveles:
00161       /// <list type="number">
00162       /// <item><b>Visual:</b> Cambia el diccionario de strings de forma dinámica mediante <see
    cref="ControladorDiccionarios"/>.</item>
00163       /// <item><b>Persistencia:</b> Si el idioma es diferente al actual, sincroniza la nueva preferencia en la base de
    datos MongoDB.</item>
00164       /// <item><b>Estado Global:</b> Actualiza la referencia en <see cref="GlobalData.Instance"/> para asegurar la
    persistencia durante la sesión activa.</item>
00165       /// </list>
00166       /// </remarks>
00167       /// <param name="indice">El índice del selector: <c>0</c> para "Spanish" y <c>1</c> para
    "English".</param>
00168       private void AplicarCambioIdioma(int indice)
00169       {
00170           string nuevoIdioma = indice == 1 ? "English" : "Spanish";
00171
00172           ControladorDiccionarios.AplicarIdioma(nuevoIdioma);
00173
00174           if (GlobalData.Instance.DiccionarioIdiomaGD != nuevoIdioma)
00175           {
00176               GuardarConfiguracionEnMongo(null, nuevoIdioma, null);
00177               GlobalData.Instance.DiccionarioIdiomaGD = nuevoIdioma;
00178           }
00179       }
00180       /// <summary>
00181       /// Ejecuta el cambio de la fuente tipográfica de la aplicación basándose en el índice seleccionado.
00182       /// </summary>
00183       /// <remarks>
00184       /// Este método gestiona la personalización visual en tres niveles:
00185       /// <list type="number">
00186       /// <item><b>Visual:</b> Cambia el diccionario de estilos de fuente de forma dinámica mediante <see
    cref="ControladorDiccionarios"/>.</item>
00187       /// <item><b>Persistencia:</b> Si la fuente es diferente a la actual, sincroniza la nueva preferencia en la base de
    datos MongoDB.</item>
00188       /// <item><b>Estado Global:</b> Actualiza la referencia en <see cref="GlobalData.Instance"/> para asegurar
    que la tipografía se mantenga durante la sesión activa.</item>
00189       /// </list>
00190       /// </remarks>
```

```
00191         /// <param name="indice">El índice del selector que determina la familia tipográfica (0: Lexend, 1: Carlito, 2: Arial, etc.).</param>
00192         private void AplicarCambioFuente(int indice)
00193         {
00194             string nuevaFuente = indice switch
00195             {
00196                 0 => "Lexend",
00197                 1 => "Carlito",
00198                 2 => "Arial",
00199                 3 => "GloriaHallelujah",
00200                 4 => "OpenSans",
00201                 5 => "Roboto",
00202                 _ => "Lexend"
00203             };
00204 
00205             ControladorDiccionarios.AplicarFuente(nuevaFuente);
00206 
00207             if (GlobalData.Instance.DiccionarioFuenteGD != nuevaFuente)
00208             {
00209                 GuardarConfiguracionEnMongo(null, null, nuevaFuente);
00210                 GlobalData.Instance.DiccionarioFuenteGD = nuevaFuente;
00211             }
00212         }
00213 
00214         /// <summary>
00215         /// Sincroniza de forma asíncrona las preferencias de personalización del usuario en la base de datos MongoDB.
00216         /// </summary>
00217         /// <remarks>
00218         /// Este método implementa una lógica de actualización parcial. Construye un objeto de configuración
00219         /// combinando los nuevos valores proporcionados con los valores actuales almacenados en <see cref="GlobalData.Instance"/>.
00220         /// Si un parámetro se recibe como <c>null</c>, se preserva el valor existente. La actualización se lanza
00221         /// mediante una tarea en segundo plano para no bloquear la interfaz.
00222         /// </remarks>
00223         /// <param name="temaNuevo">El nombre del nuevo tema visual o <c>null</c> si no ha cambiado.</param>
00224         /// <param name="idiomaNuevo">El nombre del nuevo idioma de la interfaz o <c>null</c> si no ha cambiado.</param>
00225         /// <param name="fuenteNuevo">El nombre de la nueva familia tipográfica o <c>null</c> si no ha cambiado.</param>
00226         private void GuardarConfiguracionEnMongo(string? temaNuevo, string? idiomaNuevo, string? fuenteNuevo)
00227         {
00228             if (string.IsNullOrEmpty(GlobalData.Instance.UserIdGD)) return;
00229 
00230             // Construimos el objeto Configuración mezclando lo NUEVO con lo VIEJO para actualizarlo
00231             var config = new ConfiguracionUser
00232             {
00233                 DiccionarioTema = temaNuevo ?? GlobalData.Instance.DiccionarioTemaGD,
00234                 DiccionarioIdioma = idiomaNuevo ?? GlobalData.Instance.DiccionarioIdiomaGD,
00235                 DiccionarioFuente = fuenteNuevo ?? GlobalData.Instance.DiccionarioFuenteGD
00236             };
00237 
00238             // Ejecutamos en segundo plano para actualizar la configuraciñon del usuario.
00239             _ = MongoClientSingleton.Instance.Cliente.ActualizarConfiguracionUsuario(GlobalData.Instance.UserIdGD, config);
00240         }
00241     }
00242 }
```

4.111. Referencia del archivo BetaProyecto/ViewModels/ViewCrearListaPersonalizadaViewModel.cs

4.112. ViewCrearListaPersonalizadaViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Media.Imaging;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Services;
00004 using BetaProyecto.Singleton;
00005 using ReactiveUI;
00006 using System;
00007 using System.Collections.ObjectModel;
00008 using System.Linq;
00009 using System.Reactive;
00010 using System.Reactive.Linq;
00011 using System.Threading.Tasks;
00012 
00013 namespace BetaProyecto.ViewModels
00014 {
00015     public class ViewCrearListaPersonalizadaViewModel : ViewModelBase
```

```
00016 {
00017     //Servicios
00018     private readonly IDialogoService _dialogoService;
00019     private readonly StorageService _storageService;
00020 
00021 
00022     private readonly Action _Volver;
00023 
00024     // Binding de datos
00025     private string _txtNombre;
00026     public string TxtNombre
00027     {
00028         get => _txtNombre;
00029         set => this.RaiseAndSetIfChanged(ref _txtNombre, value);
00030     }
00031 
00032     private string _txtDescripcion;
00033     public string TxtDescripcion
00034     {
00035         get => _txtDescripcion;
00036         set => this.RaiseAndSetIfChanged(ref _txtDescripcion, value);
00037     }
00038 
00039     // Biding para la imagen
00040     private string _rutaImagen;
00041     public string RutaImagen
00042     {
00043         get => _rutaImagen;
00044         set
00045         {
00046             this.RaiseAndSetIfChanged(ref _rutaImagen, value);
00047             CargarImagenLocal(value);
00048         }
00049     }
00050     public bool TieneImagen => !string.IsNullOrEmpty(RutaImagen);
00051 
00052     private Bitmap? _imagenPortada;
00053     public Bitmap? ImagenPortada
00054     {
00055         get => _imagenPortada;
00056         set => this.RaiseAndSetIfChanged(ref _imagenPortada, value);
00057     }
00058 
00059     // Binding para el buscador de canciones
00060     private string _txtBusqueda;
00061     public string TxtBusqueda
00062     {
00063         get => _txtBusqueda;
00064         set => this.RaiseAndSetIfChanged(ref _txtBusqueda, value);
00065     }
00066 
00067     private ObservableCollection<Canciones> _listaResultados;
00068     public ObservableCollection<Canciones> ListaResultados
00069     {
00070         get => _listaResultados;
00071         set => this.RaiseAndSetIfChanged(ref _listaResultados, value);
00072     }
00073 
00074     private ObservableCollection<Canciones> _listaCancionesSeleccionadas;
00075     public ObservableCollection<Canciones> ListaCancionesSeleccionadas
00076     {
00077         get => _listaCancionesSeleccionadas;
00078         set => this.RaiseAndSetIfChanged(ref _listaCancionesSeleccionadas, value);
00079     }
00080 
00081     // Para le progressbar
00082     private bool _estaCargando;
00083     public bool EstaCargando
00084     {
00085         get => _estaCargando;
00086         set => this.RaiseAndSetIfChanged(ref _estaCargando, value);
00087     }
00088 
00089     // Comandos Reactive
00090     public ReactiveCommand<Unit, Unit> BtnVolverAtras { get; }
00091     public ReactiveCommand<Unit, Unit> BtnCrear { get; }
00092     public ReactiveCommand<Unit, Unit> BtnBuscarCanciones { get; }
00093     public ReactiveCommand<Canciones, Unit> BtnAgregarCancion { get; }
00094     public ReactiveCommand<Canciones, Unit> BtnEliminarCancion { get; }
00095 
00096 
00097     public ViewCrearListaPersonalizadaViewModel(Action accionVolver)
00098     {
00099         _Volver = accionVolver;
00100         _dialogoService = new DialogoService();
00101         _storageService = new StorageService();
00102 
```

```
00103           ListaResultados = new ObservableCollection<Canciones>();
00104           ListaCancionesSeleccionadas = new ObservableCollection<Canciones>();
00105
00106           // Comandos
00107           BtnVolverAtras = ReactiveCommand.Create(accionVolver);
00108           BtnBuscarCanciones = ReactiveCommand.Create(BuscarCanciones);
00109           BtnAgregarCancion = ReactiveCommand.Create<Canciones>(AgregarCancion);
00110           BtnEliminarCancion = ReactiveCommand.Create<Canciones>(EliminarCancion);
00111
00112           // Buscador Reactivo
00113           this.WhenAnyValue(x => x.TxtBusqueda)
00114               .Throttle(TimeSpan.FromMilliseconds(500))
00115               .Where(x => !string.IsNullOrWhiteSpace(x) && x.Length > 2)
00116               .ObserveOn(RxApp.MainThreadScheduler)
00117               .Subscribe(_ => BuscarCanciones());
00118
00119           // Validación para Crear
00120           var validacionCrear = this.WhenAnyValue(
00121               x => x.TxtNombre,
00122               x => x.RutaImagen,
00123               x => x.ListaCancionesSeleccionadas.Count,
00124               (nombre, imagen, count) =>
00125                   !string.IsNullOrWhiteSpace(nombre) &&
00126                   !string.IsNullOrWhiteSpace(imagen) &&
00127                   count > 0 // Al menos una canción
00128           );
00129           //Configuración de comandos reactive
00130           BtnCrear = ReactiveCommand.CreateFromTask(CrearLista, validacionCrear);
00131       }
00132       /// <summary>
00133       /// Realiza una búsqueda asíncrona de canciones en la base de datos utilizando el texto introducido por el usuario.
00134       /// </summary>
00135       /// <remarks>
00136       /// Este método consulta MongoDB y filtra los resultados obtenidos para excluir aquellas canciones
00137       /// que ya están presentes en la lista de selección (<see cref="ListaCancionesSeleccionadas"/>).
00138       /// Esto evita duplicados visuales y actualiza la colección de resultados disponibles para añadir.
00139       /// </remarks>
00140       private async void BuscarCanciones()
00141       {
00142           if (MongoClientSingleton.Instance.Cliente != null)
00143           {
00144               var resultados = await MongoClientSingleton.Instance.Cliente.ObtenerCancionesPorBusqueda(TxtBusqueda);
00145
00146               if (resultados != null)
00147               {
00148                   // Filtramos las que ya están seleccionadas para no duplicar visualmente
00149                   var filtradas = resultados.Where(c => !ListaCancionesSeleccionadas.Any(sel => sel.Id == c.Id));
00150                   ListaResultados = new ObservableCollection<Canciones>(filtradas);
00151               }
00152           }
00153       }
00154
00155       /// <summary>
00156       /// Agrega la canción seleccionada a la lista temporal de canciones que formarán parte de la nueva playlist.
00157       /// </summary>
00158       /// <remarks>
00159       /// Este método realiza tres acciones clave:
00160       /// <list type="number">
00161       /// <item>Verifica que la canción no esté ya añadida para evitar duplicados.</item>
00162       /// <item>Mueve visualmente la canción: la añade a <see cref="ListaCancionesSeleccionadas"/> y la elimina de
    <see cref="ListaResultados"/>.</item>
00163       /// <item>Limpia el campo de búsqueda para facilitar una nueva consulta inmediata.</item>
00164       /// </list>
00165       /// </remarks>
00166       /// <param name="cancion">El objeto <see cref="Canciones"/> que el usuario ha seleccionado para
    añadir.</param>
00167       private void AgregarCancion(Canciones cancion)
00168       {
00169           if (!ListaCancionesSeleccionadas.Any(c => c.Id == cancion.Id))
00170           {
00171               ListaCancionesSeleccionadas.Add(cancion);
00172               ListaResultados.Remove(cancion); // La quitamos de resultados para que quede limpio
00173               TxtBusqueda = "";
00174           }
00175       }
00176
00177       /// <summary>
00178       /// Elimina una canción de la lista de canciones seleccionadas para la nueva playlist.
00179       /// </summary>
00180       /// <remarks>
00181       /// Permite al usuario rectificar su selección quitando canciones individuales de <see
    cref="ListaCancionesSeleccionadas"/>
00182       /// antes de guardar la lista definitiva. La interfaz de usuario refleja el cambio inmediatamente.
00183       /// </remarks>
00184       /// <param name="cancion">El objeto <see cref="Canciones"/> que se desea descartar de la selección
    actual.</param>
00185       private void EliminarCancion(Canciones cancion)
```

```
00186       {
00187           ListaCancionesSeleccionadas.Remove(cancion);
00188       }
00189
00190       /// <summary>
00191       /// Intenta cargar y visualizar una imagen local desde la ruta especificada.
00192       /// </summary>
00193       /// <remarks>
00194       /// <para>
00195       /// Este método gestiona de forma segura la carga de archivos de imagen. Si el archivo no existe
00196       /// o el formato no es válido (lanzando una excepción), la propiedad <see cref="ImagenPortada"/>
00197       /// se establece en <c>null</c> para evitar errores visuales.
00198       /// </para>
00199       /// <para>
00200       /// Al finalizar, fuerza una notificación de cambio en <see cref="TieneImagen"/> para que la interfaz
00201       /// actualice la visibilidad de los controles dependientes (como el botón de "Quitar imagen").
00202       /// </para>
00203       /// </remarks>
00204       /// <param name="ruta">La ruta absoluta del sistema de archivos donde se encuentra la imagen.</param>
00205       private void CargarImagenLocal(string ruta)
00206       {
00207           try
00208           {
00209               if (System.IO.File.Exists(ruta))
00210                   ImagenPortada = new Bitmap(ruta);
00211               else
00212                   ImagenPortada = null;
00213           }
00214           catch
00215           {
00216               ImagenPortada = null;
00217           }
00218           this.RaisePropertyChanged(nameof(TieneImagen));
00219       }
00220       /// <summary>
00221       /// Orquesta el proceso completo de creación de una nueva lista de reproducción personalizada de forma asíncrona.
00222       /// </summary>
00223       /// <remarks>
00224       /// Este método sigue un flujo de transacciones paso a paso:
00225       /// <list type="number">
00226       /// <item><b>Carga de medios:</b> Sube la imagen de portada seleccionada al servicio de almacenamiento en la
      nube.</item>
00227       /// <item><b>Construcción del modelo:</b> Crea una instancia de <see cref="ListaPersonalizada"/> con los
      metadatos y la selección de canciones actual.</item>
00228       /// <item><b>Persistencia:</b> Invoca al cliente de MongoDB para guardar la nueva lista en la base de
      datos.</item>
00229       /// </list>
00230       /// Gestiona los estados de carga (<see cref="EstaCargando"/>) para bloquear la UI durante el proceso y maneja
      excepciones globales.
00231       /// </remarks>
00232       /// <returns>Una <see cref="Task"/> que representa la operación asíncrona.</returns>
00233       private async Task CrearLista()
00234       {
00235           EstaCargando = true;
00236           try
00237           {
00238               // Subir Imagen
00239               string urlPortada = await _storageService.SubirImagen(RutaImagen);
00240
00241               // Crear Objeto
00242               var nuevaLista = new ListaPersonalizada
00243               {
00244                   Nombre = TxtNombre,
00245                   Descripcion = TxtDescripcion,
00246                   UrlPortada = urlPortada,
00247                   IdUsuario = GlobalData.Instance.UserIdGD,
00248                   IdsCanciones = ListaCancionesSeleccionadas.Select(c => c.Id).ToList()
00249               };
00250
00251               // Guardarmos en BD
00252               bool exito = await MongoClientSingleton.Instance.Cliente.CrearListaReproduccion(nuevaLista);
00253
00254               if (exito)
00255               {
00256                   EstaCargando = false;
00257                   _Volver();
00258               }
00259               else
00260               {
00261                   EstaCargando = false;
00262                   _dialogoService.MostrarAlerta("Msg_Error_CrearPlaylist");
00263               }
00264           }
00265           catch (Exception ex)
00266           {
00267               EstaCargando = false;
00268               _dialogoService.MostrarAlerta("Msg_Error_Inesperado");
```

```
00269                 System.Diagnostics.Debug.WriteLine("Error crear lista: " + ex.Message);
00270             }
00271         }
00272     }
00273 }
```

4.113. Referencia del archivo BetaProyecto/ViewModels/ViewCrearReporteViewModel.cs

4.114. ViewCrearReporteViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Singleton;
00003 using ReactiveUI;
00004 using System;
00005 using System.Collections.Generic;
00006 using System.Reactive;
00007 using System.Threading.Tasks;
00008
00009 namespace BetaProyecto.ViewModels
00010 {
00011     public class ViewCrearReporteViewModel : ViewModelBase
00012     {
00013         //Variables
00014         private Canciones _cancionAReportar;
00015         public Canciones CancionAReportar => _cancionAReportar;
00016
00017         //Actions
00018         private readonly Action _Volver;
00019
00020
00021         //Bidings
00022         public List<string> TiposDeProblema { get; } = new List<string>
00023         {
00024             "Copyright / Derechos de autor",
00025             "Contenido ofensivo o inapropiado",
00026             "Audio defectuoso o silencio",
00027             "Spam / Información falsa",
00028             "Otro"
00029         };
00030
00031         private string _tipoSeleccionado;
00032         public string TipoSeleccionado
00033         {
00034             get => _tipoSeleccionado;
00035             set => this.RaiseAndSetIfChanged(ref _tipoSeleccionado, value);
00036         }
00037
00038         private string _descripcionTexto;
00039         public string DescripcionTexto
00040         {
00041             get => _descripcionTexto;
00042             set => this.RaiseAndSetIfChanged(ref _descripcionTexto, value);
00043         }
00044
00045         private string _mensajeEstado;
00046         public string MensajeEstado
00047         {
00048             get => _mensajeEstado;
00049             set => this.RaiseAndSetIfChanged(ref _mensajeEstado, value);
00050         }
00051
00052         //Comandos reactive
00053         public ReactiveCommand<Unit, Unit> BtnEnviarReporte { get; }
00054         public ReactiveCommand<Unit, Unit> BtnCancelar { get; }
00055
00056         // Constructor
00057         public ViewCrearReporteViewModel(Canciones cancion, Action accionVolver)
00058         {
00059             _cancionAReportar = cancion;
00060             _Volver = accionVolver;
00061
00062             // Validación
00063             var validacionCampos = this.WhenAnyValue(
00064                 x => x.TipoSeleccionado,
00065                 x => x.DescripcionTexto,
00066                 (tipo, desc) => !string.IsNullOrEmpty(tipo) && !string.IsNullOrWhiteSpace(desc)
00067             );
```

```
00068
00069           // Comandos reactive
00070           BtnEnviarReporte = ReactiveCommand.CreateFromTask(EnviarReporteAsync, canExecute: validacionCampos);
00071           BtnCancelar = ReactiveCommand.Create(accionVolver);
00072       }
00073
00074       /// <summary>
00075       /// Envía un informe de forma asíncrona utilizando los detalles del informe actual y actualiza el mensaje de estado en función del
00076       /// resultado.
00077       /// </summary>
00078       /// <remarks>Si el informe se envía con éxito, se actualiza el mensaje de estado para indicar que ha tenido éxito.
00079       /// y el método navega de regreso después de un breve retraso. Si se produce un error, el mensaje de estado se actualiza a
00080       /// indica el fallo. </remarks>
00081       /// <returns>Devuelve una tarea que representa la operación asíncrona. </returns>
00082       private async Task EnviarReporteAsync()
00083       {
00084           try
00085           {
00086               var reporte = new Reportes
00087               {
00088                   TipoProblema = TipoSeleccionado,
00089                   Descripcion = DescripcionTexto,
00090                   Estado = "Pendiente",
00091                   FechaCreacion = DateTime.UtcNow,
00092                   Referencias = new ReferenciasReporte
00093                   {
00094                       CancionReportadaId = _cancionAReportar.Id,
00095                       UsuarioReportanteId = GlobalData.Instance.UserIdGD
00096                   }
00097               };
00098
00099               await MongoClientSingleton.Instance.Cliente.EnviarReporte(reporte);
00100
00101               MensajeEstado = "Msg_Exito_Reporte";
00102
00103               await Task.Delay(1500);
00104               _Volver();
00105           }
00106           catch (Exception ex)
00107           {
00108               MensajeEstado = "Msg_Error_Reporte";
00109               System.Diagnostics.Debug.WriteLine(ex.Message);
00110           }
00111       }
00112   }
00113 }
```

4.115. Referencia del archivo BetaProyecto/ViewModels/ViewCrearUsuarioViewModel.cs

4.116. ViewCrearUsuarioViewModel.cs

Ir a la documentación de este archivo.

```
00001  using Avalonia.Media.Imaging;
00002 using BetaProyecto.Helpers;
00003 using BetaProyecto.Models;
00004 using BetaProyecto.Services;
00005 using BetaProyecto.Singleton;
00006 using ReactiveUI;
00007 using System;
00008 using System.Collections.Generic;
00009 using System.Reactive;
00010 using System.Threading.Tasks;
00011
00012 namespace BetaProyecto.ViewModels
00013 {
00014     public class ViewCrearUsuarioViewModel : ViewModelBase
00015     {
00016         // Servicios
00017         private readonly IDialogoService _dialogoService;
00018         private readonly StorageService _storageService;
00019
00020         //Actions
00021         private readonly Action _accionVolver;
00022
00023         // Bidings
00024         private Usuarios _nuevoUsuario;
```

```
00025         public Usuarios NuevoUsuario
00026         {
00027             get => _nuevoUsuario;
00028             set => this.RaiseAndSetIfChanged(ref _nuevoUsuario, value);
00029         }
00030 
00031         // Propiedad extra para validar password
00032         private string _confirmarPass;
00033         public string ConfirmarPass
00034         {
00035             get => _confirmarPass;
00036             set => this.RaiseAndSetIfChanged(ref _confirmarPass, value);
00037         }
00038 
00039         // Progress bar
00040         private bool _estaCargando;
00041         public bool EstaCargando
00042         {
00043             get => _estaCargando;
00044             set => this.RaiseAndSetIfChanged(ref _estaCargando, value);
00045         }
00046 
00047         private Bitmap? _fotoPerfilBitmap;
00048         public Bitmap? FotoPerfilBitmap
00049         {
00050             get => _fotoPerfilBitmap;
00051             set => this.RaiseAndSetIfChanged(ref _fotoPerfilBitmap, value);
00052         }
00053 
00054         //Propiedades
00055         public List<string> ListaPaises { get; }
00056 
00057         // Comandos reactive
00058         public ReactiveCommand<Unit, Unit> BtnRegistrarse { get; }
00059         public ReactiveCommand<Unit, Unit> BtnVolver { get; }
00060 
00061         public ViewCrearUsuarioViewModel(Action accionVolver)
00062         {
00063             // Guardamos la acción de volver
00064             _accionVolver = accionVolver;
00065 
00066             // Inicializamos servicios
00067             _dialogoService = new DialogoService();
00068             _storageService = new StorageService();
00069 
00070             // Inicializamos el objeto vacío pero con sus estructuras listas
00071             NuevoUsuario = new Usuarios
00072             {
00073                 Perfil = new PerfilUsuario
00074                 {
00075                     FechaNacimiento = DateTime.Today
00076                 },
00077                 Estadisticas = new EstadisticasUsuario(),
00078                 Listas = new ListasUsuario()
00079             };
00080 
00081             //Inicalizamos propiedades
00082             ListaPaises = new List<string>
00083             {
00084                 "España",
00085                 "Inglaterra",
00086                 "Estados Unidos",
00087                 "Canadá",
00088                 "Suecia",
00089                 "Chile",
00090                 "Andorra",
00091                 "Francia",
00092                 "Alemania",
00093                 "Japón"
00094             };
00095 
00096             // Configuramos comandos reactive
00097             BtnRegistrarse = ReactiveCommand.CreateFromTask(RegistrarseTask);
00098             BtnVolver = ReactiveCommand.Create(() => _accionVolver?.Invoke());
00099         }
00100         /// <summary>
00101         /// Gestiona el proceso de registro de usuarios de forma asíncrona, incluida la validación, la carga de imágenes de
    perfil y
00102         /// creación de cuenta.
00103         /// </summary>
00104         /// <remarks>Muestra las alertas apropiadas al usuario en caso de errores de validación, problemas de conexión o
00105         /// resultados del registro. Si el registro es exitoso, se notifica al usuario y se lo redirige a la cuenta.
00106         /// pantalla. Este método evita los intentos de registro simultáneos comprobando y configurando una carga
00107         /// estado. </remarks>
00108         /// <returns>Devuelve una tarea que representa la operación de registro asíncrono. </returns>
00109         private async Task RegistrarseTask()
00110         {
```

