



Universidad de Jaén

Escuela Politécnica Superior de Jaén

Desarrollo de un gestor de contenido multimedia con reproductor de vídeo integrado

Autor: Juan Llaveró Company

Grado: Ingeniería Informática

Director: Prof. D. José Ignacio Gómez Espínola
Departamento del director: Informática

Fecha: 19/07/2024

Licencia CC

CREA



UNIVERSIDAD DE JAÉN

Prof. D. José Ignacio Gómez Espínola, tutor del Trabajo Fin de Grado titulado: **Desarrollo de un gestor de contenido multimedia con reproductor de vídeo integrado**, que presenta Juan Llavero Company, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, julio de 2024

El estudiante

Firmado por LLAVERO
COMPANY JUAN - ***2723**
el día 19/07/2024 con un
certificado emitido por
AC FNMT Usuarios

Juan Llavero Company

El tutor

GOMEZ
ESPINOLA
JOSE
IGNACIO -
24277832K

Firmado
digitalmente por
GOMEZ ESPINOLA
JOSE IGNACIO -
24277832K
Fecha: 2024.07.19
21:26:39 +02'00'

Prof. D. José Ignacio Gómez Espínola

Agradecimientos

Quisiera comenzar agradeciendo a mi familia, por su constante e incondicional apoyo desde que comencé este grado. Su paciencia, comprensión y motivación han sido fundamentales en cada paso de este camino.

También agradecer a mi tutor, José Ignacio Gómez Espínola, cuya guía y orientación han sido esenciales en la realización de este Trabajo de Fin de Grado.

Por último, también quisiera expresar mi gratitud a mis amigos: Miguel Ángel, Ismael y Daniel. Su apoyo constante y su ánimo inquebrantable me han acompañado a lo largo del grado. Sin ellos, no habría presentado la idea de este Trabajo de Fin de Grado.

Muchas gracias a todos.

Tabla de contenidos

1. INTRODUCCIÓN	1
1.1. Introducción al proyecto	1
1.2. Motivación	2
1.3. Objetivos	2
2. ESTADO DEL ARTE	3
2.1. Aplicaciones de escritorio existentes	3
2.1.1. Plex	4
2.1.1.1. Características	4
2.1.1.2. Ventajas	4
2.1.1.3. Desventajas	4
2.1.1.4. Funciones relevantes	5
2.1.2. Kodi	5
2.1.2.1. Características	6
2.1.2.2. Ventajas	6
2.1.2.3. Desventajas	6
2.1.2.4. Funciones relevantes	7
2.1.3. Jellyfin	7

2.1.3.1. Características	7
2.1.3.2. Ventajas	8
2.1.3.3. Desventajas	8
2.1.3.4. Funciones relevantes	8
2.1.4. Funciones relevantes para el proyecto	9
2.2. Frameworks de desarrollo más utilizados	10
2.2.1. .NET	10
2.2.1.1. Características	10
2.2.1.2. Ventajas	10
2.2.1.3. Desventajas	11
2.2.2. JavaFX	11
2.2.2.1. Características	11
2.2.2.2. Ventajas	11
2.2.2.3. Desventajas	12
2.2.3. Electron	12
2.2.3.1. Características	12
2.2.3.2. Ventajas	12
2.2.3.3. Desventajas	13
2.2.4. Flutter	13
2.2.4.1. Características	13
2.2.4.2. Ventajas	13
2.2.4.3. Desventajas	14

3. TECNOLOGÍAS UTILIZADAS

15

3.1. Framework utilizado	15
3.2. Entorno de desarrollo	16
3.3. Gestión de dependencias	16
3.4. Control de versiones	17
4. INGENIERÍA DEL SOFTWARE	19
4.1. Introducción	19
4.2. Definición de requisitos	19
4.2.1. Requisitos funcionales	19
4.2.2. Requisitos no funcionales	21
4.3. Planificación	22
4.3.1. Metodologías tradicionales	22
4.3.2. Metodologías ágiles	22
4.3.3. Metodología escogida	22
4.4. Análisis	24
4.4.1. Modo escritorio	24
4.4.2. Modo pantalla completa	25
4.4.3. Reproductor de vídeo	25
4.4.4. Diagrama de clases	26
4.5. Diseño	28
4.5.1. Arquitectura del sistema	28
4.5.2. Patrones de diseño	29
4.5.2.1. Patrón Singleton	29
4.5.2.2. Patrón Observador	29

4.5.3. Diagramas de casos de uso	30
4.5.3.1. Diagrama de casos de uso general	30
4.5.3.2. Diagrama de casos de uso: crear biblioteca	31
4.5.3.3. Diagrama de casos de uso: editar biblioteca	31
4.5.3.4. Diagrama de casos de uso: editar serie	32
4.5.3.5. Diagrama de casos de uso: editar temporada	33
4.5.3.6. Diagrama de casos de uso: editar episodio	33
4.5.3.7. Diagrama de casos de uso: eliminar biblioteca	34
4.5.3.8. Diagrama de casos de uso: eliminar serie	34
4.5.3.9. Diagrama de casos de uso: eliminar temporada	35
4.5.3.10. Diagrama de casos de uso: eliminar episodio	35
4.5.3.11. Diagrama de casos de uso: cambiar grupo de episodios	36
4.5.3.12. Diagrama de casos de uso: corregir identificación	36
4.5.4. Diagramas de secuencia	37
4.5.4.1. Diagrama de secuencia: crear biblioteca	37
4.5.4.2. Diagrama de secuencia: corregir identificación	38
4.5.4.3. Diagrama de secuencia: editar serie	39
4.5.5. Almacenamiento de datos	40
4.5.6. Interfaz de usuario	41
4.5.6.1. Wireframes	41
4.6. Implementación	45
4.6.1. Internacionalización	45
4.6.1.1. Uso de ResourceBundle	45
4.6.1.2. Proceso de Localización	47

4.6.2. Clases utilizadas	47
4.6.3. Modo escritorio	50
4.6.3.1. Ejecución concurrente	50
4.6.3.2. Análisis de archivos. Biblioteca tipo 'Series'	52
4.6.3.3. Análisis de archivos. Biblioteca tipo 'Películas'	57
4.6.3.4. Corrección de identificación para series	61
4.6.3.5. Corrección de identificación para películas	62
4.6.4. Modo pantalla completa	65
4.6.4.1. Movimiento de scroll para series	65
4.6.4.2. Movimiento de scroll para episodios	67
4.6.4.3. Reproducción de música y vídeo de fondo	68
4.6.5. Reproductor de vídeo	71
4.6.5.1. Reproductores compatibles con JavaFX	71
4.6.5.2. Reproductor MPV	72
4.6.5.3. Implementación de MPV	72
4.6.5.4. Patrón observador	74
4.7. Pruebas	75
4.7.1. Introducción	75
4.7.2. Ejemplos de pruebas realizadas	75
4.7.2.1. Prueba de corrección de identificación: series	76
4.7.2.2. Prueba de corrección de identificación: películas	80
4.7.2.3. Prueba de análisis de archivos: series	83
4.7.2.4. Prueba de análisis de archivos: películas	84

4.7.2.5. Prueba de preferencia de idioma en pistas de audio y subtítulos	87
4.7.2.6. Prueba de eliminación de contenido	92
4.7.2.7. Prueba de movimiento de scroll	95
5. CONCLUSIÓN	99
5.1. Conclusiones	99
5.2. Posibles desarrollos futuros	100
6. ANEXOS	101
6.1. Dependencias necesarias para ejecutar la aplicación	101
6.1.1. Windows	101
6.1.2. Linux	102
6.2. Estructura de carpetas y nombres	102
6.2.1. Series	102
6.2.2. Películas	103
Bibliografía	III

Lista de figuras

2.1. Logo de Plex	4
2.2. Logo de Kodi	5
2.3. Logo de Jellyfin	7
2.4. Logo de Microsoft .NET	10
2.5. Logo de JavaFX	11
2.6. Logo de Electron	12
2.7. Logo de Flutter	13
3.1. Logo de JavaFX	15
3.2. Logo de IntelliJ IDEA	16
3.3. Logo de Maven	16
3.4. Logo de Github	17
4.1. Logo de Trello	23
4.2. Diagrama de clases simplificado: Modo Escritorio	26
4.3. Diagrama de clases simplificado: Modo Pantalla Completa	27
4.4. Clases que no forman parte del diagrama de clases simplificado	27
4.5. Patrón Modelo-Vista-Controlador (MVC)	28
4.6. Diagrama de casos de uso del sistema	30

4.7. Diagrama de casos de uso: crear biblioteca	31
4.8. Diagrama de casos de uso: editar biblioteca	31
4.9. Diagrama de casos de uso: editar serie	32
4.10. Diagrama de casos de uso: editar temporada	33
4.11. Diagrama de casos de uso: editar episodio	33
4.12. Diagrama de casos de uso: eliminar biblioteca	34
4.13. Diagrama de casos de uso: eliminar serie	34
4.14. Diagrama de casos de uso: eliminar temporada	35
4.15. Diagrama de casos de uso: eliminar episodio	35
4.16. Diagrama de casos de uso: cambiar grupo de episodios	36
4.17. Diagrama de casos de uso: corregir identificación	36
4.18. Diagrama de Secuencia: Crear Biblioteca	37
4.19. Diagrama de Secuencia: Corregir Identificación	38
4.20. Diagrama de Secuencia: Editar Serie	39
4.21. Wireframe de la vista principal del modo escritorio	41
4.22. Wireframe de la vista principal del modo pantalla completa	42
4.23. Wireframe de la vista de temporada en el modo pantalla completa	43
4.24. Wireframe del reproductor de vídeo	43
4.25. Logo de VLC	71
4.26. Logo de MPV	72
4.27. Test de corrección de identificación de series: 1	76
4.28. Test de corrección de identificación de series: 2	76
4.29. Test de corrección de identificación de series: 3	77
4.30. Test de corrección de identificación de series: 4	77

4.31. Test de corrección de identificación de series: 5	78
4.32. Test de corrección de identificación de series: 6	79
4.33. Test de corrección de identificación de series: 7	79
4.34. Test de corrección de identificación de series: 8	80
4.35. Test de corrección de identificación de películas: 1	80
4.36. Test de corrección de identificación de películas: 2	81
4.37. Test de corrección de identificación de películas: 3	81
4.38. Test de corrección de identificación de películas: 4	82
4.39. Test de corrección de identificación de películas: 5	82
4.40. Test de análisis de archivos para series: 1	83
4.41. Test de análisis de archivos para series: 2	83
4.42. Test de análisis de archivos para series: 3	84
4.43. Test de análisis de archivos para películas: 1	85
4.44. Test de análisis de archivos para películas: 2	85
4.45. Test de análisis de archivos para películas: 3	86
4.46. Test de análisis de archivos para películas: 4	86
4.47. Test de preferencia de idioma: 1	87
4.48. Test de preferencia de idioma: 2	88
4.49. Test de preferencia de idioma: 3	88
4.50. Test de preferencia de idioma: 4	89
4.51. Test de preferencia de idioma: 5	89
4.52. Test de preferencia de idioma: 6	90
4.53. Test de preferencia de idioma: 7	90
4.54. Test de preferencia de idioma: 8	91

4.55. Test de preferencia de idioma: 9	91
4.56. Test de eliminación de contenido: 1	93
4.57. Test de eliminación de contenido: 2	93
4.58. Test de eliminación de contenido: 3	94
4.59. Test de eliminación de contenido: 4	94
4.60. Test de eliminación de contenido: 5	94
4.61. Test de movimiento de scroll: 1	96
4.62. Test de movimiento de scroll: 2	96
4.63. Test de movimiento de scroll: 3	97
4.64. Test de movimiento de scroll: 4	97
4.65. Test de movimiento de scroll: 5	98

Lista de listados de código

4.1. Ejemplo de contenido localizado	46
4.2. Internacionalización en código	46
4.3. Ejemplo de localización de un botón	47
4.4. Código para crear una tarea en JavaFX	50
4.5. Código para crear un hilo	50
4.6. Función que se encarga de cerrar la aplicación	51
4.7. Análisis de archivos tipo 'Series': obtención de archivos	52
4.8. Análisis de archivos tipo 'Series': creación o edición de serie	53
4.9. Análisis de archivos tipo 'Series': descarga de metadatos para temporadas y grupos de episodios	54
4.10. Análisis de archivos tipo 'Series': procesamiento de episodios	55
4.11. Función de análisis de archivos para películas	57
4.12. Función de análisis de archivos para películas: código para crear la película	58
4.13. Función para corregir identificación de serie	61
4.14. Función para corregir identificación de película	62
4.15. Fragmento de código que controla el movimiento del scroll al seleccionar una serie	65
4.16. Fragmento de código que controla el foco de los botones de series	66
4.17. Función que controla el foco de un botón de episodio	67
4.18. Función que inicializa el reproductor multimedia nativo	68
4.19. Función que busca una canción dentro de las temporadas de una serie	69
4.20. Función que asigna el contenido al reproductor nativo	69
4.21. Función que reproduce la música o vídeo de fondo de una temporada	70
4.22. Interfaz MPV	72
4.23. Fragmento de código que crea el reproductor MPV	73
4.24. Funciones para controlar el reproductor MPV	73
4.25. Funciones para controlar el reproductor MPV	74
4.26. Fragmento de código con error: corregir identificación de series	78
4.27. Solución al problema de preferencia de idiomas	92

4.28. Solución al problema del caché de imágenes descargadas	95
6.1. Estructura de carpetas Serie	102
6.2. Ejemplo de una carpeta de una serie	102
6.3. Formatos del nombre de un episodio	103
6.4. Estructura de carpetas Película	103
6.5. Ejemplos de películas	104

Capítulo 1

INTRODUCCIÓN

1.1. Introducción al proyecto

Este proyecto tiene como objetivo desarrollar una aplicación completamente funcional para la gestión y visualización de contenido multimedia en formato de vídeo. La solución se compone de dos ejecutables diferenciados, cada uno con funcionalidades y enfoques específicos:

- **Gestor de Contenido Multimedia:** el primer ejecutable es una aplicación de escritorio que permite a los usuarios organizar y gestionar su colección de vídeos. Esta aplicación ofrece una interfaz intuitiva para la creación y edición de colecciones de películas y series, la asignación de metadatos mediante la API de TheMovieDB [1], y la personalización de la presentación del contenido con imágenes, música y vídeos de fondo.
- **Interfaz de Visionado:** el segundo ejecutable es una versión adaptada de la misma aplicación, diseñada exclusivamente para la visualización del contenido multimedia. Esta versión proporciona una interfaz optimizada para el visionado, similar a las aplicaciones de TV, que permite a los usuarios disfrutar de sus vídeos en modo pantalla completa, sin opciones de gestión de contenido. Incorpora un reproductor de vídeo personalizado que mantiene el estado de visionado de cada vídeo, incluyendo el último minuto reproducido y la indicación de episodios vistos o no vistos.

1.2. Motivación

Soy usuario de Plex desde hace años y la aplicación ha cumplido siempre mis expectativas. Sin embargo, últimamente ha empezado a dar errores de reproducción que parecen aleatorios. Esto, sumado a que su plan gratuito ha dejado de incluir la posibilidad de reproducir el contenido fuera de la red local, y a que las opciones de compresión gratuitas son muy limitadas y pobres, ha hecho que pensara en utilizar alguna alternativa a este servicio.

Tras analizar las alternativas, he concluido que el apartado estético de estas es algo carente en comparación con el de Plex, y aunque las funcionalidades son similares, ninguna me ha llegado a convencer. Por todo esto he decidido crear un software que unifique funcionalidades de varias aplicaciones existentes, además de incluir algunas que no se encuentran en estas. Huelga decir que todas estas aplicaciones tienen un gran número de funcionalidades interesantes, por lo que mi papel ha sido el de seleccionar las más útiles para mi uso personal, así como el de personalizar o añadir nuevas funciones.

1.3. Objetivos

- Desarrollar una aplicación de escritorio multiplataforma en JavaFX que ofrezca una experiencia de usuario intuitiva y comparable con soluciones comerciales de gestión de contenido multimedia.
- Minimizar los posibles errores que puedan suceder en la ejecución de la misma.
- Realizar un análisis exhaustivo de las posibles mejoras y expansiones futuras de la aplicación, identificando nuevas funcionalidades y optimizaciones que puedan ser implementadas.
- Garantizar la capacidad de la aplicación para manejar colecciones de películas y series de manera eficiente, incluyendo la correcta identificación y asignación de metadatos.
- Facilitar la personalización de la experiencia del usuario mediante la implementación de características como la adición de música y vídeos de fondo.

Capítulo 2

ESTADO DEL ARTE

2.1. Aplicaciones de escritorio existentes

En esta sección haremos un análisis de las aplicaciones de escritorio que más se asemejan a este proyecto. Cabe recalcar que el análisis se realizará sobre las funciones en local de estas aplicaciones, pues todas ellas cuentan con la capacidad de transmitir el contenido a otros dispositivos; sin embargo, esta función no será implementada, ya que la aplicación a desarrollar realizará la reproducción de vídeo en local.

El motivo por el que analizaremos estas aplicaciones en vez de otras que sólo cuenten con reproducción local se debe a que las aplicaciones puramente locales son reproductores de vídeo, no gestores de contenido. Aunque dichos reproductores de vídeo pueden incluir la funcionalidad de gestionar el contenido local, no obtienen los metadatos de internet, ni se centran en realizar una gestión del contenido enfocada al diseño como lo hacen las que comentaremos; sino de una manera más estrictamente funcional.

2.1.1. Plex



Figura 2.1: Logo de Plex

Plex [2] es un gestor de contenido que consta de un servidor y varios clientes. El servidor es el encargado de analizar el contenido multimedia local y enviarlo a otros dispositivos para su posterior reproducción.

2.1.1.1. Características

- Soporte para múltiples formatos de medios.
- Transcodificación en tiempo real para adaptarse a la capacidad del dispositivo receptor.
- Acceso remoto y sincronización de contenido entre dispositivos.
- Integración con servicios de streaming y complementos para ampliar funcionalidades.

2.1.1.2. Ventajas

- Interfaz de usuario intuitiva y moderna.
- Amplias opciones de personalización y control parental.
- Comunidad activa y soporte robusto.
- Funcionalidades adicionales como DVR y Live TV.

2.1.1.3. Desventajas

- Algunas funciones avanzadas requieren una suscripción de pago (Plex Pass).
- Dependencia de servidores en la nube para algunas funcionalidades, lo que puede plantear preocupaciones de privacidad.

2.1.1.4. Funciones relevantes

Plex realiza un análisis de los archivos locales para identificar películas y series y, posteriormente, descargar y asignar los metadatos e imágenes a estas. Para descargar los metadatos hace uso de su propia base de datos, en conjunción con las bases de datos de **TheMovieDB** [1] y **TheTVDB** [3]. También cabe destacar que incorpora un reproductor multimedia para permitir la visualización de dicho contenido.

Plex se centra en presentar una aplicación cuidada y profesional, contando con un modo de suscripción que permite acceder a ciertas características. Estas características son la transcodificación y compresión de archivos multimedia para una mayor accesibilidad por condiciones de bajo ancho de banda, o incompatibilidad con el formato de audio, vídeo o subtítulos. También incluye la capacidad de descargar el contenido en otros dispositivos para permitir el visionado sin conexión, entre otras funciones.

2.1.2. Kodi



Figura 2.2: Logo de Kodi

Kodi [4] es una plataforma de código abierto para la gestión y reproducción de contenido multimedia en diferentes dispositivos. Esta aplicación tiene una interfaz básica y desactualizada, pero permite el uso de addons o complementos para que los usuarios la personalicen a su gusto.

Kodi es muy similar a Plex en lo que a funcionalidad se refiere. Sin embargo, esta aplicación es de código abierto, por lo que depende de complementos externos para mejorar su aspecto y ciertas funcionalidades. Aun así, cuenta con una amplia gama de funcionalidades avanzadas de base, como la transmisión de contenido a través de protocolos como UPnP, AirPlay, DLNA y SMB.

2.1.2.1. Características

- Soporte para una gran variedad de formatos multimedia y protocolos de transmisión.
- Interfaz altamente personalizable con múltiples skins disponibles.
- Gran biblioteca de complementos que amplían sus funcionalidades.
- Compatibilidad multiplataforma, incluyendo Windows, macOS, Linux, Android y iOS.
- Capacidad de integrar servicios de streaming mediante complementos.

2.1.2.2. Ventajas

- Gratuito y de código abierto, promoviendo la transparencia y la innovación.
- Extensible mediante complementos que permiten añadir numerosas funcionalidades adicionales.
- Fuerte soporte de la comunidad, con actualizaciones y nuevos complementos desarrollados regularmente.
- Alta flexibilidad en la personalización de la interfaz y las funcionalidades.

2.1.2.3. Desventajas

- Curva de aprendizaje pronunciada para usuarios no técnicos.
- Algunos complementos pueden ser inestables o no oficiales, lo que plantea riesgos de seguridad.
- Requiere configuración inicial y mantenimiento regular.

2.1.2.4. Funciones relevantes

Kodi es capaz de escanear y organizar archivos multimedia locales, proporcionando una biblioteca bien estructurada con metadatos descargados automáticamente. Su reproductor multimedia incorporado permite la reproducción sin conexión de contenido local. Además, Kodi ofrece amplias opciones de personalización de la interfaz y soporta la instalación de complementos que pueden añadir funcionalidades específicas según las necesidades del usuario. Esto lo convierte en una herramienta flexible para gestionar y reproducir contenido multimedia local.

2.1.3. Jellyfin



Figura 2.3: Logo de Jellyfin

Jellyfin [5] es una solución de servidor multimedia gratuita y de código abierto que se centra en la privacidad del usuario y la gestión local de contenido.