```
00111           if (EstaCargando) return;
00112           EstaCargando = true;
00113
00114           try
00115           {
00116               // Validaciones básicas
00117               if (string.IsNullOrWhiteSpace(NuevoUsuario.Username) ||
00118                   string.IsNullOrWhiteSpace(NuevoUsuario.Email) ||
00119                   string.IsNullOrWhiteSpace(NuevoUsuario.Password) ||
00120                   string.IsNullOrWhiteSpace(ConfirmarPass) ||
00121                   string.IsNullOrWhiteSpace(NuevoUsuario.Perfil.Pais) ||
00122                   string.IsNullOrWhiteSpace(NuevoUsuario.Perfil.ImagenUrl)
00123                   )
00124               {
00125                   _dialogoService.MostrarAlerta("Reg_Error_FaltanCampos");
00126                   EstaCargando = false;
00127                   return;
00128               }
00129
00130               // Validar contraseñas coinicidentes
00131               if (NuevoUsuario.Password != ConfirmarPass)
00132               {
00133                   _dialogoService.MostrarAlerta("Reg_Error_PassNoCoinciden");
00134                   EstaCargando = false;
00135                   return;
00136               }
00137
00138               // Conexión a mongo
00139               var cliente = MongoClientSingleton.Instance.Cliente;
00140               if (!await cliente.Conectar())
00141               {
00142                   _dialogoService.MostrarAlerta("Msg_Error_Conexion");
00143                   EstaCargando = false;
00144                   return;
00145               }
00146
00147               // Preparamos datos
00148               NuevoUsuario.Rol = Roles.Usuario;
00149               NuevoUsuario.FechaRegistro = DateTime.Now;
00150               NuevoUsuario.Password = Encriptador.HashPassword(NuevoUsuario.Password);
00151
00152               // Gestionamos foto de perfil
00153               try
00154               {
00155                   // Subimos a Imgbb
00156                   NuevoUsuario.Perfil.ImagenUrl = await _storageService.SubirImagen(NuevoUsuario.Perfil.ImagenUrl);
00157               }
00158               catch
00159               {
00160                   _dialogoService.MostrarAlerta("Reg_Error_SubirImagen");
00161                   NuevoUsuario.Perfil.ImagenUrl = "";
00162                   return;
00163               }
00164
00165               // Si no puso imagen o falló, ponemos una por defecto
00166               if (string.IsNullOrEmpty(NuevoUsuario.Perfil.ImagenUrl))
00167               {
00168                   NuevoUsuario.Perfil.ImagenUrl = "https://i.ibb.co/hRJ440cz/image.png";
00169               }
00170
00171               // 6. GUARDAR EN BD
00172               bool exito = await cliente.CrearUsuario(NuevoUsuario);
00173
00174               if (exito)
00175               {
00176                   _dialogoService.MostrarAlerta("Reg_Exito_CuentaCreada");
00177                   _accionVolver?.Invoke(); // Volvemos al Login automáticamente
00178               }
00179               else
00180               {
00181                   _dialogoService.MostrarAlerta("Reg_Error_UsuarioExiste");
00182               }
00183           }
00184           catch (Exception ex)
00185           {
00186               _dialogoService.MostrarAlerta("Msg_Error_Inesperado");
00187           }
00188           finally
00189           {
00190               EstaCargando = false;
00191           }
00192       }
00193       /// <summary>
00194       /// Carga una imagen de vista previa desde la ruta del archivo especificada y actualiza la referencia de imagen de
      perfil del usuario.
00195       /// </summary>
00196       /// <remarks>Si el archivo no existe o no es una imagen válida, la imagen de previsualización se borra. El
```

```
00197         /// método no genera una excepción si la carga falla. </remarks>
00198         /// <param name="ruta">La ruta del archivo de la imagen a cargar como una vista previa. Debe hacer referencia a un archivo existente. </param>
00199         public void CargarImagenPrevia(string ruta)
00200         {
00201             try
00202             {
00203                 if (System.IO.File.Exists(ruta))
00204                 {
00205                     // Cargamos el Bitmap desde el archivo
00206                     FotoPerfilBitmap = new Bitmap(ruta);
00207 
00208                     // Y guardamos la ruta en el modelo para subirla luego
00209                     NuevoUsuario.Perfil.ImagenUrl = ruta;
00210                 }
00211             }
00212             catch (Exception)
00213             {
00214                 // Si falla (no es imagen válida), ponemos null o una imagen por defecto
00215                 FotoPerfilBitmap = null;
00216             }
00217         }
00218     }
00219 }
```

4.117. Referencia del archivo BetaProyecto/ViewModels/ViewCuentaViewModel.cs

4.118. ViewCuentaViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Services;
00002 using BetaProyecto.Singleton;
00003 using ReactiveUI;
00004 using System;
00005 using System.Reactive;
00006 using System.Threading.Tasks;
00007 
00008 namespace BetaProyecto.ViewModels
00009 {
00010     public class ViewCuentaViewModel : ViewModelBase
00011     {
00012         //Servicios
00013         private readonly IDialogoService _dialogoService;
00014 
00015         //Bidings
00016         private string _nombreUsuario;
00017         public string NombreUsuario
00018         {
00019             get => _nombreUsuario;
00020             set => this.RaiseAndSetIfChanged(ref _nombreUsuario, value);
00021         }
00022 
00023         private string _email;
00024         public string Email
00025         {
00026             get => _email;
00027             set => this.RaiseAndSetIfChanged(ref _email, value);
00028         }
00029 
00030         private string _pais;
00031         public string Pais
00032         {
00033             get => _pais;
00034             set => this.RaiseAndSetIfChanged(ref _pais, value);
00035         }
00036 
00037         // El DatePicker de Avalonia usa DateTimeOffset?
00038         private DateTimeOffset? _fechaNacimiento;
00039         public DateTimeOffset? FechaNacimiento
00040         {
00041             get => _fechaNacimiento;
00042             set => this.RaiseAndSetIfChanged(ref _fechaNacimiento, value);
00043         }
00044 
00045         // 0 = Privada, 1 = Pública
00046         private int _indexPrivacidad;
00047         public int IndexPrivacidad
00048         {
00049             get => _indexPrivacidad;
00050             set => this.RaiseAndSetIfChanged(ref _indexPrivacidad, value);
```

```
00051         }
00052
00053         // Comandos Reactive
00054         public ReactiveCommand<Unit, Unit> BtnGuardar { get; }
00055         public ReactiveCommand<Unit, Unit> BtnRefrescar { get; }
00056
00057         public ViewCuentaViewModel()
00058         {
00059             // Incializamos servicios
00060             _dialogoService = new DialogoService();
00061
00062             // Configuramos comandos
00063             BtnGuardar = ReactiveCommand.CreateFromTask(GuardarCambios);
00064             BtnRefrescar = ReactiveCommand.Create(CargarDatos);
00065
00066             // Cargamos datos
00067             CargarDatos();
00068         }
00069         /// <summary>
00070         /// Recupera y sincroniza la información del perfil del usuario desde los datos globales para su edición en la interfaz.
00071         /// </summary>
00072         /// <remarks>
00073         /// Este método actúa como un mapeador entre <see cref="GlobalData.Instance"/> y las propiedades vinculadas de
    la vista.
00074         /// Realiza conversiones de tipos necesarias, como la transformación de <see cref="DateTime"/> a <see
    cref="DateTimeOffset"/>
00075         /// para el selector de fecha, y traduce el estado booleano de privacidad a un índice numérico compatible con
00076         /// los controles de selección de la UI.
00077         /// </remarks>
00078         private void CargarDatos()
00079         {
00080             // Leemos del Singleton (GlobalData)
00081             NombreUsuario = GlobalData.Instance.UsernameGD;
00082             Email = GlobalData.Instance.EmailGD;
00083             Pais = GlobalData.Instance.PaisGD;
00084
00085             // Conversión de Fechas
00086             if (GlobalData.Instance.FechaNacimientoGD != DateTime.MinValue)
00087             {
00088                 FechaNacimiento = new DateTimeOffset(GlobalData.Instance.FechaNacimientoGD);
00089             }
00090             else
00091             {
00092                 FechaNacimiento = DateTimeOffset.Now;
00093             }
00094
00095             // Conversión de Privacidad (True = Privada = Index 0)
00096             IndexPrivacidad = GlobalData.Instance.Es_PrivadaGD ? 0 : 1;
00097         }
00098         /// <summary>
00099         /// Procesa y persiste de forma asíncrona las modificaciones realizadas en el perfil del usuario tanto en la base de
    datos como en el estado global.
00100         /// </summary>
00101         /// <remarks>
00102         /// Este método realiza una validación y transformación de datos antes de la persistencia:
00103         /// <list type="number">
00104         /// <item><b>Conversión:</b> Transforma el objeto <see cref="DateTimeOffset"/> de la interfaz a <see
    cref="DateTime"/> y el índice de privacidad a un valor booleano.</item>
00105         /// <item><b>Sincronización remota:</b> Invoca al cliente de MongoDB para actualizar los documentos en la
    nube.</item>
00106         /// <item><b>Actualización local:</b> Si la operación remota es exitosa, sincroniza los nuevos valores en <see
    cref="GlobalData.Instance"/> para mantener la consistencia en la sesión actual.</item>
00107         /// </list>
00108         /// Notifica el resultado de la operación al usuario mediante el servicio de diálogos y registra errores críticos en la
    consola de depuración.
00109         /// </remarks>
00110         /// <returns>Una tarea que representa la operación de guardado asíncrona.</returns>
00111         private async Task GuardarCambios()
00112         {
00113             try
00114             {
00115                 // Convertimos los datos de la vista a los formatos de BD
00116                 var fechaParaGuardar = FechaNacimiento?.DateTime ?? DateTime.Now;
00117                 bool esCuentaPrivada = (IndexPrivacidad == 0);
00118
00119                 bool exito = await MongoClientSingleton.Instance.Cliente.ActualizarPerfilUsuario(
00120                     GlobalData.Instance.UserIdGD,
00121                     NombreUsuario,
00122                     Email,
00123                     Pais,
00124                     fechaParaGuardar,
00125                     esCuentaPrivada
00126                 );
00127
00128                 // Comprobamos si se actualizo correctamente y reflajamos los cambios en el Singleton (GlobalData)
00129                 if (exito)
00130                 {
```

```
00131                 GlobalData.Instance.UsernameGD = NombreUsuario;
00132                 GlobalData.Instance.EmailGD = Email;
00133                 GlobalData.Instance.PaisGD = Pais;
00134                 GlobalData.Instance.FechaNacimientoGD = fechaParaGuardar;
00135                 GlobalData.Instance.Es_PrivadaGD = esCuentaPrivada;
00136
00137                 _dialogoService.MostrarAlerta("MsgExitoActualizarPerfil");
00138                 System.Diagnostics.Debug.WriteLine("¡Perfil actualizado en BD y Memoria!");
00139             }
00140             else
00141             {
00142                 _dialogoService.MostrarAlerta("MsgErrorActualizarPerfil");
00143                 System.Diagnostics.Debug.WriteLine("Fallo al actualizar en Mongo.");
00144             }
00145         }
00146         catch (Exception ex)
00147         {
00148             System.Diagnostics.Debug.WriteLine("Error crítico al guardar perfil: " + ex.Message);
00149         }
00150     }
00151 }
00152 }
```

4.119. Referencia del archivo BetaProyecto/ViewModels/ViewEditarCancionViewModel.cs

4.120. ViewEditarCancionViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Media.Imaging;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Services;
00004 using BetaProyecto.Singleton;
00005 using ReactiveUI;
00006 using System;
00007 using System.Collections.ObjectModel;
00008 using System.Linq;
00009 using System.Net.Http;
00010 using System.Reactive;
00011 using System.Reactive.Linq;
00012 using System.Threading.Tasks;
00013
00014 namespace BetaProyecto.ViewModels
00015 {
00016     public class ViewEditarCancionViewModel : ViewModelBase
00017     {
00018         //Propiedad
00019         private readonly Canciones _cancionOriginal;
00020
00021         //Servicios
00022         private readonly IDialogoService _dialogoService;
00023         private readonly StorageService _storageService;
00024
00025         //Actions
00026         private readonly Action _accionVolver;
00027
00028         // Bidings
00029         private string _txtTitulo;
00030         public string TxtTitulo
00031         {
00032             get => _txtTitulo;
00033             set => this.RaiseAndSetIfChanged(ref _txtTitulo, value);
00034         }
00035
00036         private ObservableCollection<string> _listaGeneros;
00037         public ObservableCollection<string> ListaGeneros
00038         {
00039             get => _listaGeneros;
00040             set => this.RaiseAndSetIfChanged(ref _listaGeneros, value);
00041         }
00042
00043         private string _generoSeleccionado;
00044         public string GeneroSeleccionado
00045         {
00046             get => _generoSeleccionado;
00047             set => this.RaiseAndSetIfChanged(ref _generoSeleccionado, value);
00048         }
00049
00050         private ObservableCollection<string> _listaGenerosSeleccionados;
```

```
00051         public ObservableCollection<string> ListaGenerosSeleccionados
00052         {
00053             get => _listaGenerosSeleccionados;
00054             set => this.RaiseAndSetIfChanged(ref _listaGenerosSeleccionados, value);
00055         }
00056 
00057         private string _txtBusqueda;
00058         public string TxtBusqueda
00059         {
00060             get => _txtBusqueda;
00061             set => this.RaiseAndSetIfChanged(ref _txtBusqueda, value);
00062         }
00063 
00064         private ObservableCollection<Usuarios> _listaResultados;
00065         public ObservableCollection<Usuarios> ListaResultados
00066         {
00067             get => _listaResultados;
00068             set => this.RaiseAndSetIfChanged(ref _listaResultados, value);
00069         }
00070 
00071         private ObservableCollection<Usuarios> _listaArtistas;
00072         public ObservableCollection<Usuarios> ListaArtistas
00073         {
00074             get => _listaArtistas;
00075             set => this.RaiseAndSetIfChanged(ref _listaArtistas, value);
00076         }
00077 
00078         private string _rutaImagen;
00079         public string RutaImagen
00080         {
00081             get => _rutaImagen;
00082             set
00083             {
00084                 this.RaiseAndSetIfChanged(ref _rutaImagen, value);
00085                 // Si es archivo local, cargamos preview
00086                 if (!value.StartsWith("http")) CargarImagenLocal(value);
00087             }
00088         }
00089 
00090         private bool _tieneImagen;
00091         public bool TieneImagen
00092         {
00093             get => _tieneImagen;
00094             set => this.RaiseAndSetIfChanged(ref _tieneImagen, value);
00095         }
00096 
00097         private Bitmap? _imagenPortada;
00098         public Bitmap? ImagenPortada
00099         {
00100             get => _imagenPortada;
00101             set => this.RaiseAndSetIfChanged(ref _imagenPortada, value);
00102         }
00103 
00104         // Progress bar
00105         private bool _estaCargando;
00106         public bool EstaCargando
00107         {
00108             get => _estaCargando;
00109             set => this.RaiseAndSetIfChanged(ref _estaCargando, value);
00110         }
00111 
00112         //  Comandos Reactive
00113         public ReactiveCommand<Unit, Unit> BtnCancelar { get; }
00114         public ReactiveCommand<Unit, Unit> BtnGuardar { get; }
00115         public ReactiveCommand<Unit, Unit> BtnAgregarGenero { get; }
00116         public ReactiveCommand<string, Unit> BtnEliminarGenero { get; }
00117         public ReactiveCommand<Unit, Unit> BtnBuscarUsuarios { get; }
00118         public ReactiveCommand<Usuarios, Unit> BtnAgregarUsuario { get; }
00119         public ReactiveCommand<Usuarios, Unit> BtnEliminarUsuario { get; }
00120 
00121 
00122         // Constructor
00123         public ViewEditarCancionViewModel(Canciones cancion, Action accionVolver)
00124         {
00125             //Asignamos propiedades
00126             _cancionOriginal = cancion;
00127             _accionVolver = accionVolver;
00128 
00129             // Inicializamos servicios
00130             _dialogoService = new DialogoService();
00131             _storageService = new StorageService();
00132 
00133             // Inicializar listas
00134             ListaResultados = new ObservableCollection<Usuarios>();
00135             ListaArtistas = new ObservableCollection<Usuarios>();
00136             ListaGeneros = new ObservableCollection<string>();
00137             ListaGenerosSeleccionados = new ObservableCollection<string>();
```

```
00138
00139           // Cargamos datos
00140           TxtTitulo = cancion.Titulo;
00141           RutaImagen = cancion.ImagenPortadaUrl;
00142           ListaGenerosSeleccionados = new ObservableCollection<string>(cancion.Datos.Generos);
00143
00144           // Cargar imagen visualmente
00145           _ = CargarImagenDesdeUrl(cancion.ImagenPortadaUrl);
00146
00147           // Configuramos comandos reactivos
00148           BtnCancelar = ReactiveCommand.Create(() => _accionVolver());
00149           BtnAgregarGenero = ReactiveCommand.Create(AgregarGenero);
00150           BtnEliminarGenero = ReactiveCommand.Create<string>(EliminarGenero);
00151           BtnBuscarUsuarios = ReactiveCommand.Create(BuscarUsuarios);
00152           BtnAgregarUsuario = ReactiveCommand.Create<Usuarios>(AgregarUsuario);
00153           BtnEliminarUsuario = ReactiveCommand.Create<Usuarios>(EliminarUsuario);
00154
00155           // Buscador reactivo
00156           this.WhenAnyValue(x => x.TxtBusqueda)
00157               .Throttle(TimeSpan.FromMilliseconds(500))
00158               .Where(x => !string.IsNullOrWhiteSpace(x) && x.Length > 2)
00159               .ObserveOn(RxApp.MainThreadScheduler)
00160               .Subscribe(_ => BuscarUsuarios());
00161
00162           // Validación Guardar
00163           var validacionGuardar = this.WhenAnyValue(
00164               x => x.TxtTitulo,
00165               x => x.RutaImagen,
00166               x => x.ListaGenerosSeleccionados.Count,
00167               (titulo, imagen, generos) =>
00168                   !string.IsNullOrWhiteSpace(titulo) &&
00169                   !string.IsNullOrWhiteSpace(imagen) &&
00170                   generos > 0
00171           );
00172
00173           BtnGuardar = ReactiveCommand.CreateFromTask(GuardarCambios, validacionGuardar);
00174
00175           // Cargamos datos en segundo plano
00176           _ = CargarGenerosDisponibles();
00177           _ = CargarColaboradoresOriginales();
00178       }
00179       /// <summary>
00180       /// Recupera de forma asíncrona el catálogo completo de géneros musicales definidos en el sistema.
00181       /// </summary>
00182       /// <remarks>
00183       /// Este método establece conexión con la base de datos a través de <see cref="MongoClientSingleton"/> para
00184       /// obtener la lista maestra de nombres de géneros. Una vez recibidos, inicializa la propiedad
00185       /// <see cref="ListaGeneros"/> con una nueva colección observable, permitiendo que los selectores de la interfaz
00186       /// de usuario se pueblen dinámicamente con los valores actualizados.
00187       /// </remarks>
00188       /// <returns>Una tarea que representa la operación de carga asíncrona.</returns>
00189       private async Task CargarGenerosDisponibles()
00190       {
00191           if (MongoClientSingleton.Instance.Cliente != null)
00192           {
00193               var generos = await MongoClientSingleton.Instance.Cliente.ObtenerNombresGeneros();
00194               ListaGeneros = new ObservableCollection<string>(generos);
00195           }
00196       }
00197
00198       /// <summary>
00199       /// Recupera de forma asíncrona la información detallada de los colaboradores originales de la canción basándose en
      sus identificadores.
00200       /// </summary>
00201       /// <remarks>
00202       /// Este método gestiona la conversión de la lista de IDs almacenada en los metadatos de la canción a objetos de tipo
      <see cref="Usuarios"/>.
00203       /// Consulta la base de datos a través de <see cref="MongoClientSingleton"/> y, tras obtener los perfiles
      correspondientes,
00204       /// inicializa la propiedad <see cref="ListaArtistas"/> con una nueva colección observable para su representación en
      la interfaz.
00205       /// </remarks>
00206       /// <returns>Una tarea que representa la operación de carga asíncrona.</returns>
00207       private async Task CargarColaboradoresOriginales()
00208       {
00209           // Necesitamos convertir la lista de IDs en objetos Usuarios
00210           if (_cancionOriginal.AutoresIds != null && _cancionOriginal.AutoresIds.Count > 0)
00211           {
00212               var usuarios = await
      MongoClientSingleton.Instance.Cliente.ObtenerUsuariosPorListaIds(_cancionOriginal.AutoresIds);
00213
00214               if (usuarios != null)
00215                   ListaArtistas = new ObservableCollection<Usuarios>(usuarios);
00216           }
00217       }
00218       /// <summary>
00219       /// Descarga de forma asíncrona una imagen desde una dirección URL y la asigna al mapa de bits de la portada.
```

```
00220         /// </summary>
00221         /// <remarks>
00222         /// Este método gestiona la recuperación de recursos remotos mediante los siguientes pasos:
00223         /// <list type="number">
00224         /// <item><b>Petición HTTP:</b> Utiliza un <see cref="HttpClient"/> para obtener el flujo de bytes de la
      imagen desde la red.</item>
00225         /// <item><b>Procesamiento de Memoria:</b> Transfiere los bytes a un <see
      cref="System.IO.MemoryStream"/> para su decodificación.</item>
00226         /// <item><b>Asignación Visual:</b> Inicializa la propiedad <see cref="ImagenPortada"/> con el nuevo <see
      cref="Bitmap"/> y actualiza el estado de <see cref="TieneImagen"/>.</item>
00227         /// </list>
00228         /// El método incluye un bloque try-catch silencioso para asegurar que fallos en la red o URLs inválidas no
      interrumpan la ejecución de la aplicación.
00229         /// </remarks>
00230         /// <param name="url">La dirección URL completa de la imagen que se desea cargar.</param>
00231         /// <returns>Una tarea que representa la operación de carga asíncrona.</returns>
00232         private async Task CargarImagenDesdeUrl(string url)
00233         {
00234             if (string.IsNullOrEmpty(url)) return;
00235             try
00236             {
00237                 using (var client = new HttpClient())
00238                 {
00239                     var bytes = await client.GetByteArrayAsync(url);
00240                     using (var stream = new System.IO.MemoryStream(bytes))
00241                     {
00242                         ImagenPortada = new Bitmap(stream);
00243                         TieneImagen = true;
00244                     }
00245                 }
00246             }
00247             catch {
00248                 _dialogoService.MostrarAlerta("Msg_Error_CargarImagen");
00249             }
00250         }
00251 
00252         /// <summary>
00253         /// Intenta cargar y visualizar una imagen desde el almacenamiento local del sistema.
00254         /// </summary>
00255         /// <remarks>
00256         /// Este método gestiona la carga de recursos gráficos locales mediante los siguientes pasos:
00257         /// <list type="number">
00258         /// <item><b>Validación:</b> Comprueba la existencia física del archivo en la <paramref name="ruta"/>
      proporcionada.</item>
00259         /// <item><b>Decodificación:</b> Si el archivo existe, inicializa la propiedad <see cref="ImagenPortada"/> con
      un nuevo objeto <see cref="Bitmap"/>.</item>
00260         /// <item><b>Control de Estado:</b> Actualiza la propiedad booleana <see cref="TieneImagen"/> y notifica el
      cambio a la interfaz mediante <c>RaisePropertyChanged</c>.</item>
00261         /// </list>
00262         /// El método captura cualquier excepción durante la lectura para evitar interrupciones en la ejecución, asegurando
      que el estado de la UI se mantenga consistente.
00263         /// </remarks>
00264         /// <param name="ruta">La ruta absoluta del archivo de imagen en el disco local.</param>
00265         private void CargarImagenLocal(string ruta)
00266         {
00267             try
00268             {
00269                 if (System.IO.File.Exists(ruta))
00270                 {
00271                     ImagenPortada = new Bitmap(ruta);
00272                     TieneImagen = true;
00273                 }
00274                 else TieneImagen = false;
00275             }
00276             catch { TieneImagen = false; }
00277             this.RaisePropertyChanged(nameof(TieneImagen));
00278         }
00279 
00280         /// <summary>
00281         /// Añade el género seleccionado actualmente a la lista de géneros asociados, validando que no esté vacío y que no se
      haya añadido previamente.
00282         /// </summary>
00283         /// <remarks>
00284         /// Este método gestiona la selección de etiquetas musicales mediante los siguientes pasos:
00285         /// <list type="number">
00286         /// <item><b>Validación de entrada:</b> Verifica si <see cref="GeneroSeleccionado"/> contiene un valor válido
      y no nulo.</item>
00287         /// <item><b>Control de duplicados:</b> Comprueba si el género ya existe en <see
      cref="ListaGenerosSeleccionados"/> mediante una comparación insensible a mayúsculas.</item>
00288         /// <item><b>Actualización:</b> Si es un género nuevo, lo añade a la colección. En caso contrario, notifica al
      usuario a través de <see cref="_dialogoService"/>.</item>
00289         /// </list>
00290         /// Al finalizar, restablece la propiedad <see cref="GeneroSeleccionado"/> a nulo para limpiar el selector de la
      interfaz.
00291         /// </remarks>
00292         private void AgregarGenero()
00293         {
```

```
00294           if (string.IsNullOrWhiteSpace(GeneroSeleccionado))
00295           {
00296               return;
00297           }
00298
00299           bool yaEstaEnLista = ListaGenerosSeleccionados.Any(g => g.Equals(GeneroSeleccionado,
    StringComparison.OrdinalIgnoreCase));
00300
00301           if (!yaEstaEnLista)
00302           {
00303               ListaGenerosSeleccionados.Add(GeneroSeleccionado);
00304               GeneroSeleccionado = null;
00305           }
00306           else
00307           {
00308               _dialogoService.MostrarAlerta("Msg_Error_GeneroYaAnadido");
00309               GeneroSeleccionado = null;
00310           }
00311       }
00312       /// <summary>
00313       /// Elimina un género específico de la lista de géneros seleccionados para la canción.
00314       /// </summary>
00315       /// <remarks>
00316       /// Este método gestiona la edición de etiquetas musicales mediante los siguientes pasos:
00317       /// <list type="number">
00318       /// <item><b>Validación:</b> Verifica si el género proporcionado existe dentro de la colección <see
    cref="ListaGenerosSeleccionados"/>.</item>
00319       /// <item><b>Remoción:</b> Si se encuentra la coincidencia, elimina el elemento de la lista.</item>
00320       /// <item><b>Sincronización:</b> La interfaz de usuario se actualiza automáticamente al ser una colección de
    tipo observable.</item>
00321       /// </list>
00322       /// </remarks>
00323       /// <param name="genero">El nombre del género que se desea remover de la selección actual.</param>
00324       private void EliminarGenero(string genero)
00325       {
00326           if (ListaGenerosSeleccionados.Contains(genero))
00327           {
00328               ListaGenerosSeleccionados.Remove(genero);
00329           }
00330       }
00331
00332       /// <summary>
00333       /// Realiza una búsqueda asíncrona de usuarios en la base de datos basándose en el texto introducido, filtrando
    aquellos que ya han sido seleccionados.
00334       /// </summary>
00335       /// <remarks>
00336       /// Este método gestiona la recuperación de perfiles mediante los siguientes pasos:
00337       /// <list type="number">
00338       /// <item><b>Validación de servicio:</b> Verifica la disponibilidad del cliente de MongoDB a través de <see
    cref="MongoClientSingleton"/>.</item>
00339       /// <item><b>Consulta con exclusión:</b> Ejecuta la búsqueda utilizando <see cref="TxtBusqueda"/> y envía
    una lista de IDs de <see cref="ListaArtistas"/> para evitar resultados duplicados.</item>
00340       /// <item><b>Actualización de interfaz:</b> Si se obtienen resultados, inicializa <see cref="ListaResultados"/>
    con una nueva colección observable para refrescar la vista.</item>
00341       /// </list>
00342       /// </remarks>
00343       private async void BuscarUsuarios()
00344       {
00345           if (MongoClientSingleton.Instance.Cliente != null)
00346           {
00347               var resultados = await MongoClientSingleton.Instance.Cliente.ObtenerUsuariosPorBusqueda(TxtBusqueda,
    ListaArtistas.Select(x => x.Id).ToList());
00348               if (resultados != null) ListaResultados = new ObservableCollection<Usuarios>(resultados);
00349           }
00350       }
00351       /// <summary>
00352       /// Añade un usuario a la lista de artistas seleccionados, evitando duplicados y limpiando los resultados de búsqueda
    actuales.
00353       /// </summary>
00354       /// <remarks>
00355       /// Este método gestiona la selección de colaboradores mediante los siguientes pasos:
00356       /// <list type="number">
00357       /// <item><b>Validación de existencia:</b> Verifica mediante el identificador único si el <paramref
    name="usuario"/> ya se encuentra en <see cref="ListaArtistas"/>.</item>
00358       /// <item><b>Actualización de colección:</b> Si el usuario no es un duplicado, se añade a la lista de artistas
    vinculados.</item>
00359       /// <item><b>Limpieza de interfaz:</b> Independientemente del resultado, se restablece <see
    cref="TxtBusqueda"/> y se vacía <see cref="ListaResultados"/> para preparar una nueva consulta.</item>
00360       /// </list>
00361       /// </remarks>
00362       /// <param name="usuario">El objeto de tipo <see cref="Usuarios"/> que se desea vincular a la canción o
    lista.</param>
00363       private void AgregarUsuario(Usuarios usuario)
00364       {
00365           bool yaExiste = ListaArtistas.Any(u => u.Id == usuario.Id);
00366           if (!yaExiste)
00367           {
```

```
00368               ListaArtistas.Add(usuario);
00369           }
00370           TxtBusqueda = string.Empty;
00371           ListaResultados.Clear();
00372       }
00373
00374       /// <summary>
00375       /// Elimina un usuario de la lista de artistas seleccionados, validando que no sea el usuario que ha iniciado sesión.
00376       /// </summary>
00377       /// <remarks>
00378       /// Este método gestiona la remoción de colaboradores mediante los siguientes pasos:
00379       /// <list type="number">
00380       /// <item><b>Validación de identidad:</b> Comprueba si el <paramref name="usuario"/> a eliminar coincide con el ID del usuario actual en <see cref="GlobalData.Instance.UserIdGD"/>.</item>
00381       /// <item><b>Restricción de seguridad:</b> Si coinciden, se muestra una alerta de error mediante <see cref="_dialogoService"/> para impedir que el usuario se elimine a sí mismo.</item>
00382       /// <item><b>Actualización:</b> Si la validación es correcta y el usuario existe en la colección, se procede a removerlo de <see cref="ListaArtistas"/>.</item>
00383       /// </list>
00384       /// </remarks>
00385       /// <param name="usuario">El objeto de tipo <see cref="Usuarios"/> que se desea remover de la selección actual.</param>
00386       private void EliminarUsuario(Usuarios usuario)
00387       {
00388           if (usuario.Id == GlobalData.Instance.UserIdGD)
00389           {
00390               _dialogoService.MostrarAlerta("Msg_Error_BorrarPropioUser");
00391               return;
00392           }
00393           if (ListaArtistas.Contains(usuario))
00394           {
00395               ListaArtistas.Remove(usuario);
00396           }
00397       }
00398
00399
00400       /// <summary>
00401       /// Procesa y persiste de forma asíncrona las modificaciones realizadas en una canción existente, gestionando la actualización de medios y metadatos.
00402       /// </summary>
00403       /// <remarks>
00404       /// Este método orquesta la actualización de la canción siguiendo un flujo transaccional:
00405       /// <list type="number">
00406       /// <item><b>Gestión de Imagen:</b> Detecta si la ruta de la imagen es local o remota. Si es local, procede a subir el nuevo archivo mediante <see cref="_storageService"/>.</item>
00407       /// <item><b>Sincronización remota:</b> Envía los nuevos títulos, IDs de colaboradores y géneros al cliente de MongoDB para actualizar el registro físico.</item>
00408       /// <item><b>Actualización de estado local:</b> Si la persistencia es exitosa, sincroniza los cambios en el objeto <c>_cancionOriginal</c> para asegurar la consistencia visual al regresar a la vista anterior.</item>
00409       /// </list>
00410       /// Durante el proceso, controla la propiedad <see cref="EstaCargando"/> para feedback visual y gestiona posibles excepciones mediante el servicio de diálogos.
00411       /// </remarks>
00412       /// <returns>Una tarea que representa la operación de guardado asíncrona.</returns>
00413       private async Task GuardarCambios()
00414       {
00415           EstaCargando = true;
00416           try
00417           {
00418               string urlPortadaFinal = RutaImagen;
00419
00420               // Si la ruta NO es web (es decir, es un archivo local C:\Users\...), subimos la nueva.
00421               if (!RutaImagen.StartsWith("http"))
00422               {
00423                   urlPortadaFinal = await _storageService.SubirImagen(RutaImagen);
00424               }
00425
00426               // Recogemos los datos nuevos
00427               var nuevosAutoresIds = ListaArtistas.Select(u => u.Id).ToList();
00428               var nuevosGeneros = ListaGenerosSeleccionados.ToList();
00429
00430               // Y los actualizamos a mongo
00431               bool exito = await MongoClientSingleton.Instance.Cliente.ActualizarCancion(
00432                   TxtTitulo,
00433                   urlPortadaFinal,
00434                   nuevosAutoresIds,
00435                   nuevosGeneros,
00436                   _cancionOriginal
00437               );
00438
00439               if (exito)
00440               {
00441                   // Actualizamos el objeto original para que los cambios se reflejen al volver a la vista anterior
00442                   _cancionOriginal.Titulo = TxtTitulo;
00443                   _cancionOriginal.ImagenPortadaUrl = urlPortadaFinal;
00444                   _cancionOriginal.AutoresIds = nuevosAutoresIds;
00445
```

```
00446                 if (_cancionOriginal.Datos == null) _cancionOriginal.Datos = new DatosCancion();
00447                 _cancionOriginal.Datos.Generos = nuevosGeneros;
00448
00449                 // Actualizar nombre artista principal (display)
00450                 if (ListaArtistas.Count > 0)
00451                     _cancionOriginal.NombreArtista = ListaArtistas[0].Username;
00452
00453                 _accionVolver();
00454             }
00455             else
00456             {
00457                 _dialogoService.MostrarAlerta("Msg_Error_GuardarCambios");
00458             }
00459         }
00460         catch (Exception ex)
00461         {
00462             _dialogoService.MostrarAlerta("Msg_Error_Inesperado");
00463             System.Diagnostics.Debug.WriteLine(ex.Message);
00464         }
00465         finally
00466         {
00467             EstaCargando = false;
00468         }
00469     }
00470 }
00471 }
```

4.121. Referencia del archivo BetaProyecto/ViewModels/ViewEditarListaPersonalizadaViewModel.cs

4.122. ViewEditarListaPersonalizadaViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Media.Imaging;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Services;
00004 using BetaProyecto.Singleton;
00005 using ReactiveUI;
00006 using System;
00007 using System.Collections.ObjectModel;
00008 using System.Linq;
00009 using System.Reactive;
00010 using System.Reactive.Linq;
00011 using System.Threading.Tasks;
00012 using System.Net.Http;
00013
00014 namespace BetaProyecto.ViewModels
00015 {
00016     public class ViewEditarListaPersonalizadaViewModel : ViewModelBase
00017     {
00018         //Variables
00019         private readonly ListaPersonalizada _playlistOriginal;
00020
00021         //Sercivios
00022         private readonly IDialogoService _dialogoService;
00023         private readonly StorageService _storageService;
00024
00025         //Actions
00026         private readonly Action? _accionVolver;
00027
00028         // Bidings
00029         private string _txtNombre;
00030         public string TxtNombre
00031         {
00032             get => _txtNombre;
00033             set => this.RaiseAndSetIfChanged(ref _txtNombre, value);
00034         }
00035
00036         private string _txtDescripcion;
00037
00038         public string TxtDescripcion
00039         {
00040             get => _txtDescripcion;
00041             set => this.RaiseAndSetIfChanged(ref _txtDescripcion, value);
00042         }
00043
00044         private string _rutaImagen;
00045         public string RutaImagen
00046         {
```

```
00047           get => _rutaImagen;
00048           set
00049           {
00050               this.RaiseAndSetIfChanged(ref _rutaImagen, value);
00051               // Si cambiamos la ruta a un archivo local, cargamos la preview
00052               if (!value.StartsWith("http"))
00053                   CargarImagenLocal(value);
00054           }
00055       }
00056
00057       private bool _tieneImagen;
00058       public bool TieneImagen
00059       {
00060           get => _tieneImagen;
00061           set => this.RaiseAndSetIfChanged(ref _tieneImagen, value);
00062       }
00063
00064       private Bitmap? _imagenPortada;
00065       public Bitmap? ImagenPortada
00066       {
00067           get => _imagenPortada;
00068           set => this.RaiseAndSetIfChanged(ref _imagenPortada, value);
00069       }
00070
00071       private string _txtBusqueda;
00072       public string TxtBusqueda
00073       {
00074           get => _txtBusqueda;
00075           set => this.RaiseAndSetIfChanged(ref _txtBusqueda, value);
00076       }
00077
00078       private ObservableCollection<Canciones> _listaResultados;
00079       public ObservableCollection<Canciones> ListaResultados
00080       {
00081           get => _listaResultados;
00082           set => this.RaiseAndSetIfChanged(ref _listaResultados, value);
00083       }
00084
00085       private ObservableCollection<Canciones> _listaCancionesSeleccionadas;
00086       public ObservableCollection<Canciones> ListaCancionesSeleccionadas
00087       {
00088           get => _listaCancionesSeleccionadas;
00089           set => this.RaiseAndSetIfChanged(ref _listaCancionesSeleccionadas, value);
00090       }
00091
00092       // Para el progess bar
00093       private bool _estaCargando;
00094       public bool EstaCargando
00095       {
00096           get => _estaCargando;
00097           set => this.RaiseAndSetIfChanged(ref _estaCargando, value);
00098       }
00099
00100       // Comandos Reactive
00101       public ReactiveCommand<Unit, Unit> BtnAtras { get; }
00102       public ReactiveCommand<Unit, Unit> BtnGuardar { get; }
00103       public ReactiveCommand<Unit, Unit> BtnBuscarCanciones { get; }
00104       public ReactiveCommand<Canciones, Unit> BtnAgregarCancion { get; }
00105       public ReactiveCommand<Canciones, Unit> BtnEliminarCancion { get; }
00106
00107       // Constructor
00108       public ViewEditarListaPersonalizadaViewModel(ListaPersonalizada playlist, Action accionVolver)
00109       {
00110           _playlistOriginal = playlist;
00111           _accionVolver = accionVolver;
00112
00113           //Inicializamos servicios
00114           _dialogoService = new DialogoService();
00115           _storageService = new StorageService();
00116
00117           ListaResultados = new ObservableCollection<Canciones>();
00118
00119           // Cargar datos originales
00120           TxtNombre = playlist.Nombre;
00121           TxtDescripcion = playlist.Descripcion;
00122           RutaImagen = playlist.UrlPortada; // Inicialmente es la URL de la nube
00123
00124           // Cargar canciones
00125           ListaCancionesSeleccionadas = new ObservableCollection<Canciones>(playlist.CancionesCompletas);
00126
00127           // Cargar la imagen visualmente
00128           _ = CargarImagenDesdeUrl(playlist.UrlPortada);
00129
00130           //Configuramos comandos reative
00131           BtnAtras = ReactiveCommand.Create(() => _accionVolver());
00132           BtnBuscarCanciones = ReactiveCommand.Create(BuscarCanciones);
00133           BtnAgregarCancion = ReactiveCommand.Create<Canciones>(AgregarCancion);
```

```
00134           BtnEliminarCancion = ReactiveCommand.Create<Canciones>(EliminarCancion);
00135
00136           // Buscador Reactivo
00137           this.WhenAnyValue(x => x.TxtBusqueda)
00138               .Throttle(TimeSpan.FromMilliseconds(500))
00139               .Where(x => !string.IsNullOrWhiteSpace(x) && x.Length > 2)
00140               .ObserveOn(RxApp.MainThreadScheduler)
00141               .Subscribe(_ => BuscarCanciones());
00142
00143           // Validación para Guardar
00144           var validacionGuardar = this.WhenAnyValue(
00145               x => x.TxtNombre,
00146               x => x.RutaImagen,
00147               x => x.ListaCancionesSeleccionadas.Count,
00148               (nombre, imagen, count) =>
00149                   !string.IsNullOrWhiteSpace(nombre) &&
00150                   !string.IsNullOrWhiteSpace(imagen) &&
00151                   count > 0
00152           );
00153
00154           BtnGuardar = ReactiveCommand.CreateFromTask(GuardarCambios, validacionGuardar);
00155       }
00156
00157       /// <summary>
00158       /// Descarga de forma asíncrona una imagen desde una dirección URL y la asigna al mapa de bits de la portada.
00159       /// </summary>
00160       /// <remarks>
00161       /// Este método gestiona la recuperación de recursos remotos mediante los siguientes pasos:
00162       /// <list type="number">
00163       /// <item><b>Petición HTTP:</b> Utiliza un <see cref="HttpClient"/> para obtener el flujo de bytes de la
      imagen desde la red.</item>
00164       /// <item><b>Procesamiento de Memoria:</b> Transfiere los bytes a un <see
      cref="System.IO.MemoryStream"/> para su decodificación.</item>
00165       /// <item><b>Asignación Visual:</b> Inicializa la propiedad <see cref="ImagenPortada"/> con el nuevo <see
      cref="Bitmap"/> y actualiza el estado de <see cref="TieneImagen"/>.</item>
00166       /// </list>
00167       /// En caso de error en la red o formato inválido, se captura la excepción y se notifica al usuario mediante <see
      cref="_dialogoService"/>.
00168       /// </remarks>
00169       /// <param name="url">La dirección URL completa de la imagen que se desea cargar.</param>
00170       /// <returns>Una tarea que representa la operación de carga asíncrona.</returns>
00171       private async Task CargarImagenDesdeUrl(string url)
00172       {
00173           if (string.IsNullOrEmpty(url)) return;
00174           try
00175           {
00176               using (var client = new HttpClient())
00177               {
00178                   var bytes = await client.GetByteArrayAsync(url);
00179                   using (var stream = new System.IO.MemoryStream(bytes))
00180                   {
00181                       ImagenPortada = new Bitmap(stream);
00182                       TieneImagen = true;
00183                   }
00184               }
00185           }
00186           catch {
00187               _dialogoService.MostrarAlerta("Msg_Error_CargarImagen");
00188           }
00189       }
00190       /// <summary>
00191       /// Intenta cargar y asignar una imagen desde el almacenamiento local del sistema de archivos.
00192       /// </summary>
00193       /// <remarks>
00194       /// Este método gestiona la carga de recursos gráficos locales mediante los siguientes pasos:
00195       /// <list type="number">
00196       /// <item><b>Validación de ruta:</b> Verifica la existencia física del archivo mediante <see
      cref="System.IO.File.Exists"/>.</item>
00197       /// <item><b>Instanciación:</b> Si el archivo es válido, crea un nuevo objeto <see cref="Bitmap"/> y lo asigna
      a <see cref="ImagenPortada"/>.</item>
00198       /// <item><b>Control de estado:</b> Actualiza la propiedad booleana <see cref="TieneImagen"/> para reflejar
      el éxito o fallo de la operación en la interfaz.</item>
00199       /// </list>
00200       /// El bloque <c>try-catch</c> asegura que errores de formato o permisos de lectura no interrumpan la ejecución
      del programa.
00201       /// </remarks>
00202       /// <param name="ruta">La ruta absoluta en el disco local donde se encuentra el archivo de imagen.</param>
00203       private void CargarImagenLocal(string ruta)
00204       {
00205           try
00206           {
00207               if (System.IO.File.Exists(ruta))
00208               {
00209                   ImagenPortada = new Bitmap(ruta);
00210                   TieneImagen = true;
00211               }
00212               else
```

```
00213                {
00214                    TieneImagen = false;
00215                }
00216            }
00217            catch
00218            {
00219                TieneImagen = false;
00220            }
00221        }
00222
00223        /// <summary>
00224        /// Realiza una búsqueda asíncrona de canciones en la base de datos y filtra aquellas que ya han sido seleccionadas para la lista actual.
00225        /// </summary>
00226        /// <remarks>
00227        /// Este método gestiona el filtrado dinámico de contenido mediante los siguientes pasos:
00228        /// <list type="number">
00229        /// <item><b>Consulta remota:</b> Solicita al cliente de MongoDB las canciones que coincidan con el término almacenado en <see cref="TxtBusqueda"/>.</item>
00230        /// <item><b>Filtrado local:</b> Aplica una operación LINQ para excluir de los resultados cualquier canción cuyo identificador ya se encuentre en <see cref="ListaCancionesSeleccionadas"/>.</item>
00231        /// <item><b>Actualización de UI:</b> Inicializa la propiedad <see cref="ListaResultados"/> con una nueva colección observable, permitiendo que la interfaz muestre únicamente las opciones elegibles.</item>
00232        /// </list>
00233        /// </remarks>
00234        private async void BuscarCanciones()
00235        {
00236            if (MongoClientSingleton.Instance.Cliente != null)
00237            {
00238                var resultados = await MongoClientSingleton.Instance.Cliente.ObtenerCancionesPorBusqueda(TxtBusqueda);
00239                if (resultados != null)
00240                {
00241                    var filtradas = resultados.Where(c => !ListaCancionesSeleccionadas.Any(sel => sel.Id == c.Id));
00242                    ListaResultados = new ObservableCollection<Canciones>(filtradas);
00243                }
00244            }
00245        }
00246        /// <summary>
00247        /// Incorpora una canción específica a la lista de selección actual y limpia el estado de búsqueda.
00248        /// </summary>
00249        /// <remarks>
00250        /// Este método gestiona la selección de pistas musicales mediante los siguientes pasos:
00251        /// <list type="number">
00252        /// <item><b>Validación de unicidad:</b> Verifica que la canción no haya sido agregada previamente comparando su identificador único.</item>
00253        /// <item><b>Transferencia de estado:</b> Añade la canción a <see cref="ListaCancionesSeleccionadas"/> y la remueve simultáneamente de la lista de resultados de búsqueda para evitar duplicidad visual.</item>
00254        /// <item><b>Reinicio de filtros:</b> Restablece la cadena de búsqueda <see cref="TxtBusqueda"/> para facilitar una nueva consulta.</item>
00255        /// </list>
00256        /// </remarks>
00257        /// <param name="cancion">El objeto de tipo <see cref="Canciones"/> que se desea añadir a la lista o playlist.</param>
00258        private void AgregarCancion(Canciones cancion)
00259        {
00260            if (!ListaCancionesSeleccionadas.Any(c => c.Id == cancion.Id))
00261            {
00262                ListaCancionesSeleccionadas.Add(cancion);
00263                ListaResultados.Remove(cancion);
00264                TxtBusqueda = "";
00265            }
00266        }
00267        /// <summary>
00268        /// Remueve una canción específica de la colección de pistas seleccionadas para la lista de reproducción.
00269        /// </summary>
00270        /// <remarks>
00271        /// Este método gestiona la edición de la lista mediante los siguientes pasos:
00272        /// <list type="number">
00273        /// <item><b>Identificación:</b> Localiza la instancia del objeto <see cref="Canciones"/> dentro de la colección <see cref="ListaCancionesSeleccionadas"/>.</item>
00274        /// <item><b>Remoción:</b> Elimina el elemento de la lista, lo cual desencadena automáticamente la actualización de la interfaz de usuario al ser una colección observable.</item>
00275        /// </list>
00276        /// </remarks>
00277        /// <param name="cancion">El objeto de tipo <see cref="Canciones"/> que se desea retirar de la selección actual.</param>
00278        private void EliminarCancion(Canciones cancion)
00279        {
00280            ListaCancionesSeleccionadas.Remove(cancion);
00281        }
00282
00283        /// <summary>
00284        /// Procesa y persiste de forma asíncrona las modificaciones realizadas en una lista de reproducción existente, incluyendo la gestión de medios y la estructura de pistas.
00285        /// </summary>
00286        /// <remarks>
00287        /// Este método orquesta la actualización de la playlist mediante el siguiente flujo de trabajo:
```

```
00288         /// <list type="number">
00289         /// <item><b>Sincronización de Imagen:</b> Evalúa si la ruta de la portada es local o remota. En caso de ser local, sube el archivo a la nube mediante <see cref="_storageService"/> para obtener una URL persistente.</item>
00290         /// <item><b>Preparación de Metadatos:</b> Extrae y proyecta los identificadores únicos de la colección <see cref="ListaCancionesSeleccionadas"/>.</item>
00291         /// <item><b>Persistencia en BD:</b> Invoca al cliente de MongoDB para actualizar el nombre, descripción, lista de IDs y URL de portada en el documento correspondiente.</item>
00292         /// <item><b>Finalización:</b> Tras el éxito, libera el estado de carga y ejecuta la acción de retorno a la vista anterior.</item>
00293         /// </list>
00294         /// En caso de error, se notifica al usuario mediante el servicio de diálogos y se registra la excepción para depuración.
00295         /// </remarks>
00296         /// <returns>Una tarea que representa la operación de guardado asíncrona.</returns>
00297         private async Task GuardarCambios()
00298         {
00299             EstaCargando = true;
00300             try
00301             {
00302                 string urlPortadaFinal = RutaImagen;
00303 
00304                 //Si la imagen es local, la subimos y obtenemos la URL de la nube
00305                 if (!RutaImagen.StartsWith("http"))
00306                 {
00307                     urlPortadaFinal = await _storageService.SubirImagen(RutaImagen);
00308                 }
00309 
00310                 // Extraeremos IDs de canciones
00311                 var nuevosIds = ListaCancionesSeleccionadas.Select(c => c.Id).ToList();
00312 
00313                 // Actualizamos en la BD
00314                 await MongoClientSingleton.Instance.Cliente.ActualizarPlaylist(
00315                     TxtNombre,
00316                     TxtDescripcion,
00317                     nuevosIds,
00318                     urlPortadaFinal,
00319                     _playlistOriginal
00320                     );
00321 
00322                 EstaCargando = false;
00323                 _accionVolver();
00324             }
00325             catch (Exception ex)
00326             {
00327                 EstaCargando = false;
00328                 _dialogoService.MostrarAlerta("Msg_Error_ActualizarPlaylist");
00329                 System.Diagnostics.Debug.WriteLine($"Error al actualizar playlist: {ex.Message}");
00330             }
00331         }
00332     }
00333 }
```

4.123. Referencia del archivo BetaProyecto/ViewModels/ViewGestionarBDViewModel.cs

4.124. ViewGestionarBDViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Controls;
00002 using BetaProyecto.Helpers;
00003 using BetaProyecto.Models;
00004 using BetaProyecto.Services;
00005 using BetaProyecto.Singleton;
00006 using MongoDB.Bson;
00007 using MongoDB.Driver;
00008 using ReactiveUI;
00009 using System;
00010 using System.Collections.Generic;
00011 using System.Collections.ObjectModel;
00012 using System.IO;
00013 using System.Linq;
00014 using System.Reactive;
00015 using System.Reactive.Linq;
00016 using System.Threading.Tasks;
00017 
00018 namespace BetaProyecto.ViewModels
00019 {
00020     public class ViewGestionarBDViewModel : ViewModelBase
00021     {
00022         //Servicios
```

```
00023         private readonly IDialogoService _dialogoService;
00024         private readonly StorageService _storageService;
00025         private readonly AudioService _audioService;
00026 
00027         //Comandos reactive
00028         public ReactiveCommand<Unit, Unit> BtnRecargar { get; }
00029         public ReactiveCommand<Unit, Unit> BtnGuardarCambios { get; }
00030 
00031         //Biding
00032         // Para saber en qué pestaña estamos (0=Usuarios, 1=Canciones, etc.)
00033         private int _indiceTab;
00034         public int IndiceTab
00035         {
00036             get => _indiceTab;
00037             set => this.RaiseAndSetIfChanged(ref _indiceTab, value);
00038         }
00039 
00040         // ============================================
00041         // LOGICA PESTAÑA USUARIOS
00042         // ============================================
00043 
00044         // --- DataGrid / Colecciones ---
00045         public ObservableCollection<Usuarios> ListaUsuarios { get; }
00046         public ObservableCollection<string> RolesDisponibles { get; } // Auxiliar
00047 
00048         // --- Crear ---
00049         private Usuarios _nuevoUsuario;
00050         public Usuarios NuevoUsuario
00051         {
00052             get => _nuevoUsuario;
00053             set => this.RaiseAndSetIfChanged(ref _nuevoUsuario, value);
00054         }
00055         //Comandos reactive
00056         public ReactiveCommand<Unit, Unit> BtnCrearUsuario { get; }
00057         public ReactiveCommand<Unit, Unit> BtnEliminarUsuario { get; }
00058 
00059         // --- Editar / Eliminar ---
00060         private Usuarios _selectedUsuario;
00061         public Usuarios SelectedUsuario
00062         {
00063             get => _selectedUsuario;
00064             set => this.RaiseAndSetIfChanged(ref _selectedUsuario, value);
00065         }
00066 
00067 
00068         // ============================================
00069         // LOGICA PESTAÑA CANCIONES
00070         // ============================================
00071 
00072         // --- DataGrid / Colecciones ---
00073         public ObservableCollection<Canciones> ListaCanciones { get; }
00074         public ObservableCollection<string> ListaGenerosCombox { get; } // Auxiliar
00075 
00076         // --- Crear ---
00077         private Canciones _nuevaCancion;
00078         public Canciones NuevaCancion
00079         {
00080             get => _nuevaCancion;
00081             set => this.RaiseAndSetIfChanged(ref _nuevaCancion, value);
00082         }
00083 
00084         // Buscador Artistas (Crear)
00085         private string _txtBusquedaCrear;
00086         public string TxtBusquedaCrear
00087         {
00088             get => _txtBusquedaCrear;
00089             set => this.RaiseAndSetIfChanged(ref _txtBusquedaCrear, value);
00090         }
00091 
00092         private bool _hayResultadosCrear;
00093         public bool HayResultadosCrear
00094         {
00095             get => _hayResultadosCrear;
00096             set => this.RaiseAndSetIfChanged(ref _hayResultadosCrear, value);
00097         }
00098         public ObservableCollection<Usuarios> ListaResultadosCrear { get; }
00099         public ObservableCollection<Usuarios> ListaArtistasCrear { get; } // Seleccionados
00100 
00101         // Gestión Géneros (Crear)
00102         private string _generoSeleccionadoCrear;
00103         public string GeneroSeleccionadoCrear
00104         {
00105             get => _generoSeleccionadoCrear;
00106             set => this.RaiseAndSetIfChanged(ref _generoSeleccionadoCrear, value);
00107         }
00108         public ObservableCollection<string> ListaGenerosSeleccionadosCrear { get; }
00109 
```