2.1.3.1. Características

- Soporte para una amplia variedad de formatos de vídeo y audio.
- Transcodificación en tiempo real y acceso remoto.
- Interfaz web para administración y reproducción.
- Organización automatizada de bibliotecas de medios y descarga de metadatos.
- Compatibilidad multiplataforma, incluyendo Windows, macOS, Linux y más.

2.1.3.2. Ventajas

- Completamente gratuito y de código abierto, sin características bloqueadas detrás de un muro de pago.
- Enfoque en la privacidad del usuario, ya que no depende de servicios en la nube.
- Comunidad activa que contribuye con mejoras y nuevos complementos.
- Amplias capacidades de personalización a través de plugins y ajustes.

2.1.3.3. Desventajas

- Menor integración con servicios de streaming en comparación con alternativas comerciales.
- Requiere mayor configuración inicial y mantenimiento para aprovechar todas sus funcionalidades.
- La interfaz, aunque funcional, puede no ser tan pulida como la de competidores comerciales.

2.1.3.4. Funciones relevantes

Jellyfin escanea y organiza automáticamente las bibliotecas locales, proporcionando una interfaz accesible para la gestión y reproducción de contenido multimedia. Permite la reproducción de medios locales sin conexión y ofrece opciones de personalización a través de plugins. Estas características, junto con su enfoque en la privacidad y su naturaleza de código abierto, hacen de Jellyfin una opción atractiva para proyectos que requieren una gestión eficiente del contenido multimedia local.

2.1.4. Funciones relevantes para el proyecto

Las aplicaciones analizadas anteriormente tienen una amplia gama de funcionalidades, entre las cuales destacan para el proyecto la capacidad de analizar el contenido multimedia del usuario y permitir su reproducción de manera local. Para cumplir con este cometido, existen otras aplicaciones más simples que no requieren de conexión a internet, como reproductores de vídeo. Estos suelen incluir la funcionalidad de gestionar el contenido multimedia, aunque se limitan a utilizar los metadatos que contienen los propios archivos, en lugar de acceder a bases de datos de contenido multimedia como sí lo hacen las aplicaciones antes mencionadas.

Resumiendo las funciones más relevantes, obtenemos la siguiente lista:

- Analizar el contenido local, detectando a qué película/serie pertenece cada vídeo.
- Descargar los metadatos del contenido detectado, utilizando la API de alguna base de datos de contenido multimedia.
- Organizar el contenido y mostrarlo en forma de aplicación que permite la visualización del mismo.
- Permitir la reproducción del contenido por medio de un reproductor de contenido multimedia.
- Permitir la edición manual de los metadatos del contenido del usuario.

2.2. Frameworks de desarrollo más utilizados

En la actualidad, existen múltiples frameworks que permiten el desarrollo de aplicaciones de escritorio. En este apartado veremos algunos de los más relevantes para este tipo de proyecto.

2.2.1. .NET



Figura 2.4: Logo de Microsoft .NET

El framework .NET [6], desarrollado por Microsoft, es una plataforma muy utilizada para el desarrollo de aplicaciones de escritorio y web. Destaca por su robustez, flexibilidad y el amplio soporte de herramientas.

2.2.1.1. Características

- Soporta múltiples lenguajes de programación como C#, F# y Visual Basic.
- Ofrece una gran cantidad de bibliotecas y herramientas integradas para el desarrollo de interfaces de usuario, acceso a bases de datos y más.
- Incluye Windows Presentation Foundation (WPF) y Windows Forms para el desarrollo de aplicaciones de escritorio ricas en funcionalidades.

2.2.1.2. Ventajas

- Gran rendimiento y escalabilidad.
- Excelente soporte y documentación.
- Integración nativa con el ecosistema de Windows.

2.2.1.3. Desventajas

- Principalmente orientado a sistemas Windows, aunque .NET Core y .NET 5+ han mejorado la compatibilidad multiplataforma.
- Curva de aprendizaje considerable para aprovechar todas sus capacidades.

2.2.2. JavaFX



Figura 2.5: Logo de JavaFX

JavaFX [7] es una plataforma de desarrollo de aplicaciones rica en funcionalidades y gráficos, que es una evolución de Swing, el framework tradicional de Java para interfaces de usuario.

2.2.2.1. Características

- Permite la creación de aplicaciones con interfaces de usuario modernas y visualmente atractivas.
- Ofrece capacidades de manipulación de medios, gráficos 2D y 3D.
- Se integra bien con el ecosistema Java, beneficiándose de su portabilidad y robustez.

2.2.2.2. Ventajas

- Es multiplataforma, permitiendo desarrollar aplicaciones que corren en Windows, macOS y Linux.
- Amplia comunidad y abundante documentación.
- Fuerte enfoque en la creación de interfaces de usuario dinámicas y multimedia.

2.2.2.3. Desventajas

- Puede ser menos eficiente en términos de rendimiento en comparación con tecnologías más modernas.
- Requiere de conocimiento profundo de Java y del propio framework JavaFX.

2.2.3. Electron



Figura 2.6: Logo de Electron

Electron [8] es un framework desarrollado por GitHub que permite la creación de aplicaciones de escritorio utilizando tecnologías web como HTML, CSS y JavaScript.

2.2.3.1. Características

- Utiliza Chromium y Node.js, permitiendo a los desarrolladores web crear aplicaciones de escritorio.
- Ofrece soporte para funcionalidades de escritorio avanzadas como notificaciones, gestión de archivos y más.
- Es conocido por ser la base de aplicaciones populares como Visual Studio Code, Slack y Discord.

2.2.3.2. Ventajas

- Permite reutilizar conocimientos y código de desarrollo web.
- Multiplataforma, compatible con Windows, macOS y Linux. Gran flexibilidad y rapidez en el desarrollo.

2.2.3.3. Desventajas

- Puede resultar en aplicaciones con mayor consumo de recursos (memoria y CPU).
- El tamaño del paquete de la aplicación tiende a ser mayor en comparación con otras tecnologías.

2.2.4. Flutter



Figura 2.7: Logo de Flutter

Flutter [9] es un framework de desarrollo de aplicaciones de código abierto creado por Google. Este se ha vuelto muy popular para el desarrollo de aplicaciones móviles y de escritorio.

2.2.4.1. Características

- Utiliza Dart como lenguaje de programación.
- Proporciona una rica colección de widgets que permiten crear interfaces de usuario atractivas y personalizables.
- Ofrece soporte nativo para la reproducción de vídeo y otras funcionalidades multimedia.

2.2.4.2. Ventajas

- Alto rendimiento debido a su motor de renderizado nativo.
- Desarrollar una sola base de código para aplicaciones tanto en iOS como en Android, y ahora también para aplicaciones de escritorio.
- Rápido ciclo de desarrollo gracias a la funcionalidad de "hot reload".

2.2.4.3. Desventajas

- Comunidad y recursos disponibles aún en crecimiento comparado con tecnologías más maduras.
- Necesidad de aprender Dart, un lenguaje menos conocido.

Capítulo 3

TECNOLOGÍAS UTILIZADAS

3.1. Framework utilizado



Figura 3.1: Logo de JavaFX

El framework utilizado ha sido JavaFX. La elección ha sido entre los frameworks de desarrollo nativos, como .NET, JavaFX y Flutter; principalmente debido a la prioridad de un rendimiento óptimo en una aplicación que cuenta con reproductor de vídeo y necesita un rendimiento gráfico mínimo.

Cualquier framework de desarrollo nativo sería óptimo para desarrollar una aplicación como esta, por lo que la elección se reduce más bien a la familiaridad que tenga el desarrollador con cada framework. Por tanto, dado que he programado en Java, y en concreto he utilizado el framework JavaFX, esta resulta la opción más óptima.

3.2. Entorno de desarrollo

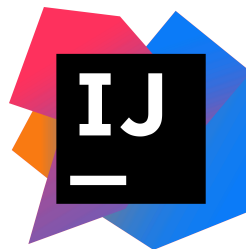


Figura 3.2: Logo de IntelliJ IDEA

El entorno de desarrollo escogido ha sido IntelliJ IDEA [10] de JetBrains. El principal motivo de su elección ha sido el lenguaje de programación, pues este IDE está especialmente enfocado en Java y por tanto incorpora todas las funcionalidades necesarias para desarrollar software en dicho lenguaje.

Dado que el framework escogido ha sido JavaFX y este utiliza archivos FXML para las vistas, era necesario un IDE que permitiera la edición visual y de código de dichos archivos. Además, IntelliJ IDEA cuenta con soporte para SceneBuilder [11], que es una herramienta de diseño visual para JavaFX que permite crear interfaces de usuario sin escribir manualmente el código de las mismas, simplificando así el desarrollo de las vistas FXML.

3.3. Gestión de dependencias



Figura 3.3: Logo de Maven

Para la gestión de dependencias en el desarrollo de la aplicación, se ha optado por utilizar Apache Maven [12]. Esta es una herramienta de gestión y construcción de proyectos que se basa en el concepto de POM (Project Object Model). Este modelo permite gestionar de forma eficiente las dependencias, el ciclo de vida del proyecto y la automatización de tareas de construcción.

Maven simplifica la gestión de dependencias, unificándolas en un archivo, permitiendo el manejo de versiones y automatizando la descarga de las mismas.

3.4. Control de versiones

La herramienta escogida para realizar el control de versiones ha sido Git [13]. Git es un sistema de control de versiones distribuido ampliamente utilizado en proyectos de desarrollo de software.



Figura 3.4: Logo de Github

La gestión del repositorio se ha realizado a través de la plataforma GitHub [14], que ofrece herramientas colaborativas y una interfaz web intuitiva para la administración de versiones. El uso de Git en el proyecto se ha integrado con el entorno de desarrollo IntelliJ IDEA, aprovechando las herramientas y la integración nativa que ofrece este IDE para Git. Esto incluye soporte para la creación de ramas, fusiones de código, gestión de conflictos y otras operaciones de control de versiones directamente desde el entorno de desarrollo.

Capítulo 4

INGENIERÍA DEL SOFTWARE

4.1. Introducción

En este apartado vamos a explicar con detalle los aspectos de diseño y desarrollo del proyecto. Para ello, comenzaremos definiendo los requisitos de la aplicación, luego realizaremos un análisis del software a desarrollar y explicaremos el diseño de las clases y vistas de la misma. Para terminar, explicaremos con detalle algunas de las funciones implementadas y las pruebas que se han realizado para asegurar el buen funcionamiento del software desarrollado.

4.2. Definición de requisitos

4.2.1. Requisitos funcionales

- La aplicación ha de contar con dos modos, 'Pantalla Completa' y 'Escritorio', así como la capacidad de alternar entre estos distintos modos desde cada uno de ellos.
- La aplicación en modo 'Pantalla Completa' se controlará mediante teclado, no mediante ratón. Este modo no permitirá la edición de contenido, sino la visualización del mismo.
- La aplicación en modo 'Escritorio' debe permitir la visualización del contenido de una biblioteca en formato lista, y al seleccionar una serie o película, se ha de poder visualizar el contenido de la misma en un formato más extenso; pudiendo

visualizar los logos, fondos e imágenes de póster seleccionados, así como los vídeos o episodios que contienen las temporadas.

- La aplicación en modo pantalla completa debe permitir la visualización del contenido en formato grid, con las imágenes de poster de cada serie o película y una lista de bibliotecas en la parte superior que permita cambiar de biblioteca para la visualización de los elementos de la misma. Al seleccionar un elemento, la aplicación mostrará los detalles de la temporada seleccionada por defecto, o la última seleccionada por el usuario, mostrando los episodios de la misma en formato de lista con las imágenes de fondo de los episodios.
- La aplicación debe permitir al usuario la creación de bibliotecas que simbolicen un conjunto de carpetas con contenido multimedia. Cada biblioteca ha de ser de un tipo específico, pudiendo ser de tipo 'Series' o 'Películas'.
- La aplicación debe permitir al usuario la edición de las bibliotecas creadas, pudiendo modificar el nombre, idioma y carpetas asociadas; además de borrar la propia biblioteca.
- La aplicación debe analizar las carpetas asociadas a la biblioteca creada y procesar las distintas series o películas; buscando por nombre en la API de **TheMovieDB** y descargando los metadatos del primer resultado en el idioma de la biblioteca. Además, se descargará una canción por defecto para cada serie o película.
- La aplicación debe permitir al usuario corregir la identificación de cualquier película o serie; mostrando una ventana de búsqueda que permita buscar el contenido accediendo a la API de **TheMovieDB**. Este proceso reemplazará los metadatos actuales por los del contenido identificado en última instancia.
- La aplicación debe permitir al usuario modificar el grupo de episodios de una serie utilizando el mismo concepto que la corrección de identificación. Esto permitirá organizar la serie en temporadas con distintos números de episodios, según el grupo de episodios escogido.
- La aplicación debe permitir al usuario la edición manual de metadatos tanto en series y películas, como en sus elementos subyacentes: temporadas y episodios.
- La aplicación debe permitir al usuario eliminar cualquier elemento de una biblioteca, ya sea un episodio, una temporada, una serie completa o una película. Debe solicitar confirmación antes de la eliminación.

- La aplicación debe permitir al usuario añadir imágenes de portada y de fondo para las películas y serie, así como imagen de fondo para cada temporada y para los episodios. Estas imágenes se podrán añadir subiendo un archivo, descargando a través de una ventana emergente con buscador, o descargando a través de la URL de la imagen. Además, también ha de permitir añadir logos a las películas, así como a las temporadas de una serie.
- La aplicación debe permitir al usuario añadir música o vídeo de fondo para cada temporada. Si el usuario añade un vídeo de fondo, este tendrá prioridad sobre la música de fondo, y la música no será reproducida.
- La aplicación debe permitir al usuario reproducir el contenido añadido a la misma mediante un reproductor que sea compatible con la mayoría de formatos comúnmente utilizados de audio, vídeo y subtítulos; así como interactuar con el reproductor, pudiendo parar, avanzar o retroceder la reproducción, así como seleccionar el capítulo del vídeo que se está reproduciendo.
- La aplicación debe mantener un registro de las reproducciones del usuario, permitiéndole marcar episodios o películas como 'vista' o 'no vista'. Cuando el usuario salga del reproductor, se guardará el instante en el que se ha quedado para continuar la reproducción la próxima vez que el usuario quiera visualizar dicho vídeo.
- La aplicación debe permitir al usuario cambiar las pistas de audio, vídeo y subtítulos entre las disponibles.
- La aplicación debe permitir al usuario escoger el idioma de la misma, así como el idioma preferido para pistas de audio y subtítulos. La modificación del idioma ha de realizarse de forma dinámica, sin reiniciar la aplicación.

4.2.2. Requisitos no funcionales

- La aplicación debe permitir la interacción del usuario mientras procesa la descarga de metadatos, no impidiendo el uso de la misma al usuario.
- La aplicación debe ser compatible con los sistemas operativos Windows y Linux.
- La aplicación debe tener una interfaz de usuario intuitiva y fácil de usar, siguiendo las pautas de diseño de interfaces de usuario estándar.
- La aplicación debe almacenar todos los datos del usuario, como bibliotecas y configuraciones, localmente en formato JSON.

4.3. Planificación

Existen diversas metodologías de desarrollo, entre las cuales destacan las metodologías tradicionales y las metodologías ágiles, como se explica en [15].

4.3.1. Metodologías tradicionales

Las metodologías tradicionales, como el modelo en cascada, se caracterizan por seguir un enfoque secuencial en el que cada fase del desarrollo debe completarse antes de pasar a la siguiente. Este enfoque estructurado y lineal facilita la planificación y el seguimiento del progreso del proyecto. Sin embargo, puede ser inflexible ante cambios imprevistos o requisitos adicionales que surjan durante el desarrollo. En proyectos donde los requisitos están bien definidos y son poco susceptibles a cambios, estas metodologías pueden resultar eficaces.

4.3.2. Metodologías ágiles

Por otro lado, las metodologías ágiles han ganado popularidad en el desarrollo de software debido a su flexibilidad y capacidad para adaptarse a cambios continuos. Estas promueven un desarrollo iterativo e incremental, permitiendo ajustes rápidos en respuesta a nuevos requisitos o retroalimentación del usuario. Algunas de las metodologías ágiles más conocidas incluyen Scrum, Kanban y XP (Extreme Programming).

4.3.3. Metodología escogida

En este proyecto se utilizará la metodología ágil, específicamente el enfoque Kanban [16], para gestionar y planificar el desarrollo de la aplicación de escritorio. Kanban es una metodología visual que ayuda a mejorar la eficiencia y a gestionar el flujo de trabajo, permitiendo una adaptación rápida a los cambios y una entrega continua de resultados. Esta elección se justifica por varias razones:

- **Flexibilidad y Adaptabilidad:** Kanban permite un flujo continuo de trabajo, lo que facilita la incorporación de cambios en cualquier etapa del desarrollo.
- **Visualización del Progreso:** una de las principales características de Kanban es su tablero visual, que muestra las tareas en diferentes estados. Esta visualización

ayuda a identificar cuellos de botella y a gestionar mejor el tiempo y los recursos disponibles.

- **Enfoque en la Mejora Continua:** Kanban fomenta la mejora continua a través de revisiones frecuentes y ajustes en el proceso de trabajo. Esto permite optimizar la eficiencia y la productividad de manera constante.
- **Ajuste Perfecto para Desarrollo Individual:** a diferencia de Scrum, que requiere la formación de equipos y roles específicos (como el Scrum Master y el Product Owner), Kanban es más adecuado para proyectos desarrollados por una sola persona. La simplicidad y flexibilidad de Kanban permiten que un desarrollador individual gestione eficazmente el flujo de trabajo sin la necesidad de estructura adicional.
- **Reducción del Tiempo de Entrega:** al centrarse en la entrega continua de pequeñas mejoras y actualizaciones, Kanban reduce el tiempo de entrega de nuevas funcionalidades o correcciones, lo que resulta en un producto que puede evolucionar rápidamente en respuesta a las necesidades del usuario.



Figura 4.1: Logo de Trello

Para implementar Kanban, se utilizará la herramienta de gestión de proyectos Trello [17]. Se creará un tablero en Trello con la siguiente estructura:

- **To Do:** tareas que han sido identificadas y priorizadas, pero que aún no se han comenzado.
- **In Progress:** tareas que están siendo trabajadas actualmente.
- **Test:** tareas que han sido completadas y están pendientes de revisión y/o pruebas.
- **Done:** tareas que han sido completadas y revisadas.

Dado que el desarrollo es llevado a cabo por una única persona, no se realizarán reuniones, pero sí se mantendrá la revisión semanal para modificar las tareas a realizar. Esta revisión tiene como objetivo evaluar el progreso del proyecto, ajustar las prioridades de las tareas pendientes y asegurar que el desarrollo se mantenga en el camino correcto.

4.4. Análisis

En este apartado analizaremos los dos 'modos' o 'versiones' de la aplicación, el modo base o 'Escritorio' y el modo 'Pantalla Completa', así como el reproductor de vídeo; con la intención de descubrir las características a implementar.

4.4.1. Modo escritorio

El modo escritorio será una aplicación nativa con estética de aplicación de escritorio, pero adaptada a las aplicaciones modernas que no cuentan con la clásica barra superior con los ajustes y opciones de la aplicación. Este estilo de diseño se debe principalmente a la escasa cantidad de opciones que presentará, junto con la persecución de un diseño moderno y minimalista. Este modo nos proporcionará un selector para escoger la biblioteca a mostrar, y mostrará una lista vertical con los elementos de la biblioteca; dejando un espacio a la derecha de la ventana para mostrar en detalle la información del elemento seleccionado.

Se busca un diseño basado en una ventana principal y ventanas emergentes para realizar las acciones de adición, edición y borrado, mostrando así todo el contenido de los elementos de la biblioteca en una misma ventana.

Este modo permite añadir, editar y eliminar contenido, por lo que su función más compleja e importante es la búsqueda de archivos. Esta función se encargará de buscar dentro de las carpetas de una biblioteca los archivos de vídeo compatibles para, posteriormente, descargar los metadatos y añadir el contenido. Para ello es necesario analizar los nombres de los archivos y carpetas y detectar a qué serie o película corresponde cada uno. Esta tarea requerirá del uso de expresiones regulares y un recorrido ordenado y claro de los elementos en las carpetas seleccionadas. Una vez detectado el contenido, se descargarán los metadatos utilizando una petición HTTPS a la API de TheMovieDB [18] y se crearán los objetos pertinentes con los datos descargados. También se habrá de utilizar un script de Python [19] para buscar y descargar la música de dicho contenido.

Este modo permitirá la edición del contenido, por lo que será necesario implementar un código para descargar imágenes, preferiblemente incluyendo un buscador integrado. Los vídeos se subirán localmente, pero la música podrá subirse o descargarse desde la aplicación.

4.4.2. Modo pantalla completa

El modo pantalla completa alternará entre diferentes vistas, centrando la importancia en el contenido y no en su gestión. La primera vista mostrará un grid con los elementos de la biblioteca por defecto, teniendo una lista con las bibliotecas en la parte superior. De este modo, podremos seleccionar un elemento de la biblioteca y esto nos llevará a otra ventana, esta vez una centrada en las temporadas. Esta nueva ventana, que idealmente sustituye a la anterior, por lo que no será una ventana emergente; mostrará los detalles de la temporada, así como los episodios de la misma. Se podrá seleccionar un episodio, lo que abrirá una nueva ventana con el reproductor de vídeo.

4.4.3. Reproductor de vídeo

El reproductor de vídeo ha de estar integrado con JavaFX, pues se necesita la capacidad de interactuar con él dentro de la aplicación. La vista del reproductor tendrá los clásicos botones que permiten la interacción con un reproductor de vídeo, y mostrará información acerca del mismo. También permitirá la selección de las pistas de vídeo, audio y subtítulos, por lo que probablemente requiera la creación de un menú para realizar la interacción con cada lista de pistas.

4.4.4. Diagrama de clases

Para asegurar una clara visualización de las clases, el diagrama de clases se ha dividido en dos, uno para cada modo de la aplicación.