```
00110         // Audio (Crear)
00111         private bool _esArchivoLocal;
00112         public bool EsArchivoLocal
00113         {
00114             get => _esArchivoLocal;
00115             set
00116             {
00117                 this.RaiseAndSetIfChanged(ref _esArchivoLocal, value);
00118                 this.RaisePropertyChanged(nameof(EsYoutube));
00119             }
00120         }
00121         public bool EsYoutube => !EsArchivoLocal;
00122 
00123         // Comandos reactivos para buscador de usuarios (Crear)
00124         public ReactiveCommand<Unit, Unit> BtnBuscarUsuariosCrear { get; }
00125         public ReactiveCommand<Usuarios, Unit> BtnAgregarUsuarioCrear { get; }
00126         public ReactiveCommand<Usuarios, Unit> BtnEliminarUsuarioCrear { get; }
00127 
00128         // Comandos reactivos para gestión de géneros (Crear)
00129         public ReactiveCommand<Unit, Unit> BtnAgregarGeneroCrear { get; }
00130         public ReactiveCommand<string, Unit> BtnEliminarGeneroCrear { get; }
00131 
00132         // Comando reactive para crear una canción
00133         public ReactiveCommand<Unit, Unit> BtnCrearCancion { get; }
00134 
00135         // Comando reactive para eliminar una canción
00136         public ReactiveCommand<Unit, Unit> BtnEliminarCancion { get; }
00137 
00138         // --- Editar / Eliminar ---
00139         private Canciones _selectedCancion;
00140         public Canciones SelectedCancion
00141         {
00142             get => _selectedCancion;
00143             set
00144             {
00145                 this.RaiseAndSetIfChanged(ref _selectedCancion, value);
00146                 CargarDatosEditarCancion(); // Carga artistas, géneros y audio
00147             }
00148         }
00149 
00150         // Buscador Artistas (Editar)
00151         private string _txtBusquedaEditar;
00152         public string TxtBusquedaEditar
00153         {
00154             get => _txtBusquedaEditar;
00155             set => this.RaiseAndSetIfChanged(ref _txtBusquedaEditar, value);
00156         }
00157 
00158         private bool _hayResultadosEditar;
00159         public bool HayResultadosEditar
00160         {
00161             get => _hayResultadosEditar;
00162             set => this.RaiseAndSetIfChanged(ref _hayResultadosEditar, value);
00163         }
00164         public ObservableCollection<Usuarios> ListaResultadosEditar { get; }
00165         public ObservableCollection<Usuarios> ListaArtistasEditar { get; } // Seleccionados
00166 
00167         // Gestión Géneros (Editar)
00168         private string _generoSeleccionadoEditar;
00169         public string GeneroSeleccionadoEditar
00170         {
00171             get => _generoSeleccionadoEditar;
00172             set => this.RaiseAndSetIfChanged(ref _generoSeleccionadoEditar, value);
00173         }
00174         public ObservableCollection<string> ListaGenerosSeleccionadosEditar { get; }
00175 
00176         // Audio (Editar)
00177         private string _txtRutaArchivoEditar;
00178         public string TxtRutaArchivoEditar
00179         {
00180             get => _txtRutaArchivoEditar;
00181             set => this.RaiseAndSetIfChanged(ref _txtRutaArchivoEditar, value);
00182         }
00183 
00184         private string _txtUrlYoutubeEditar;
00185         public string TxtUrlYoutubeEditar
00186         {
00187             get => _txtUrlYoutubeEditar;
00188             set => this.RaiseAndSetIfChanged(ref _txtUrlYoutubeEditar, value);
00189         }
00190 
00191         private bool _esArchivoLocalEditar;
00192         public bool EsArchivoLocalEditar
00193         {
00194             get => _esArchivoLocalEditar;
00195             set
00196             {
```

```
00197             this.RaiseAndSetIfChanged(ref _esArchivoLocalEditar, value);
00198             this.RaisePropertyChanged(nameof(EsYoutubeEditar));
00199         }
00200     }
00201     public bool EsYoutubeEditar => !EsArchivoLocalEditar;
00202
00203     // Comandos reactivos para buscador de usuarios (Editar)
00204     public ReactiveCommand<Unit, Unit> BtnBuscarUsuariosEditar { get; }
00205     public ReactiveCommand<Usuarios, Unit> BtnAgregarUsuarioEditar { get; }
00206     public ReactiveCommand<Usuarios, Unit> BtnEliminarUsuarioEditar { get; }
00207
00208     // Comandos reactivos para gestión de géneros (Editar)
00209     public ReactiveCommand<Unit, Unit> BtnAgregarGeneroEditar { get; }
00210     public ReactiveCommand<string, Unit> BtnEliminarGeneroEditar { get; }
00211
00212     // ==========================================
00213     // LOGICA PESTAÑA GÉNEROS
00214     // ==========================================
00215
00216     // --- DataGrid / Colecciones ---
00217     public ObservableCollection<Generos> ListaGeneros { get; }
00218
00219     // --- Crear ---
00220     private string _nuevoGeneroTxt;
00221     public string NuevoGeneroTxt
00222     {
00223         get => _nuevoGeneroTxt;
00224         set => this.RaiseAndSetIfChanged(ref _nuevoGeneroTxt, value);
00225     }
00226     //Comandos reactive para crear género
00227     public ReactiveCommand<Unit, Unit> BtnCrearGenero { get; }
00228
00229     //Comandos reactive para eliminar género
00230     public ReactiveCommand<Unit, Unit> BtnEliminarGenero { get; }
00231
00232     // --- Editar / Eliminar ---
00233     private Generos _selectedGenero;
00234     public Generos SelectedGenero
00235     {
00236         get => _selectedGenero;
00237         set => this.RaiseAndSetIfChanged(ref _selectedGenero, value);
00238     }
00239
00240     // ==========================================
00241     // LOGICA PESTAÑA PLAYLISTS
00242     // ==========================================
00243
00244     // --- DataGrid / Colecciones ---
00245     public ObservableCollection<ListaPersonalizada> ListaPlaylists { get; }
00246
00247     // --- Crear ---
00248     private ListaPersonalizada _nuevaPlaylist;
00249     public ListaPersonalizada NuevaPlaylist
00250     {
00251         get => _nuevaPlaylist;
00252         set => this.RaiseAndSetIfChanged(ref _nuevaPlaylist, value);
00253     }
00254
00255     // Buscador Canciones para Playlist (Crear)
00256     private string _txtBusquedaCancionCrear;
00257     public string TxtBusquedaCancionCrear
00258     {
00259         get => _txtBusquedaCancionCrear;
00260         set => this.RaiseAndSetIfChanged(ref _txtBusquedaCancionCrear, value);
00261
00262     }
00263
00264     private bool _hayResultadosCancionCrear;
00265     public bool HayResultadosCancionCrear
00266     {
00267         get => _hayResultadosCancionCrear;
00268         set => this.RaiseAndSetIfChanged(ref _hayResultadosCancionCrear, value);
00269     }
00270
00271     public ObservableCollection<Canciones> ListaResultadosCancionesCrear { get; }
00272     public ObservableCollection<Canciones> ListaCancionesPlaylistCrear { get; } // La lista visual de objetos
00273
00274     // Comandos reactivos para buscador de canciones (Crear)
00275     public ReactiveCommand<Unit, Unit> BtnBuscarCancionesCrear { get; }
00276     public ReactiveCommand<Canciones, Unit> BtnAgregarCancionPlaylistCrear { get; }
00277     public ReactiveCommand<Canciones, Unit> BtnEliminarCancionPlaylistCrear { get; }
00278
00279     // Comando reactive para crear una playlist
00280     public ReactiveCommand<Unit, Unit> BtnCrearPlaylist { get; }
00281
00282     // Comando reactive para eliminar una playlist
00283     public ReactiveCommand<Unit, Unit> BtnEliminarPlaylist { get; }
```

```
00284
00285         // --- Editar / Eliminar ---
00286         private ListaPersonalizada _selectedPlaylist;
00287         public ListaPersonalizada SelectedPlaylist
00288         {
00289             get => _selectedPlaylist;
00290             set
00291             {
00292                 this.RaiseAndSetIfChanged(ref _selectedPlaylist, value);
00293                 CargarDatosEditarPlaylist(); // Convierte IDs a Objetos Cancion
00294             }
00295         }
00296
00297         // Buscador Canciones para Playlist (Editar)
00298         private string _txtBusquedaCancionEditar;
00299         public string TxtBusquedaCancionEditar
00300         {
00301             get => _txtBusquedaCancionEditar;
00302             set => this.RaiseAndSetIfChanged(ref _txtBusquedaCancionEditar, value);
00303         }
00304
00305         private bool _hayResultadosCancionEditar;
00306         public bool HayResultadosCancionEditar
00307         {
00308             get => _hayResultadosCancionEditar;
00309             set => this.RaiseAndSetIfChanged(ref _hayResultadosCancionEditar, value);
00310
00311         }
00312         public ObservableCollection<Canciones> ListaResultadosCancionesEditar { get; }
00313         public ObservableCollection<Canciones> ListaCancionesPlaylistEditar { get; } // La lista visual de objetos
00314
00315         // Comandos reactivos para buscador de canciones (Editar)
00316         public ReactiveCommand<Unit, Unit> BtnBuscarCancionesEditar { get; }
00317         public ReactiveCommand<Canciones, Unit> BtnAgregarCancionPlaylistEditar { get; }
00318         public ReactiveCommand<Canciones, Unit> BtnEliminarCancionPlaylistEditar { get; }
00319
00320
00321         // ==========================================
00322         // LOGICA PESTAÑA REPORTES
00323         // ==========================================
00324
00325         // --- DataGrid / Colecciones ---
00326         public ObservableCollection<Reportes> ListaReportes { get; }
00327         public List<string> EstadosReporte { get; } = new List<string>
00328         {
00329             "Pendiente",
00330             "Investigando",
00331             "Finalizado"
00332         };
00333         public ObservableCollection<Reportes> ListaTipoProblema { get; }
00334         public List<string> TipoProblema { get; } = new List<string>
00335         {
00336             "Copyright / Derechos de autor",
00337             "Contenido ofensivo o inapropiado",
00338             "Audio defectuoso o silencio",
00339             "Spam / Información falsa",
00340             "Otro"
00341         };
00342
00343         // --- Crear ---
00344         private Reportes _nuevoReporte;
00345         public Reportes NuevoReporte
00346         {
00347             get => _nuevoReporte;
00348             set => this.RaiseAndSetIfChanged(ref _nuevoReporte, value);
00349         }
00350         //Comandos reactive para crear un reporte
00351         public ReactiveCommand<Unit, Unit> BtnCrearReporte { get; }
00352
00353         //Comandos reactive para eliminar un reporte
00354         public ReactiveCommand<Unit, Unit> BtnEliminarReporte { get; }
00355
00356         // --- Editar ---
00357         private Reportes _selectedReporte;
00358         public Reportes SelectedReporte
00359         {
00360             get => _selectedReporte;
00361             set => this.RaiseAndSetIfChanged(ref _selectedReporte, value);
00362         }
00363         // Progress bar
00364         private bool _estaCargando;
00365         public bool EstaCargando
00366         {
00367             get => _estaCargando;
00368             set
00369             {
00370                 this.RaiseAndSetIfChanged(ref _estaCargando, value);
```

```
00371             this.RaisePropertyChanged(nameof(NoEstaCargando));
00372         }
00373     }
00374     public bool NoEstaCargando => !EstaCargando;
00375
00376     private string _mensajeCarga;
00377     public string MensajeCarga
00378     {
00379         get => _mensajeCarga;
00380         set => this.RaiseAndSetIfChanged(ref _mensajeCarga, value);
00381     }
00382
00383     // =============
00384     // CONSTRUCTOR
00385     // =============
00386     public ViewGestionarBDViewModel()
00387     {
00388         // Inicializamos servicios
00389         _dialogoService = new DialogoService();
00390         _storageService = new StorageService();
00391         _audioService = new AudioService();
00392
00393         // Inicialización Colecciones Tablas
00394         ListaUsuarios = new ObservableCollection<Usuarios>();
00395         ListaCanciones = new ObservableCollection<Canciones>();
00396         ListaPlaylists = new ObservableCollection<ListaPersonalizada>();
00397         ListaReportes = new ObservableCollection<Reportes>();
00398         ListaGeneros = new ObservableCollection<Generos>();
00399
00400         // Inicialización Auxiliares
00401         RolesDisponibles = new ObservableCollection<string> { "SuperAdmin", "Admin", "Usuario" };
00402         ListaGenerosCombox = new ObservableCollection<string>();
00403
00404         // Inicialización Listas y Variables CANCIONES (Crear)
00405         ListaResultadosCrear = new ObservableCollection<Usuarios>();
00406         ListaArtistasCrear = new ObservableCollection<Usuarios>();
00407         ListaGenerosSeleccionadosCrear = new ObservableCollection<string>();
00408         _esArchivoLocal = true;
00409
00410         // Inicialización Listas CANCIONES (Editar)
00411         ListaResultadosEditar = new ObservableCollection<Usuarios>();
00412         ListaArtistasEditar = new ObservableCollection<Usuarios>();
00413         ListaGenerosSeleccionadosEditar = new ObservableCollection<string>();
00414         _esArchivoLocalEditar = true;
00415
00416         // Inicialización Listas PLAYLISTS (Crear y Editar)
00417         ListaResultadosCancionesCrear = new ObservableCollection<Canciones>();
00418         ListaCancionesPlaylistCrear = new ObservableCollection<Canciones>();
00419         ListaResultadosCancionesEditar = new ObservableCollection<Canciones>();
00420         ListaCancionesPlaylistEditar = new ObservableCollection<Canciones>();
00421
00422
00423         //Validación campos reactivos
00424
00425         // Validaciones Eliminar
00426         var hayUsuario = this.WhenAnyValue(x => x.SelectedUsuario)
00427                               .Select(u => u != null);
00428
00429         var hayCancion = this.WhenAnyValue(x => x.SelectedCancion)
00430                               .Select(c => c != null);
00431
00432         var hayPlaylist = this.WhenAnyValue(x => x.SelectedPlaylist)
00433                                .Select(p => p != null);
00434
00435         var hayGenero = this.WhenAnyValue(x => x.SelectedGenero)
00436                              .Select(g => g != null);
00437
00438         var hayReporte = this.WhenAnyValue(x => x.SelectedReporte)
00439                               .Select(r => r != null);
00440
00441         // Comandos reactive generales
00442         BtnRecargar = ReactiveCommand.CreateFromTask(CargarTodo);
00443         BtnGuardarCambios = ReactiveCommand.CreateFromTask(() =>
00444             EjecutarConCarga(GuardarSeleccionado, "Msg_Carga_Guardando"));
00445
00446         // Comandos crear
00447         BtnCrearCancion = ReactiveCommand.CreateFromTask(() =>
00448             EjecutarConCarga(CrearCancionTask, "Msg_Carga_CreandoCancion"));
00449
00450         BtnCrearUsuario = ReactiveCommand.CreateFromTask(() =>
00451             EjecutarConCarga(CrearUsuarioTask, "Msg_Carga_CreandoUsuario"));
00452
00453         BtnCrearPlaylist = ReactiveCommand.CreateFromTask(() =>
00454             EjecutarConCarga(CrearPlaylistTask, "Msg_Carga_CreandoPlaylist"));
00455
00456         BtnCrearReporte = ReactiveCommand.CreateFromTask(() =>
00457             EjecutarConCarga(CrearReporteTask, "Msg_Carga_CreandoReporte"));
```

```
00458
00459           BtnCrearGenero = ReactiveCommand.CreateFromTask(() =>
00460               EjecutarConCarga(AgregarGeneroBD, "Msg_Carga_CreandoGenero"));
00461
00462           // Comandos eliminar
00463           BtnEliminarUsuario = ReactiveCommand.CreateFromTask(EliminarUsuarioTask, hayUsuario);
00464           BtnEliminarCancion = ReactiveCommand.CreateFromTask(EliminarCancionTask, hayCancion);
00465           BtnEliminarPlaylist = ReactiveCommand.CreateFromTask(EliminarPlaylistTask, hayPlaylist);
00466           BtnEliminarGenero = ReactiveCommand.CreateFromTask(EliminarGeneroTask, hayGenero);
00467           BtnEliminarReporte = ReactiveCommand.CreateFromTask(EliminarReporteTask, hayReporte);
00468
00469           // Comandos Artistas/Generos (Crear)
00470           BtnBuscarUsuariosCrear = ReactiveCommand.Create(() => BuscarUsuarios(TxtBusquedaCrear,
    ListaResultadosCrear, ListaArtistasCrear, val => HayResultadosCrear = val));
00471           BtnAgregarUsuarioCrear = ReactiveCommand.Create<Usuarios>(u => AgregarUsuario(u, ListaArtistasCrear, ()
    => { TxtBusquedaCrear = ""; HayResultadosCrear = false; }));
00472           BtnEliminarUsuarioCrear = ReactiveCommand.Create<Usuarios>(u => EliminarUsuario(u, ListaArtistasCrear));
00473           BtnAgregarGeneroCrear = ReactiveCommand.Create(() => AgregarGenero(GeneroSeleccionadoCrear,
    ListaGenerosSeleccionadosCrear, () => GeneroSeleccionadoCrear = null));
00474           BtnEliminarGeneroCrear = ReactiveCommand.Create<string>(g => EliminarGenero(g,
    ListaGenerosSeleccionadosCrear));
00475
00476           // Comandos Artistas/Generos (Editar)
00477           BtnBuscarUsuariosEditar = ReactiveCommand.Create(() => BuscarUsuarios(TxtBusquedaEditar,
    ListaResultadosEditar, ListaArtistasEditar, val => HayResultadosEditar = val));
00478           BtnAgregarUsuarioEditar = ReactiveCommand.Create<Usuarios>(u => AgregarUsuario(u, ListaArtistasEditar,
    () => { TxtBusquedaEditar = ""; HayResultadosEditar = false; }));
00479           BtnEliminarUsuarioEditar = ReactiveCommand.Create<Usuarios>(u => EliminarUsuario(u,
    ListaArtistasEditar));
00480           BtnAgregarGeneroEditar = ReactiveCommand.Create(() => AgregarGenero(GeneroSeleccionadoEditar,
    ListaGenerosSeleccionadosEditar, () => GeneroSeleccionadoEditar = null));
00481           BtnEliminarGeneroEditar = ReactiveCommand.Create<string>(g => EliminarGenero(g,
    ListaGenerosSeleccionadosEditar));
00482
00483           // Comandos Canciones en Playlist (Crear)
00484           BtnBuscarCancionesCrear = ReactiveCommand.Create(() => BuscarCanciones(TxtBusquedaCancionCrear,
    ListaResultadosCancionesCrear, ListaCancionesPlaylistCrear, val => HayResultadosCancionCrear = val));
00485           BtnAgregarCancionPlaylistCrear = ReactiveCommand.Create<Canciones>(c => AgregarCancionAPlaylist(c,
    ListaCancionesPlaylistCrear, () => { TxtBusquedaCancionCrear = ""; HayResultadosCancionCrear = false; }));
00486           BtnEliminarCancionPlaylistCrear = ReactiveCommand.Create<Canciones>(c =>
    ListaCancionesPlaylistCrear.Remove(c));
00487
00488           // Comandos Canciones en Playlist (Editar)
00489           BtnBuscarCancionesEditar = ReactiveCommand.Create(() => BuscarCanciones(TxtBusquedaCancionEditar,
    ListaResultadosCancionesEditar, ListaCancionesPlaylistEditar, val => HayResultadosCancionEditar = val));
00490           BtnAgregarCancionPlaylistEditar = ReactiveCommand.Create<Canciones>(c => AgregarCancionAPlaylist(c,
    ListaCancionesPlaylistEditar, () => { TxtBusquedaCancionEditar = ""; HayResultadosCancionEditar = false; }));
00491           BtnEliminarCancionPlaylistEditar = ReactiveCommand.Create<Canciones>(c =>
    ListaCancionesPlaylistEditar.Remove(c));
00492
00493
00494           // Lógica reactiva (BUSCADORES)
00495           // Artistas
00496           this.WhenAnyValue(x => x.TxtBusquedaCrear).Throttle(TimeSpan.FromMilliseconds(500)).Where(x =>
    !string.IsNullOrWhiteSpace(x)).ObserveOn(RxApp.MainThreadScheduler).Subscribe(_ =>
    BtnBuscarUsuariosCrear.Execute().Subscribe());
00497           this.WhenAnyValue(x => x.TxtBusquedaEditar).Throttle(TimeSpan.FromMilliseconds(500)).Where(x =>
    !string.IsNullOrWhiteSpace(x)).ObserveOn(RxApp.MainThreadScheduler).Subscribe(_ =>
    BtnBuscarUsuariosEditar.Execute().Subscribe());
00498           // Canciones en Playlist
00499           this.WhenAnyValue(x => x.TxtBusquedaCancionCrear).Throttle(TimeSpan.FromMilliseconds(500)).Where(x =>
    !string.IsNullOrWhiteSpace(x)).ObserveOn(RxApp.MainThreadScheduler).Subscribe(_ =>
    BtnBuscarCancionesCrear.Execute().Subscribe());
00500           this.WhenAnyValue(x => x.TxtBusquedaCancionEditar).Throttle(TimeSpan.FromMilliseconds(500)).Where(x =>
    !string.IsNullOrWhiteSpace(x)).ObserveOn(RxApp.MainThreadScheduler).Subscribe(_ =>
    BtnBuscarCancionesEditar.Execute().Subscribe());
00501
00502           // Carga Inicial
00503           ResetearBorradores();
00504           _ = CargarTodo();
00505       }
00506
00507
00508       // ============================================================
00509       // MÉTODOS DE APOYO Y CARGA DE DATOS EDITAR
00510       // ============================================================
00511       /// <summary>
00512       /// Reinicializa todos los objetos de borrador y limpia las listas temporales utilizadas en los formularios de creación.
00513       /// </summary>
00514       private void ResetearBorradores()
00515       {
00516           NuevoUsuario = new Usuarios { Perfil = new PerfilUsuario(), Estadisticas = new EstadisticasUsuario(), Listas =
    new ListasUsuario() };
00517           NuevaCancion = new Canciones ();
00518           NuevaPlaylist = new ListaPersonalizada();
00519           NuevoReporte = new Reportes { Referencias = new ReferenciasReporte() };
00520
```

```
00521           // Reset Listas Crear Canción
00522           ListaArtistasCrear.Clear();
00523           ListaGenerosSeleccionadosCrear.Clear();
00524           ListaResultadosCrear.Clear();
00525           TxtBusquedaCrear = "";
00526           HayResultadosCrear = false;
00527
00528           // Reset Listas Crear Playlist
00529           ListaCancionesPlaylistCrear.Clear();
00530           ListaResultadosCancionesCrear.Clear();
00531           TxtBusquedaCancionCrear = "";
00532           HayResultadosCancionCrear = false;
00533
00534           NuevoGeneroTxt = "";
00535           GeneroSeleccionadoCrear = "";
00536       }
00537
00538       // Cargar datos en listas temporales EDITAR CANCION + AUDIO
00539       /// <summary>
00540       /// Prepara el formulario de edición de canciones cargando los metadatos y detectando el tipo de origen del audio (Local vs YouTube).
00541       /// </summary>
00542       private void CargarDatosEditarCancion()
00543       {
00544           ListaArtistasEditar.Clear();
00545           ListaGenerosSeleccionadosEditar.Clear();
00546           ListaResultadosEditar.Clear();
00547           TxtBusquedaEditar = "";
00548           HayResultadosEditar = false;
00549
00550           if (SelectedCancion != null)
00551           {
00552               // Listas
00553               if (SelectedCancion.Datos != null && SelectedCancion.Datos.Generos != null)
00554                   foreach (var g in SelectedCancion.Datos.Generos) ListaGenerosSeleccionadosEditar.Add(g);
00555
00556               if (SelectedCancion.AutoresIds != null)
00557               {
00558                   foreach (var id in SelectedCancion.AutoresIds)
00559                   {
00560                       var user = ListaUsuarios.FirstOrDefault(u => u.Id == id);
00561                       if (user != null) ListaArtistasEditar.Add(user);
00562                   }
00563               }
00564
00565               // Audio (Detectar Youtube vs Local)
00566               if (!string.IsNullOrEmpty(SelectedCancion.UrlCancion))
00567               {
00568                   bool esLink = SelectedCancion.UrlCancion.StartsWith("http", StringComparison.OrdinalIgnoreCase) ||
00569                                 SelectedCancion.UrlCancion.StartsWith("www", StringComparison.OrdinalIgnoreCase);
00570
00571                   if (esLink)
00572                   {
00573                       EsArchivoLocalEditar = false; TxtUrlYoutubeEditar = SelectedCancion.UrlCancion; TxtRutaArchivoEditar = string.Empty;
00574                   }
00575                   else
00576                   {
00577                       EsArchivoLocalEditar = true; TxtRutaArchivoEditar = SelectedCancion.UrlCancion; TxtUrlYoutubeEditar = string.Empty;
00578                   }
00579               }
00580               else
00581               {
00582                   EsArchivoLocalEditar = true; TxtRutaArchivoEditar = ""; TxtUrlYoutubeEditar = "";
00583               }
00584           }
00585       }
00586
00587       // Cargar datos en listas temporales EDITAR PLAYLIST (IDs -> Objetos)
00588       /// <summary>
00589       /// Mapea los identificadores de canciones de una playlist seleccionada a objetos completos para su edición en la lista de pistas.
00590       /// </summary>
00591       private void CargarDatosEditarPlaylist()
00592       {
00593           ListaCancionesPlaylistEditar.Clear();
00594           ListaResultadosCancionesEditar.Clear();
00595           TxtBusquedaCancionEditar = "";
00596           HayResultadosCancionEditar = false;
00597
00598           if (SelectedPlaylist != null && SelectedPlaylist.IdsCanciones != null)
00599           {
00600               foreach (var id in SelectedPlaylist.IdsCanciones)
00601               {
00602                   var cancionReal = ListaCanciones.FirstOrDefault(c => c.Id == id);
00603                   if (cancionReal != null) ListaCancionesPlaylistEditar.Add(cancionReal);
```

```
00604                 }
00605             }
00606         }
00607 
00608 
00609         // ============================================================
00610         // MÉTODOS LÓGICOS DE AGREGAR/BUSCAR
00611         // ============================================================
00612         /// <summary>
00613         /// Filtra la lista global de usuarios basándose en un criterio de búsqueda y excluye a los que ya han sido seleccionados.
00614         /// </summary>
00615         /// <param name="texto">Cadena de búsqueda.</param>
00616         /// <param name="resultados">Colección donde se volcarán los resultados.</param>
00617         /// <param name="yaSeleccionados">Colección de usuarios a excluir.</param>
00618         /// <param name="setHayResultados">Acción para actualizar el estado de visibilidad de resultados.</param>
00619         private void BuscarUsuarios(string texto, ObservableCollection<Usuarios> resultados, ObservableCollection<Usuarios> yaSeleccionados, Action<bool> setHayResultados)
00620         {
00621             if (string.IsNullOrWhiteSpace(texto)) { resultados.Clear(); setHayResultados(false); return; }
00622             var res = ListaUsuarios.Where(u => u.Username != null && u.Username.Contains(texto, StringComparison.OrdinalIgnoreCase) && !yaSeleccionados.Contains(u)).ToList();
00623             resultados.Clear();
00624             foreach (var r in res) resultados.Add(r);
00625             setHayResultados(resultados.Count > 0);
00626         }
00627         /// <summary>
00628         /// Añade un usuario a la lista de destino y ejecuta una acción de limpieza en la interfaz.
00629         /// </summary>
00630         private void AgregarUsuario(Usuarios usuario, ObservableCollection<Usuarios> listaDestino, Action limpiarUI)
00631         {
00632             if (usuario != null && !listaDestino.Contains(usuario)) { listaDestino.Add(usuario); limpiarUI(); }
00633         }
00634         /// <summary>
00635         /// Remueve un usuario de la colección especificada.
00636         /// </summary>
00637         private void EliminarUsuario(Usuarios usuario, ObservableCollection<Usuarios> listaObjetivo)
00638         {
00639             if (usuario != null && listaObjetivo.Contains(usuario)) listaObjetivo.Remove(usuario);
00640         }
00641 
00642         /// <summary>
00643         /// Valida y añade un nuevo género a la lista de selección, evitando duplicados.
00644         /// </summary>
00645         private void AgregarGenero(string genero, ObservableCollection<string> listaDestino, Action limpiarCombo)
00646         {
00647             if (!string.IsNullOrWhiteSpace(genero))
00648             {
00649                 if (!listaDestino.Any(g => g.Equals(genero, StringComparison.OrdinalIgnoreCase))) { listaDestino.Add(genero); limpiarCombo(); }
00650                 else _dialogoService.MostrarAlerta("Ese género ya está añadido.");
00651             }
00652         }
00653         /// <summary>
00654         /// Remueve un género de la colección especificada.
00655         /// </summary>
00656         private void EliminarGenero(string genero, ObservableCollection<string> listaObjetivo)
00657         {
00658             if (listaObjetivo.Contains(genero)) listaObjetivo.Remove(genero);
00659         }
00660 
00661         // Lógica Playlist (Buscar Canciones)
00662         /// <summary>
00663         /// Remueve un género de la colección especificada.
00664         /// </summary>
00665         private void BuscarCanciones(string texto, ObservableCollection<Canciones> resultados, ObservableCollection<Canciones> yaSeleccionadas, Action<bool> setHayResultados)
00666         {
00667             if (string.IsNullOrWhiteSpace(texto)) { resultados.Clear(); setHayResultados(false); return; }
00668             var res = ListaCanciones.Where(c => c.Titulo != null && c.Titulo.Contains(texto, StringComparison.OrdinalIgnoreCase) && !yaSeleccionadas.Contains(c)).ToList();
00669             resultados.Clear();
00670             foreach (var r in res) resultados.Add(r);
00671             setHayResultados(resultados.Count > 0);
00672         }
00673         /// <summary>
00674         /// Añade una canción a la colección de destino de la playlist y limpia el estado de búsqueda.
00675         /// </summary>
00676         private void AgregarCancionAPlaylist(Canciones c, ObservableCollection<Canciones> destino, Action limpiar)
00677         {
00678             if (c != null && !destino.Contains(c)) {
00679                 destino.Add(c); limpiar();
00680             }
00681         }
00682 
00683         // ============================================================
00684         // OPERACIONES BASE DE DATOS (CARGAR, CREAR, EDITAR, ELIMINAR)
```

```
00685       // ============================================================
00686
00687       // --- CARGAR TODO ---
00688       /// <summary>
00689       /// Realiza una carga masiva inicial de usuarios, canciones, playlists, reportes y géneros desde MongoDB.
00690       /// </summary>
00691       private async Task CargarTodo()
00692       {
00693           var cliente = MongoClientSingleton.Instance.Cliente;
00694
00695           // Cargar datos
00696           var users = await cliente.ObtenerTodosLosUsuarios();
00697           ListaUsuarios.Clear();
00698           foreach (var u in users) ListaUsuarios.Add(u);
00699
00700           var songs = await cliente.ObtenerCanciones();
00701           ListaCanciones.Clear();
00702           foreach (var s in songs) ListaCanciones.Add(s);
00703
00704           var plays = await cliente.ObtenerListasReproduccion();
00705           ListaPlaylists.Clear();
00706           foreach (var p in plays) ListaPlaylists.Add(p);
00707
00708           var reps = await cliente.ObtenerReportes();
00709           ListaReportes.Clear();
00710           foreach (var r in reps) ListaReportes.Add(r);
00711
00712           var gens = await cliente.ObtenerGenerosCompletos();
00713           ListaGeneros.Clear();
00714           ListaGenerosCombox.Clear();
00715           foreach (var g in gens) { ListaGeneros.Add(g); ListaGenerosCombox.Add(g.Nombre); }
00716       }
00717
00718       // --- CREAR ---
00719       /// <summary>
00720       /// Orquesta el proceso de creación de un nuevo usuario, incluyendo validación de duplicados, carga de imagen a la nube y encriptación de credenciales.
00721       /// </summary>
00722       private async Task CrearUsuarioTask()
00723       {
00724           var cliente = MongoClientSingleton.Instance.Cliente;
00725           if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
00726
00727           // Buscamos si OTRO usuario ya tiene ese email en el datagrid
00728           bool emailDuplicado = ListaUsuarios.Any(u => u.Email.Equals(NuevoUsuario.Email, StringComparison.OrdinalIgnoreCase));
00729           if (emailDuplicado)
00730           {
00731               _dialogoService.MostrarAlerta("Msg_Error_EmailDuplicado");
00732               return;
00733           }
00734
00735           // Validación de campos vacios
00736           if (string.IsNullOrWhiteSpace(NuevoUsuario.Username) ||
00737               string.IsNullOrWhiteSpace(NuevoUsuario.Email) ||
00738               string.IsNullOrWhiteSpace(NuevoUsuario.Password) ||
00739               string.IsNullOrWhiteSpace(NuevoUsuario.Rol) ||
00740               string.IsNullOrWhiteSpace(NuevoUsuario.Perfil.Pais) ||
00741               string.IsNullOrWhiteSpace(NuevoUsuario.Perfil.ImagenUrl) ||
00742               NuevoUsuario.Perfil.FechaNacimiento == default)
00743           {
00744               _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosUser");
00745               return;
00746           }
00747
00748           // Subir imagen (Si es un archivo local)
00749           if (File.Exists(NuevoUsuario.Perfil.ImagenUrl))
00750           {
00751               try
00752               {
00753                   string urlNube = await _storageService.SubirImagen(NuevoUsuario.Perfil.ImagenUrl);
00754                   NuevoUsuario.Perfil.ImagenUrl = urlNube;
00755               }
00756               catch (Exception ex)
00757               {
00758                   _dialogoService.MostrarAlerta("Msg_Error_SubirImagen" + ex.Message);
00759                   return;
00760               }
00761           }
00762
00763           // Encripatar contraseña
00764           NuevoUsuario.Password = Encriptador.HashPassword(NuevoUsuario.Password);
00765
00766           // Preparamos de datos
00767           NuevoUsuario.Id = ObjectId.GenerateNewId().ToString();
00768           NuevoUsuario.FechaRegistro = DateTime.Now;
00769           if (NuevoUsuario.Listas == null) NuevoUsuario.Listas = new ListasUsuario();
```

```
00770           if (NuevoUsuario.Estadisticas == null) NuevoUsuario.Estadisticas = new EstadisticasUsuario();
00771
00772           // Guardamos en la base de datos
00773           bool exito = await cliente.CrearUsuario(NuevoUsuario);
00774
00775           if (exito)
00776           {
00777               ListaUsuarios.Add(NuevoUsuario);
00778               ResetearBorradores();
00779               _dialogoService.MostrarAlerta("Msg_Exito_UsuarioCreado");
00780           }
00781           else
00782           {
00783               _dialogoService.MostrarAlerta("Msg_Error_CrearUsuario");
00784           }
00785       }
00786       /// <summary>
00787       /// Gestiona la publicación de una nueva canción, procesando la subida de archivos multimedia y el cálculo de
      duraciones mediante APIs externas o análisis local.
00788       /// </summary>
00789       private async Task CrearCancionTask()
00790       {
00791           var cliente = MongoClientSingleton.Instance.Cliente;
00792
00793           if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
00794           // Validamos
00795           if (string.IsNullOrWhiteSpace(NuevaCancion.Titulo) ||
00796               string.IsNullOrWhiteSpace(NuevaCancion.UrlCancion) ||
00797               string.IsNullOrWhiteSpace(NuevaCancion.ImagenPortadaUrl))
00798           {
00799               _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosCancion");
00800               return;
00801           }
00802           if (ListaArtistasCrear.Count == 0 || ListaGenerosSeleccionadosCrear.Count == 0)
00803           {
00804               _dialogoService.MostrarAlerta("Msg_Error_FaltanArtistasGeneros");
00805               return;
00806           }
00807
00808           try
00809           {
00810               // Subimos portada
00811               if (File.Exists(NuevaCancion.ImagenPortadaUrl))
00812               {
00813                   NuevaCancion.ImagenPortadaUrl = await _storageService.SubirImagen(NuevaCancion.ImagenPortadaUrl);
00814               }
00815
00816               // Subimos audio y obtenemos la duración
00817               if (EsYoutube)
00818               {
00819                   // Si es YouTube, usamos la API para sacar la duración
00820                   var info = await _audioService.ObtenerMp3(NuevaCancion.UrlCancion);
00821                   if (info != null && info.DuracionSegundos > 0)
00822                   {
00823                       NuevaCancion.Datos.DuracionSegundos = info.DuracionSegundos;
00824                   }
00825               }
00826               else
00827               {
00828                   // Si es archivo local
00829                   if (File.Exists(NuevaCancion.UrlCancion))
00830                   {
00831                       // Calculamos duración antes de subir
00832                       NuevaCancion.Datos.DuracionSegundos = ObtenerDuracionLocal(NuevaCancion.UrlCancion);
00833                       // Subimos el archivo y guardamos la URL de la nube
00834                       NuevaCancion.UrlCancion = await _storageService.SubirCancion(NuevaCancion.UrlCancion);
00835                   }
00836               }
00837           }
00838           catch (Exception ex)
00839           {
00840               _dialogoService.MostrarAlerta("Msg_Error_GestionArchivos" + ex.Message);
00841               return;
00842           }
00843
00844           // Preparamos datos
00845           if (NuevaCancion.Datos == null) NuevaCancion.Datos = new DatosCancion();
00846           NuevaCancion.Datos.Generos = ListaGenerosSeleccionadosCrear.ToList();
00847           NuevaCancion.AutoresIds = ListaArtistasCrear.Select(u => u.Id).ToList();
00848
00849           NuevaCancion.Datos.FechaLanzamiento = DateTime.Now;
00850           // Generar nombre artista para la vista rápida
00851           NuevaCancion.NombreArtista = string.Join(", ", ListaArtistasCrear.Select(u => u.Username));
00852
00853           if (EsYoutube)
00854           {
00855               // Avisamos al usuario en el mensaje de carga (ya que esto tarda un poco)
```

```
00856             MensajeCarga = "Msg_Carga_AnalizandoDuracion";
00857             var info = await _audioService.ObtenerMp3(NuevaCancion.UrlCancion);
00858             if (info != null && info.DuracionSegundos > 0)
00859             {
00860                 NuevaCancion.Datos.DuracionSegundos = info.DuracionSegundos;
00861             }
00862         }
00863         // Guardamos en la base de datos
00864         bool exito = await cliente.PublicarCancion(NuevaCancion);
00865 
00866         if (exito)
00867         {
00868             if (NuevaCancion.AutoresIds != null)
00869             {
00870                 foreach (var idAutor in NuevaCancion.AutoresIds)
00871                 {
00872                     await cliente.IncrementarContadorCancionesUsuario(idAutor, 1);
00873                 }
00874             }
00875             ListaCanciones.Add(NuevaCancion);
00876             ResetearBorradores();
00877             _dialogoService.MostrarAlerta("Msg_Exito_CancionPublicada");
00878         }
00879         else
00880         {
00881             _dialogoService.MostrarAlerta("Msg_Error_SubirCancion");
00882         }
00883     }
00884     /// <summary>
00885     /// Inserta un nuevo género musical en la base de datos tras validar su inexistencia previa.
00886     /// </summary>
00887     private async Task AgregarGeneroBD()
00888     {
00889         var cliente = MongoClientSingleton.Instance.Cliente;
00890         if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
00891 
00892         // Validamos
00893         if (string.IsNullOrWhiteSpace(NuevoGeneroTxt))
00894         {
00895             _dialogoService.MostrarAlerta("Msg_Error_NombreGeneroVacio");
00896             return;
00897         }
00898 
00899         // Validamos que no exista ya ese género
00900         bool generoExiste = ListaGeneros.Any(g => g.Nombre.Equals(NuevoGeneroTxt,
    StringComparison.OrdinalIgnoreCase));
00901         if (generoExiste)
00902         {
00903             _dialogoService.MostrarAlerta("Msg_Error_GeneroExiste");
00904             return;
00905         }
00906 
00907         // Guardamos
00908         bool exito = await cliente.CrearGenero(NuevoGeneroTxt);
00909 
00910         if (exito)
00911         {
00912             NuevoGeneroTxt = "";
00913             await CargarTodo();
00914             _dialogoService.MostrarAlerta("Msg_Exito_GeneroAnadido");
00915         }
00916         else
00917         {
00918             _dialogoService.MostrarAlerta("Msg_Error_CrearGenero");
00919         }
00920     }
00921     /// <summary>
00922     /// Persiste una nueva playlist en MongoDB, gestionando la subida de la imagen de portada y la vinculación de IDs
    de canciones.
00923     /// </summary>
00924     private async Task CrearPlaylistTask()
00925     {
00926         var cliente = MongoClientSingleton.Instance.Cliente;
00927 
00928         if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
00929 
00930         // Validamos
00931         if (string.IsNullOrWhiteSpace(NuevaPlaylist.Nombre) ||
00932             string.IsNullOrWhiteSpace(NuevaPlaylist.IdUsuario) |
00933             string.IsNullOrWhiteSpace(NuevaPlaylist.Descripcion) ||
00934             string.IsNullOrWhiteSpace(NuevaPlaylist.UrlPortada))
00935         {
00936             _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosPlaylist");
00937             return;
00938         }
00939         // Subimos portada
00940         if (File.Exists(NuevaPlaylist.UrlPortada))
```

```
00941           {
00942               try
00943               {
00944                   NuevaPlaylist.UrlPortada = await _storageService.SubirImagen(NuevaPlaylist.UrlPortada);
00945               }
00946               catch (Exception ex)
00947               {
00948                   _dialogoService.MostrarAlerta("Msg_Error_SubirImagen" + ex.Message);
00949                   return;
00950               }
00951           }
00952
00953           // Preparamos datos
00954           NuevaPlaylist.IdsCanciones = ListaCancionesPlaylistCrear.Select(c => c.Id).ToList();
00955
00956           // Guardamos en la base de datos
00957           bool exito = await cliente.CrearListaReproduccion(NuevaPlaylist);
00958
00959           if (exito)
00960           {
00961               ListaPlaylists.Add(NuevaPlaylist);
00962               ResetearBorradores();
00963               _dialogoService.MostrarAlerta("Msg_Exito_PlaylistCreada");
00964           }
00965           else
00966           {
00967               _dialogoService.MostrarAlerta("Msg_Error_CrearPlaylist");
00968           }
00969       }
00970       /// <summary>
00971       /// Registra un nuevo reporte de error o infracción en el sistema.
00972       /// </summary>
00973       private async Task CrearReporteTask()
00974       {
00975           var cliente = MongoClientSingleton.Instance.Cliente;
00976
00977           if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
00978
00979           // Validamos
00980           if (string.IsNullOrWhiteSpace(NuevoReporte.TipoProblema) ||
00981               string.IsNullOrWhiteSpace(NuevoReporte.Descripcion) ||
00982               string.IsNullOrWhiteSpace(NuevoReporte.Estado))
00983           {
00984               _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosReporte");
00985               return;
00986           }
00987           if (string.IsNullOrWhiteSpace(NuevoReporte.Referencias.CancionReportadaId) ||
00988               string.IsNullOrWhiteSpace(NuevoReporte.Referencias.UsuarioReportanteId))
00989           {
00990               _dialogoService.MostrarAlerta("Msg_Error_FaltanRefReporte");
00991               return;
00992           }
00993
00994           // Preparamos datos
00995           NuevoReporte.FechaCreacion = DateTime.Now;
00996
00997           // Guardamos
00998           bool exito = await cliente.EnviarReporte(NuevoReporte);
00999
01000           if (exito)
01001           {
01002               ResetearBorradores();
01003               _ = CargarTodo();
01004               _dialogoService.MostrarAlerta("Msg_Exito_ReporteRegistrado");
01005           }
01006           else
01007           {
01008               _dialogoService.MostrarAlerta("Msg_Exito_ReporteRegistrado");
01009           }
01010       }
01011
01012       // --- GUARDAR EDICIÓN (UPDATE) ---
01013       /// <summary>
01014       /// Método central de edición que detecta la pestaña activa del panel (Usuario, Canción, Género, Playlist, Reporte) y
      sincroniza los cambios con MongoDB.
01015       /// </summary>
01016       private async Task GuardarSeleccionado()
01017       {
01018           var cliente = MongoClientSingleton.Instance.Cliente;
01019           bool exito = false;
01020           switch (IndiceTab)
01021           {
01022               // -----------
01023               // 0. USUARIOS
01024               // -----------
01025               case 0:
01026                   if (SelectedUsuario != null)
```

```
01027                 {
01028                     // 1. Validaciones
01029                     if (string.IsNullOrWhiteSpace(SelectedUsuario.Username) ||
01030                         string.IsNullOrWhiteSpace(SelectedUsuario.Email) ||
01031                         string.IsNullOrWhiteSpace(SelectedUsuario.Perfil.Pais))
01032                     {
01033                         _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosObligatorios");
01034                         return;
01035                     }
01036 
01037                     Usuarios usuarioOriginal = null;
01038                     try
01039                     {
01040                         var lista = await cliente.ObtenerUsuariosPorListaIds(new List<string> { SelectedUsuario.Id });
01041                         usuarioOriginal = lista.FirstOrDefault();
01042 
01043                         if (usuarioOriginal != null && usuarioOriginal.Perfil == null)
01044                             usuarioOriginal.Perfil = new PerfilUsuario { ImagenUrl = "" };
01045                     }
01046                     catch (Exception x )
01047                     {
01048                         System.Diagnostics.Debug.WriteLine("[ERROR] Fallo al recuperar original: " + x.Message);
01049                     }
01050 
01051                     if (SelectedUsuario.Password != usuarioOriginal.Password)
01052                     {
01053                         SelectedUsuario.Password = Encriptador.HashPassword(SelectedUsuario.Password);
01054                     }
01055 
01056                     // 3. GESTIÓN DE FOTO DE PERFIL
01057                     try
01058                     {
01059                         // Comparamos la URL actual del TextBox con la de la BD
01060                         if (SelectedUsuario.Perfil.ImagenUrl != usuarioOriginal.Perfil.ImagenUrl)
01061                         {
01062                             // Si son diferentes y es un archivo local -> SUBIR
01063                             if (File.Exists(SelectedUsuario.Perfil.ImagenUrl))
01064                             {
01065                                 MensajeCarga = "Msg_Carga_SubiendoAvatar";
01066                                 SelectedUsuario.Perfil.ImagenUrl = await
    _storageService.SubirImagen(SelectedUsuario.Perfil.ImagenUrl);
01067                             }
01068                         }
01069                         else
01070                         {
01071                             // Si es igual, restauramos la original para asegurar integridad
01072                             SelectedUsuario.Perfil.ImagenUrl = usuarioOriginal.Perfil.ImagenUrl;
01073                         }
01074                     }
01075                     catch (Exception ex)
01076                     {
01077                         _dialogoService.MostrarAlerta("[ERROR] Error subiendo imagen: " + ex.Message);
01078                         return;
01079                     }
01080 
01081                     if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01082 
01083                     // 4. GUARDAR CAMBIOS
01084                     exito = await cliente.ActualizarUsuario(
01085                         SelectedUsuario.Id,
01086                         SelectedUsuario.Username,
01087                         SelectedUsuario.Email,
01088                         SelectedUsuario.Password,
01089                         SelectedUsuario.Rol,
01090                         SelectedUsuario.Perfil.Pais,
01091                         SelectedUsuario.Perfil.ImagenUrl,
01092                         SelectedUsuario.Perfil.FechaNacimiento,
01093                         SelectedUsuario.Perfil.EsPrivada);
01094 
01095                     if (exito)
01096                     {
01097                         // 6. ACTUALIZAR SESIÓN ACTUAL (Si me he editado a mí mismo)
01098                         if (SelectedUsuario.Id == GlobalData.Instance.UserIdGD)
01099                         {
01100                             GlobalData.Instance.SetUserData(SelectedUsuario);
01101                             System.Diagnostics.Debug.WriteLine("[INFO] Datos de sesión actualizados tras edición.");
01102                         }
01103                         await CargarTodo();
01104                         _dialogoService.MostrarAlerta("Msg_Exito_UsuarioActualizado.");
01105                     }
01106                     else
01107                     {
01108                         _dialogoService.MostrarAlerta("Msg_Error_ActualizarUsuario");
01109                     }
01110                 }
01111                 break;
01112 
```

```
01113             // -------------------------
01114             // 1. CANCIONES
01115             // -------------------------
01116             case 1:
01117                 if (SelectedCancion != null)
01118                 {
01119                     // 1. Validaciones
01120                     if (string.IsNullOrWhiteSpace(SelectedCancion.Titulo) ||
01121                         string.IsNullOrWhiteSpace(SelectedCancion.UrlCancion) ||
01122                         string.IsNullOrWhiteSpace(SelectedCancion.ImagenPortadaUrl))
01123                     {
01124                         _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosObligatorios");
01125                         return;
01126                     }
01127 
01128                     // ============================================================
01129                     // 2. RECUPERAR VERSIÓN ORIGINAL (PARA COMPARAR)
01130                     // ============================================================
01131                     Canciones cancionOriginal = null;
01132                     List<string> idsViejos = new List<string>();
01133 
01134                     try
01135                     {
01136                         // Traemos la canción tal cual está en la base de datos ahora mismo
01137                       var listaTemp = await cliente.ObtenerCancionesPorListaIds(new List<string> { SelectedCancion.Id });
01138                         cancionOriginal = listaTemp.FirstOrDefault();
01139 
01140                         if (cancionOriginal != null)
01141                         {
01142                             // Guardamos ids viejos para el contador luego
01143                             idsViejos = cancionOriginal.AutoresIds != null ? cancionOriginal.AutoresIds.ToList() : new
    List<string>();
01144 
01145                             // Si en la BD los campos son null, los convertimos a listas vacías
01146                             if (cancionOriginal.AutoresIds == null) cancionOriginal.AutoresIds = new List<string>();
01147                             if (cancionOriginal.Datos == null) cancionOriginal.Datos = new DatosCancion();
01148                             if (cancionOriginal.Datos.Generos == null) cancionOriginal.Datos.Generos = new List<string>();
01149                         }
01150                     }
01151                     catch (Exception ex)
01152                     {
01153                         System.Diagnostics.Debug.WriteLine("[ERROR] Fallo al recuperar original: " + ex.Message);
01154                     }
01155 
01156                     // Si falló la recuperación, creamos uno vacío seguro para no bloquear
01157                     if (cancionOriginal == null)
01158                     {
01159                         cancionOriginal = new Canciones
01160                         {
01161                             Id = SelectedCancion.Id,
01162                             AutoresIds = new List<string>(), // Lista vacía, no null
01163                             Datos = new DatosCancion { Generos = new List<string>() } // Lista vacía, no null
01164                         };
01165                     }
01166 
01167                     // 3. ACTUALIZAR OBJETO LOCAL
01168                     if (SelectedCancion.Datos == null) SelectedCancion.Datos = new DatosCancion();
01169                     SelectedCancion.Datos.Generos = ListaGenerosSeleccionadosEditar.ToList();
01170                     SelectedCancion.AutoresIds = ListaArtistasEditar.Select(u => u.Id).ToList();
01171 
01172                     // 4. GESTIÓN DE ARCHIVOS
01173                     try
01174                     {
01175                         // Solo procesamos si la URL es DIFERENTE a la que había antes
01176                         if (SelectedCancion.ImagenPortadaUrl != cancionOriginal.ImagenPortadaUrl)
01177                         {
01178                             // Si ha cambiado y ADEMÁS es un archivo físico en el PC -> Subimos
01179                             if (File.Exists(SelectedCancion.ImagenPortadaUrl))
01180                             {
01181                                 MensajeCarga = "Msg_Carga_SubiendoPortada";
01182                                 SelectedCancion.ImagenPortadaUrl = await
    _storageService.SubirImagen(SelectedCancion.ImagenPortadaUrl);
01183                             }
01184                         }
01185                         else
01186                         {
01187                             // Si no ha cambiado, aseguramos que se queda la original (por seguridad)
01188                             System.Diagnostics.Debug.WriteLine("[INFO] La portada no ha cambiado.");
01189                             SelectedCancion.ImagenPortadaUrl = cancionOriginal.ImagenPortadaUrl;
01190                         }
01191                         // GESTIÓN DE AUDIO
01192                         // 1. Determinamos cuál es la URL nueva propuesta según el TextBox activo
01193                         string nuevaUrlAudio = EsArchivoLocalEditar ? TxtRutaArchivoEditar : TxtUrlYoutubeEditar;
01194 
01195                         // 2. Solo procesamos si la URL HA CAMBIADO respecto a la base de datos
01196                         if (nuevaUrlAudio != cancionOriginal.UrlCancion)
01197                         {
```