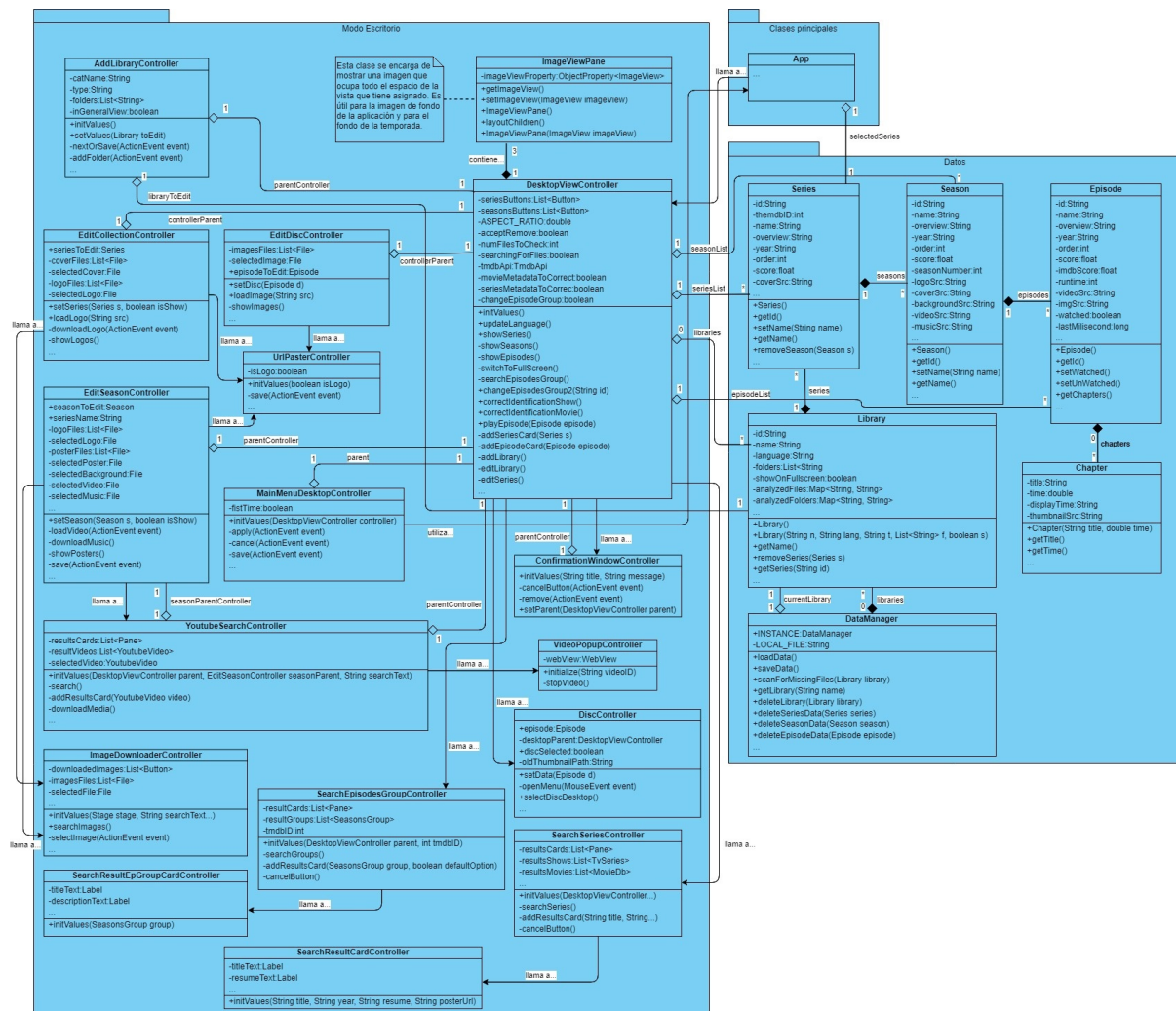


Figura 4.2: Diagrama de clases simplificado: Modo Escritorio

Debido al gran número de métodos y atributos presentes en algunas clases, como los más de 120 métodos de la clase `DesktopViewController`; se ha optado por simplificar el diseño mostrando únicamente algunos de dichos métodos y atributos. Además, para poder representar el diagrama de clases de manera correcta, se ha optado por prescindir de la representación de las clases destinadas a procesar los metadatos descargados utilizando la API de **TheMovieDB** [18], al igual que las clases encargadas de almacenar los capítulos extraídos de un archivo de vídeo utilizando **FFMPEG** [20], y la clase que representa un vídeo de **Youtube** para descargar la música.

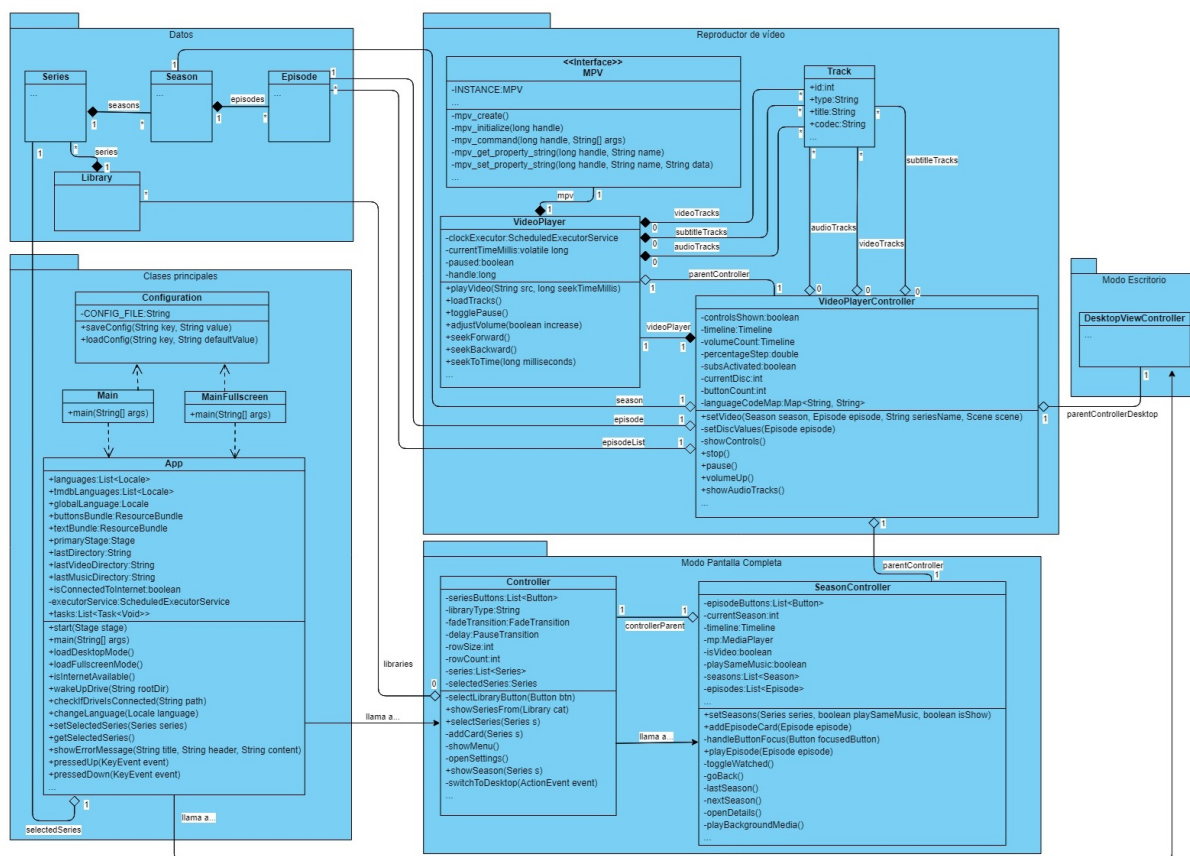


Figura 4.3: Diagrama de clases simplificado: Modo Pantalla Completa

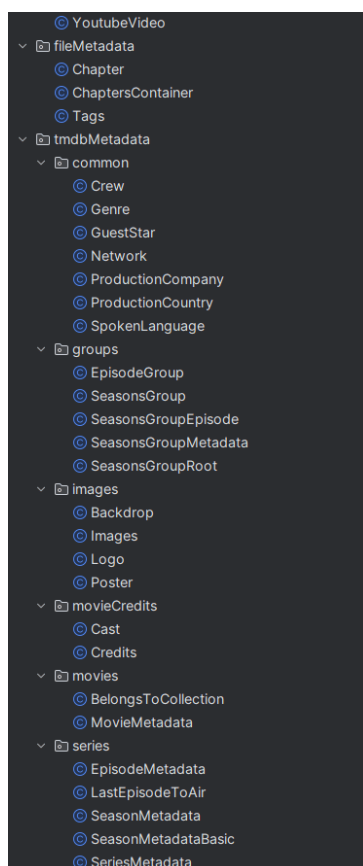


Figura 4.4: Clases que no forman parte del diagrama de clases simplificado

4.5. Diseño

En este apartado veremos en profundidad cómo se realizará el diseño de cada parte del proyecto, desde su arquitectura y patrones de diseño, hasta el almacenamiento de datos y la interfaz de usuario.

4.5.1. Arquitectura del sistema

La aplicación utiliza el patrón arquitectónico Modelo-Vista-Controlador (MVC) [21] para separar las responsabilidades y facilitar la mantenibilidad y escalabilidad. Los componentes principales son el Modelo, la Vista y el Controlador. Aplicación del patrón MVC en JavaFX en [22].

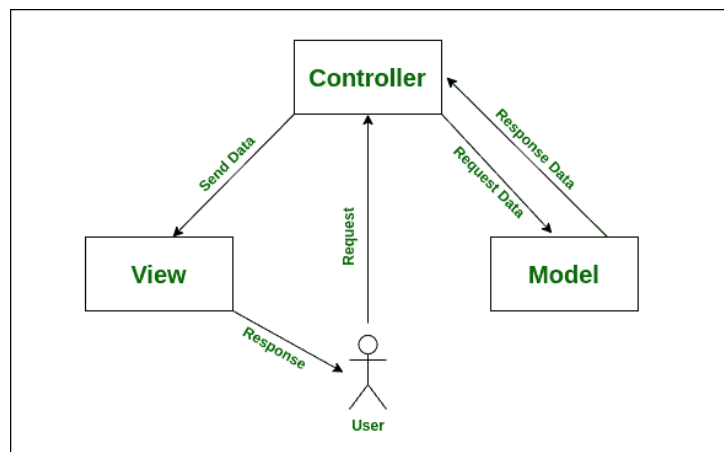


Figura 4.5: Patrón Modelo-Vista-Controlador (MVC)

JavaFX utiliza archivos .java para los modelos y controladores, y archivos .fxml para las vistas. Los archivos .fxml representan la interfaz de usuario, creando la vista que presenta la información al usuario. Los modelos serán archivos .java y la clase que contienen representará el propio modelo, incluyendo las funciones para obtener y modificar los atributos del mismo. Por parte de los controladores, cada vista crea por defecto un archivo .java que cumple con la función de controlador, modificando los datos de la misma. El controlador asociado a una vista es inicializado cuando esta se crea.

Para la comunicación con la API de TheMovieDB se utilizarán peticiones HTTPS, utilizando una API key personal, obtenida desde la propia página.

4.5.2. Patrones de diseño

Se han utilizado varios patrones de diseño para mejorar la organización del código y facilitar la mantenibilidad. Estos patrones son el patrón singleton y el patrón observador, explicados en [\[23\]](#).

4.5.2.1. Patrón Singleton

El patrón singleton asegura que una clase tenga únicamente una instancia, permitiendo así el acceso global a la misma y simplificando el código. El patrón singleton se ha implementado en la clase **DataManager**, encargada de gestionar los datos de las bibliotecas del usuario. Esto permite realizar las operaciones de CRUD con los datos desde cualquier clase externa de manera simple y eficaz. También se ha implementado en la interfaz **MPV**, que representa el reproductor de vídeo y es necesario asegurar que únicamente existe una instancia del mismo para prevenir problemas de rendimiento o bugs derivados de la utilización de varios reproductores simultáneos.

4.5.2.2. Patrón Observador

El patrón observador se ha implementado en el controlador del reproductor de vídeo y la clase encargada de mostrar el reproductor. Esto ha sido necesario para sincronizar el slider de la vista con el instante de reproducción actual, para permitir al usuario avanzar y retroceder el vídeo mediante la vista. La implementación detallada de este patrón se explicará en el apartado [4.6.5.4 Patrón observador](#) (véase la pág. [74](#)).

4.5.3. Diagramas de casos de uso

En este apartado veremos los diagramas de casos de uso, explicando cada uno de ellos.

4.5.3.1. Diagrama de casos de uso general

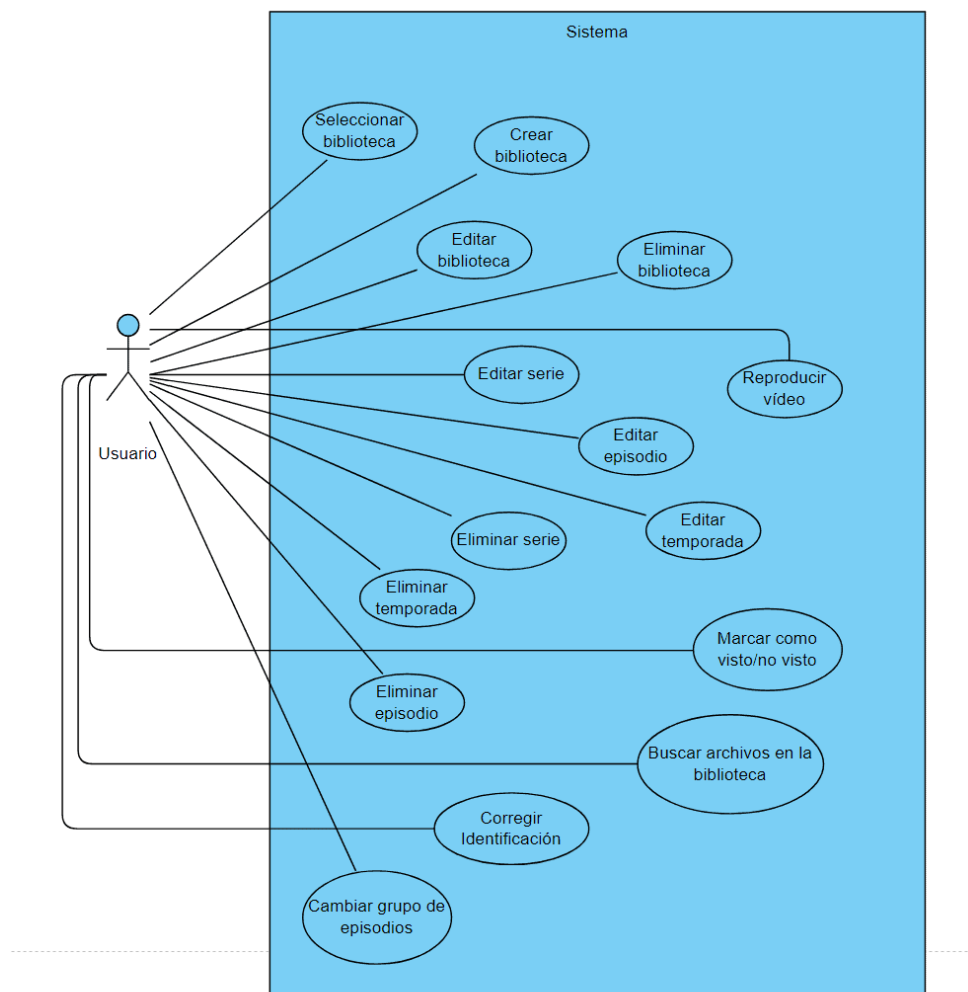


Figura 4.6: Diagrama de casos de uso del sistema

Este diagrama representa la interacción del usuario con el sistema a través de una serie de casos de uso. El usuario es el que comienza la interacción, pudiendo crear bibliotecas, así como editar el contenido de las mismas o eliminarlo. El sistema, en su caso, se encarga de realizar el procesamiento necesario sobre los datos para llevar a cabo las peticiones del usuario.

4.5.3.2. Diagrama de casos de uso: crear biblioteca

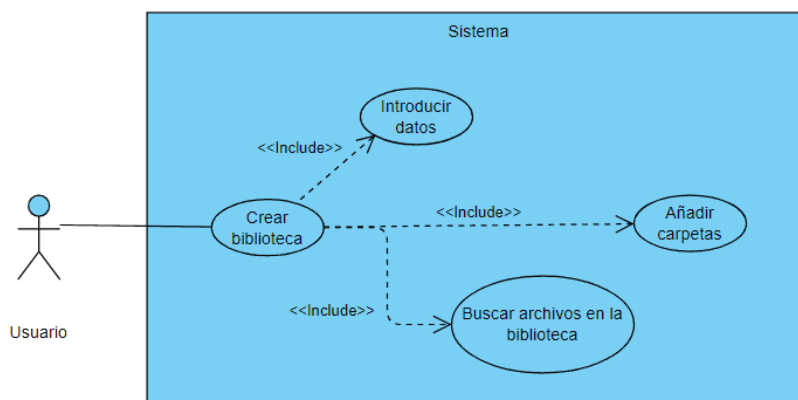


Figura 4.7: Diagrama de casos de uso: crear biblioteca

La creación de una biblioteca comienza con una ventana en la que el usuario ha de escoger el tipo de biblioteca, el idioma y el nombre de la misma. Al introducir dichos datos, podrá acceder a otro apartado de la ventana, en el cual aparecerá un botón para añadir carpetas a la biblioteca. Una vez se ha añadido al menos una carpeta, el usuario puede guardar la biblioteca. Al hacerlo, comenzará la función de búsqueda de archivos y se añadirán automáticamente las series o películas.

4.5.3.3. Diagrama de casos de uso: editar biblioteca

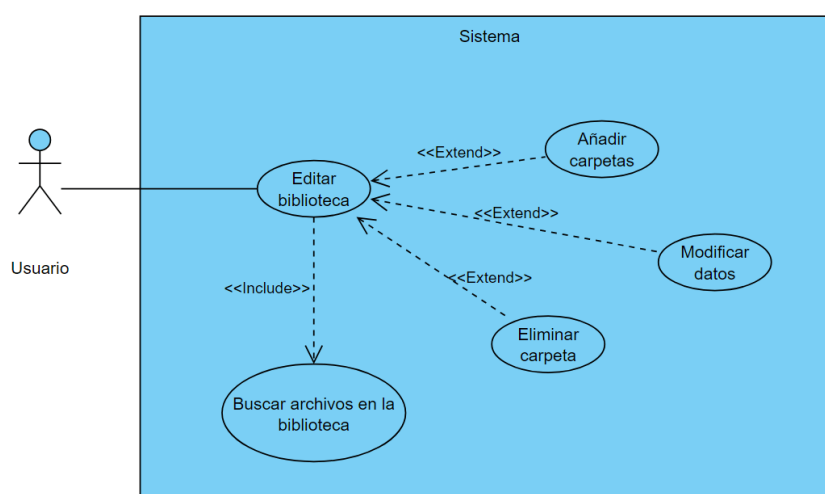


Figura 4.8: Diagrama de casos de uso: editar biblioteca

La edición de una biblioteca es el proceso de modificar el nombre, idioma y carpetas de esta. Para ello, se mostrará la misma ventana que al añadir una biblioteca, pero con los datos de esta previamente introducidos. Se permitirá la modificación de todos esos datos, menos del tipo de biblioteca. Una vez guardada la edición, se procederá a realizar la búsqueda de archivos.

4.5.3.4. Diagrama de casos de uso: editar serie

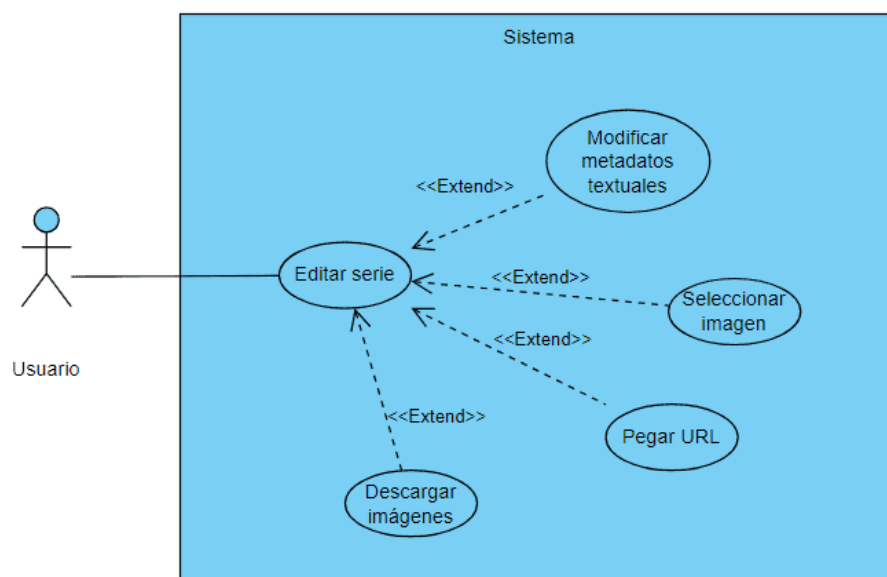


Figura 4.9: Diagrama de casos de uso: editar serie

La edición de una serie permite la modificación de sus datos, al igual que la descarga de imágenes. Si se trata de una película, el objeto Serie se tratará como una colección de películas (siendo estas las temporadas), por lo que las únicas imágenes a descargar serían las imágenes de póster de la colección. En caso de ser una serie, se descargarían tanto imágenes de póster como logos.

4.5.3.5. Diagrama de casos de uso: editar temporada

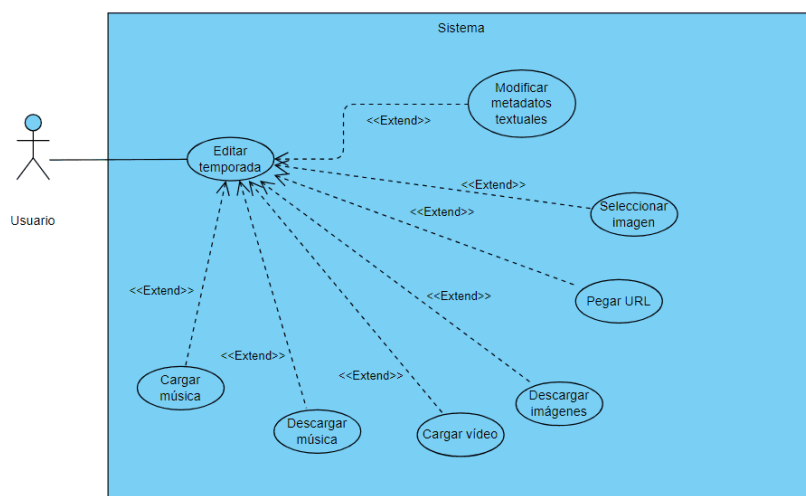


Figura 4.10: Diagrama de casos de uso: editar temporada

La temporada hace referencia a una temporada de una serie, o a una película en una colección. Por tanto, se podrán descargar en ambos casos imágenes de fondo. Sin embargo, en caso de ser una película, se podrán descargar imágenes de póster y logos. La temporada también almacena la música y vídeo de fondo, por lo que se podrán añadir o modificar estos archivos, al igual que los datos de la temporada.

4.5.3.6. Diagrama de casos de uso: editar episodio

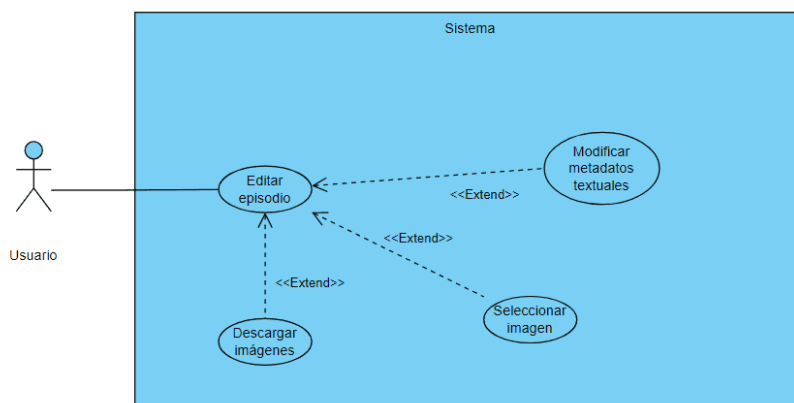


Figura 4.11: Diagrama de casos de uso: editar episodio

La edición de un episodio permitirá la modificación del nombre del mismo, al igual que la descarga de imágenes de fondo.

4.5.3.7. Diagrama de casos de uso: eliminar biblioteca

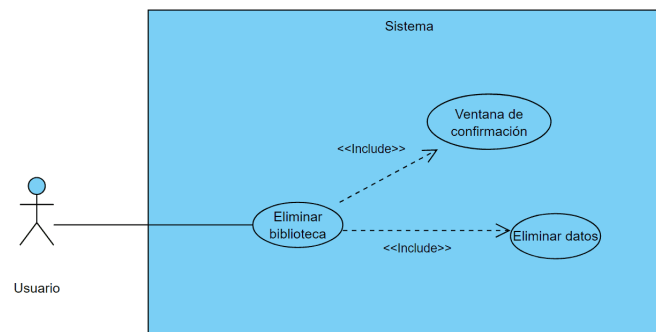


Figura 4.12: Diagrama de casos de uso: eliminar biblioteca

Al seleccionar el botón de eliminar biblioteca, se mostrará una ventana de confirmación. Al aceptar la eliminación, se borrarán los datos asociados a cada uno de los elementos de la biblioteca, al igual que las imágenes, música y vídeos de fondo asociados a cada uno.

4.5.3.8. Diagrama de casos de uso: eliminar serie

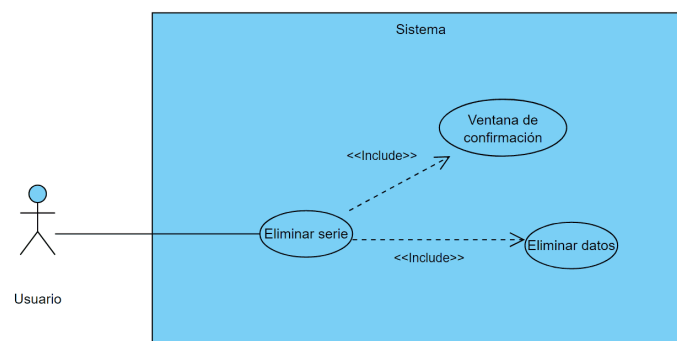


Figura 4.13: Diagrama de casos de uso: eliminar serie

La eliminación de una serie es similar a la de una biblioteca, mostrando la ventana de confirmación. Una vez aceptada la eliminación, se procederá a borrar los datos y contenido descargado de dicha serie.

4.5.3.9. Diagrama de casos de uso: eliminar temporada

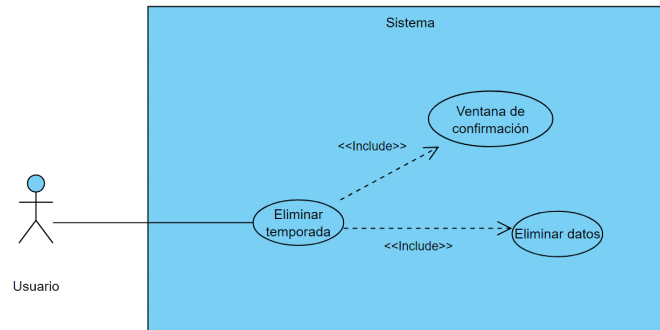


Figura 4.14: Diagrama de casos de uso: eliminar temporada

Al igual que al eliminar una serie, la eliminación de una temporada muestra la ventana de confirmación y borra los datos y archivos descargados de la temporada.

4.5.3.10. Diagrama de casos de uso: eliminar episodio

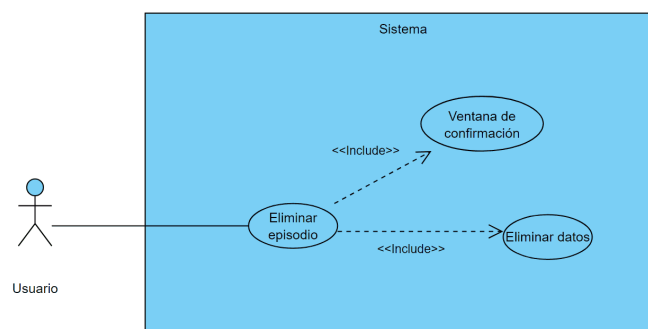


Figura 4.15: Diagrama de casos de uso: eliminar episodio

La eliminación de un episodio muestra una ventana de confirmación y elimina los datos e imágenes de fondo del episodio seleccionado.

4.5.3.11. Diagrama de casos de uso: cambiar grupo de episodios

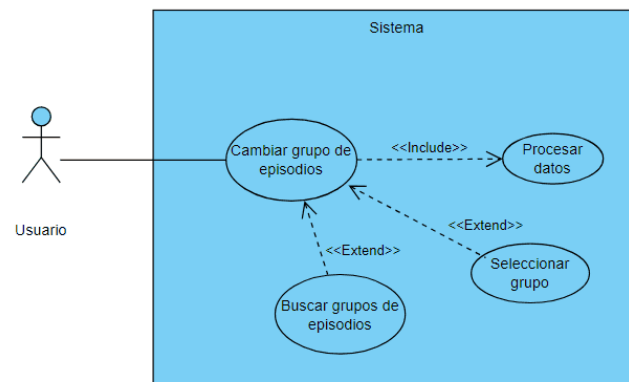


Figura 4.16: Diagrama de casos de uso: cambiar grupo de episodios

Esta ventana mostrará una lista con los diferentes grupos de episodios que tiene una serie. Cada grupo contará con información acerca de cuántas temporadas y episodios tiene. El usuario podrá seleccionar uno de estos grupos, lo cual modificará las temporadas de dicha serie.

4.5.3.12. Diagrama de casos de uso: corregir identificación

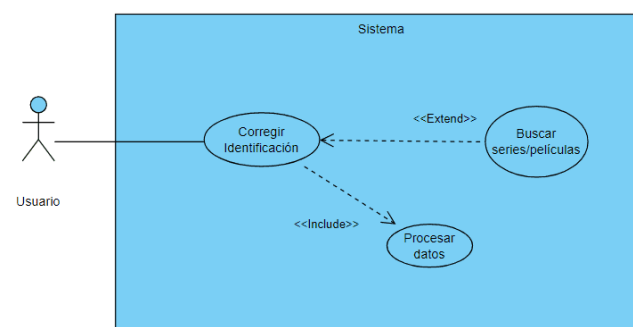


Figura 4.17: Diagrama de casos de uso: corregir identificación

La ventana de corrección de identificación mostrará un buscador de series o películas, dependiendo del tipo de biblioteca actual, y la lista de contenido encontrado. El usuario podrá entonces seleccionar la nueva identificación, lo que modificará los datos de dicho contenido, asignando una nueva identificación al mismo.

4.5.4. Diagramas de secuencia

En este apartado veremos algunos de los diagramas de secuencia, explicando el flujo de operaciones que se realiza en cada uno de ellos.

4.5.4.1. Diagrama de secuencia: crear biblioteca

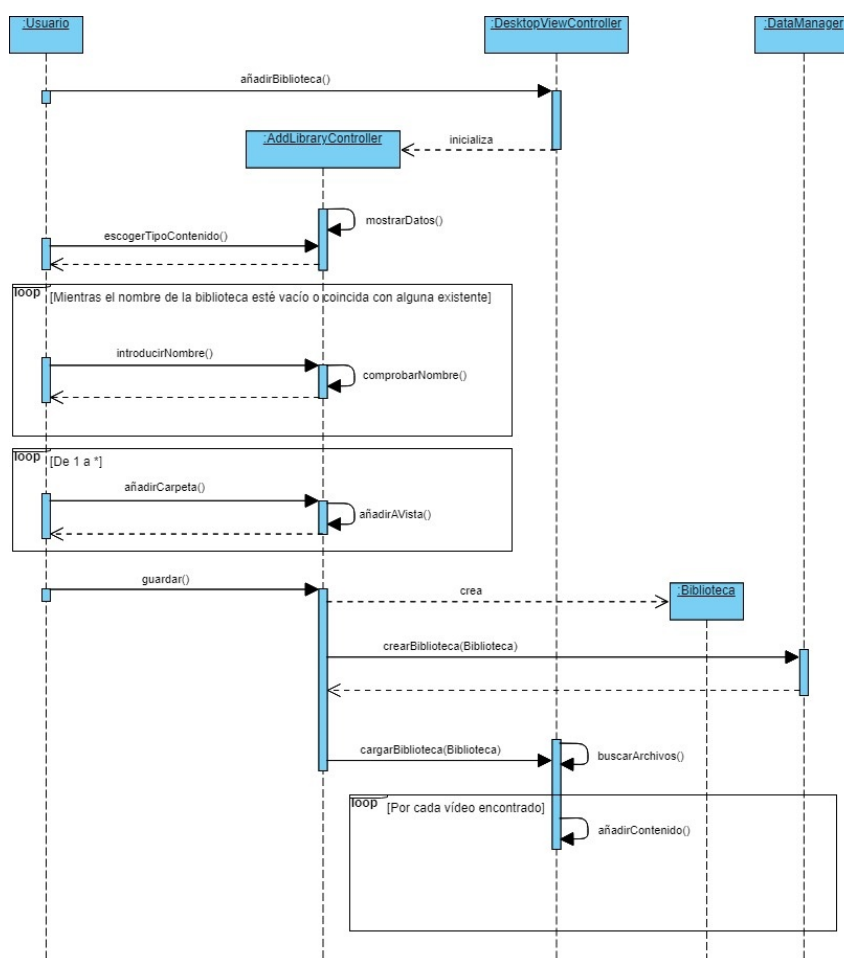


Figura 4.18: Diagrama de Secuencia: Crear Biblioteca

En este diagrama podemos ver que el usuario comienza su interacción pulsando el botón de añadir biblioteca en la vista principal, controlada por DesktopViewController. Tras esto, dicha clase crea otra vista, una ventana con el formulario para crear la biblioteca. El usuario ahora introduce los datos de la biblioteca, no pudiendo continuar si el nombre está vacío o coincide con el de una biblioteca existente. Una vez avanza, se le permite añadir las carpetas de la biblioteca. Puede ser una única carpeta o varias, pero no puede ser ninguna.

Una vez pulsa el botón para guardar, se crea la biblioteca y se añade a la lista de bibliotecas almacenada en DataManager. Tras esto, el procesamiento vuelve a DesktopViewController, que busca los archivos de vídeo dentro de dichas carpetas, los procesa y los añade.

4.5.4.2. Diagrama de secuencia: corregir identificación

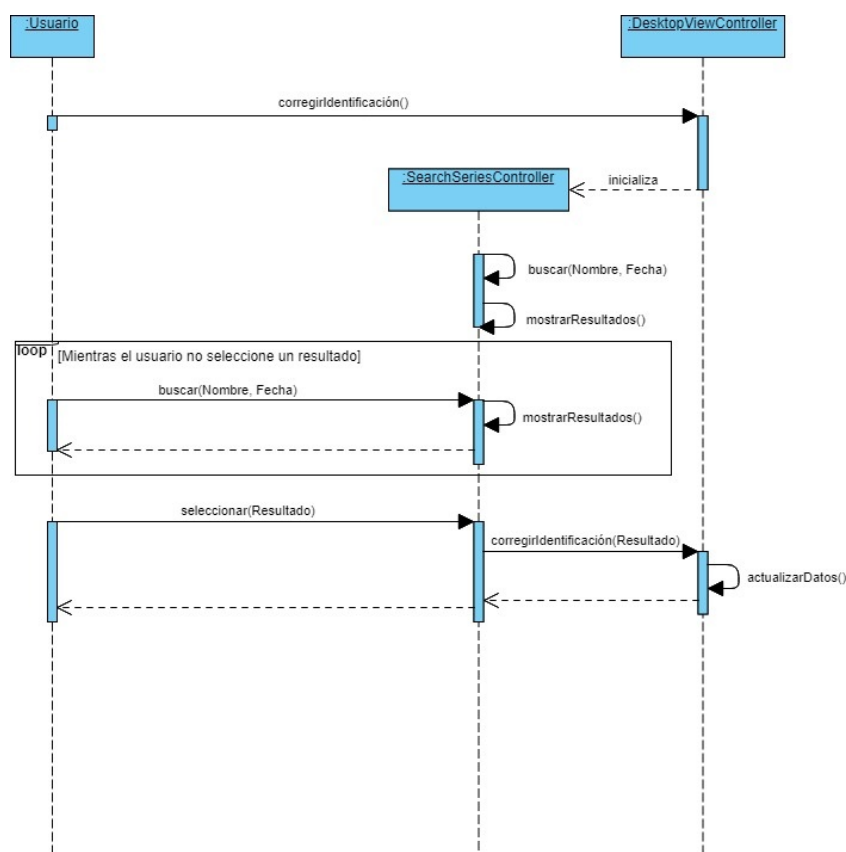


Figura 4.19: Diagrama de Secuencia: Corregir Identificación

El siguiente diagrama es el de *Corregir Identificación*. Este comienza con el usuario pulsando el botón para corregir identificación, situado en el menú emergente de una serie o de la temporada de una película; pues la serie sería la colección y la temporada, la película, como está explicado en [4.5.5 Almacenamiento de datos](#) (véase la pág. 40). Tras esto, se abre una ventana que muestra los resultados de buscar en la API de TheMovieDB por nombre y fecha la serie o película en cuestión. El usuario entonces selecciona el contenido correcto, o realiza otra búsqueda si lo ve oportuno. Una vez seleccionada la serie o película correcta, la clase DesktopViewController se encarga de actualizar los datos del contenido para reflejar la nueva identificación.

4.5.4.3. Diagrama de secuencia: editar serie

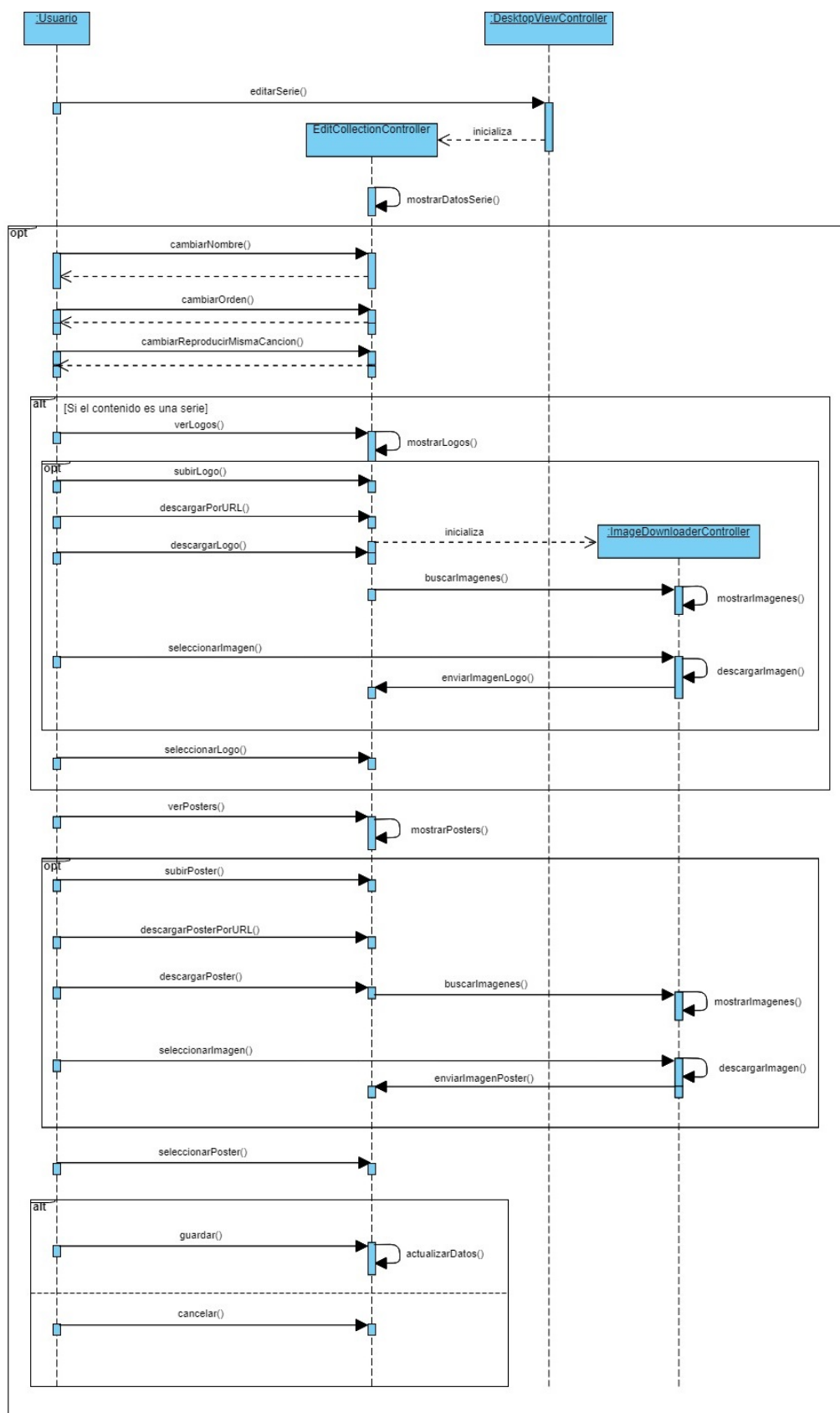


Figura 4.20: Diagrama de Secuencia: Editar Serie

Este diagrama comienza con el usuario pulsando el botón de editar serie. Al hacerlo, una ventana muestra los detalles de la serie en un formulario para que el usuario los edite. Si el contenido es una serie, aparecerá un botón extra para mostrar los logos. El usuario puede pulsar los botones para mostrar los logos o los pósters, y al hacerlo aparecerán todas las imágenes descargadas de dicha categoría. El usuario puede subir sus imágenes, al igual que pegar la URL para descargarlas o abrir una ventana que muestre un buscador de imágenes y permita descargar la que el usuario seleccione.

4.5.5. Almacenamiento de datos

El almacenamiento de los datos se llevará a cabo en un archivo **JSON** [24] que incluirá una lista con las bibliotecas del usuario. Estas bibliotecas incluyen a su vez el contenido multimedia en forma de series con temporadas y episodios.

Se ha decidido que las películas y las series se tratarán de la misma manera a la hora de almacenar los datos. Esto se debe a que las películas pueden formar parte de colecciones, como una trilogía, por ejemplo; debido a esto, una trilogía de películas puede verse como una serie (colección) con temporadas (películas) y episodios (vídeo de la película, vídeos extra, etc.). En caso de que una película no forme parte de una colección, la serie y la temporada tendrán los mismos datos. De esta manera, unificamos el almacenamiento y simplificamos el diseño para contar con únicamente tres clases básicas que almacenarán el contenido del usuario: series, season, episode.

La elección de utilizar **JSON** en vez de una base de datos se debe a que el propósito de la aplicación es visualizar el contenido en conjunto, por lo que no requerimos de carga en cascada, sino que cargamos todos los datos al inicio y los utilizamos desde memoria. Esto podría suponer un problema si se maneja una cantidad de datos muy grande, pero una colección de vídeos personal no supone un problema para este cometido.