```
01198                         System.Diagnostics.Debug.WriteLine("[DEBUG] El audio ha cambiado. Procesando...");
01199
01200                         if (EsArchivoLocalEditar)
01201                         {
01202                             // CASO LOCAL: Solo subimos si es un archivo físico en el disco
01203                             if (File.Exists(TxtRutaArchivoEditar))
01204                             {
01205                                 System.Diagnostics.Debug.WriteLine($"[DEBUG] Subiendo nuevo archivo local:
    {TxtRutaArchivoEditar}");
01206
01207                                 // Calculamos duración con TagLib
01208                                 SelectedCancion.Datos.DuracionSegundos = ObtenerDuracionLocal(TxtRutaArchivoEditar);
01209
01210                                 // Subimos a Cloudinary y obtenemos la nueva URL
01211                                 SelectedCancion.UrlCancion = await _storageService.SubirCancion(TxtRutaArchivoEditar);
01212                             }
01213                             else
01214                             {
01215                                 // Si no existe el archivo (ej: url rota), mantenemos el texto tal cual por si acaso
01216                                 SelectedCancion.UrlCancion = TxtRutaArchivoEditar;
01217                                 _dialogoService.MostrarAlerta("El archivo local especificado no existe. Comprueba");
01218                             }
01219                         }
01220                         else
01221                         {
01222                             // CASO YOUTUBE: Asignamos la nueva URL y consultamos la API
01223                             SelectedCancion.UrlCancion = TxtUrlYoutubeEditar;
01224
01225                             if (!string.IsNullOrEmpty(SelectedCancion.UrlCancion) &&
01226                                 (SelectedCancion.UrlCancion.Contains("youtube") ||
    SelectedCancion.UrlCancion.Contains("youtu.be")))
01227                             {
01228                                 System.Diagnostics.Debug.WriteLine($"[DEBUG] Analizando nueva URL de YouTube...");
01229                                 var info = await _audioService.ObtenerMp3(SelectedCancion.UrlCancion);
01230                                 if (info != null && info.DuracionSegundos > 0)
01231                                 {
01232                                     SelectedCancion.Datos.DuracionSegundos = info.DuracionSegundos;
01233                                 }
01234                             }
01235                         }
01236                     }
01237                     else
01238                     {
01239                         // NO HA CAMBIADO: Asignamos la URL vieja y nos ahorramos el trabajo
01240                         System.Diagnostics.Debug.WriteLine("[DEBUG] El audio NO ha cambiado. Se mantiene.");
01241                         SelectedCancion.UrlCancion = cancionOriginal.UrlCancion;
01242                     }
01243                 }
01244                 catch (Exception ex)
01245                 {
01246                     _dialogoService.MostrarAlerta("Msg_Error_GestionArchivos" + ex.Message);
01247                     return;
01248                 }
01249
01250                 if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01251
01252                 // 5. LLAMADA A ACTUALIZAR
01253                 // Ahora 'cancionOriginal' es segura (no tiene nulls), así que MongoAtlas no fallará.
01254                 exito = await cliente.ActualizarCancion(
01255                     SelectedCancion.Titulo,
01256                     SelectedCancion.ImagenPortadaUrl,
01257                     SelectedCancion.AutoresIds,
01258                     SelectedCancion.Datos.Generos,
01259                     cancionOriginal
01260                 );
01261
01262                 // 6. ACTUALIZAR CONTADORES DE USUARIOS
01263                 if (exito)
01264                 {
01265                     try
01266                     {
01267                         var idsNuevos = SelectedCancion.AutoresIds;
01268
01269                         // A. Sumar a los nuevos
01270                         var nuevosAutores = idsNuevos.Except(idsViejos).ToList();
01271                         foreach (var id in nuevosAutores)
01272                             await cliente.IncrementarContadorCancionesUsuario(id, 1);
01273
01274                         // B. Restar a los viejos
01275                         var autoresEliminados = idsViejos.Except(idsNuevos).ToList();
01276                         foreach (var id in autoresEliminados)
01277                             await cliente.IncrementarContadorCancionesUsuario(id, -1);
01278                     }
01279                     catch (Exception ex)
01280                     {
01281                         System.Diagnostics.Debug.WriteLine("Warning contadores: " + ex.Message);
01282                     }
```

```
01283
01284                       await CargarTodo();
01285                       _dialogoService.MostrarAlerta("Msg_Exito_CancionActualizada");
01286                   }
01287                   else
01288                   {
01289                       // Si devuelve false, es que no detectó cambios reales (o hubo error interno controlado)
01290                       _dialogoService.MostrarAlerta("Msg_Error_NoCambios");
01291                   }
01292               }
01293               break;
01294
01295           // ----------
01296           // 2. GÉNEROS
01297           // ----------
01298           case 2:
01299               if (SelectedGenero != null)
01300               {
01301                   if (string.IsNullOrWhiteSpace(SelectedGenero.Nombre))
01302                   {
01303                       _dialogoService.MostrarAlerta("Msg_Error_NombreGeneroVacio");
01304                       return;
01305                   }
01306
01307                   // Comprobamos si hay OTRO género con ese nombre (excluyendo al propio) en el propio datagrid
01308                   bool existeGenero = ListaGeneros.Any(g => g.Nombre.Equals(SelectedGenero.Nombre,
    StringComparison.OrdinalIgnoreCase)
01309                                                        && g.Id != SelectedGenero.Id);
01310                   if (existeGenero)
01311                   {
01312                       _dialogoService.MostrarAlerta("Msg_Error_GeneroExiste");
01313                       return;
01314                   }
01315
01316                   if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01317
01318                   exito = await cliente.ActualizarGenero(SelectedGenero.Id, SelectedGenero.Nombre);
01319
01320                   if (exito)
01321                   {
01322                       await CargarTodo();
01323                       _dialogoService.MostrarAlerta("Msg_Exito_GeneroActualizado");
01324                   }
01325                   else
01326                   {
01327                       _dialogoService.MostrarAlerta("Msg_Error_ActualizarGenero");
01328                   }
01329               }
01330               break;
01331
01332           // ------------
01333           // 3. PLAYLISTS
01334           // ------------
01335           case 3:
01336               if (SelectedPlaylist != null)
01337               {
01338                   // 1. Validaciones
01339                   if (string.IsNullOrWhiteSpace(SelectedPlaylist.Nombre) ||
01340                       string.IsNullOrWhiteSpace(SelectedPlaylist.Descripcion) ||
01341                       string.IsNullOrWhiteSpace(SelectedPlaylist.UrlPortada))
01342                   {
01343                       _dialogoService.MostrarAlerta("Msg_Error_FaltanDatosPlaylist");
01344                       return;
01345                   }
01346                   if (ListaCancionesPlaylistEditar.Count == 0)
01347                   {
01348                       _dialogoService.MostrarAlerta("Msg_Error_PlaylistVacia");
01349                       return;
01350                   }
01351
01352                   ListaPersonalizada playlistOriginal = null;
01353                   try
01354                   {
01355                       // Accedemos directo a la colección para buscar por ID
01356                       // (Asegúrate de tener 'using MongoDB.Driver;' arriba si te marca error en 'Builders')
01357                       var col = cliente.Database.GetCollection<ListaPersonalizada>("listapersonalizada");
01358                       var filtro = Builders<ListaPersonalizada>.Filter.Eq(x => x.Id, SelectedPlaylist.Id);
01359
01360                       playlistOriginal = await col.Find(filtro).FirstOrDefaultAsync();
01361                   }
01362                   catch {
01363                   }
01364
01365                   // Fallback de seguridad si no se encontró
01366                   if (playlistOriginal == null)
01367                   {
01368                       playlistOriginal = new ListaPersonalizada
```

```
01369                       {
01370                           Id = SelectedPlaylist.Id,
01371                           UrlPortada = "", // Para que la comparación no falle
01372                           IdsCanciones = new List<string>()
01373                       };
01374                   }
01375
01376                   // 3. ACTUALIZAR OBJETO LOCAL
01377                   SelectedPlaylist.IdsCanciones = ListaCancionesPlaylistEditar.Select(c => c.Id).ToList();
01378
01379                   // 4. GESTIÓN DE PORTADA (SI CAMBIÓ)
01380                   try
01381                   {
01382                       // Comparamos la URL del TextBox con la de la BD
01383                       if (SelectedPlaylist.UrlPortada != playlistOriginal.UrlPortada)
01384                       {
01385                           // Si es diferente y es un archivo local -> SUBIR
01386                           if (File.Exists(SelectedPlaylist.UrlPortada))
01387                           {
01388                               MensajeCarga = "Msg_Carga_SubiendoPortadaPly";
01389                               SelectedPlaylist.UrlPortada = await _storageService.SubirImagen(SelectedPlaylist.UrlPortada);
01390                           }
01391                       }
01392                       else
01393                       {
01394                           // Si es igual, restauramos la original para asegurar
01395                           SelectedPlaylist.UrlPortada = playlistOriginal.UrlPortada;
01396                       }
01397                   }
01398                   catch (Exception ex)
01399                   {
01400                       _dialogoService.MostrarAlerta("Msg_Error_SubirImagen" + ex.Message);
01401                       return;
01402                   }
01403
01404                   if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01405
01406                   // 5. GUARDAR EN BD
01407                   exito = await cliente.ActualizarPlaylist(
01408                       SelectedPlaylist.Nombre,
01409                       SelectedPlaylist.Descripcion,
01410                       SelectedPlaylist.IdsCanciones,
01411                       SelectedPlaylist.UrlPortada,
01412                       playlistOriginal
01413                   );
01414
01415                   if (exito)
01416                   {
01417                       await CargarTodo();
01418                       _dialogoService.MostrarAlerta("Msg_Exito_PlaylistActualizada");
01419                   }
01420                   else
01421                   {
01422                       _dialogoService.MostrarAlerta("Msg_Error_NoCambios");
01423                   }
01424               }
01425               break;
01426
01427           // -----------
01428           // 4. REPORTES
01429           // -----------
01430           case 4:
01431               if (SelectedReporte != null)
01432               {
01433                   if (string.IsNullOrWhiteSpace(SelectedReporte.Estado))
01434                   {
01435                       _dialogoService.MostrarAlerta("Msg_Error_FaltaEstadoReporte");
01436                       return;
01437                   }
01438
01439                   if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01440
01441                   exito = await cliente.ActualizarEstadoReporte(SelectedReporte.Estado, SelectedReporte.Resolucion,
      SelectedReporte);
01442
01443                   if (exito)
01444                   {
01445                       await CargarTodo();
01446                       _dialogoService.MostrarAlerta("Msg_Exito_ReporteActualizado");
01447                   }
01448                   else
01449                   {
01450                       _dialogoService.MostrarAlerta("Msg_Error_ActualizarReporte");
01451                   }
01452               }
01453               break;
01454       }
```

```
01455         }
01456
01457         // --- ELIMINAR (DELETE) ---
01458         /// <summary>
01459         /// Elimina permanentemente un usuario de la base de datos tras confirmar la acción y validar que no sea el usuario en sesión.
01460         /// </summary>
01461         private async Task EliminarUsuarioTask()
01462         {
01463             var cliente = MongoClientSingleton.Instance.Cliente;
01464             if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01465             // TRADUCCIÓN DEL DIÁLOGO DE CONFIRMACIÓN
01466             if (!await _dialogoService.Preguntar(
01467                 "Msg_Confirmar_Titulo",
01468                 "Msg_Confirmar_BorrarUsuario",
01469                 "Msg_Confirmar_BtnSi",
01470                 "Msg_Confirmar_BtnNo")) return;
01471
01472             // --- BLOQUEO DE SEGURIDAD ---
01473             if (SelectedUsuario.Id == GlobalData.Instance.UserIdGD)
01474             {
01475                 _dialogoService.MostrarAlerta("Msg_Error_BorrarPropioUser");
01476                 return;
01477             }
01478
01479             await EjecutarConCarga(async () =>
01480             {
01481                 bool exito = await cliente.EliminarUsuario(SelectedUsuario.Id);
01482                 if (exito)
01483                 {
01484                     await CargarTodo();
01485                     _dialogoService.MostrarAlerta("Msg_Exito_UsuarioEliminado");
01486                 }
01487                 else _dialogoService.MostrarAlerta("Msg_Error_EliminarUsuario");
01488             }, "Msg_Carga_EliminandoUser");
01489         }
01490         /// <summary>
01491         /// Ejecuta el proceso de borrado de una canción, eliminando el archivo físico de Cloudinary si aplica y actualizando los contadores de sus autores.
01492         /// </summary>
01493         private async Task EliminarCancionTask()
01494         {
01495             var cliente = MongoClientSingleton.Instance.Cliente;
01496             if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01497
01498             if (!await _dialogoService.Preguntar(
01499                 "Msg_Confirmar_Titulo",
01500                 "Msg_Confirmar_BorrarCancion",
01501                 "Msg_Confirmar_BtnSi",
01502                 "Msg_Confirmar_BtnNo")) return;
01503
01504             // 2. EJECUTAR (CON CARGA)
01505             await EjecutarConCarga(async () =>
01506             {
01507                 // Borrar de Cloudinary si hace falta
01508                 bool esYoutube = !string.IsNullOrEmpty(SelectedCancion.UrlCancion) &&
01509                                  (SelectedCancion.UrlCancion.Contains("youtube.com") || SelectedCancion.UrlCancion.Contains("youtu.be"));
01510
01511                 if (!esYoutube && !string.IsNullOrEmpty(SelectedCancion.UrlCancion))
01512                 {
01513                     if (SelectedCancion.UrlCancion.Contains("cloudinary"))
01514                     {
01515                         bool seEliminoNube = await _storageService.EliminarArchivo(SelectedCancion.UrlCancion);
01516                         if (!seEliminoNube)
01517                         {
01518                             _dialogoService.MostrarAlerta("Msg_Error_EliminarAudioNube");
01519                             return;
01520                         }
01521                     }
01522                 }
01523
01524                 // Borrar de BD
01525                 bool exito = await cliente.EliminarCancionPorId(SelectedCancion.Id);
01526
01527                 if (exito)
01528                 {
01529                     // Restar contadores
01530                     if (SelectedCancion.AutoresIds != null)
01531                     {
01532                         foreach (var idAutor in SelectedCancion.AutoresIds)
01533                             await cliente.IncrementarContadorCancionesUsuario(idAutor, -1);
01534                     }
01535                     await CargarTodo();
01536                     _dialogoService.MostrarAlerta("Msg_Exito_CancionEliminada");
01537                 }
01538                 else _dialogoService.MostrarAlerta("Msg_Error_EliminarCancion");
```

```
01539
01540           }, "Msg_Carga_EliminandoCancion");
01541       }
01542       /// <summary>
01543       /// Remueve una playlist de la base de datos tras confirmación del usuario.
01544       /// </summary>
01545       private async Task EliminarPlaylistTask()
01546       {
01547           var cliente = MongoClientSingleton.Instance.Cliente;
01548           if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01549
01550           if (!await _dialogoService.Preguntar(
01551               "Msg_Confirmar_Titulo",
01552               "Msg_Confirmar_BorrarPlaylist",
01553               "Msg_Confirmar_BtnSi",
01554               "Msg_Confirmar_BtnNo")) return;
01555
01556           await EjecutarConCarga(async () =>
01557           {
01558               bool exito = await cliente.EliminarPlaylistPorId(SelectedPlaylist.Id);
01559               if (exito)
01560               {
01561                   await CargarTodo();
01562                   _dialogoService.MostrarAlerta("Msg_Exito_PlaylistEliminada");
01563               }
01564               else _dialogoService.MostrarAlerta("Msg_Error_EliminarPlaylist");
01565           }, "Msg_Carga_BorrandoPlaylist");
01566       }
01567       /// <summary>
01568       /// Elimina un género musical del catálogo global de la aplicación.
01569       /// </summary>
01570       private async Task EliminarGeneroTask()
01571       {
01572           var cliente = MongoClientSingleton.Instance.Cliente;
01573           if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01574
01575           if (!await _dialogoService.Preguntar(
01576               "Msg_Confirmar_Titulo",
01577               "Msg_Confirmar_BorrarGenero",
01578               "Msg_Confirmar_BtnSi",
01579               "Msg_Confirmar_BtnNo")) return;
01580
01581           await EjecutarConCarga(async () =>
01582           {
01583               bool exito = await cliente.EliminarGenero(SelectedGenero);
01584               if (exito)
01585               {
01586                   await CargarTodo();
01587                   _dialogoService.MostrarAlerta("Msg_Exito_GeneroEliminado");
01588               }
01589               else _dialogoService.MostrarAlerta("Msg_Error_EliminarGenero");
01590           }, "Msg_Carga_EliminandoGenero");
01591       }
01592       /// <summary>
01593       /// Remueve el registro de un reporte del sistema.
01594       /// </summary>
01595       private async Task EliminarReporteTask()
01596       {
01597           var cliente = MongoClientSingleton.Instance.Cliente;
01598           if (cliente == null) { _dialogoService.MostrarAlerta("Msg_Error_Conexion"); return; }
01599
01600           if (!await _dialogoService.Preguntar(
01601               "Msg_Confirmar_Titulo",
01602               "Msg_Confirmar_BorrarReporte",
01603               "Msg_Confirmar_BtnSi",
01604               "Msg_Confirmar_BtnNo")) return;
01605
01606           await EjecutarConCarga(async () =>
01607           {
01608               bool exito = await cliente.EliminarReporte(SelectedReporte.Id);
01609               if (exito)
01610               {
01611                   await CargarTodo();
01612                   _dialogoService.MostrarAlerta("Msg_Exito_ReporteEliminado");
01613               }
01614               else _dialogoService.MostrarAlerta("Msg_Error_EliminarReporte");
01615           }, "Msg_Carga_EliminandoReporte");
01616       }
01617       /// <summary>
01618       /// Wrapper para ejecutar tareas asíncronas controlando los estados de carga y visualización de mensajes para el
      usuario.
01619       /// </summary>
01620       /// <param name="tarea">Función asíncrona a ejecutar.</param>
01621       /// <param name="mensaje">Mensaje de carga a mostrar en la interfaz.</param>
01622       private async Task EjecutarConCarga(Func<Task> tarea, string mensaje = "Procesando...")
01623       {
01624           if (EstaCargando) return; // Evitar doble clic
```

```
01625
01626           try
01627           {
01628               EstaCargando = true;
01629               MensajeCarga = mensaje;
01630               await tarea();
01631           }
01632           catch (Exception ex)
01633           {
01634               _dialogoService.MostrarAlerta($"Error: {ex.Message}");
01635           }
01636           finally
01637           {
01638               EstaCargando = false;
01639           }
01640       }
01641       /// <summary>
01642       /// Analiza un archivo multimedia local para extraer su duración exacta en segundos utilizando la librería TagLib.
01643       /// </summary>
01644       /// <param name="rutaArchivo">Ruta física del archivo en el disco.</param>
01645       /// <returns>Segundos de duración (0 en caso de fallo).</returns>
01646       private int ObtenerDuracionLocal(string rutaArchivo)
01647       {
01648           try
01649           {
01650               if (System.IO.File.Exists(rutaArchivo))
01651               {
01652                   var archivo = TagLib.File.Create(rutaArchivo);
01653                   return (int)archivo.Properties.Duration.TotalSeconds;
01654               }
01655           }
01656           catch (Exception x)
01657           {
01658               System.Diagnostics.Debug.WriteLine("Error al obtener duración lo  cal: " + x.Message);
01659           }
01660           return 0;
01661       }
01662   }
01663 }
```

4.125. Referencia del archivo BetaProyecto/ViewModels/ViewGestionarCuentaViewModel.cs

4.126. ViewGestionarCuentaViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Services;
00003 using BetaProyecto.Singleton;
00004 using ReactiveUI;
00005 using System;
00006 using System.Collections.Generic;
00007 using System.Collections.ObjectModel;
00008 using System.Reactive;
00009 using System.Threading.Tasks;
00010
00011 namespace BetaProyecto.ViewModels
00012 {
00013     public class ViewGestionarCuentaViewModel : ViewModelBase
00014     {
00015         // Servicios
00016         private readonly IDialogoService _dialogoService;
00017         private readonly StorageService _storageService;
00018
00019         //Actions
00020         public Action<ListaPersonalizada>? SolicitudIrAEditarPlaylist { get; set; }
00021         public Action<Canciones>? SolicitudIrAEditarCanciones { get; set; }
00022
00023         // Bidings
00024         private ObservableCollection<Canciones> _misCanciones;
00025         public ObservableCollection<Canciones> MisCanciones
00026         {
00027             get => _misCanciones;
00028             set => this.RaiseAndSetIfChanged(ref _misCanciones, value);
00029         }
00030
00031         private ObservableCollection<ListaPersonalizada> _misPlaylists;
00032         public ObservableCollection<ListaPersonalizada> MisPlaylists
00033         {
```

```
00034           get => _misPlaylists;
00035           set => this.RaiseAndSetIfChanged(ref _misPlaylists, value);
00036       }
00037
00038       // Comandos Reactive
00039       public ReactiveCommand<Canciones, Unit> BtnEditarCancion { get; }
00040       public ReactiveCommand<Canciones, Unit> BtnEliminarCancion { get; }
00041
00042       public ReactiveCommand<ListaPersonalizada, Unit> BtnEditarPlaylist { get; }
00043       public ReactiveCommand<ListaPersonalizada, Unit> BtnEliminarPlaylist { get; }
00044
00045       public ReactiveCommand<Unit, Unit> BtnRefrescar { get; }
00046
00047       // Constructor
00048       public ViewGestionarCuentaViewModel()
00049       {
00050           //Inicializamos servicios
00051           _dialogoService = new DialogoService();
00052           _storageService = new StorageService();
00053
00054           // Inicializar listas
00055           MisCanciones = new ObservableCollection<Canciones>();
00056           MisPlaylists = new ObservableCollection<ListaPersonalizada>();
00057
00058           // Configurar comandos
00059           BtnEditarCancion = ReactiveCommand.Create<Canciones>(cancion =>
00060           {
00061               System.Diagnostics.Debug.WriteLine($"Editar Canción: {cancion.Titulo}");
00062               SolicitudIrAEditarCanciones?.Invoke(cancion);
00063           });
00064
00065           BtnEliminarCancion = ReactiveCommand.Create<Canciones>(async cancion =>
00066           {
00067               System.Diagnostics.Debug.WriteLine($"Eliminar Canción: {cancion.Titulo}");
00068               await EliminarCancion(cancion);
00069           });
00070
00071           BtnEditarPlaylist = ReactiveCommand.Create<ListaPersonalizada>(playlist =>
00072           {
00073               System.Diagnostics.Debug.WriteLine($"Editar Playlist: {playlist.Nombre}");
00074               SolicitudIrAEditarPlaylist?.Invoke(playlist);
00075           });
00076
00077           BtnEliminarPlaylist = ReactiveCommand.Create<ListaPersonalizada>(async playlist =>
00078           {
00079               System.Diagnostics.Debug.WriteLine($"Eliminar Playlist: {playlist.Nombre}");
00080               await EliminarPlaylist(playlist);
00081           });
00082
00083           BtnRefrescar = ReactiveCommand.CreateFromTask(CargarContenidoUsuario);
00084
00085           // Cargar datos al iniciar en segundo plano
00086           _ = CargarContenidoUsuario();
00087       }
00088       /// <summary>
00089       /// Gestiona el proceso integral de eliminación de una canción, incluyendo la limpieza de recursos en la nube y la
      actualización de la base de datos.
00090       /// </summary>
00091       /// <remarks>
00092       /// Este método ejecuta un flujo de borrado seguro mediante los siguientes pasos:
00093       /// <list type="number">
00094       /// <item><b>Confirmación:</b> Solicita permiso al usuario mediante <see cref="_dialogoService"/> para
      evitar eliminaciones accidentales.</item>
00095       /// <item><b>Limpieza de Almacenamiento:</b> Identifica si el archivo reside en Cloudinary y lo elimina
      físicamente mediante <see cref="_storageService"/>.</item>
00096       /// <item><b>Persistencia y Métricas:</b> Remueve el registro en MongoDB y decrementa el contador de
      canciones publicadas del usuario.</item>
00097       /// <item><b>Actualización de UI:</b> Remueve la instancia de la colección local <see cref="MisCanciones"/>
      para reflejar el cambio instantáneamente.</item>
00098       /// </list>
00099       /// </remarks>
00100       /// <param name="cancion">El objeto <see cref="Canciones"/> que se desea eliminar definitivamente del
      sistema.</param>
00101       /// <returns>Una tarea que representa la operación de eliminación asíncrona.</returns>
00102       private async Task EliminarCancion(Canciones cancion)
00103       {
00104           // Preguntar antes de borrar para evitar accidentes
00105           var confirm = await _dialogoService.Preguntar("MsgConfirmEliminar", "MsgPreguntaEliminarCancion",
      "BtnEliminar", "BtnCancelar");
00106           if (!confirm)
00107           {
00108               return;
00109           }
00110           try
00111           {
00112               // Borrar Audio en la nube (Cloudinary)
00113               if (!string.IsNullOrEmpty(cancion.UrlCancion) && cancion.UrlCancion.Contains("cloudinary"))
```

```
00114               {
00115                   await _storageService.EliminarArchivo(cancion.UrlCancion);
00116               }
00117               //Borrar datos en MongoDB
00118               bool exito = await MongoClientSingleton.Instance.Cliente.EliminarCancionPorId(cancion.Id);
00119
00120               if (exito)
00121               {
00122                   //Restar 1 al contador de canciones del usuario en MongoDB
00123                   await
    MongoClientSingleton.Instance.Cliente.IncrementarContadorCancionesUsuario(GlobalData.Instance.UserIdGD, -1);
00124
00125                   //Actualizar pantalla
00126                   MisCanciones.Remove(cancion);
00127                   _dialogoService.MostrarAlerta("MsgExitoBorrado");
00128               }
00129               else
00130               {
00131                   _dialogoService.MostrarAlerta("MsgErrorBorradoDB");
00132               }
00133
00134           }
00135           catch (Exception ex)
00136           {
00137               _dialogoService.MostrarAlerta("MsgErrorBorradoDB");
00138               System.Diagnostics.Debug.WriteLine($"Error base: {ex}");
00139
00140           }
00141       }
00142       /// <summary>
00143       /// Gestiona el proceso de eliminación de una lista de reproducción personalizada de la base de datos y de la interfaz
    de usuario.
00144       /// </summary>
00145       /// <remarks>
00146       /// Este método ejecuta un flujo de borrado seguro estructurado en los siguientes pasos:
00147       /// <list type="number">
00148       /// <item><b>Confirmación:</b> Solicita una validación explícita al usuario a través de <see
    cref="_dialogoService"/> para prevenir eliminaciones accidentales.</item>
00149       /// <item><b>Persistencia:</b> Invoca al cliente de MongoDB para eliminar el registro físico de la lista mediante
    su identificador único.</item>
00150       /// <item><b>Actualización de UI:</b> Si la operación en la base de datos es exitosa, remueve la instancia de la
    colección <see cref="MisPlaylists"/> para refrescar la vista inmediatamente.</item>
00151       /// </list>
00152       /// Notifica al usuario el resultado de la operación mediante mensajes de alerta traducidos.
00153       /// </remarks>
00154       /// <param name="playlist">El objeto <see cref="ListaPersonalizada"/> que se desea eliminar definitivamente del
    sistema.</param>
00155       /// <returns>Una tarea que representa la operación de eliminación asíncrona.</returns>
00156       private async Task EliminarPlaylist(ListaPersonalizada playlist)
00157       {
00158           var confirm = await _dialogoService.Preguntar("MsgConfirmEliminar", "MsgPreguntaEliminarPlaylist",
    "BtnEliminar", "BtnCancelar");
00159           if (!confirm)
00160           {
00161               return;
00162           }
00163
00164           bool exito = await MongoClientSingleton.Instance.Cliente.EliminarPlaylistPorId(playlist.Id);
00165
00166           if (exito)
00167           {
00168               MisPlaylists.Remove(playlist);
00169               _dialogoService.MostrarAlerta("MsgExitoBorrado");
00170           }
00171           else
00172           {
00173               _dialogoService.MostrarAlerta("MsgErrorBorradoDB");
00174           }
00175       }
00176       /// <summary>
00177       /// Recupera y carga de forma asíncrona el catálogo de canciones y listas de reproducción creadas por el usuario
    actual.
00178       /// </summary>
00179       /// <remarks>
00180       /// Este método gestiona la carga de contenido personal en dos fases:
00181       /// <list type="number">
00182       /// <item><b>Consulta paralela:</b> Lanza simultáneamente las peticiones a MongoDB para obtener las
    canciones por autor y las playlists por creador utilizando el ID de <see cref="GlobalData.Instance.UserIdGD"/>.</item>
00183       /// <item><b>Sincronización:</b> Utiliza <see cref="Task.WhenAll"/> para optimizar el tiempo de respuesta y,
    una vez recibidos los datos, inicializa las colecciones <see cref="MisCanciones"/> y <see cref="MisPlaylists"/>.</item>
00184       /// </list>
00185       /// Esto asegura que la interfaz de usuario se actualice con todo el contenido propio del usuario de una sola vez.
00186       /// </remarks>
00187       /// <returns>Una tarea que representa la operación de carga asíncrona.</returns>
00188       private async Task CargarContenidoUsuario()
00189       {
00190           if(MongoClientSingleton.Instance.Cliente != null)
```

```
00191           {
00192               string miId = GlobalData.Instance.UserIdGD;
00193
00194               var listaCanciones = MongoClientSingleton.Instance.Cliente.ObtenerCancionesPorAutor(miId);
00195               var listaPlaylist = MongoClientSingleton.Instance.Cliente.ObtenerPlaylistsPorCreador(miId);
00196
00197
00198               await Task.WhenAll(listaCanciones,listaPlaylist);
00199
00200               MisCanciones = new ObservableCollection<Canciones>(listaCanciones.Result);
00201               MisPlaylists = new ObservableCollection<ListaPersonalizada>(listaPlaylist.Result);
00202           }
00203       }
00204   }
00205 }
```