4.5.6. Interfaz de usuario

En este proyecto se ha seguido un enfoque metódico para diseñar y desarrollar una interfaz de usuario que sea intuitiva, eficiente y atractiva. Este proceso incluye la creación de bocetos y wireframes iniciales, seguidos de la implementación de las vistas en FXML utilizando SceneBuilder [11], una herramienta de diseño de interfaz gráfica, para crear y editar archivos FXML de manera visual e intuitiva.

4.5.6.1. Wireframes

El diseño se comenzó con bocetos digitales en una tablet y, posteriormente, la generación de wireframes gracias a los mismos. Los wireframes permiten definir claramente la jerarquía de la información, la navegación, y la disposición de los componentes de la interfaz antes de proceder a la implementación.

A continuación veremos los wireframes de las vistas principales de la aplicación:

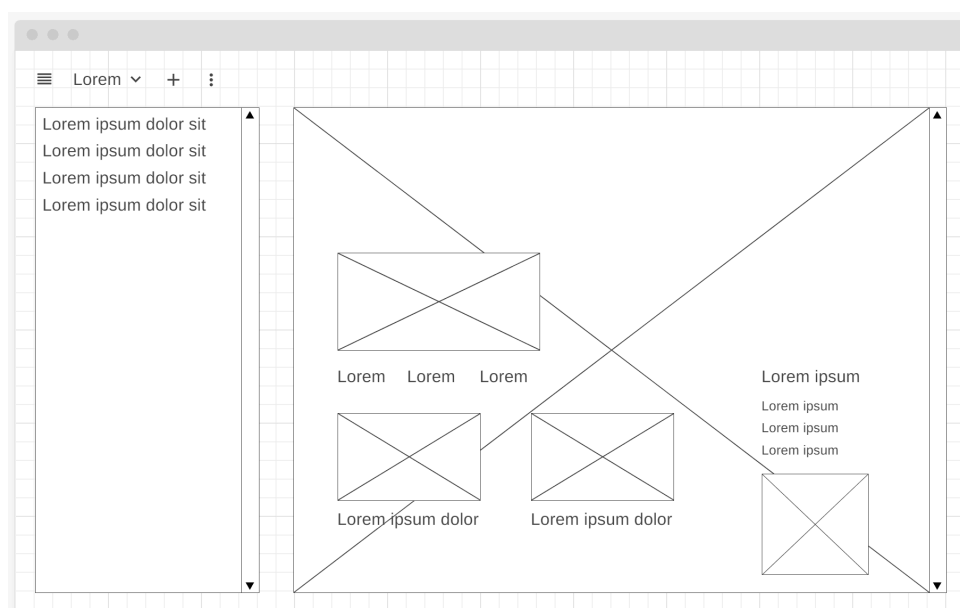


Figura 4.21: Wireframe de la vista principal del modo escritorio

Este wireframe (fig. 4.21) representa el diseño de la vista de la aplicación en modo escritorio. Este modo tiene una única vista base y una serie de ventanas flotantes para representar los menús de configuración, adición, edición y eliminación de contenido.

Como podemos observar, en la parte superior izquierda contamos con un icono para acceder a un menú contextual que nos permitirá acceder a preferencias, cambiar

al modo pantalla completa, o salir de la aplicación. A su derecha nos encontramos con un desplegable que representa la lista de bibliotecas del usuario, permitiéndonos cambiar entre estas haciendo click. A la derecha del desplegable tenemos el icono para añadir una nueva biblioteca y el icono que abre un menú contextual para realizar las siguientes acciones: editar biblioteca, buscar archivos en la biblioteca, y eliminar la biblioteca.

Debajo de la sección analizada, tenemos una lista con los elementos de la biblioteca seleccionada. Al seleccionar uno de estos elementos, la sección de su derecha se rellenará con la información de la serie o película. Esta sección cuenta con una imagen de fondo, y si analizamos en vertical encontramos el logo, las temporadas en una lista horizontal, y debajo los vídeos o episodios en formato de imagen y título. A la derecha de los episodios tenemos una sección de detalles sobre la serie o película, junto con la carátula de la misma debajo de esta sección.

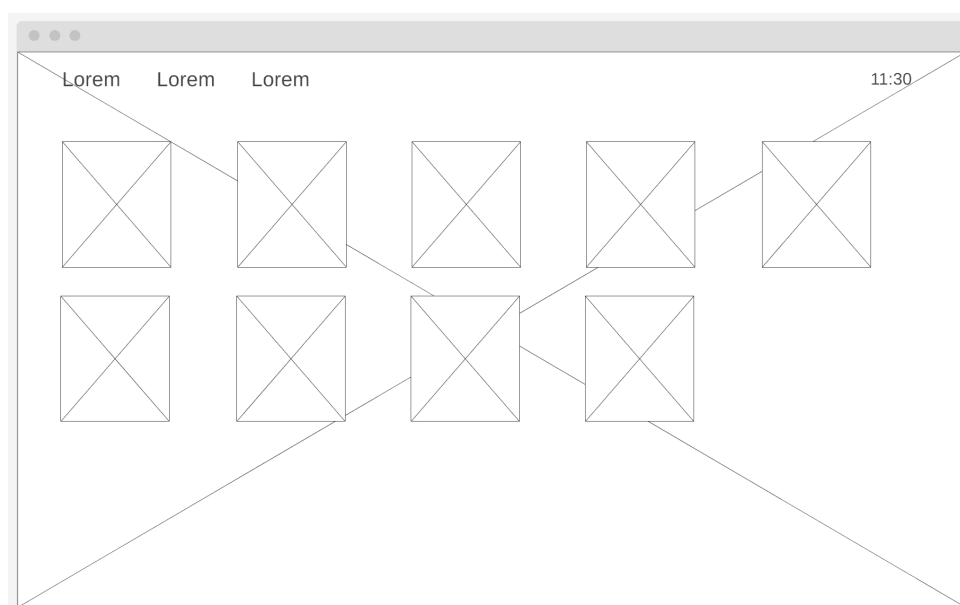


Figura 4.22: Wireframe de la vista principal del modo pantalla completa

El siguiente wireframe (fig. 4.22) es muy simple, contando con una lista horizontal con las bibliotecas del usuario en la parte superior izquierda y la hora en la parte superior derecha. Debajo de esta sección tenemos un grid con los distintos elementos de la colección, representados por la carátula o póster de cada uno. De fondo contamos con la imagen de fondo con efecto blur de la primera temporada del elemento seleccionado.

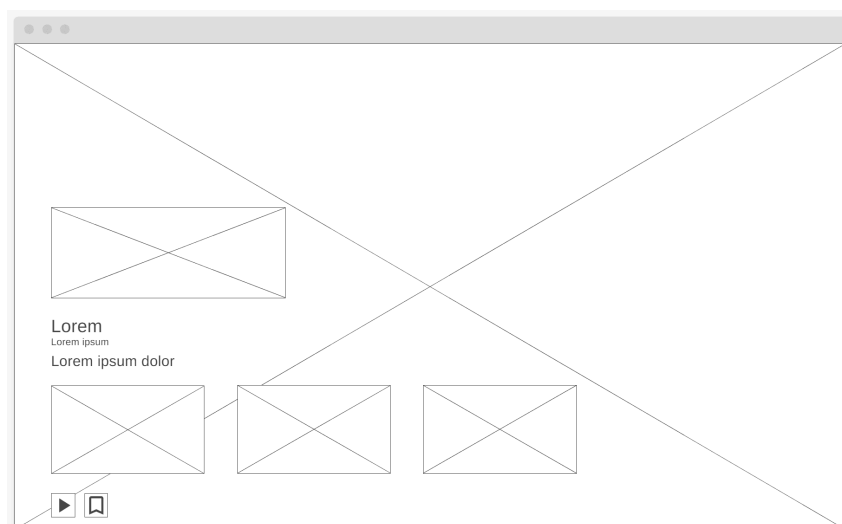


Figura 4.23: Wireframe de la vista de temporada en el modo pantalla completa

Esta es la vista resultante de seleccionar un elemento en el modo pantalla completa (fig. 4.23). La vista en cuestión nos mostrará los datos de la temporada y nos permitirá movernos entre temporadas pulsando las teclas de movimiento hacia izquierda o derecha. Los datos que muestra son el fondo de la temporada, y en vertical tenemos el logo de la serie o película, el título de la temporada, diversos datos y la sinopsis. Debajo de la sección de datos nos encontramos con una lista horizontal de imágenes de cada episodio, que al seleccionar abrirá el reproductor con el vídeo seleccionado. Esta lista sólo se mostrará si hay más de un vídeo en la temporada. Tras esta sección, tenemos los botones para reproducir el vídeo seleccionado y para marcar como visto o no visto.

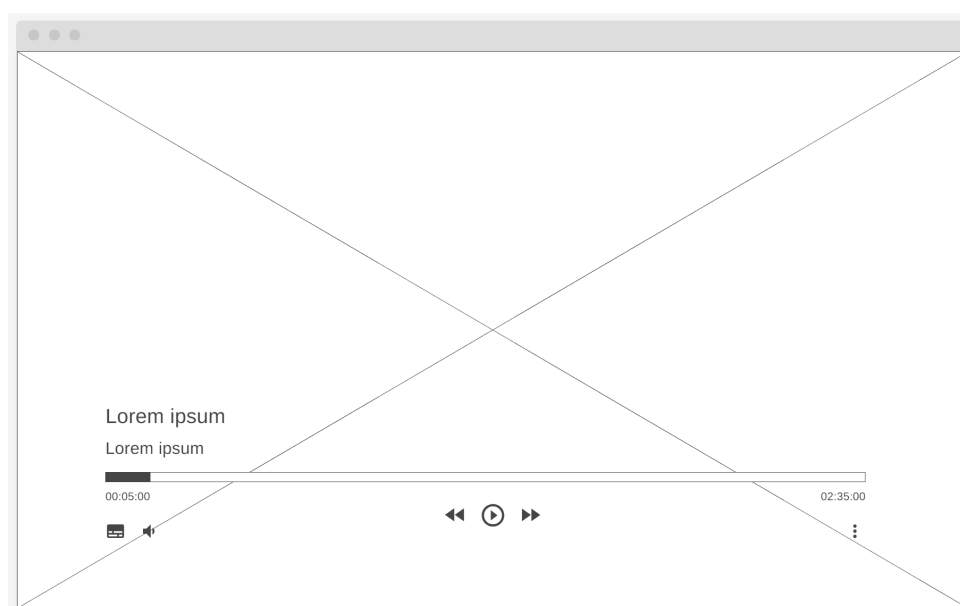


Figura 4.24: Wireframe del reproductor de vídeo

El último de los wireframe que veremos es el reproductor de vídeo (fig. 4.24). Esta vista proporciona los controles para interactuar con el reproductor, por lo que cuenta con botones para pausar y continuar, moverse entre vídeos (aunque en el wireframe parecen iconos de avanzar y retroceder), opciones a la derecha, y menús para seleccionar las pistas de audio y subtítulos. El menú de opciones permitirá seleccionar la pista de vídeo, junto con la modificación del zoom del mismo. También contará con un slider para avanzar o retroceder el vídeo, y con información acerca del vídeo que se está reproduciendo, como el nombre del mismo, el de la serie o película y la fecha.

4.6. Implementación

En este apartado describiremos con detalle la funcionalidad implementada, explicando el código de manera general y simplificada, pero proporcionando los ejemplos del mismo para una mejor comprensión.

4.6.1. Internacionalización

La internacionalización se ha logrado gracias al uso de la clase `ResourceBundle` [25] de Java. Este enfoque permite que la aplicación soporte múltiples idiomas y se adapte automáticamente al idioma del usuario final sin la necesidad de recompilar la misma.

4.6.1.1. Uso de `ResourceBundle`

Para manejar la internacionalización, se han creado dos tipos de archivos de propiedades:

- `buttons_idioma.properties`: contiene las traducciones de los textos de los botones.
- `text_idioma.properties`: contiene las traducciones de los textos generales de la aplicación.

Cada archivo de propiedades está localizado para diferentes idiomas, lo que permite que la interfaz de usuario se muestre en el idioma preferido del usuario. Los idiomas soportados actualmente son:

- Árabe (Arabia Saudita): `buttons_ar_SA.properties` y `text_ar_SA.properties`
- Alemán (Alemania): `buttons_de_DE.properties` y `text_de_DE.properties`
- Inglés (Estados Unidos): `buttons_en_US.properties` y `text_en_US.properties`
- Español (España): `buttons_es_ES.properties` y `text_es_ES.properties`
- Francés (Francia): `buttons_fr_FR.properties` y `text_fr_FR.properties`

- Hindi (India): `buttons_hi_IN.properties` y `text_hi_IN.properties`
- Japonés (Japón): `buttons_ja_JP.properties` y `text_ja_JP.properties`
- Coreano (Corea del Sur): `buttons_ko_KR.properties` y `text_ko_KR.properties`
- Chino Simplificado (China): `buttons_zh_CN.properties` y `text_zh_CN.properties`

A continuación, se muestra un fragmento del archivo **`buttons_es_ES.properties`**:

```
1 playButton=Reproducir
2 editButton=Editar
3 removeButton=Eliminar
4 applyButton=Aplicar
5 addButton=Añadir
```

Listado 4.1: Ejemplo de contenido localizado

El uso de `ResourceBundle` en Java ha sido el siguiente:

```
1 //Set global language
2 globalLanguage = Locale.forLanguageTag(Configuration.loadConfig("currentLanguageTag", "en
   -US"));
3
4 //Add languages
5 String languageDir = "resources";
6 File[] filesInDir = new File(languageDir).listFiles();
7
8 if (filesInDir != null){
9     for (File file : filesInDir){
10         if (file.getName().matches("^buttons.*")){
11             String language = "";
12
13             language = file.getName().substring(file.getName().indexOf("_") + 1, file.
               getName().lastIndexOf("."));
14
15             String tag = language.substring(0, language.indexOf("_"));
16             String country = language.substring(language.indexOf("_") + 1);
17
18             languages.add(Locale.of(tag, country));
19         }
20     }
21 }
22
23 //Set resource bundles
24 File file = new File("resources/");
25 URL[] urls = {file.toURI().toURL()};
26 ClassLoader loader = new URLClassLoader(urls);
27 buttonsBundle = ResourceBundle.getBundle("buttons", globalLanguage, loader);
28 textBundle = ResourceBundle.getBundle("text", globalLanguage, loader);
```

Listado 4.2: Internacionalización en código

Como podemos observar, el idioma global de la aplicación se establece según un archivo de configuración, mediante la clase 'Configuration'. Tras esto, se analizan los nombres de los archivos de texto de botones para detectar qué idiomas tiene la aplicación. Esto permite que se puedan añadir nuevos idiomas, pues cada vez que se ejecuta la aplicación, se analizan los archivos de idiomas y se guardan sus respectivos objetos 'Locale' en una lista. Una vez tenemos los idiomas, procesamos los archivos, creando los ResourceBundle.

Para su uso en la aplicación, simplemente es necesario reemplazar todas las cadenas de caracteres por accesos a los ResourceBundle. Aquí podemos observar un ejemplo que establece el texto del botón para eliminar un episodio:

```
1 removeEpisodeButton.setText(App.buttonsBundle.getString("removeButton"));
```

Listado 4.3: Ejemplo de localización de un botón

4.6.1.2. Proceso de Localización

Para realizar las localizaciones, se ha utilizado ChatGPT [26] como traductor. Esto ha permitido obtener traducciones rápidas y precisas para los diferentes textos de la aplicación. Cada archivo de propiedades ha sido revisado y ajustado para asegurar la coherencia y precisión en el contexto de uso específico de la aplicación.

El uso de ChatGPT ha facilitado la internacionalización de la aplicación, asegurando que los usuarios de diferentes regiones puedan utilizarla en su idioma nativo, mejorando así la accesibilidad y la experiencia de usuario.

4.6.2. Clases utilizadas

En este apartado explicaremos cada una de las clases presentes en la aplicación, sin contar con las clases excluidas del diagrama de clases, que como se comentó anteriormente, son clases auxiliares para procesar los metadatos descargados de **The-MovieDB**.

- **Library**: representa una biblioteca, es decir, un conjunto de series o películas.
- **Series**: representa una serie, película o colección de estas. Si es una película única, tendrá una temporada única con la misma información de la película.

- **Season:** representa una temporada de una serie, o una película de una colección.
- **Episode:** representa un vídeo, ya sea una película o un episodio.
- **Chapter:** representa un capítulo de un vídeo.
- **DataManager:** clase encargada de leer y guardar los datos del usuario, así como gestionar las bibliotecas.
- **Configuration:** clase encargada de leer y guardar datos en el archivo de configuración. Este archivo es *config.properties* y almacena valores que el usuario puede editar en el menú de configuración de la aplicación.
- **Utils:** clase que contiene los comparadores para ordenar las bibliotecas, series, temporadas y episodios dentro de las listas utilizadas para mostrarlas.
- **App:** clase principal. Está encargada de ejecutar la vista inicial de JavaFX, cargando los idiomas previamente.
- **Main:** clase auxiliar para permitir la ejecución del archivo JAR. Esta se encarga de modificar el archivo de configuración antes de llamar a App, para así lanzar la aplicación con la vista del modo escritorio.
- **MainFullscreen:** clase auxiliar para permitir la ejecución del archivo JAR. También modifica el archivo de configuración antes de llamar a App, para ahora lanzar la aplicación con la vista del modo pantalla completa.
- **Controller:** controlador de la vista principal del modo pantalla completa.
- **SeasonController:** controlador de la vista de temporadas del modo pantalla completa.
- **DesktopViewController:** controlador de la vista principal del modo escritorio. Esta clase realiza toda la gestión de análisis de archivos y descarga de metadatos.
- **MPV:** interfaz encargada de ejecutar el reproductor MPV.
- **Track:** clase que representa una pista de audio, vídeo o subtítulos de un vídeo.
- **VideoPlayer:** clase que utiliza la interfaz MPV para ejecutar el reproductor en una vista de JavaFX. También proporciona las funciones de interacción con el reproductor.
- **AddLibraryController:** controlador de la vista encargada de añadir o editar una biblioteca.

- `ConfirmationWindowController`: clase que genera una ventana de confirmación con los datos que se le proporcionan.
- `DiscController`: clase que representa un episodio o vídeo en el modo escritorio, proporcionando elementos de interacción para seleccionarlo, editarlo, eliminarlo o reproducirlo.
- `EditCollectionController`: controlador de la vista encargada de editar una serie.
- `EditSeasonController`: controlador de la vista encargada de editar una temporada.
- `EditDiscController`: controlador de la vista encargada de editar un episodio.
- `ImageDownloaderController`: controlador de la vista que muestra un buscador de imágenes, haciendo uso de un script de python para descargar imágenes.
- `ImageViewPane`: clase que crea una imagen que se ajusta al tamaño de la sección en la que se encuentra. Especialmente útil a la hora de modificar el tamaño de la imagen de fondo de una temporada cuando la ventana se redimensiona.
- `MainMenuDesktopController`: controlador de la vista del menú de configuración del modo escritorio.
- `SearchEpisodesGroupController`: controlador de la vista que genera una ventana que permite buscar grupos de episodios para cambiar el grupo actual de una serie.
- `SearchResultEpGroupCardController`: controlador de la tarjeta que muestra la información de un grupo de episodios.
- `SearchSeriesController`: controlador de la vista que genera una ventana que permite buscar series o películas para corregir la identificación de la serie o película seleccionada.
- `SearchResultCardController`: controlador de la tarjeta que muestra información de una serie o película.
- `UrlPasterController`: controlador de la vista que genera una ventana que permite al usuario introducir una URL y descarga la imagen correspondiente.
- `VideoPlayerController`: controlador de la vista del reproductor de vídeo. Esta vista permite interactuar con la clase `VideoPlayer` por medio de botones en la misma.

- `YoutubeSearchController`: controlador de la vista que genera un buscador de vídeos de youtube. Este buscador tiene como objetivo descargar la música del vídeo seleccionado y añadirla a la temporada que se está editando.
- `VideoPopupController`: controlador de la vista que genera una ventana de reproducción para vídeos en youtube a la hora de descargar la música.

4.6.3. Modo escritorio

En esta sección veremos algunas de las funciones más importantes del modo escritorio, explicando su comportamiento paso a paso y ejemplificando con código.

4.6.3.1. Ejecución concurrente

La ejecución concurrente se ha llevado a cabo gracias a la clase `Task` de JavaFX. Esta clase permite crear una tarea, con una función que se ejecuta al inicializarla tal y como se muestra en el siguiente fragmento de código:

```
1 Task<Void> thumbnailTask = new Task<>() {  
2     @Override  
3     protected Void call() {  
4         for (int i = 1; i < thumbnailsUrls.size(); i++){  
5             saveThumbnail(episode, thumbnailsUrls.get(i), i);  
6         }  
7         return null;  
8     }  
9 };
```

Listado 4.4: Código para crear una tarea en JavaFX

Las tareas permiten establecer comportamientos a realizar tras completarse o al cancelarse, de modo que podemos gestionar lo que ocurre al finalizar una tarea que se ejecuta en segundo plano. Dicha tarea se puede ejecutar de manera concurrente creando un nuevo hilo de Java de la siguiente manera:

```
1 Thread thread = new Thread(thumbnailTask);  
2 thread.setDaemon(true);  
3 thread.start();
```

Listado 4.5: Código para crear un hilo

Al cerrar la aplicación es muy importante gestionar la terminación de las tareas en segundo plano, pues de no hacerlo estas podrían seguir ejecutándose. Para ello, las

principales tareas son añadidas a la clase *App* y esta se encarga de forzar su terminación en la función de cierre de la aplicación tal y como se muestra en el siguiente código:

```
1 public static void close(){
2     executorService.shutdown();
3     try {
4         executorService.awaitTermination(5, TimeUnit.SECONDS);
5     } catch (InterruptedException e) {
6         e.printStackTrace();
7     }
8
9     for (Task<Void> task : tasks)
10         if (task.isRunning())
11             task.cancel();
12
13     DataManager.INSTANCE.saveData();
14     Platform.exit();
15 }
```

Listado 4.6: Función que se encarga de cerrar la aplicación

En el fragmento de código anterior, *executorService* es un objeto de la clase *ScheduledExecutorService* que se encarga de llamar periódicamente a la función *isInternetAvailable*, que comprueba si hay conexión a internet y actualiza una variable booleana en la clase *App* en consecuencia.

4.6.3.2. Análisis de archivos. Biblioteca tipo 'Series'

El análisis de archivos comienza recorriendo las carpetas de la biblioteca. Por cada carpeta raíz, almacenamos en una lista sus carpetas y dividimos la búsqueda en dos hilos, uno para cada mitad de la lista.

Para las series, por cada carpeta detectada se realizará el siguiente procesamiento:

```
1 private void scanTVShow(Library library, File directory, File[] filesInDir, boolean
  updateMetadata){
2     //All video files in directory (not taking into account two subdirectories ahead, like
      Series/Folder1/Folder2/File)
3     List<File> videoFiles = new ArrayList<>();
4
5     //Analyze all files in directory except two subdirectories ahead
6     for (File file : filesInDir){
7         if (file.isDirectory()){
8             File[] filesInDirectory = file.listFiles();
9
10            if (filesInDirectory == null)
11                continue;
12
13            for (File f : filesInDirectory){
14                if (f.isFile() && validVideoFile(f))
15                    videoFiles.add(f);
16            }
17        }else{
18            if (validVideoFile(file))
19                videoFiles.add(file);
20        }
21    }
22
23    if (videoFiles.isEmpty())
24        return;
25
26    ...
```

Listado 4.7: Análisis de archivos tipo 'Series': obtención de archivos

Para comenzar, almacenamos en una lista todos los archivos de vídeo compatibles, sin tener en cuenta los archivos que estén a dos o más carpetas de profundidad. Si se han encontrado archivos, procedemos a procesarlos.

```
1 //region CREATE/EDIT SERIES
2 Series series;
3 boolean exists = false;
4
5 if (library.getAnalyzedFolders().get(directory.getAbsolutePath()) != null){
6     series = library.getSeries(library.getAnalyzedFolders().get(directory.getAbsolutePath()
7     ));
8     exists = true;
9 }else{
10     series = new Series();
11     series.setFolder(directory.getAbsolutePath());
12     library.getAnalyzedFolders().put(directory.getAbsolutePath(), series.getId());
13 }
14
15 int themdbID = series.getThemdbID();
16
17 //This means that this is a new Folder to be analyzed, so we need to search for the show metadata
18 if (themdbID == -1){
19     String finalName = directory.getName();
20     String nameWithoutParenthesis = finalName.replaceAll("[()]", "");
21
22     //Pattern to extract name and year
23     Pattern pattern = Pattern.compile("^(.*)?(?:\\s\\d{4})?$");
24
25     String year = "";
26     Matcher matcher = pattern.matcher(nameWithoutParenthesis);
27     if (matcher.matches()) {
28         finalName = matcher.group(1);
29         year = matcher.group(2);
30     }
31
32     if (year == null)
33         year = "1";
34
35     TvResultsPage tvResults = tmdbApi.getSearch().searchTv(finalName, Integer.valueOf(
36         year), library.getLanguage(), true, 1);
37
38     if (tvResults.getTotalResults() == 0)
39         return;
40
41     themdbID = tvResults.getResults().get(0).getId();
42     series.setThemdbID(themdbID);
43
44     //Set values in order to set the attributes from the search result in the new series and download
45     //images
46     updateMetadata = true;
47 }
48
49 SeriesMetadata seriesMetadata = downloadSeriesMetadata(library, themdbID);
50
51 if (seriesMetadata == null)
52     return;
53
54 if (updateMetadata)
55     setSeriesMetadataAndImages(library, series, seriesMetadata, updateMetadata);
56 //endregion
```

Listado 4.8: Análisis de archivos tipo 'Series': creación o edición de serie

Una vez tenemos los archivos, comprobamos si existe una serie con raíz en la carpeta actual. En caso de que exista, la utilizaremos de base para añadir los nuevos episodios que se detecten. Si no existe, se crea una nueva serie, se busca por el nombre de la carpeta con la API de *TheMovieDB* y se descargan los metadatos asociados al primer resultado.

Para buscar por el nombre de la carpeta, se hace uso de una expresión regular. Esta expresión detecta el nombre de la serie y el año, en caso de que se incluya en el nombre.

```
1 //Download metadata for each season of the show
2 List<SeasonMetadata> seasonsMetadata = new ArrayList<>();
3 for (SeasonMetadataBasic seasonMetadataBasic : seriesMetadata.seasons){
4     SeasonMetadata seasonMetadata = downloadSeasonMetadata(library, series.getThemdbID(),
5         seasonMetadataBasic.season_number);
6
7     if (seasonMetadata != null) {
8         seasonsMetadata.add(seasonMetadata);
9     }
10 }
11 //Download Episode Group Metadata
12 SeasonsGroupMetadata episodesGroup = null;
13 if (!series.getEpisodeGroupID().isEmpty())
14     episodesGroup = getEpisodesGroup(series);
```

Listado 4.9: Análisis de archivos tipo 'Series': descarga de metadatos para temporadas y grupos de episodios

Tras procesar la serie, procedemos a descargar los metadatos de cada una de las temporadas de esta. Además, descargamos los metadatos del grupo de episodios en caso de que dicha serie tenga un grupo asociado. Esto nos permitirá asignar las temporadas y episodios en base al grupo de episodios que el usuario ha seleccionado para la serie.

```
1 //Process each episode
2 for (File video : videoFiles){
3     if (library.getAnalyzedFiles().get(video.getAbsolutePath()) != null)
4         continue;
5
6     processEpisode(library, series, video, seasonsMetadata, episodesGroup);
7 }
8
9 //Change the name of every season to match the Episodes Group
10 if (!series.getEpisodeGroupID().isEmpty() && episodesGroup != null){
11     for (Season season : series.getSeasons()){
12         for (EpisodeGroup episodeGroup : episodesGroup.groups){
13             if (episodeGroup.order == season.getSeasonNumber()){
14                 season.setName(episodeGroup.name);
15             }
16         }
17     }
18 }
19
20 if (exists)
21     updateSeries(series);
22 else
23     addSeries(library, series);
```

Listado 4.10: Análisis de archivos tipo 'Series': procesamiento de episodios

Para cada episodio que no haya sido procesado anteriormente, se realiza un procesamiento extenso mediante la función *processEpisode()*. Dicha función es extremadamente extensa, por lo que se resumirá su procesamiento en:

- Extracción del número de temporada y episodio: se han utilizado expresiones regulares, explicadas en [27], para detectar el patrón S00E00 en el nombre del archivo, o simplemente un número en su defecto, pues los episodios también pueden nombrarse en base al número absoluto de episodio, sin hacer referencia a una temporada concreta.
- En caso de utilizar el formato de un único número de episodio, se buscará en los metadatos descargados anteriormente el episodio correspondiente y se descargarán sus metadatos.
- En caso de utilizar el formato S00E00, tendremos que comprobar primero si la serie tiene un grupo de episodios asignado. En caso de tener grupo de episodios, hemos de buscar el episodio correspondiente en dicho grupo y, una vez encontrado, obtener el episodio real en los metadatos de las temporadas que hemos descargado anteriormente.
- Tras obtener los metadatos del episodio, comprobamos si existe la temporada. Si la temporada no existe, la creamos, asignamos los metadatos y descargamos

las imágenes correspondientes.

- Para finalizar, asignamos los metadatos al episodio y descargamos la imagen de miniatura del mismo.

Tras procesar los episodios, si existe grupo de episodios asignado se procederá a renombrar cada temporada de acuerdo al nombre de la misma en el grupo de episodios. Para finalizar, se añadirá un nuevo botón para la serie en la vista, o se actualizarán los datos en caso de que existiera previamente. Una vez finalizada la tarea, se ejecutará una nueva tarea en segundo plano encargada de descargar la música y asignarla a la primera temporada de la serie.

4.6.3.3. Análisis de archivos. Biblioteca tipo 'Películas'

El procesamiento de las carpetas se realiza de la misma forma que en el análisis descrito para las series, de modo que explicaremos directamente el procesamiento que se realiza sobre cada archivo dentro de la carpeta raíz.

```
1 private void scanMovie(Library library, File f){
2     boolean exists = false;
3     if (!f.isDirectory()){
4         //region MOVIE FILE ONLY
5         if (!validVideoFile(f) || library.getAnalyzedFiles().get(f.getAbsolutePath()) !=
6             null)
7             return;
8
9         //CREATE/EDIT SERIES
10        ...
11    }else {
12        File[] filesInDir = f.listFiles();
13        if (filesInDir == null)
14            return;
15
16        List<File> folders = new ArrayList<>();
17        List<File> filesInRoot = new ArrayList<>();
18        for (File file : filesInDir){
19            if (file.isDirectory())
20                folders.add(file);
21            else
22                filesInRoot.add(file);
23        }
24
25        Series series;
26        if (library.getAnalyzedFolders().get(f.getAbsolutePath()) != null) {
27            series = library.getSeries(library.getAnalyzedFolders().get(f.getAbsolutePath())
28                ());
29            exists = true;
30
31            if (series == null)
32                return;
33        }else{
34            series = new Series();
35            series.setName(f.getName());
36            series.setFolder(f.getAbsolutePath());
37            library.getAnalyzedFolders().put(f.getAbsolutePath(), series.getId());
38        }
39
40        if (!folders.isEmpty()){
41            //FOLDERS CORRESPONDING DIFFERENT MOVIES FROM A COLLECTION
42            ...
43        }else if (!filesInRoot.isEmpty()){
44            //MOVIE FILE/CONCERT FILES INSIDE FOLDER
45            ...
46        }
47    }
48 }
```

Listado 4.11: Función de análisis de archivos para películas

El primer paso es comprobar si se trata de un archivo o un directorio. Si es un archivo, lo tomaremos como una película única y lo procesaremos como tal. Si es un directorio, puede ser una carpeta que contenga una película y/o vídeos extra, o una carpeta con una colección de películas.

Procesamiento realizado en caso de contar con un único archivo:

```
1 //region CREATE/EDIT SERIES
2 Series series = new Series();
3 series.setFolder(f.getAbsolutePath());
4 library.getAnalyzedFolders().put(f.getAbsolutePath(), series.getId());
5
6 NameYearContainer result = extractNameAndYear(f.getName().substring(0, f.getName().
    lastIndexOf(".")));
7
8 String movieName = result.name;
9 String year = result.year;
10
11 if (year.isEmpty())
12     year = "1";
13
14 //Search show by name of the folder
15 int themdbID = searchFirstMovie(library, movieName, Integer.parseInt(year));
16
17 if (themdbID == -1) {
18     series.setName(movieName);
19     Season newSeason = new Season();
20     series.addSeason(newSeason);
21     newSeason.setName(movieName);
22     newSeason.setYear(year);
23     newSeason.setSeriesID(series.getId());
24     newSeason.setSeasonNumber(series.getSeasons().size());
25
26     library.getAnalyzedFolders().put(f.getAbsolutePath(), series.getId());
27
28     saveDiscWithoutMetadata(library, f, newSeason);
29     addSeries(library, series);
30     return;
31 }
32
33 MovieMetadata movieMetadata = downloadMovieMetadata(library, themdbID);
34
35 series.setName(movieMetadata.title);
36 Season season = new Season();
37 series.addSeason(season);
38 season.setFolder(f.getAbsolutePath());
39 season.setSeriesID(series.getId());
40 season.setThemdbID(themdbID);
41 season.setImdbID(movieMetadata.imdb_id);
42
43 if (season.getImdbID() == null)
44     season.setImdbID("");
45
46 season.setName(movieMetadata.title);
47 season.setOverview(movieMetadata.overview);
48 season.setYear(movieMetadata.release_date.substring(0, movieMetadata.release_date.indexOf
    ("-")));
```

```
49 season.setScore(((float) ((int) (movieMetadata.vote_average * 10.0)) / 10.0f));
50 season.setSeasonNumber(series.getSeasons().size() + 1);
51
52 List<Genre> genres = movieMetadata.genres;
53
54 if (genres != null){
55     List<String> genreList = new ArrayList<>();
56     for (Genre genre : genres){
57         genreList.add(genre.name);
58     }
59
60     season.setGenres(genreList);
61 }
62
63 library.getSeasonFolders().put(f.getAbsolutePath(), season.getId());
64
65 Images images = downloadImages(library, season.getThemdbID());
66
67 if (images != null){
68     loadImages(library, season.getId(), images, season.getThemdbID(), false);
69     loadImages(library, series.getId(), images, season.getThemdbID(), true);
70 }
71
72 File posterDir = new File("resources/img/seriesCovers/" + season.getId() + "/0.jpg");
73 if (posterDir.exists()){
74     season.setCoverSrc("resources/img/seriesCovers/" + season.getId() + "/0.jpg");
75 }
76
77 posterDir = new File("resources/img/seriesCovers/" + series.getId() + "/0.jpg");
78 if (posterDir.exists()){
79     series.setCoverSrc("resources/img/seriesCovers/" + season.getId() + "/0.jpg");
80 }
81
82 downloadLogos(library, series, season, season.getThemdbID());
83 saveBackground(season, "resources/img/DownloadCache/" + season.getThemdbID() + ".jpg");
84
85 processMovie(library, f, season, movieMetadata.runtime);
86
87 addSeries(library, series);
88 //endregion
```

Listado 4.12: Función de análisis de archivos para películas: código para crear la película

Comenzamos creando la serie y buscando los metadatos. Al no contar con una carpeta, utilizamos el nombre del archivo para buscar en la API de *TheMovieDB*, extrayendo el nombre mediante la función ***extractNameAndYear()***.

En caso de no encontrar coincidencias, la variable ***themdbID*** será igual a -1, por lo que habremos de crear la temporada, añadir el episodio y agregar la serie a la vista; todo ello sin utilizar metadatos descargados.

Si se ha encontrado la película, descargamos y asignamos los metadatos de esta a la serie, creamos la temporada y asignamos los mismos datos. Tras tener serie y temporada, ambas correspondiendo a la misma película, descargamos las imágenes correspondientes y llamamos a la función ***processMovie()***, que se encarga de crear el episodio, asignarle los metadatos y descargar las imágenes de miniatura. Para finalizar, añadimos la serie a la vista.

El procesamiento en caso de ser un directorio sería el siguiente:

- Almacenar en dos listas las carpetas y los archivos que contiene el directorio.
- Comprobar si existe una serie con dicho directorio raíz, y crearla en caso contrario.
- En caso de contar con carpetas, lo tomaremos como una colección y procesaremos el directorio como tal. Se procesará cada carpeta del directorio como si fuera una película. El nombre utilizado en la búsqueda será el nombre de la carpeta, separando nombre de la película y año mediante el uso de una expresión regular en la función ***extractNameAndYear()***. El procesamiento posterior será exactamente igual al descrito en caso de tener un archivo y no un directorio.
- En caso de no haber carpetas, sino archivos, lo tomaremos como una película y no una colección. Por tanto, el procesamiento será igual al descrito anteriormente, utilizando el nombre del directorio para buscar la película con la API de ***TheMovieDB*** y descargando los metadatos necesarios tal y como se ha descrito anteriormente en este apartado.

4.6.3.4. Corrección de identificación para series

La corrección de identificación se encarga de borrar los datos pertenecientes a una serie y descargar los datos correspondientes a la nueva identificación asignada por el usuario.

```
1 public void correctIdentificationShow(){
2     showBackgroundShadow();
3     downloadingContentTextStatic.setText(App.textBundle.getString("downloadingMessage"));
4     downloadingContentWindowStatic.setVisible(true);
5     Task<Void> correctIS = new Task<>() {
6         @Override
7         protected Void call() {
8             File folder = new File(selectedSeries.getFolder());
9             File[] filesInFolder = folder.listFiles();
10
11             if (filesInFolder == null)
12                 return null;
13
14             DataManager.INSTANCE.deleteSeriesData(selectedSeries);
15             selectedSeries.getSeasons().clear();
16
17             scanTVShow(currentLibrary, folder, filesInFolder, true);
18
19             if (selectedSeries.getSeasons().isEmpty())
20                 return null;
21
22             Season season = selectedSeries.getSeasons().get(0);
23
24             //Process background music
25             List<YoutubeVideo> results = searchYoutube(season.getName() + "▯main▯theme");
26
27             if (results != null)
28                 downloadMedia(season, results.get(0).watch_url);
29
30             return null;
31         }
32     };
33
34     correctIS.setOnSucceeded(e -> {
35         downloadingContentWindowStatic.setVisible(false);
36         hideBackgroundShadow();
37         refreshSeries();
38     });
39     correctIS.setOnFailed(e -> {
40         downloadingContentWindowStatic.setVisible(false);
41         hideBackgroundShadow();
42         refreshSeries();
43     });
44
45     Thread thread = new Thread(correctIS);
46     thread.setDaemon(true);
47     thread.start();
48 }
```

Listado 4.13: Función para corregir identificación de serie

Comenzamos mostrando un mensaje al usuario mientras se descargan los datos y creando una tarea a ejecutar en segundo plano. Esta tarea se encarga de borrar los datos de la serie y volver a realizar el análisis de archivos sobre la carpeta raíz de dicha serie mediante la función ***scanTVShow()*** que hemos descrito en apartados anteriores. Una vez tenemos la serie, descargamos la canción de fondo y la asignamos a la primera temporada de esta.

Al finalizar la tarea, se programa el cierre de la ventana flotante y se ejecuta una función que actualiza el contenido de la serie en la vista para reflejar la nueva identificación.

4.6.3.5. Corrección de identificación para películas

La corrección de identificación en películas se realiza sobre una temporada, pues la información de la película se guarda en la temporada en caso de contar con una colección de películas. Por ello, el procesamiento se realiza directamente en la función de corrección de identificación; a diferencia de la corrección de identificación de series, que llama a la función de análisis de archivos.

```
1 public void correctIdentificationMovie(){
2     showBackgroundShadow();
3     downloadingContentTextStatic.setText(App.textBundle.getString("downloadingMessage"));
4     downloadingContentWindowStatic.setVisible(true);
5     Task<Void> correctIM = new Task<>() {
6         @Override
7         protected Void call() {
8             MovieMetadata metadata = downloadMovieMetadata(currentLibrary, selectedSeason
9                 .getThemdbID());
10
11             if (metadata == null)
12                 return null;
13
14             selectedSeason.setName(metadata.title);
15             selectedSeason.setOverview(metadata.overview);
16             selectedSeason.setYear(metadata.release_date.substring(0, metadata.
17                 release_date.indexOf("-")));
18             selectedSeason.setScore(((float) ((int) (metadata.vote_average * 10.0)) / 10.0
19                 f));
20             selectedSeason.setImdbID(metadata.imdb_id);
21
22             if (selectedSeason.getImdbID() == null)
23                 selectedSeason.setImdbID("");
24
25             Images images = downloadImages(currentLibrary, selectedSeason.getThemdbID());
26
27             if (images != null)
28                 loadImages(currentLibrary, selectedSeason.getId(), images, selectedSeason
29                     .getThemdbID(), false);
```

```
27 File posterDir = new File("resources/img/seriesCovers/" + selectedSeason.  
28     getId() + "/0.jpg");  
29 if (posterDir.exists()){  
30     selectedSeason.setCoverSrc("resources/img/seriesCovers/" + selectedSeason.  
31         getId() + "/0.jpg");  
32  
33     if (selectedSeries.getSeasons().size() == 1 || selectedSeries.getCoverSrc()  
34         ().equals("resources/img/DefaultPoster.png") || selectedSeries.  
35         getCoverSrc().isEmpty()){  
36         posterDir = new File("resources/img/seriesCovers/" + selectedSeason.  
37             getId() + "/0.jpg");  
38         if (posterDir.exists()){  
39             Path sourcePath = Paths.get(posterDir.toURI());  
40             Path destinationPath = Paths.get("resources/img/seriesCovers/" +  
41                 selectedSeries.getId() + "/0.jpg");  
42  
43             try{  
44                 Files.createDirectories(Paths.get("resources/img/seriesCovers  
45                     /" + selectedSeries.getId() + "/"));  
46             } catch (IOException e) {  
47                 System.err.println("correctIdentificationMovie: series covers  
48                     folder could not be created");  
49             }  
50  
51             try {  
52                 Files.copy(sourcePath, destinationPath, StandardCopyOption.  
53                     REPLACE_EXISTING);  
54             } catch (IOException e) {  
55                 System.err.println("correctIdentificationMovie: error copying  
56                     season cover image");  
57             }  
58  
59             selectedSeries.setCoverSrc("resources/img/seriesCovers/" +  
60                 selectedSeries.getId() + "/0.jpg");  
61         }  
62     }  
63 }  
64  
65 downloadLogos(currentLibrary, selectedSeries, selectedSeason, selectedSeason.  
66     getThemdbID());  
67 saveBackground(selectedSeason, "resources/img/DownloadCache/" +  
68     selectedSeason.getThemdbID() + ".jpg");  
69  
70 //Process background music  
71 List<YoutubeVideo> results = searchYoutube(selectedSeason.getName() + "main  
    theme");  
72  
73 if (results != null)  
74     downloadMedia(selectedSeason, results.get(0).watch_url);  
75  
76 if (selectedSeason.getEpisodes().size() == 1){  
77     Episode episode = selectedSeason.getEpisodes().get(0);  
78     episode.setName(selectedSeason.getName());  
79 }  
80  
81 for (Episode episode : selectedSeason.getEpisodes()){  
82     if (episode == null)  
83         continue;
```

```
72         episode.setScore(selectedSeason.getScore());
73         episode.setOverview(selectedSeason.getOverview());
74         episode.setYear(selectedSeason.getYear());
75         episode.setSeasonNumber(selectedSeason.getSeasonNumber());
76         episode.setRuntime(metadata.runtime);
77
78         if (!selectedSeason.getImdbID().isEmpty())
79             setIMDBScore(selectedSeason.getImdbID(), episode);
80
81         setMovieThumbnail(episode, selectedSeason.getThemdbID());
82     }
83
84     refreshSeason(selectedSeason);
85     return null;
86 }
87 };
88
89 correctIM.setOnSucceeded(e -> {
90     downloadingContentWindowStatic.setVisible(false);
91     hideBackgroundShadow();
92 });
93 correctIM.setOnFailed(e -> {
94     downloadingContentWindowStatic.setVisible(false);
95     hideBackgroundShadow();
96 });
97
98 Thread thread = new Thread(correctIM);
99 thread.setDaemon(true);
100 thread.start();
101 }
```

Listado 4.14: Función para corregir identificación de película

Al igual que la función de corregir identificación de series, esta comienza mostrando un mensaje al usuario para indicarle que se están descargando los nuevos metadatos. En la tarea a ejecutar en segundo plano, esta vez no borramos el contenido, sino que descargamos los metadatos de la película con la nueva identificación y reemplazamos los actuales. Además, descargamos las imágenes de fondo, logos y pósters; además de la música de fondo.