4.127. Referencia del archivo BetaProyecto/ViewModels/ViewGestionarReportesViewModel.cs

4.128. ViewGestionarReportesViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Services;
00003 using BetaProyecto.Singleton;
00004 using ReactiveUI;
00005 using System.Collections.ObjectModel;
00006 using System.Linq;
00007 using System.Reactive;
00008 using System.Threading.Tasks;
00009
00010 namespace BetaProyecto.ViewModels
00011 {
00012     public class ViewGestionarReportesViewModel : ViewModelBase
00013     {
00014         //Servicios
00015         private readonly IDialogoService _dialogoService;
00016
00017         // Lista
00018         public ObservableCollection<Reportes> ListaPendientes { get; }
00019         public ObservableCollection<Reportes> ListaInvestigando { get; }
00020         public ObservableCollection<Reportes> ListaFinalizados { get; }
00021         public ObservableCollection<string> OpcionesEstado { get; } = new()
00022         { "Pendiente", "Investigando", "Finalizado" };
00023
00024         //Bindings
00025         private Reportes _reporteSeleccionado;
00026         public Reportes ReporteSeleccionado
00027         {
00028             get => _reporteSeleccionado;
00029             set
00030             {
00031                 this.RaiseAndSetIfChanged(ref _reporteSeleccionado, value);
00032                 if (value != null)
00033                 {
00034                     EstadoEdit = value.Estado;
00035                     ResolucionEdit = value.Resolucion;
00036                 }
00037             }
00038         }
00039         //Hacemos selecciones individuales para cada lista para evitar conflictos al seleccionar en una y que se marque en las
      otras
00040         // Selección Pendientes
00041         private Reportes _selPendiente;
00042         public Reportes SelectedPendiente
00043         {
00044             get => _selPendiente;
00045             set
00046             {
00047                 this.RaiseAndSetIfChanged(ref _selPendiente, value);
00048                 if (value != null)
00049                 {
00050                     ReporteSeleccionado = value;
00051                     SelectedInvestigando = null;
00052                     SelectedFinalizado = null;
00053                 }
00054             }
00055         }
00056
```

```
00057         // Selección Investigando
00058         private Reportes _selInvestigando;
00059         public Reportes SelectedInvestigando
00060         {
00061             get => _selInvestigando;
00062             set
00063             {
00064                 this.RaiseAndSetIfChanged(ref _selInvestigando, value);
00065                 if (value != null)
00066                 {
00067                     ReporteSeleccionado = value;
00068                     SelectedPendiente = null;
00069                     SelectedFinalizado = null;
00070                 }
00071             }
00072         }
00073 
00074         // Selección Finalizados
00075         private Reportes _selFinalizado;
00076         public Reportes SelectedFinalizado
00077         {
00078             get => _selFinalizado;
00079             set
00080             {
00081                 this.RaiseAndSetIfChanged(ref _selFinalizado, value);
00082                 if (value != null)
00083                 {
00084                     ReporteSeleccionado = value;
00085                     SelectedPendiente = null;
00086                     SelectedInvestigando = null;
00087                 }
00088             }
00089         }
00090 
00091         private string _estadoEdit;
00092         public string EstadoEdit
00093         {
00094             get => _estadoEdit;
00095             set => this.RaiseAndSetIfChanged(ref _estadoEdit, value);
00096         }
00097 
00098         private string _resolucionEdit;
00099         public string ResolucionEdit
00100         {
00101             get => _resolucionEdit;
00102             set => this.RaiseAndSetIfChanged(ref _resolucionEdit, value);
00103         }
00104 
00105         //Comandos reactive
00106         public ReactiveCommand<Unit, Unit> BtnRefrescar { get; }
00107         public ReactiveCommand<Unit, Unit> BtnGuardar { get; }
00108 
00109         public ViewGestionarReportesViewModel()
00110         {
00111             //Inicializamos servicios
00112             _dialogoService = new DialogoService();
00113             // Inicializamos listas
00114             ListaPendientes = new ObservableCollection<Reportes>();
00115             ListaInvestigando = new ObservableCollection<Reportes>();
00116             ListaFinalizados = new ObservableCollection<Reportes>();
00117 
00118             // Configuramos comandos
00119             BtnRefrescar = ReactiveCommand.CreateFromTask(CargarDatos);
00120             BtnGuardar = ReactiveCommand.CreateFromTask(GuardarCambios);
00121             _ = CargarDatos();
00122         }
00123         /// <summary>
00124         /// Recupera todos los reportes de la base de datos y los clasifica en colecciones independientes según su estado actual.
00125         /// </summary>
00126         /// <remarks>
00127         /// Este método realiza una limpieza integral de las listas y selecciones actuales para evitar duplicidad visual.
00128         /// Posteriormente, consulta MongoDB y distribuye cada reporte en las categorías de "Pendiente", "Investigando"
00129         /// o "Finalizado" basándose en el valor de su propiedad <c>Estado</c>, facilitando la organización por columnas
    en la interfaz.
00130         /// </remarks>
00131         /// <returns>Una tarea que representa la operación de carga y clasificación asíncrona.</returns>
00132         private async Task CargarDatos()
00133         {
00134             // Limpiamos todo
00135             ListaPendientes.Clear();
00136             ListaInvestigando.Clear();
00137             ListaFinalizados.Clear();
00138 
00139             // Limpiamos selecciones
00140             SelectedPendiente = null;
00141             SelectedInvestigando = null;
00142             SelectedFinalizado = null;
```

```
00143             ReporteSeleccionado = null;
00144
00145             var todos = await MongoClientSingleton.Instance.Cliente.ObtenerReportes();
00146
00147             foreach (var r in todos)
00148             {
00149                 switch (r.Estado?.Trim())
00150                 {
00151                     case "Pendiente": ListaPendientes.Add(r); break;
00152                     case "Investigando": ListaInvestigando.Add(r); break;
00153                     case "Finalizado": ListaFinalizados.Add(r); break;
00154                     default: ListaPendientes.Add(r); break;
00155                 }
00156             }
00157         }
00158         /// <summary>
00159         /// Persiste de forma asíncrona las modificaciones realizadas en el estado y la resolución del reporte seleccionado.
00160         /// </summary>
00161         /// <remarks>
00162         /// Este método sincroniza los cambios con la base de datos MongoDB mediante los siguientes pasos:
00163         /// <list type="number">
00164         /// <item><b>Validación:</b> Verifica que exista una instancia válida en <see
      cref="ReporteSeleccionado"/>.</item>
00165         /// <item><b>Sincronización:</b> Envía los nuevos valores de <c>EstadoEdit</c> y <c>ResolucionEdit</c>
      al servidor.</item>
00166         /// <item><b>Refresco:</b> Si la operación es exitosa, notifica al usuario y reejecuta <see cref="CargarDatos"/>
      para reorganizar los reportes en sus respectivas columnas visuales.</item>
00167         /// </list>
00168         /// </remarks>
00169         /// <returns>Una tarea que representa la operación de actualización asíncrona.</returns>
00170         private async Task GuardarCambios()
00171         {
00172             if (ReporteSeleccionado == null) return;
00173
00174             bool exito = await MongoClientSingleton.Instance.Cliente.ActualizarEstadoReporte(
00175                 EstadoEdit, ResolucionEdit, ReporteSeleccionado
00176             );
00177
00178             if (exito)
00179             {
00180                 _dialogoService.MostrarAlerta("Reportes_MsgActualizado");
00181                 await CargarDatos();
00182             }
00183             else
00184             {
00185                 _dialogoService.MostrarAlerta("Reportes_MsgSinCambios");
00186             }
00187         }
00188     }
00189 }
```

4.129. Referencia del archivo BetaProyecto/ViewModels/ViewListaPersonalizadaViewModel.cs

4.130. ViewListaPersonalizadaViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using ReactiveUI;
00003 using System;
00004 using System.Reactive;
00005
00006 namespace BetaProyecto.ViewModels
00007 {
00008     public class ViewListaPersonalizadaViewModel : ViewModelBase
00009     {
00010         public ListaPersonalizada Playlist { get; }
00011
00012         // Comandos Reactive
00013         public ReactiveCommand<Unit, Unit> BtnVolver { get; }
00014
00015         //Propiedad calculada
00016         public int CantidadCanciones => Playlist.CancionesCompletas?.Count ?? 0;
00017
00018         public ViewListaPersonalizadaViewModel(ListaPersonalizada playlist, Action accionVolver)
00019         {
00020             Playlist = playlist;
00021
00022             // Configuramos comando reactive
```

```
00023             BtnVolver = ReactiveCommand.Create(accionVolver);
00024         }
00025
00026     }
00027 }
```

4.131. Referencia del archivo BetaProyecto/ViewModels/ViewModelBase.cs

4.132. ViewModelBase.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Media.Imaging;
00002 using Avalonia.Platform;
00003 using ReactiveUI;
00004 using System;
00005
00006 namespace BetaProyecto.ViewModels
00007 {
00008     public class ViewModelBase : ReactiveObject
00009     {
00010
00011     }
00012 }
```

4.133. Referencia del archivo BetaProyecto/ViewModels/ViewPerfilViewModel.cs

4.134. ViewPerfilViewModel.cs

Ir a la documentación de este archivo.

```
00001 using BetaProyecto.Models;
00002 using BetaProyecto.Singleton;
00003 using ReactiveUI;
00004 using System;
00005 using System.Collections.ObjectModel;
00006 using System.Reactive;
00007 using System.Threading;
00008 using System.Threading.Tasks;
00009
00010 namespace BetaProyecto.ViewModels
00011 {
00012     public class ViewPerfilViewModel : ViewModelBase
00013     {
00014         //Bidings
00015         private string _nombreUsuario;
00016         public string NombreUsuario { get => _nombreUsuario; set => this.RaiseAndSetIfChanged(ref _nombreUsuario,
      value); }
00017
00018         private string _imagenPerfil;
00019         public string ImagenPerfil { get => _imagenPerfil; set => this.RaiseAndSetIfChanged(ref _imagenPerfil, value); }
00020
00021         private ObservableCollection<ListaUsuarios> _secciones;
00022         public ObservableCollection<ListaUsuarios> Secciones
00023         {
00024             get => _secciones;
00025             set => this.RaiseAndSetIfChanged(ref _secciones, value);
00026         }
00027         //Comandos Reactive
00028         public ReactiveCommand<Unit, Unit> BtnRefrescar { get; }
00029
00030         public ViewPerfilViewModel()
00031         {
00032             //Inicializamos listas
00033             Secciones = new ObservableCollection<ListaUsuarios>();
00034
00035             //Configuramos comandos reactive
00036             BtnRefrescar = ReactiveCommand.CreateFromTask(async () => await CargarDatos());
00037
00038             _ = CargarDatos(); // Cargamos los datos en segundo plano
00039         }
00040         /// <summary>
00041         /// Recupera y organiza de forma asíncrona la información del perfil, artistas sugeridos y usuarios seguidos.
00042         /// </summary>
00043         /// <remarks>
```

```
00044         /// Este método gestiona la carga de la red social del usuario mediante los siguientes pasos:
00045         /// <list type="number">
00046         /// <item><b>Inicialización:</b> Carga la identidad básica (nombre y foto) desde <see
      cref="GlobalData.Instance"/>.</item>
00047         /// <item><b>Carga Paralela:</b> Ejecuta simultáneamente las peticiones a MongoDB para obtener los perfiles
      seguidos y el catálogo global de usuarios mediante <see cref="Task.WhenAll"/>.</item>
00048         /// <item><b>Categorización:</b> Estructura los resultados en secciones diferenciadas ("Descubre Artistas" y
      "Siguiendo") utilizando claves de traducción para los encabezados.</item>
00049         /// <item><b>Asignación:</b> Actualiza la propiedad <see cref="Secciones"/>, lo que dispara la actualización
      de los controles agrupados en la interfaz.</item>
00050         /// </list>
00051         /// Cualquier fallo durante la consulta se registra en la consola de depuración para evitar el colapso de la vista.
00052         /// </remarks>
00053         /// <returns>Una tarea que representa la operación de carga y estructuración asíncrona.</returns>
00054         private async Task CargarDatos()
00055         {
00056             try
00057             {
00058                 // Carga datos básicos
00059                 NombreUsuario = GlobalData.Instance.UsernameGD ?? "Usuario";
00060                 ImagenPerfil = GlobalData.Instance.UrlFotoPerfilGD ?? "https://i.ibb.co/dbQSrpB/Perfil.png";
00061 
00062                 // Carga listas en paralelo
00063                 var listaIds = GlobalData.Instance.SeguidoresGD;
00064                 var taskSeguidores = MongoClientSingleton.Instance.Cliente.ObtenerUsuariosPorListaIds(listaIds);
00065                 var taskTodos = MongoClientSingleton.Instance.Cliente.ObtenerTodosLosUsuarios();
00066 
00067                 await Task.WhenAll(taskSeguidores, taskTodos);
00068 
00069                 var listaSecciones = new ObservableCollection<ListaUsuarios>();
00070 
00071                 // Sección 1: Descubre Artistas
00072                 listaSecciones.Add(new ListaUsuarios("Perfil_SecDescubre", new
      ObservableCollection<Usuarios>(taskTodos.Result)));
00073 
00074                 // Sección 2: Siguiendo
00075                 listaSecciones.Add(new ListaUsuarios("Perfil_SecSiguiendo",new
      ObservableCollection<Usuarios>(taskSeguidores.Result)));
00076 
00077                 // Asignamos a la propiedad pública
00078                 Secciones = listaSecciones;
00079             }
00080             catch (Exception ex)
00081             {
00082                 System.Diagnostics.Debug.WriteLine("Error cargando perfil: " + ex.Message);
00083             }
00084         }
00085     }
00086 }
```

4.135. Referencia del archivo BetaProyecto/ViewModels/ViewPublicarCancionViewModel.cs

4.136. ViewPublicarCancionViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Media.Imaging;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Services;
00004 using BetaProyecto.Singleton;
00005 using ReactiveUI;
00006 using System;
00007 using System.Collections.ObjectModel;
00008 using System.Linq;
00009 using System.Reactive;
00010 using System.Reactive.Linq;
00011 using System.Threading.Tasks;
00012 
00013 namespace BetaProyecto.ViewModels
00014 {
00015     public class ViewPublicarCancionViewModel : ViewModelBase
00016     {
00017         // Servicios
00018         private readonly IDialogoService _dialogoService;
00019         private readonly StorageService _storageService;
00020         private readonly AudioService _audioService;
00021 
00022         // Actions
00023         private readonly Action _Volver;
```

```
00024
00025         // Datos de canciones
00026         private string _txtTitulo;
00027         public string TxtTitulo
00028         {
00029             get => _txtTitulo;
00030             set => this.RaiseAndSetIfChanged(ref _txtTitulo, value);
00031         }
00032
00033         // Listas de Géneros
00034         private ObservableCollection<string> _listaGeneros;
00035         public ObservableCollection<string> ListaGeneros
00036         {
00037             get => _listaGeneros;
00038             set => this.RaiseAndSetIfChanged(ref _listaGeneros, value);
00039         }
00040
00041         private string _generoSeleccionado;
00042         public string GeneroSeleccionado
00043         {
00044             get => _generoSeleccionado;
00045             set => this.RaiseAndSetIfChanged(ref _generoSeleccionado, value);
00046         }
00047
00048         // Lista de etiquetas seleccionadas
00049         private ObservableCollection<string> _listaGenerosSeleccionados;
00050         public ObservableCollection<string> ListaGenerosSeleccionados
00051         {
00052             get => _listaGenerosSeleccionados;
00053             set => this.RaiseAndSetIfChanged(ref _listaGenerosSeleccionados, value);
00054         }
00055
00056         // Gestión de colaboradores
00057         private string _txtBusqueda;
00058         public string TxtBusqueda
00059         {
00060             get => _txtBusqueda;
00061             set => this.RaiseAndSetIfChanged(ref _txtBusqueda, value);
00062         }
00063
00064         private ObservableCollection<Usuarios> _listaResultados;
00065         public ObservableCollection<Usuarios> ListaResultados
00066         {
00067             get => _listaResultados;
00068             set => this.RaiseAndSetIfChanged(ref _listaResultados, value);
00069         }
00070
00071         private ObservableCollection<Usuarios> _listaArtitas;
00072         public ObservableCollection<Usuarios> ListaArtistas
00073         {
00074             get => _listaArtitas;
00075             set => this.RaiseAndSetIfChanged(ref _listaArtitas, value);
00076         }
00077
00078         // Imagen y audio
00079         private string _rutaImagen;
00080         public string RutaImagen
00081         {
00082             get => _rutaImagen;
00083             set
00084             {
00085                 this.RaiseAndSetIfChanged(ref _rutaImagen, value);
00086                 CargarImagenLocal(value);
00087             }
00088         }
00089         public bool TieneImagen => !string.IsNullOrEmpty(RutaImagen);
00090
00091         private Bitmap? _imagenPortada;
00092         public Bitmap? ImagenPortada
00093         {
00094             get => _imagenPortada;
00095             set => this.RaiseAndSetIfChanged(ref _imagenPortada, value);
00096         }
00097
00098         // Binding Declarativo
00099         private bool _esYoutube = true;
00100         public bool EsYoutube
00101         {
00102             get => _esYoutube;
00103             set
00104             {
00105                 this.RaiseAndSetIfChanged(ref _esYoutube, value);
00106                 this.RaisePropertyChanged(nameof(EsArchivo));
00107             }
00108         }
00109         public bool EsArchivo => !EsYoutube;
00110
```

```
00111         private string _linkYoutube;
00112         public string LinkYoutube
00113         {
00114             get => _linkYoutube;
00115             set => this.RaiseAndSetIfChanged(ref _linkYoutube, value);
00116         }
00117
00118         private string _rutaMp3;
00119         public string RutaMp3
00120         {
00121             get => _rutaMp3;
00122             set
00123             {
00124                 this.RaiseAndSetIfChanged(ref _rutaMp3, value);
00125                 if (!string.IsNullOrEmpty(value) && EsArchivo)
00126                 {
00127                     _duracionCalculada = ObtenerDuracionMp3(value);
00128                 }
00129             }
00130         }
00131         private int _duracionCalculada = 0;
00132
00133         // Estado
00134         private bool _estaCargando;
00135         public bool EstaCargando
00136         {
00137             get => _estaCargando;
00138             set => this.RaiseAndSetIfChanged(ref _estaCargando, value);
00139         }
00140
00141         // Comandos reactive
00142         public ReactiveCommand<Unit, Unit> BtnVolverAtras { get; }
00143         public ReactiveCommand<Unit, Unit> BtnPublicar { get; }
00144         public ReactiveCommand<Unit, Unit> BtnBuscarUsuarios { get; }
00145         public ReactiveCommand<Usuarios, Unit> BtnAgregarUsuario { get; }
00146         public ReactiveCommand<Usuarios, Unit> BtnEliminarUsuario { get; }
00147         public ReactiveCommand<Unit, Unit> BtnAgregarGenero { get; }
00148         public ReactiveCommand<string, Unit> BtnEliminarGenero { get; }
00149
00150         //Constructor
00151         public ViewPublicarCancionViewModel(Action accionVolver)
00152         {
00153             //Heredamos actions
00154             _Volver = accionVolver;
00155
00156             //Inicializamos servicios
00157             _dialogoService = new DialogoService();
00158             _storageService = new StorageService();
00159             _audioService = new AudioService();
00160
00161             //Inicializamos listas
00162             ListaResultados = new ObservableCollection<Usuarios>();
00163             ListaArtistas = new ObservableCollection<Usuarios>();
00164             ListaGeneros = new ObservableCollection<string>();
00165             ListaGenerosSeleccionados = new ObservableCollection<string>();
00166
00167             // Nos añadimos como colaborador automáticamente
00168             var miUsuario = GlobalData.Instance.GetUsuarioObject();
00169             if (miUsuario != null) ListaArtistas.Add(miUsuario);
00170
00171             // Configuración del Buscador Reactivo
00172             this.WhenAnyValue(x => x.TxtBusqueda)
00173                 .Throttle(TimeSpan.FromMilliseconds(500))
00174                 .Where(x => !string.IsNullOrWhiteSpace(x) && x.Length > 2)
00175                 .ObserveOn(RxApp.MainThreadScheduler)
00176                 .Subscribe(_ => BuscarUsuarios());
00177
00178             // Validación para Publicar
00179             var validacionPublicar = this.WhenAnyValue(
00180                 x => x.TxtTitulo,
00181                 x => x.LinkYoutube,
00182                 x => x.RutaMp3,
00183                 x => x.EsYoutube,
00184                 x => x.RutaImagen,
00185                 x => x.ListaGenerosSeleccionados.Count,
00186                 (titulo, link, mp3, esYt, imagen, cantidadGeneros) =>
00187                     !string.IsNullOrWhiteSpace(titulo) &&
00188                     !string.IsNullOrWhiteSpace(imagen) &&
00189                     cantidadGeneros > 0 &&
00190                     (esYt ? !string.IsNullOrWhiteSpace(link) : !string.IsNullOrWhiteSpace(mp3))
00191             );
00192
00193             // Configuramos comandos reactive
00194             BtnAgregarGenero = ReactiveCommand.Create(AgregarGenero);
00195             BtnEliminarGenero = ReactiveCommand.Create<string>(EliminarGenero);
00196             BtnBuscarUsuarios = ReactiveCommand.Create(BuscarUsuarios);
00197             BtnAgregarUsuario = ReactiveCommand.Create<Usuarios>(AgregarUsuario);
```

```
00198           BtnEliminarUsuario = ReactiveCommand.Create<Usuarios>(EliminarUsuario);
00199           BtnVolverAtras = ReactiveCommand.Create(accionVolver);
00200           BtnPublicar = ReactiveCommand.CreateFromTask(PublicarCancion, validacionPublicar);
00201
00202           // Carga inicial
00203           _ = CargarGeneros();
00204       }
00205       /// <summary>
00206       /// Añade el género seleccionado actualmente a la lista de géneros asociados, validando que no esté vacío y que no se
    haya añadido previamente.
00207       /// </summary>
00208       /// <remarks>
00209       /// El método verifica si <see cref="GeneroSeleccionado"/> contiene un valor válido. Si el género ya existe en
00210       /// <see cref="ListaGenerosSeleccionados"/> (comparación insensible a mayúsculas), se muestra una alerta de error
00211       /// a través de <see cref="_dialogoService"/>. En cualquier caso, tras el intento de adición, se restablece
00212       /// la propiedad <see cref="GeneroSeleccionado"/> a nulo para limpiar la selección de la interfaz.
00213       /// </remarks>
00214       private void AgregarGenero()
00215       {
00216           if (string.IsNullOrWhiteSpace(GeneroSeleccionado))
00217           {
00218               return;
00219           }
00220
00221           bool yaEstaEnLista = ListaGenerosSeleccionados.Any(g => g.Equals(GeneroSeleccionado,
    StringComparison.OrdinalIgnoreCase));
00222
00223           if (!yaEstaEnLista)
00224           {
00225               ListaGenerosSeleccionados.Add(GeneroSeleccionado);
00226               GeneroSeleccionado = null;
00227           }
00228           else
00229           {
00230               _dialogoService.MostrarAlerta("Msg_Error_GeneroYaAnadido");
00231               GeneroSeleccionado = null;
00232           }
00233       }
00234       /// <summary>
00235       /// Elimina un género específico de la lista de géneros seleccionados para la canción.
00236       /// </summary>
00237       /// <remarks>
00238       /// Este método verifica si el género proporcionado existe dentro de <see cref="ListaGenerosSeleccionados"/>.
00239       /// Si se encuentra, lo elimina, lo que actualiza automáticamente cualquier control de la interfaz
00240       /// vinculado a esta colección.
00241       /// </remarks>
00242       /// <param name="genero">El nombre del género que se desea remover de la selección actual.</param>
00243       private void EliminarGenero(string genero)
00244       {
00245           if (ListaGenerosSeleccionados.Contains(genero))
00246           {
00247               ListaGenerosSeleccionados.Remove(genero);
00248           }
00249       }
00250       /// <summary>
00251       /// Añade un usuario a la lista de artistas seleccionados, evitando duplicados y limpiando los resultados de búsqueda
    actuales.
00252       /// </summary>
00253       /// <remarks>
00254       /// El método verifica mediante el ID si el <paramref name="usuario"/> ya se encuentra en <see
    cref="ListaArtistas"/>.
00255       /// Tras la validación, independientemente de si se añadió o no, se restablece <see cref="TxtBusqueda"/>
00256       /// y se vacía <see cref="ListaResultados"/> para limpiar la interfaz de búsqueda.
00257       /// </remarks>
00258       /// <param name="usuario">El objeto de tipo <see cref="Usuarios"/> que se desea vincular o añadir.</param>
00259       private void AgregarUsuario(Usuarios usuario)
00260       {
00261           bool yaExiste = ListaArtistas.Any(u => u.Id == usuario.Id);
00262           if (!yaExiste)
00263           {
00264               ListaArtistas.Add(usuario);
00265           }
00266           TxtBusqueda = string.Empty;
00267           ListaResultados.Clear();
00268       }
00269       /// <summary>
00270       /// Elimina un usuario de la lista de artistas seleccionados, validando que no sea el usuario que ha iniciado sesión.
00271       /// </summary>
00272       /// <remarks>
00273       /// El método comprueba si el <paramref name="usuario"/> a eliminar coincide con el ID del usuario actual en
00274       /// <see cref="GlobalData.Instance.UserIdGD"/>. Si coinciden, se muestra una alerta de error mediante
00275       /// <see cref="_dialogoService"/> para impedir que un usuario se elimine a sí mismo de una lista.
00276       /// Si la validación es correcta, procede a removerlo de <see cref="ListaArtistas"/>.
00277       /// </remarks>
00278       /// <param name="usuario">El objeto de tipo <see cref="Usuarios"/> que se desea remover de la
    selección.</param>
00279       private void EliminarUsuario(Usuarios usuario)
```

```
00280         {
00281             if (usuario.Id == GlobalData.Instance.UserIdGD)
00282             {
00283                 _dialogoService.MostrarAlerta("Msg_Error_BorrarPropioUser");
00284                 return;
00285             }
00286             if (ListaArtistas.Contains(usuario))
00287             {
00288                 ListaArtistas.Remove(usuario);
00289             }
00290         }
00291         /// <summary>
00292         /// Realiza una búsqueda asíncrona de usuarios en la base de datos basada en el texto introducido, filtrando aquellos que ya han sido seleccionados.
00293         /// </summary>
00294         /// <remarks>
00295         /// Este método utiliza <see cref="MongoClientSingleton"/> para consultar usuarios cuyo nombre coincida con <see cref="TxtBusqueda"/>.
00296         /// Para evitar duplicados, se envían los IDs de la <see cref="ListaArtistas"/> actual como lista de exclusión.
00297         /// Si se encuentran resultados, se actualiza <see cref="ListaResultados"/>; de lo contrario, se limpia.
00298         /// En caso de fallo en la conexión, se muestra una alerta mediante <see cref="_dialogoService"/>.
00299         /// </remarks>
00300         private async void BuscarUsuarios()
00301         {
00302             if (MongoClientSingleton.Instance.Cliente != null)
00303             {
00304                 var listaResultadosBusqueda = await MongoClientSingleton.Instance.Cliente.ObtenerUsuariosPorBusqueda(TxtBusqueda, ListaArtistas.Select(x => x.Id).ToList());
00305                 if (listaResultadosBusqueda != null && listaResultadosBusqueda.Count > 0)
00306                 {
00307                     ListaResultados = new ObservableCollection<Usuarios>(listaResultadosBusqueda);
00308                 }
00309                 else
00310                 {
00311                     ListaResultados.Clear();
00312                 }
00313             }
00314             else
00315             {
00316                 _dialogoService.MostrarAlerta("Msg_Error_Conexion");
00317             }
00318         }
00319         /// <summary>
00320         /// Carga una imagen desde una ruta local y la asigna a la propiedad ImagenPortada.
00321         /// </summary>
00322         /// <remarks>
00323         /// Intenta crear un objeto <see cref="Bitmap"/> a partir de la ruta proporcionada. Si el archivo no existe o ocurre un error
00324         /// durante la lectura, se asigna <c>null</c> a <see cref="ImagenPortada"/> para evitar fallos visuales.
00325         /// Finalmente, notifica el cambio de la propiedad <see cref="TieneImagen"/> para actualizar la UI.
00326         /// </remarks>
00327         /// <param name="ruta">La ruta del sistema de archivos donde se encuentra la imagen.</param>
00328         private void CargarImagenLocal(string ruta)
00329         {
00330             try
00331             {
00332                 if (System.IO.File.Exists(ruta))
00333                 {
00334                     ImagenPortada = new Bitmap(ruta);
00335                 }
00336                 else
00337                 {
00338                     ImagenPortada = null;
00339                 }
00340             }
00341             catch
00342             {
00343                 ImagenPortada = null;
00344             }
00345             this.RaisePropertyChanged(nameof(TieneImagen));
00346         }
00347         /// <summary>
00348         /// Carga la lista de géneros disponibles desde la base de datos y los asigna a la propiedad ListaGeneros.
00349         /// </summary>
00350         /// <remarks>
00351         /// Este método recupera todos los nombres de géneros registrados en MongoDB mediante el cliente singleton.
00352         /// Si la conexión es exitosa, inicializa <see cref="ListaGeneros"/>; de lo contrario, registra el error en
00353         /// el flujo de depuración del sistema.
00354         /// </remarks>
00355         /// <returns>Una tarea que representa la operación asíncrona.</returns>
00356         private async Task CargarGeneros()
00357         {
00358             if (MongoClientSingleton.Instance.Cliente != null)
00359             {
00360                 var listadeGeneros = await MongoClientSingleton.Instance.Cliente.ObtenerNombresGeneros();
00361                 ListaGeneros = new ObservableCollection<string>(listadeGeneros);
```

```
00362           }
00363           else
00364           {
00365               System.Diagnostics.Debug.WriteLine("Error en la conexión de la base de datos");
00366           }
00367       }
00368       /// <summary>
00369       /// Obtiene la duración de un archivo MP3 en segundos utilizando la biblioteca TagLib#.
00370       /// </summary>
00371       /// <remarks>
00372       /// Accede a las propiedades del archivo en disco para extraer su duración total. Si el archivo no existe
00373       /// o se produce una excepción al intentar leer los metadatos de audio, el error se captura y el método
00374       /// devuelve 0 segundos para no interrumpir el flujo.
00375       /// </remarks>
00376       /// <param name="rutaArchivo">La ruta completa del archivo de audio local.</param>
00377       /// <returns>La duración total en segundos.</returns>
00378       private int ObtenerDuracionMp3(string rutaArchivo)
00379       {
00380           try
00381           {
00382               if (System.IO.File.Exists(rutaArchivo))
00383               {
00384                   var archivo = TagLib.File.Create(rutaArchivo);
00385                   return (int)archivo.Properties.Duration.TotalSeconds;
00386               }
00387           }
00388           catch (Exception ex)
00389           {
00390               System.Diagnostics.Debug.WriteLine("Error leyendo duración: " + ex.Message);
00391           }
00392           return 0;
00393       }
00394       /// <summary>
00395       /// Realiza el proceso completo de publicación de una canción, integrando subida de archivos, obtención de datos y
      persistencia.
00396       /// </summary>
00397       /// <remarks>
00398       /// Este método orquesta un flujo complejo dividido en cuatro fases principales:
00399       /// <list type="number">
00400       /// <item><b>Subida de imagen:</b> Sube la portada seleccionada a la nube.</item>
00401       /// <item><b>Gestión de audio:</b> Sube el archivo MP3 o procesa el enlace de YouTube para obtener la
      duración y URL final.</item>
00402       /// <item><b>Creación de modelo:</b> Construye el objeto <see cref="Canciones"/> con autores y géneros
      seleccionados.</item>
00403       /// <item><b>Persistencia:</b> Guarda la canción en la BD y actualiza el contador de canciones del
      usuario.</item>
00404       /// </list>
00405       /// Durante la ejecución, se controla la propiedad <see cref="EstaCargando"/> para feedback visual en la UI.
00406       /// </remarks>
00407       /// <returns>Una tarea que representa la operación de publicación asíncrona.</returns>
00408       private async Task PublicarCancion()
00409       {
00410           EstaCargando = true;
00411
00412           try
00413           {
00414               // Subimos Imagen
00415               string urlImagenNube = await _storageService.SubirImagen(RutaImagen);
00416
00417               string urlAudioFinal = "";
00418               int duracionFinal = 0;
00419
00420               // Subimos Audio / Obtenemos Info
00421               if (EsYoutube)
00422               {
00423                   var infoYoutube = await _audioService.ObtenerMp3(LinkYoutube);
00424
00425                   if (infoYoutube != null)
00426                   {
00427                       urlAudioFinal = LinkYoutube; // Guardamos el enlace original
00428                       duracionFinal = infoYoutube.DuracionSegundos;
00429                   }
00430                   else
00431                   {
00432                       throw new InvalidOperationException("Msg_Error_InfoYoutube");
00433                   }
00434               }
00435               else
00436               {
00437                   urlAudioFinal = await _storageService.SubirCancion(RutaMp3);
00438                   duracionFinal = _duracionCalculada;
00439               }
00440
00441               // Creamos Objeto
00442               var nuevaCancion = new Canciones
00443               {
00444                   Titulo = TxtTitulo,
```

```
00445                 AutoresIds = ListaArtistas.Select(u => u.Id).ToList(),
00446                 ImagenPortadaUrl = urlImagenNube,
00447                 UrlCancion = urlAudioFinal,
00448                 Datos = new DatosCancion
00449                 {
00450                     FechaLanzamiento = DateTime.Now,
00451                     DuracionSegundos = duracionFinal,
00452                     Generos = ListaGenerosSeleccionados.ToList() // Guardamos la lista completa
00453                 }
00454             };
00455
00456             // Guardamos en BD
00457             bool exito = await MongoClientSingleton.Instance.Cliente.PublicarCancion(nuevaCancion);
00458
00459             if (exito)
00460             {
00461                 await
      MongoClientSingleton.Instance.Cliente.IncrementarContadorCancionesUsuario(GlobalData.Instance.UserIdGD, 1);
00462                 EstaCargando = false;
00463
00464                 _dialogoService.MostrarAlerta("Msg_Exito_CancionPublicada");
00465                 _Volver();
00466             }
00467             else
00468             {
00469                 EstaCargando = false;
00470                 _dialogoService.MostrarAlerta("Msg_Error_SubirCancion");
00471             }
00472         }
00473         catch (InvalidOperationException ex)
00474         {
00475             EstaCargando = false;
00476             _dialogoService.MostrarAlerta("Msg_Error_OperacionInvalida");
00477             System.Diagnostics.Debug.WriteLine(ex.Message);
00478         }
00479         catch (Exception ex)
00480         {
00481             EstaCargando = false;
00482             _dialogoService.MostrarAlerta("Msg_Error_Inesperado");
00483             System.Diagnostics.Debug.WriteLine(ex.Message);
00484         }
00485     }
00486 }
00487 }
```