Si tenemos únicamente un vídeo, este será el de la película, por lo que cambiamos el nombre del mismo por el de la película. Para terminar, asignamos los datos de la película a los vídeos y descargamos las miniaturas.

Si la película tiene asociado un ID de **IMDB**, se realizará un pequeño scraping para obtener la puntuación de esta.

4.6.4. Modo pantalla completa

El modo pantalla completa se centra en mostrar el contenido, por lo que las funciones más importantes son las relacionadas con la interacción del usuario. Por ello, nos centraremos en la movilidad entre series y episodios, así como en la reproducción automática de música y vídeos de fondo.

4.6.4.1. Movimiento de scroll para series

```
1 btn.focusedProperty().addListener((obs, oldVal, newVal) -> {
2     if (newVal) {
3         playInteractionSound();
4
5         ScaleTransition scaleTransition = new ScaleTransition(Duration.seconds(0.15), btn
6             );
7         scaleTransition.setToX(1.1);
8         scaleTransition.setToY(1.1);
9
10        scaleTransition.setOnFinished(e -> {
11            CompletableFuture.runAsync(() -> selectSeries(series.get(seriesButtons.
12                indexOf(btn)))));
13        });
14
15        scaleTransition.play();
16        int index = (seriesButtons.indexOf(btn) / rowSize);
17
18        if (index != scrollPane.getVvalue()) {
19            Bounds cardBounds = btn.localToScene(btn.getBoundsInLocal());
20            Bounds scrollBounds = scrollPane.localToScene(scrollPane.getBoundsInLocal());
21
22            //Check if the button is outside the view
23            if (cardBounds.getMinY() < scrollBounds.getMinY() || cardBounds.getMaxY() >
24                scrollBounds.getMaxY()){
25                Timeline timeline = new Timeline();
26                KeyValue keyValue = new KeyValue(scrollPane.vvalueProperty(), index);
27                KeyFrame keyFrame = new KeyFrame(Duration.seconds(0.1), keyValue);
28                timeline.getKeyFrames().add(keyFrame);
29                timeline.play();
30            }
31        }
32    }else{
33        ScaleTransition scaleTransition = new ScaleTransition(Duration.seconds(0.15), btn
34            );
35        scaleTransition.setToX(1);
36        scaleTransition.setToY(1);
37
38        scaleTransition.play();
39    }
40 });
```

Listado 4.15: Fragmento de código que controla el movimiento del scroll al seleccionar una serie

Este fragmento de código pertenece a la función **addCard()** que se encarga de crear un botón con la imagen de póster de la serie. Al hacer foco en el botón, este aumentará su tamaño y su CSS asociado también se verá modificado. Además, en el fragmento anterior se puede observar el código que comprueba si el botón está fuera de pantalla. De estarlo, el panel que contiene todos los botones de las series se movería con una animación imitando el efecto *scroll* hasta que el botón esté en pantalla.

```
1 btn.addEventHandler(KeyEvent.KEY_PRESSED, (KeyEvent event) ->{
2     if (App.pressedSelect(event)){
3         if (selectedSeries != null) {
4             playCategoriesSound();
5             showSeason(selectedSeries);
6         }
7     }
8
9     int index = seriesButtons.indexOf(btn);
10
11     if (App.pressedUp(event)){
12         if (index < rowSize)
13             librariesBox.getChildren().get(libraries.indexOf(currentLibrary)).
14                 requestFocus();
15         else
16             seriesButtons.get(seriesButtons.indexOf(btn) - rowSize).requestFocus();
17     }else if (App.pressedDown(event)){
18         if (index + rowSize < seriesButtons.size())
19             seriesButtons.get(seriesButtons.indexOf(btn) + rowSize).requestFocus();
20     }else if (App.pressedLeft(event)){
21         if (index > 0)
22             seriesButtons.get(seriesButtons.indexOf(btn) - 1).requestFocus();
23     }else if (App.pressedRight(event)){
24         if (index < seriesButtons.size() - 1)
25             seriesButtons.get(seriesButtons.indexOf(btn) + 1).requestFocus();
26     }
27 }}
```

Listado 4.16: Fragmento de código que controla el foco de los botones de series

El código anterior va justo después del que controla el comportamiento del botón al hacer foco en este. Este código se encarga de controlar la interacción del usuario, permitiendo seleccionar la serie que tiene el foco si se pulsa **Intro**. Además de esto, permite el uso de las flechas del teclado para moverse entre los diferentes botones en pantalla, siendo estos los botones pertenecientes a las series de la biblioteca, y los botones que representan las diferentes bibliotecas.

4.6.4.2. Movimiento de scroll para episodios

```
1 private void handleButtonFocus(Button focusedButton) {
2     double screenCenter, buttonCenterX, offset, finalPos;
3
4     if (episodeButtons.indexOf(focusedButton) <= buttonCount / 2 || episodeButtons.size()
5         <= 3 || episodeButtons.size() == buttonCount){
6         finalPos = 0;
7     }else{
8         //Get center of screen
9         screenCenter = Screen.getPrimary().getBounds().getWidth() / 2;
10
11         //Check aspect ratio to limit the right end of the button list
12         double aspectRatio = Screen.getPrimary().getBounds().getWidth() / Screen.
13             getPrimary().getBounds().getHeight();
14         int maxButtons = (episodeButtons.size() - (buttonCount / 2));
15         if (aspectRatio > 1.8)
16             maxButtons--;
17
18         if (episodeButtons.indexOf(focusedButton) >= (episodeButtons.size() - (
19             buttonCount / 2)))
20             focusedButton = episodeButtons.get(Math.max(2, maxButtons));
21
22         //Get center of button in the screen
23         Bounds buttonBounds = focusedButton.localToScene(focusedButton.getBoundsInLocal()
24             );
25         buttonCenterX = buttonBounds.getMinX() + buttonBounds.getWidth() / 2;
26
27         //Calculate offset
28         offset = screenCenter - buttonCenterX;
29
30         //Calculate position
31         finalPos = cardContainer.getTranslateX() + offset;
32     }
33
34     //Translation with animation
35     TranslateTransition transition = new TranslateTransition(Duration.seconds(0.2),
36         cardContainer);
37     transition.setToX(finalPos);
38     transition.play();
39 }
```

Listado 4.17: Función que controla el foco de un botón de episodio

Esta función se encarga que mover el panel que contiene la lista de episodios de una temporada para simular el comportamiento de un **ScrollPane**. El comportamiento se basa en:

- Si el movimiento es hacia la derecha: se comprueba si el siguiente botón está más allá del centro de la pantalla y se ejecuta una animación para mover el panel y centrar dicho botón.
- Si el movimiento es hacia la izquierda: se comprueba si el siguiente botón está

situado a la izquierda del centro de la pantalla y se ejecuta una animación para centrar dicho botón, moviendo el panel en consecuencia.

4.6.4.3. Reproducción de música y vídeo de fondo

```
1 private void setMediaPlayer(){
2     Task<Void> mpTask = new Task<>() {
3         @Override
4         protected Void call() {
5             int volume = Integer.parseInt(Configuration.loadConfig("backgroundVolume", "
6                 0.4"));
7             mp.setVolume((double) volume / 100);
8             mp.setOnEndOfMedia(() -> stopPlayer());
9
10            double delay = 0.5;
11            if (isVideo)
12                delay = Double.parseDouble(Configuration.loadConfig("backgroundDelay", "3
13                    "));
14            timeline = new javafx.animation.Timeline(
15                new javafx.animation.KeyFrame(Duration.seconds(delay), event ->
16                    playBackgroundMedia())
17            );
18            timeline.play();
19            return null;
20        }
21    };
22
23    Thread thread = new Thread(mpTask);
24    thread.setDaemon(true);
25    thread.start();
26
27    App.tasks.add(mpTask);
28 }
```

Listado 4.18: Función que inicializa el reproductor multimedia nativo

Esta función inicializa el reproductor multimedia de JavaFX. Para ello, hacemos uso de las clases **Task** y **Thread** como ya hemos explicado anteriormente, para ejecutar el código en un nuevo hilo. Este reproductor será encargado de reproducir música o vídeo, según el contenido presente en la temporada mostrada por la vista, y se inicializará con un volumen establecido por un parámetro modificable desde el menú de preferencias de la aplicación.

```
1 private void findAndPlaySong(){
2     for (Season season : seasons){
3         if (!season.getMusicSrc().isEmpty()){
4             Platform.runLater(() -> {
5                 File file = new File(season.getMusicSrc());
6                 Media media = new Media(file.toURI().toString());
7                 mp = new MediaPlayer(media);
8                 isVideo = false;
9
10                setMediaPlayer();
11            });
12            break;
13        }
14    }
15 }
```

Listado 4.19: Función que busca una canción dentro de las temporadas de una serie

La función ***findAndPlaySong()*** es llamada cuando la temporada no tiene vídeo de fondo asociado y el usuario ha marcado la casilla **Reproducir la misma música en todas las temporadas**, la cual es marcada por defecto al añadir una serie, pues es más óptimo descargar únicamente una canción por serie de manera automática. La función en cuestión se encarga de buscar por orden en cada temporada de una serie si tiene una canción asociada. De tenerla, se ejecutaría el reproductor multimedia con esa canción.

```
1 private void processBackgroundMedia(){
2     if (!seasons.get(currentSeason).getVideoSrc().isEmpty()){
3         if (mp != null)
4             mp.stop();
5
6         File file = new File(seasons.get(currentSeason).getVideoSrc());
7         Media media = new Media(file.toURI().toString());
8         mp = new MediaPlayer(media);
9         backgroundVideo.setMediaPlayer(mp);
10        backgroundVideo.setVisible(false);
11        isVideo = true;
12
13        setMediaPlayer();
14    }else if (!playSameMusic && !seasons.get(currentSeason).getMusicSrc().isEmpty()){
15        File file = new File(seasons.get(currentSeason).getMusicSrc());
16        Media media = new Media(file.toURI().toString());
17        mp = new MediaPlayer(media);
18        isVideo = false;
19
20        setMediaPlayer();
21    }
22 }
```

Listado 4.20: Función que asigna el contenido al reproductor nativo

La función anterior se encarga de procesar el contenido multimedia asociado a la temporada y cargarlo en el reproductor, para posteriormente reproducirlo según si es un vídeo o una canción.

```
1 private void playBackgroundMedia(){
2     Platform.runLater(() ->{
3         if (isVideo){
4             double screenRatio = Screen.getPrimary().getBounds().getWidth() / Screen.
5                 getPrimary().getBounds().getHeight();
6             double mediaRatio = (double) backgroundVideo.getMediaPlayer().getMedia().
7                 getWidth() / backgroundVideo.getMediaPlayer().getMedia().getHeight();
8
9             backgroundVideo.setPreserveRatio(true);
10
11             if (screenRatio > 1.8f && mediaRatio > 1.8f){
12                 backgroundVideo.setPreserveRatio(false);
13             }
14
15             backgroundVideo.setVisible(true);
16
17             //Fade In Transition
18             FadeTransition fadeIn = new FadeTransition(Duration.seconds(1),
19                 backgroundVideo);
20             fadeIn.setFromValue(0);
21             fadeIn.setToValue(1.0);
22             fadeIn.play();
23             mp.stop();
24             mp.seek(mp.getStartTime());
25             mp.play();
26         }else{
27             if (mp != null) {
28                 mp.stop();
29                 mp.seek(mp.getStartTime());
30                 mp.play();
31             }
32         }
33     });
34 }
```

Listado 4.21: Función que reproduce la música o vídeo de fondo de una temporada

Esta función reproduce el vídeo o la música de fondo. En caso de que la temporada tenga un vídeo asociado, este tendrá prioridad y se iniciará una animación para mostrarlo en la vista.

4.6.5. Reproductor de vídeo

La aplicación requiere de un reproductor de vídeo integrado, por lo que se han analizado las distintas opciones compatibles con JavaFX hasta encontrar la idónea.

4.6.5.1. Reproductores compatibles con JavaFX

La primera opción fue el reproductor nativo de JavaFX, MediaPlayer [28]. Este reproductor viene integrado en el propio framework y tiene una integración simple e intuitiva con las vistas FXML. Sin embargo, únicamente es compatible con vídeos en formato MP4, por lo que esta opción se descartó rápidamente.



Figura 4.25: Logo de VLC

El siguiente reproductor fue VideoLAN VLC [29]. Este reproductor es compatible con la mayoría de formatos de audio, vídeo y subtítulos, y cuenta con una librería [30] para Java con integración en JavaFX. El reproductor se integró en el proyecto y se consiguió que funcionara, pero apareció un problema: el contenido HDR (High Dynamic Range) no se procesaba como tal. Los vídeos con HDR se mostraban de manera incorrecta, por lo que se optó por buscar una alternativa.

4.6.5.2. Reproductor MPV



Figura 4.26: Logo de MPV

El reproductor MPV [31] funciona por comandos, por lo que no cuenta con una librería en Java que facilite las funciones de interacción con el mismo. De hecho, para hacerlo funcionar en JavaFX se ha necesitado de un código en github que proporciona una interfaz que hace uso de JNA (Java Native Access) [32] para acceder a las funciones de libmpv.dll (el reproductor MPV) y mover el vídeo directamente a un Stage de JavaFX.

Al no contar con una librería específica, se ha tenido que desarrollar una clase Java que proporcione las funciones de interacción con el reproductor. Esto ha sido posible gracias al manual de MPV [33], que documenta las funciones del mismo. Así mismo, se ha tenido que crear la interfaz mediante SceneBuilder [11] para que el usuario interactúe con el reproductor.

4.6.5.3. Implementación de MPV

```
1 import com.sun.jna.Native;
2 import com.sun.jna.Pointer;
3 import com.sun.jna.Structure;
4 import com.sun.jna.win32.StdCallLibrary;
5
6 public interface MPV extends StdCallLibrary {
7     MPV INSTANCE = Native.load("resources/lib/libmpv-2.dll", MPV.class);
8     ...
9
10    long mpv_create();
11    int mpv_initialize(long handle);
12    int mpv_command(long handle, String[] args);
13    int mpv_command_string(long handle, String args);
14    Pointer mpv_get_property_string(long handle, String name);
15    int mpv_set_property_string(long handle, String name, String data);
16    ...
17 }
```

Listado 4.22: Interfaz MPV

La interfaz MPV realiza la carga del reproductor mediante **JNA**.

```
1 public void playVideo(String src, long seekTimeMillis) {
2     //Get interface to MPV DLL
3     MPV mpv = MPV.INSTANCE;
4
5     //Create MPV player instance
6     handle = mpv.mpv_create();
7
8     //Get the native window id by looking up a window by title:
9     WinDef.HWND hwnd = User32.INSTANCE.FindWindow(null, App.textBundle.getString("season"
10     ));
11
12     //Tell MPV on which window video should be displayed:
13     LongByReference longByReference =
14         new LongByReference(Pointer.nativeValue(hwnd.getPointer()));
15     mpv.mpv_set_option(handle, "wid", 4, longByReference.getPointer());
16
17     int error;
18
19     //Initialize MPV after setting basic options:
20     if((error = mpv.mpv_initialize(handle)) != 0) {
21         throw new IllegalStateException("Initialization failed with error: " + error);
22     }
23
24     //Load and play a video:
25     if((error = mpv.mpv_command(handle, new String[] {"loadfile", src})) != 0) {
26         throw new IllegalStateException("Playback failed with error: " + error);
27     }
28
29     ...
30 }
```

Listado 4.23: Fragmento de código que crea el reproductor MPV

En la clase **VideoPlayer** se llama a la función **playVideo**, encargada de crear la instancia del reproductor MPV y asociarlo con una ventana de JavaFX para mostrarlo sobre una vista. Para asociarlo a una vista se hace uso del nombre de la ventana, en este caso dicho texto está contenido en los archivos de idiomas y lo podemos obtener mediante **App.textBundle.getString("season")**.

```
1 public void seekForward() {
2     mpvCommand("seek", "10");
3 }
4 public void seekBackward() {
5     mpvCommand("seek", "-5");
6 }
7 public void seekToTime(long milliseconds) {
8     mpvCommand("seek", String.valueOf(milliseconds / 1000.0));
9 }
10 public void fixZoom(double zoomValue) {
11     mpvSetProperty("video-zoom", Double.toString(zoomValue));
12 }
```

Listado 4.24: Funciones para controlar el reproductor MPV

Para utilizar el reproductor en la vista, se han creado funciones en la clase **VideoPlayer** que nos permiten interactuar con la interfaz MPV y ejecutar comandos de control.

4.6.5.4. Patrón observador

```
1 public class VideoPlayer extends MediaView {
2     ...
3     private ScheduledExecutorService clockExecutor;
4     ...
5     public void pauseClock() {
6         if (clockExecutor != null && !clockExecutor.isShutdown()) {
7             clockExecutor.shutdown();
8         }
9     }
10    public void startClock() {
11        if (clockExecutor == null || clockExecutor.isShutdown()) {
12            clockExecutor = Executors.newSingleThreadScheduledExecutor();
13            int UPDATE_INTERVAL = 100;
14            clockExecutor.scheduleAtFixedRate(() -> Platform.runLater(this::
15                updateCurrentTime), 0, UPDATE_INTERVAL, TimeUnit.MILLISECONDS);
16        }
17        private void updateCurrentTime() {
18            long newCurrentTime = getCurrentTime();
19            if (newCurrentTime != currentTimeMillis) {
20                currentTimeMillis = newCurrentTime;
21                parentController.notifyChanges(currentTimeMillis);
22            }
23
24            if (getDuration() - newCurrentTime <= 200 && !nextVideo){
25                nextVideo = true;
26                parentController.nextEpisode();
27            }
28        }
29        ...
30    }
```

Listado 4.25: Funciones para controlar el reproductor MPV

Para implementar el patrón observador se ha utilizado la clase **ScheduledExecutorService**. Esta clase nos permite ejecutar un código periódicamente, gracias a lo cual podemos llamar a la función **updateCurrentTime()** cada milisegundo y actualizar en tiempo real el tiempo que lleva el reproductor ejecutándose. En dicha función, llamamos al controlador de la vista del reproductor y le notificamos los cambios, permitiendo así que la vista se modifique en tiempo real, actualizando el valor del slider del reproductor.

4.7. Pruebas

4.7.1. Introducción

Para garantizar la calidad y funcionalidad de la aplicación, se ha implementado un enfoque iterativo en la realización de pruebas, involucrando tanto al desarrollador como a un grupo limitado de usuarios. A continuación, se detalla el proceso seguido:

- **Pruebas del desarrollador:** durante el desarrollo, el propio desarrollador llevó a cabo pruebas exhaustivas tras la finalización de cada tarea. Estas pruebas incluyeron la verificación de la funcionalidad implementada, la detección de posibles fallos y su corrección inmediata. Cada tarea se consideraba completada y se movía a la columna 'Done' en el tablero Kanban de Trello solo después de superar estas pruebas internas. Este enfoque ayudó a asegurar que cada componente de la aplicación funcionara correctamente antes de avanzar al siguiente.
- **Pruebas por usuarios:** se contó con la participación de un número limitado de usuarios que probaron la aplicación en distintas etapas del desarrollo. Estos usuarios proporcionaron retroalimentación valiosa, especialmente durante las etapas finales del proyecto, cuando la aplicación estaba más cercana a su versión final. Los usuarios destacaron los fallos que detectaron, permitiendo su solución antes del lanzamiento final. Esta retroalimentación fue crucial para identificar problemas que podrían no haber sido detectados por el desarrollador, proporcionando una perspectiva externa sobre la usabilidad y funcionalidad de la aplicación.

El proceso de pruebas llevado a cabo ha sido efectivo para detectar y corregir una amplia gama de problemas, contribuyendo significativamente a la estabilidad y usabilidad de la aplicación final.

4.7.2. Ejemplos de pruebas realizadas

Dado que se han realizado pruebas por cada tarea implementada, en esta sección mostraremos un subconjunto de estas, destacando errores encontrados y las soluciones a estos. Las pruebas se han realizado ejecutando la aplicación mediante la terminal, para así poder observar las excepciones producidas y solucionar algunos fallos de manera más eficaz y rápida.

4.7.2.1. Prueba de corrección de identificación: series

Esta prueba se realiza para comprobar que la identificación de una serie se corrige de manera correcta.