4.137. Referencia del archivo BetaProyecto/ViewModels/ViewSobreNosotrosViewModel.cs

4.138. ViewSobreNosotrosViewModel.cs

Ir a la documentación de este archivo.

```
00001 using ReactiveUI;
00002 using System;
00003 using System.Diagnostics;
00004 using System.Reactive;
00005
00006 namespace BetaProyecto.ViewModels
00007 {
00008
00009     public class ViewSobreNosotrosViewModel : ViewModelBase, INavegable
00010     {
00011         public Action? VolverAtras { get; set; }
00012         public ReactiveCommand<Unit, Unit> btnVolverAtras { get; }
00013         public ReactiveCommand<Unit, Unit> BtnAbrirGitHub { get; }
00014         public ViewSobreNosotrosViewModel()
00015         {
00016             // Configuramos el comando reactive
00017             btnVolverAtras = ReactiveCommand.Create(() =>
00018             {
00019                 Debug.WriteLine("Volviendo desde el Sobre Nosotros...");
00020                 VolverAtras?.Invoke();
00021             });
00022             BtnAbrirGitHub = ReactiveCommand.Create(() =>
00023             {
00024                 try
00025                 {
00026                     // Intentamos abrir la URL en el navegador predeterminado
00027                     Process.Start(new ProcessStartInfo
00028                     {
```

```
00029                     FileName = "https://github.com/GabrielC777/ProyectoIntermodularDAM.git",
00030                     UseShellExecute = true
00031                 });
00032             }
00033             catch (System.Exception ex)
00034             {
00035                 System.Diagnostics.Debug.WriteLine("Error al abrir GitHub: " + ex.Message);
00036             }
00037         });
00038
00039     }
00040 }
00041 }
```

4.139. Referencia del archivo BetaProyecto/ViewModels/ViewUsuariosViewModel.cs

4.140. ViewUsuariosViewModel.cs

Ir a la documentación de este archivo.

```
00001 using Avalonia.Threading;
00002 using BetaProyecto.Models;
00003 using BetaProyecto.Services;
00004 using BetaProyecto.Singleton;
00005 using ReactiveUI;
00006 using System;
00007 using System.Collections.Generic;
00008 using System.Linq;
00009 using System.Reactive;
00010 using System.Threading;
00011 using System.Threading.Tasks;
00012
00013 namespace BetaProyecto.ViewModels
00014 {
00015     public class ViewUsuariosViewModel : ViewModelBase
00016     {
00017         //Servicios
00018         private readonly IDialogoService _dialogoService;
00019
00020
00021         //Dato principal
00022         private string _idUsuarioCargado; // Guardamos el ID
00023
00024         //El usuario completo que como lo actualizamos cada X segundos, lo hacemos reactive
00025         private Usuarios _usuario;
00026         public Usuarios Usuario
00027         {
00028             get => _usuario;
00029             set => this.RaiseAndSetIfChanged(ref _usuario, value);
00030         }
00031
00032         //Bidings
00033         private List<Canciones> _cancionesSubidas;
00034         public List<Canciones> CancionesSubidas
00035         {
00036             get => _cancionesSubidas;
00037             set => this.RaiseAndSetIfChanged(ref _cancionesSubidas, value);
00038         }
00039
00040         private List<ListaPersonalizada> _playlistsCreadas;
00041         public List<ListaPersonalizada> PlaylistsCreadas
00042         {
00043             get => _playlistsCreadas;
00044             set => this.RaiseAndSetIfChanged(ref _playlistsCreadas, value);
00045         }
00046
00047         // --- TEMPORIZADOR DE ACTUALIZACIÓN ---
00048
00049         private string _txtMensajeTimer = "VisorUser_Timer_Iniciando";
00050         public string TxtMensajeTimer
00051         {
00052             get => _txtMensajeTimer;
00053             set => this.RaiseAndSetIfChanged(ref _txtMensajeTimer, value);
00054         }
00055
00056         // 2. Variable numérica (Ej: " 3s")
00057         private string _txtVariableTimer = "";
00058         public string TxtVariableTimer
00059         {
00060             get => _txtVariableTimer;
00061             set => this.RaiseAndSetIfChanged(ref _txtVariableTimer, value);
```

```
00062         }
00063
00064         private bool _esSeguido;
00065         public bool EsSeguido
00066         {
00067             get => _esSeguido;
00068             set
00069             {
00070                 this.RaiseAndSetIfChanged(ref _esSeguido, value);
00071                 // Actualizamos el texto (CLAVE) y color automáticamente
00072                 TextoBotonSeguir = value ? "VisorUser_Btn_DejarSeguir" : "VisorUser_Btn_Seguir";
00073                 ColorBotonSeguir = value ? "#D32F2F" : "#4939DC"; // Rojo si sigues, Azul si no
00074             }
00075         }
00076
00077         private string _textoBotonSeguir = "VisorUser_Btn_Seguir";
00078         public string TextoBotonSeguir
00079         {
00080             get => _textoBotonSeguir;
00081             set => this.RaiseAndSetIfChanged(ref _textoBotonSeguir, value);
00082         }
00083
00084         private string _colorBotonSeguir = "#4939DC";
00085         public string ColorBotonSeguir
00086         {
00087             get => _colorBotonSeguir;
00088             set => this.RaiseAndSetIfChanged(ref _colorBotonSeguir, value);
00089         }
00090
00091         // Propiedades formateas
00092         public string FechaNacimientoFormateada =>
00093                 _usuario?.Perfil?.FechaNacimiento.ToString("dd MMMM yyyy") ?? "";
00094         public int CantidadCanciones =>
00095             _usuario?.Estadisticas?.NumCancionesSubidas ?? 0;
00096
00097         //Comandos Reactive
00098         public ReactiveCommand<Unit, Unit> BtnVolver { get; }
00099         public ReactiveCommand<Unit, Unit> BtnSeguir { get; }
00100
00101         //Control de hilos
00102         private CancellationTokenSource _cancelToken;
00103
00104         //Constructor
00105         public ViewUsuariosViewModel(string idUsuario, Action accionVolver)
00106         {
00107             //Inicializamos servicio
00108             _dialogoService = new DialogoService();
00109
00110             _idUsuarioCargado = idUsuario;
00111
00112             // Inicializamos listas vacías para evitar errores null en el XAML al arrancar
00113             _cancionesSubidas = new List<Canciones>();
00114             _playlistsCreadas = new List<ListaPersonalizada>();
00115
00116             // Configurar comandos reactive
00117             BtnVolver = ReactiveCommand.Create(() =>
00118             {
00119                 _cancelToken.Cancel(); // Matamos el hilo al salir
00120                 accionVolver();
00121             });
00122             BtnSeguir = ReactiveCommand.CreateFromTask(AlterarSeguimiento);
00123
00124             // Arrancar el Hilo de PSP
00125             _cancelToken = new CancellationTokenSource();
00126             IniciarHiloActualizacion(_cancelToken.Token);
00127             ActualizarBtnSeguir();
00128         }
00129         /// <summary>
00130         /// Cambia el estado de seguimiento del usuario cargado actualmente para el usuario activo. Si el usuario activo ya
    está
00131         /// siguiendo al usuario cargado, este método dejará de seguir; de lo contrario, iniciará un seguimiento.
00132         /// </summary>
00133         /// <remarks>Si el usuario activo intenta seguirse a sí mismo, se muestra una alerta y no hay acción
00134         /// se toma. El método actualiza tanto el estado de seguimiento como la lista local de seguidores al éxito. </remarks>
00135         /// <returns>Devuelve una tarea que representa la operación asíncrona. </returns>
00136         private async Task AlterarSeguimiento()
00137         {
00138             string miId = GlobalData.Instance.UserIdGD;
00139
00140             if (_idUsuarioCargado == miId)
00141             {
00142                 // "No puedes seguirte a ti mismo"
00143                 _dialogoService.MostrarAlerta("Msg_Error_SeguirseMismo");
00144                 return;
00145             }
00146
00147             bool exito;
```

```
00148
00149           if (EsSeguido)
00150           {
00151               exito = await MongoClientSingleton.Instance.Cliente.DejarDeSeguirUsuario(miId, _idUsuarioCargado);
00152               if (exito)
00153               {
00154                   EsSeguido = false;
00155                   // Actualizamos la lista local
00156                   Usuario.Listas.Seguidores?.Remove(_idUsuarioCargado);
00157               }
00158           }
00159           else
00160           {
00161               exito = await MongoClientSingleton.Instance.Cliente.SeguirUsuario(miId, _idUsuarioCargado);
00162               if (exito)
00163               {
00164                   EsSeguido = true;
00165                   // Actualizamos la lista local
00166                   Usuario.Listas.Seguidores?.Add(_idUsuarioCargado);
00167               }
00168           }
00169       }
00170       /// <summary>
00171       /// Actualiza el indicador de estado de seguimiento según si el usuario cargado está presente en los seguidores globales
00172       /// lista.
00173       /// </summary>
00174       /// <remarks>Este método establece el valor de la propiedad EsSeguido para reflejar si el actualmente
00175       /// el usuario cargado está siendo seguido. Debe llamarse cada vez que la lista de seguidores o el usuario cargado
      cambie a
00176       /// asegúrese de que el estado de seguimiento siga siendo preciso. </remarks>
00177       private void ActualizarBtnSeguir()
00178       {
00179           List<string> lista = GlobalData.Instance.SeguidoresGD;
00180
00181           if (lista != null && lista.Contains(_idUsuarioCargado))
00182           {
00183               EsSeguido = true;
00184           }
00185           else
00186           {
00187               EsSeguido = false;
00188           }
00189       }
00190       /// <summary>
00191       /// Inicia un ciclo de actualización en segundo plano que actualiza periódicamente el usuario y las listas relacionadas
      hasta que se cancele
00192       /// solicitado.
00193       /// </summary>
00194       /// <remarks>El bucle de actualización se ejecuta de forma asíncrona y actualiza los elementos de la interfaz de
      usuario para reflejar el estado actual.
00195       /// estado de actualización. El método no bloquea el hilo de llamada. Para detener el proceso de actualización, indica
00196       /// cancelación a través del token proporcionado. </remarks>
00197       /// <param name="token">Un token de cancelación que se puede usar para solicitar la finalización del ciclo de
      actualización. </param>
00198       private void IniciarHiloActualizacion(CancellationToken token)
00199       {
00200           Task.Run(async () =>
00201           {
00202               try
00203               {
00204                   //Carga inicial
00205                   await CargarUsuario();
00206
00207                   // Si encontramos al usuario, cargamos sus listas
00208                   if (_usuario != null)
00209                   {
00210                       await CargarListasDetalladas(_usuario.Id);
00211
00212                   }
00213
00214                   //Bucle de actualización cada 5 segundos
00215                   while (!token.IsCancellationRequested)
00216                   {
00217                       for (int i = 5; i > 0; i--)
00218                       {
00219                           await Dispatcher.UIThread.InvokeAsync(() =>
00220                           {
00221                               TxtMensajeTimer = "VisorUser_Timer_Refrescando";
00222                               TxtVariableTimer = $" {i}s";
00223                           });
00224                           await Task.Delay(1000, token);
00225                       }
00226
00227                       await Dispatcher.UIThread.InvokeAsync(() =>
00228                       {
00229                           TxtMensajeTimer = "VisorUser_Timer_Consultando";
00230                           TxtVariableTimer = "";
```

```
00231                     });
00232
00233                     // Refrescamos usuario y listas
00234                     await CargarUsuario();
00235                     if (_usuario != null)
00236                     {
00237                         // Actualizamos la UI visualmente
00238                         await Dispatcher.UIThread.InvokeAsync(() =>
00239                         {
00240                             TxtMensajeTimer = "VisorUser_Timer_Actualizado";
00241                             TxtVariableTimer = "";
00242                         });
00243                     }
00244                 }
00245             }
00246             catch (TaskCanceledException) { }
00247         });
00248     }
00249     /// <summary>
00250     /// Carga asincrónicamente los datos de usuario del identificador de usuario seleccionado actualmente y actualiza el relacionado
00251     /// propiedades.
00252     /// </summary>
00253     /// <remarks>Este método recupera la información del usuario de la fuente de datos basada en el usuario actual
00254     /// identificador y actualiza el usuario vinculado y las propiedades calculadas relacionadas en el hilo de la interfaz. Destinado a
00255     /// uso interno dentro del modelo de vista para asegurar la consistencia de la interfaz de usuario después de cambios en los datos del usuario. </remarks>
00256     /// <returns>Devuelve una tarea que representa la operación de carga asíncrona. </returns>
00257     private async Task CargarUsuario()
00258     {
00259         var listaUnId = new List<string> { _idUsuarioCargado };
00260
00261         var resultados = await MongoClientSingleton.Instance.Cliente.ObtenerUsuariosPorListaIds(listaUnId);
00262
00263         var usuarioEncontrado = resultados.FirstOrDefault();
00264
00265         if (usuarioEncontrado != null)
00266         {
00267             await Dispatcher.UIThread.InvokeAsync(() =>
00268             {
00269                 Usuario = usuarioEncontrado;
00270                 // Forzamos actualización de las propiedades calculadas
00271                 this.RaisePropertyChanged(nameof(FechaNacimientoFormateada));
00272                 this.RaisePropertyChanged(nameof(CantidadCanciones));
00273             });
00274         }
00275     }
00276     /// <summary>
00277     /// Carga de forma asíncrona las listas detalladas de canciones y listas de reproducción creadas por el usuario especificado y actualiza el
00278     /// propiedades correspondientes en el hilo de la interfaz.
00279     /// </summary>
00280     /// <remarks>Este método recupera las canciones y listas de reproducción del usuario desde la fuente de datos y las actualizaciones
00281     /// las propiedades vinculadas a la interfaz de usuario. Las actualizaciones se envían al hilo de la interfaz para garantizar la seguridad del hilo al modificarlo
00282     /// elementos de la interfaz de usuario. </remarks>
00283     /// <param name="idUser">El identificador único del usuario cuyas canciones cargadas y listas de reproducción creadas se deben cargar. No puede ser
00284     /// nulo. </param>
00285     /// <returns>Devuelve una tarea que representa la operación de carga asíncrona. </returns>
00286     private async Task CargarListasDetalladas(string idUser)
00287     {
00288         // Buscamos las lista que nos interesan
00289         var canciones = await MongoClientSingleton.Instance.Cliente.ObtenerCancionesPorAutor(idUser);
00290         var playlists = await MongoClientSingleton.Instance.Cliente.ObtenerPlaylistsPorCreador(idUser);
00291
00292         // Las mandamos al hilo principal para que las actualize
00293         await Dispatcher.UIThread.InvokeAsync(() =>
00294         {
00295             CancionesSubidas = canciones;
00296             PlaylistsCreadas = playlists;
00297         });
00298     }
00299 }
00300 }
```

Índice alfabético