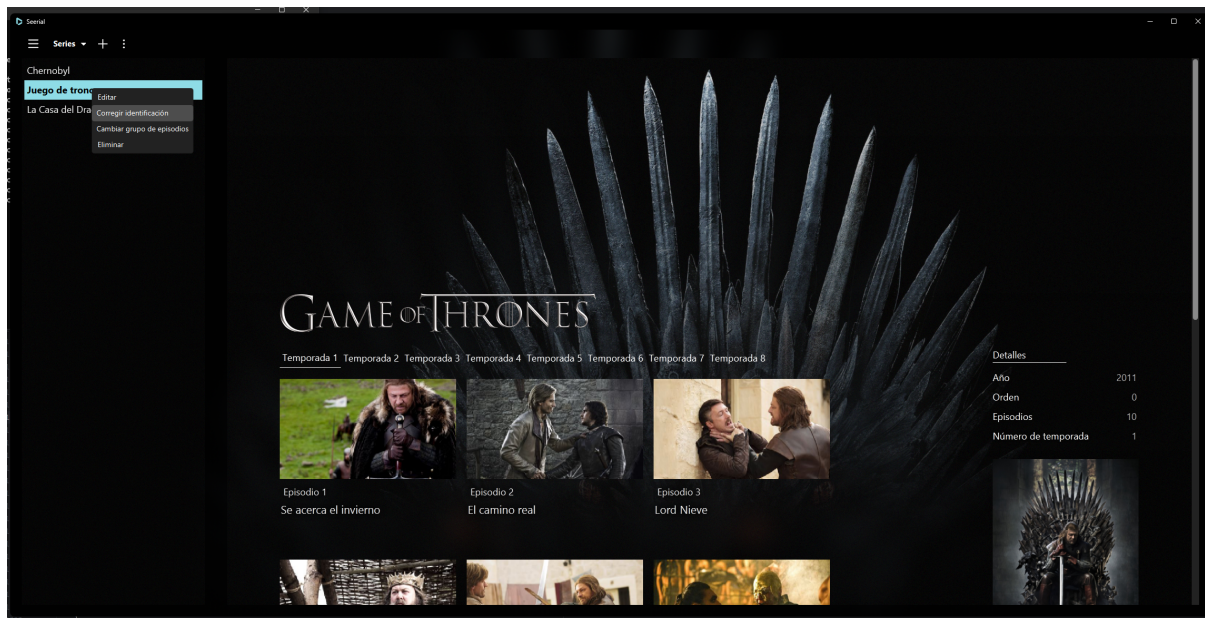


Figura 4.27: Test de corrección de identificación de series: 1

La prueba comienza en el modo escritorio, donde seleccionaremos una serie y con el botón derecho del ratón abriremos su menú contextual (fig. 4.27). Una vez abierto, seleccionaremos la opción '*Corregir Identificación*'.

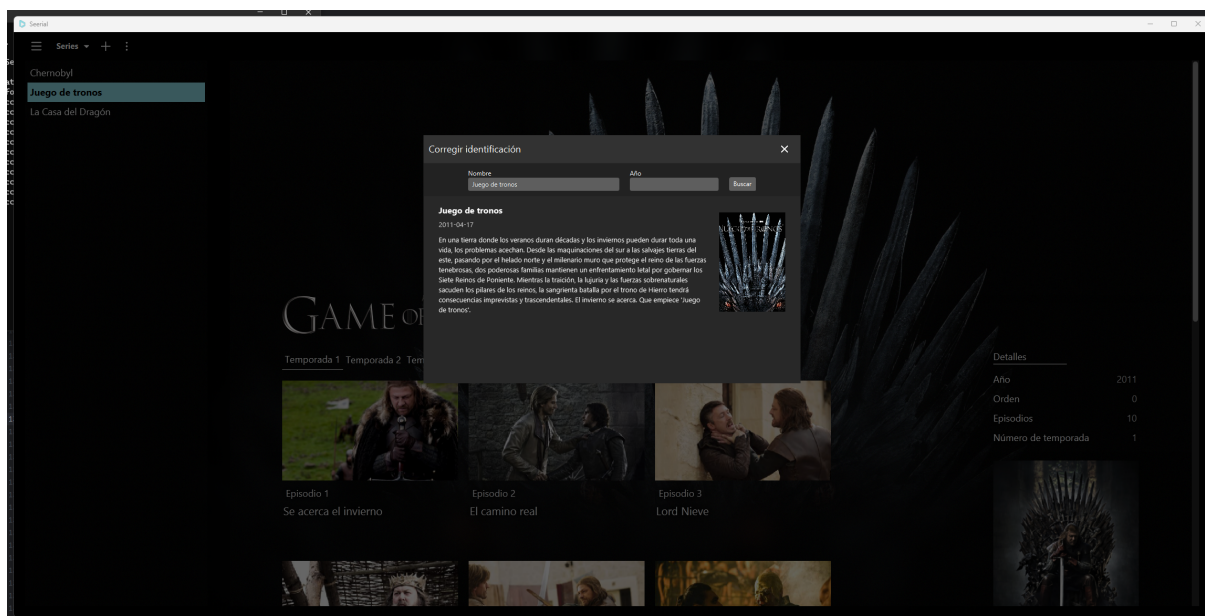


Figura 4.28: Test de corrección de identificación de series: 2

Ahora se nos muestra una nueva ventana (fig. 4.28). Esta busca automáticamente

por el nombre de la serie y nos muestra los resultados. Sin embargo, para la prueba nos interesa buscar una serie distinta para aplicar una nueva identificación.

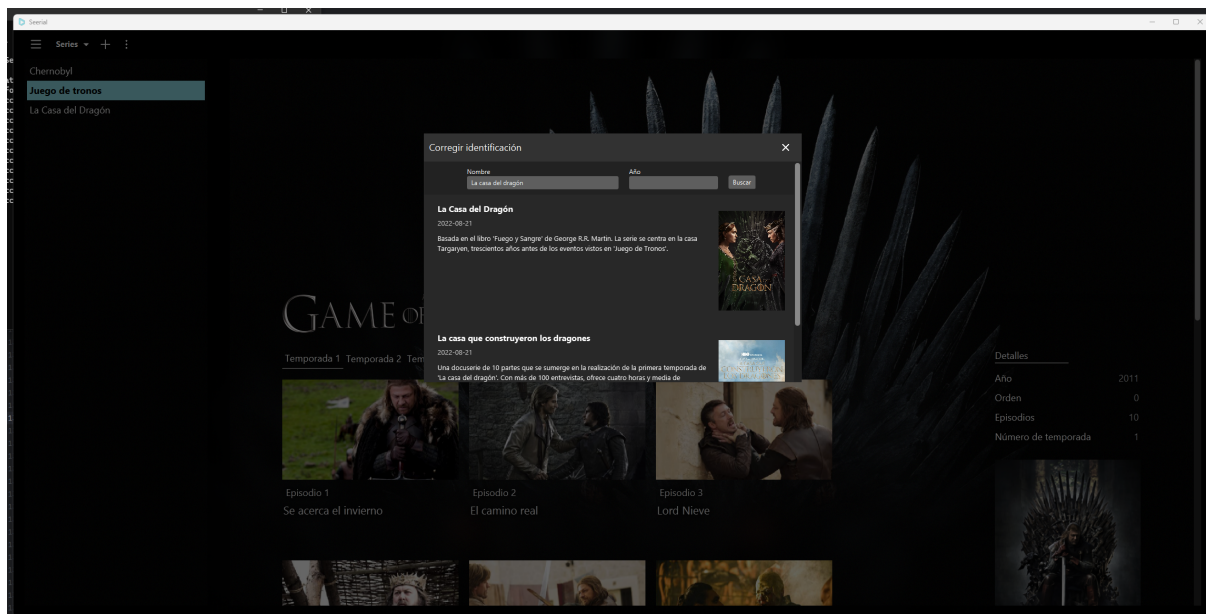


Figura 4.29: Test de corrección de identificación de series: 3

Una vez buscamos otra serie, seleccionamos la primera opción (fig. 4.29). El comportamiento esperado es la aparición de una nueva ventana que nos indique que se está realizando dicha corrección.



Figura 4.30: Test de corrección de identificación de series: 4

Como podemos observar, dicha ventana no se muestra y, en su lugar, se nos muestra la pantalla principal con el contenido de la serie eliminado (fig. 4.30).

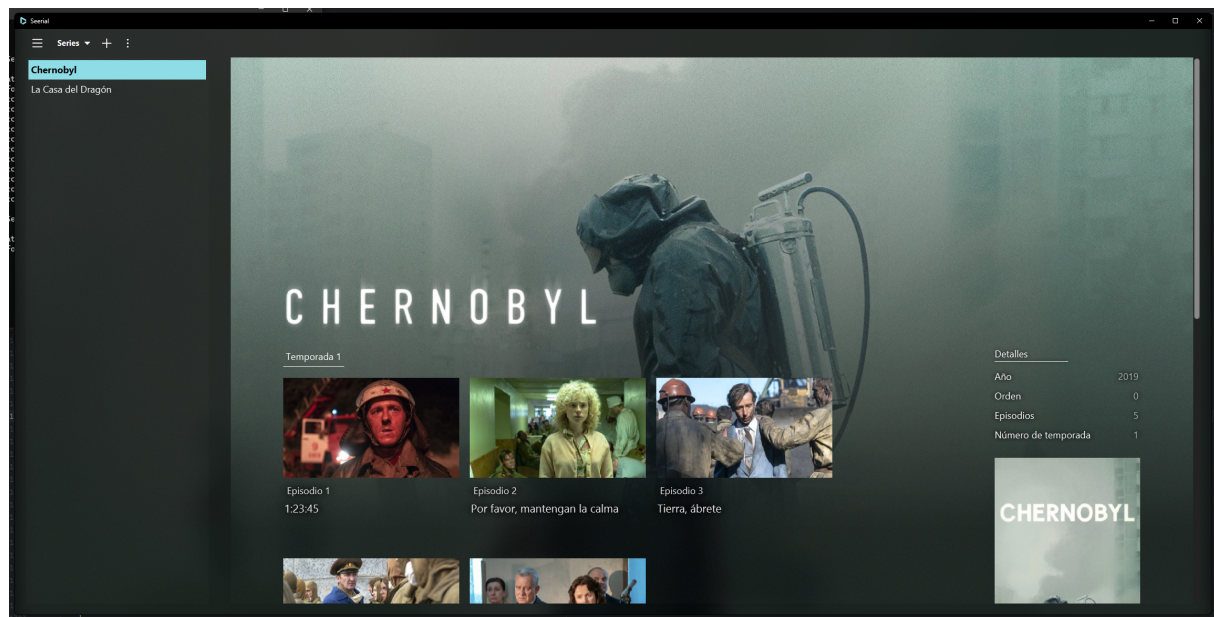


Figura 4.31: Test de corrección de identificación de series: 5

Cuando reiniciamos la aplicación, la serie desaparece (fig. 4.31). De aquí podemos deducir, conociendo el código, que el procesamiento previo a la corrección de identificación está funcionando, pero la corrección en sí misma no. El proceso previo se encarga de eliminar los datos de los episodios y temporadas de la serie, para reemplazar los datos de la serie por los datos seleccionados por el usuario en la ventana de corrección de identificación, y volver a analizar las carpetas para añadir las temporadas y episodios.

Teniendo en cuenta que el error parecía estar en el análisis posterior, se analizó la función encargada de realizarlo y se encontró el error en el siguiente fragmento de código:

```
1 if (library.getAnalyzedFolders().get(directory.getAbsolutePath()) != null){
2     series = library.getSeries(library.getAnalyzedFiles().get(directory.getAbsolutePath())
3     exists = true;
4 }else{
5     series = new Series();
6     series.setFolder(directory.getAbsolutePath());
7     library.getAnalyzedFolders().put(directory.getAbsolutePath(), series.getId());
8 }
```

Listado 4.26: Fragmento de código con error: corregir identificación de series

La segunda línea utiliza el método `.getAnalyzedFiles()` de `Library` para comprobar que la carpeta de la serie está dentro de la base de datos, cuando debería utilizar el método `.getAnalyzedFolders()` en su lugar. Este error es una confusión del desarrollador, por lo que ha sido más difícil de detectar que un error con excepción asociada,

pero tras revisar detenidamente el código se ha podido encontrar y subsanar.

Una vez solucionado el error, volvemos a realizar la prueba:

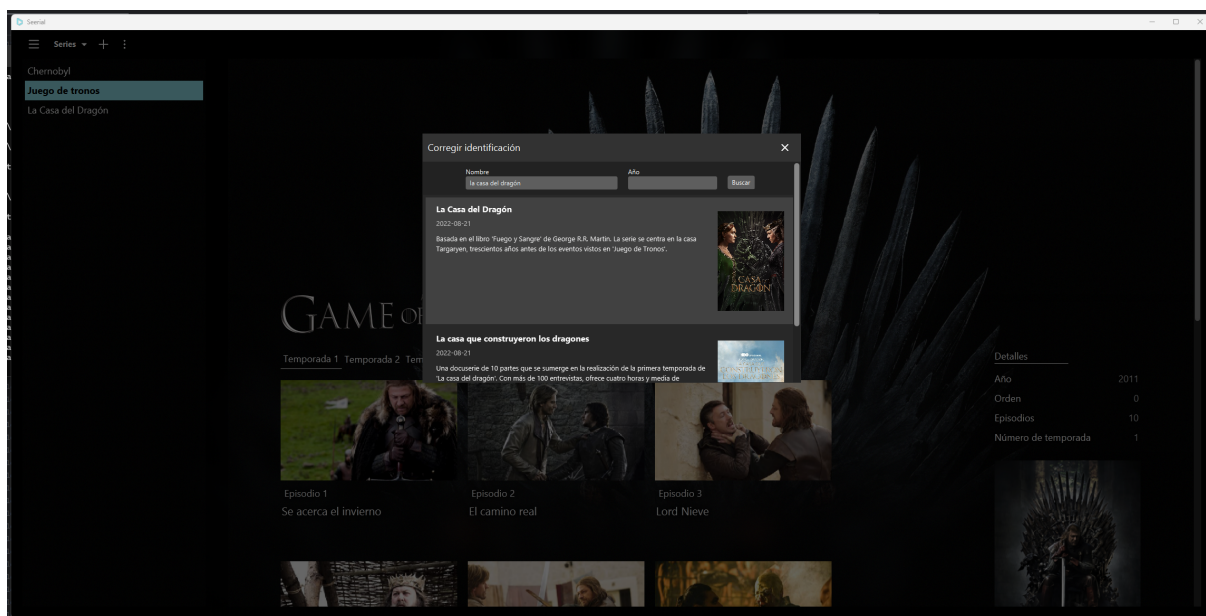


Figura 4.32: Test de corrección de identificación de series: 6

Para realizar la prueba en las mismas condiciones, primero eliminamos la biblioteca y la volvemos a añadir, para así volver a contar con las mismas series que en la primera prueba. Tras esto, realizamos los mismos pasos que antes, buscando la nueva serie y seleccionando el botón de corregir identificación (fig. 4.32).

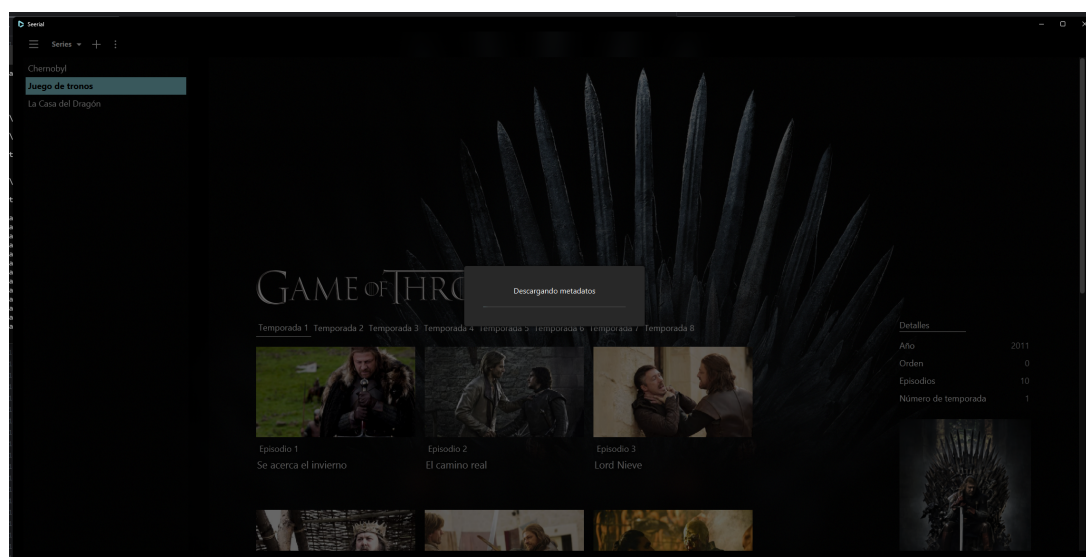


Figura 4.33: Test de corrección de identificación de series: 7

Esta vez sí aparece la ventana de carga que indica que se está procesando la nueva identificación (fig. 4.33).

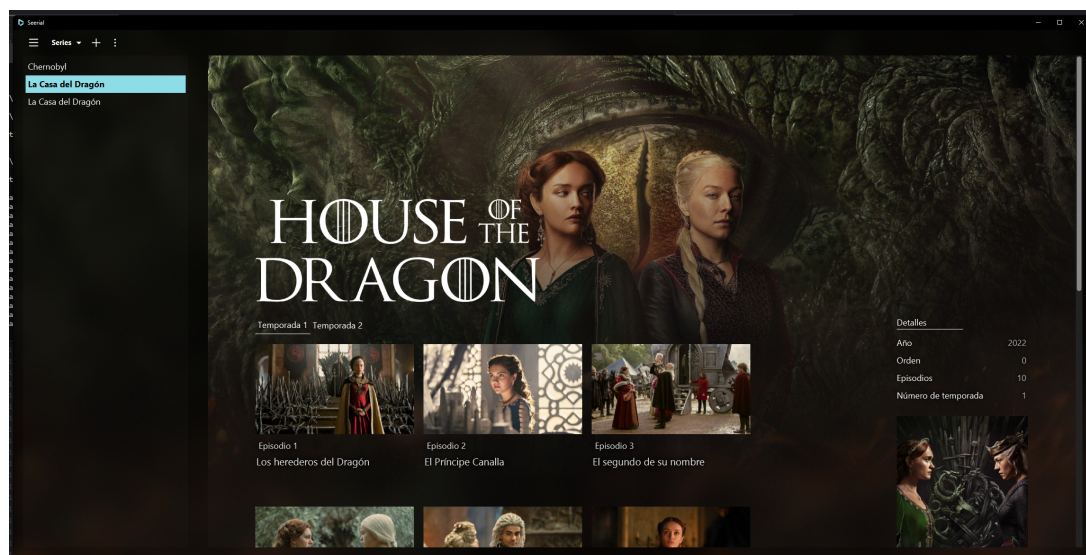


Figura 4.34: Test de corrección de identificación de series: 8

Una vez terminado el proceso, la ventana de carga se cierra y se nos muestra la serie que teníamos seleccionada (fig. 4.34). Como podemos observar, la serie que antes correspondía a '*Juego de Tronos*', ahora es '*La Casa del Dragón*'. Por tanto, la corrección de identificación se ha llevado a cabo de forma correcta.

4.7.2.2. Prueba de corrección de identificación: películas

Esta prueba es muy similar a la anterior, salvo por cómo se accede a la opción en cuestión.

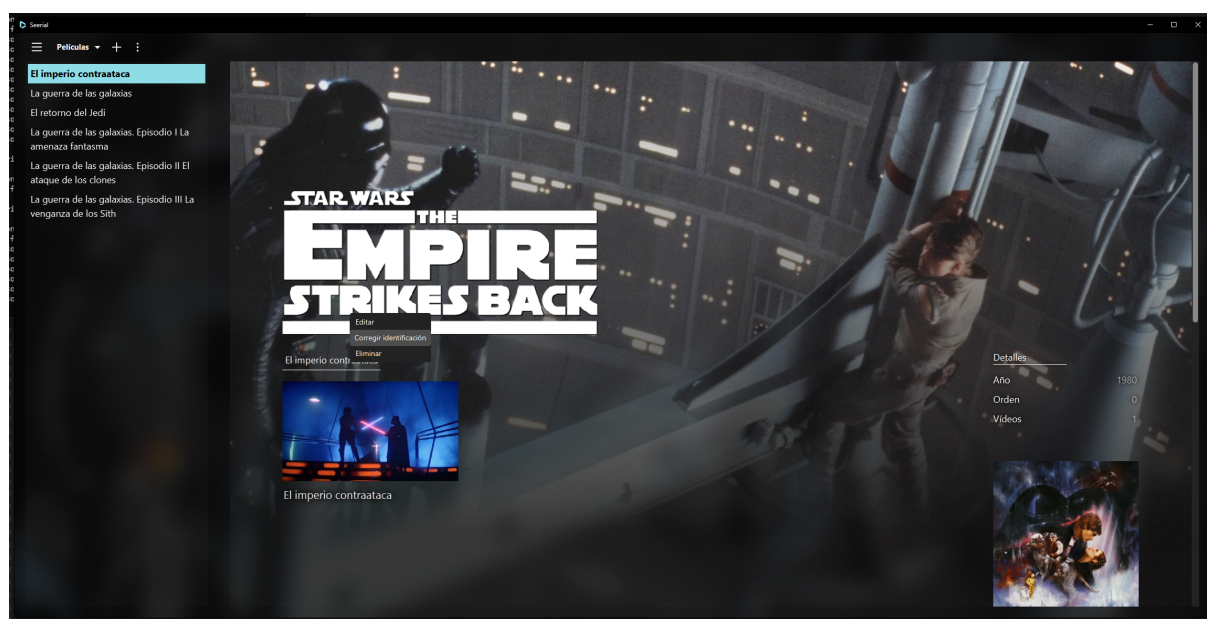


Figura 4.35: Test de corrección de identificación de películas: 1

Para seleccionar la opción de '*Corregir Identificación*' para una película, hemos de seleccionar primero una temporada, ya que esta representa una película. Pulsamos el botón derecho del ratón y abrimos el menú contextual de la temporada, pulsando ahora en la opción de '*Corregir Identificación*' (fig. 4.35).

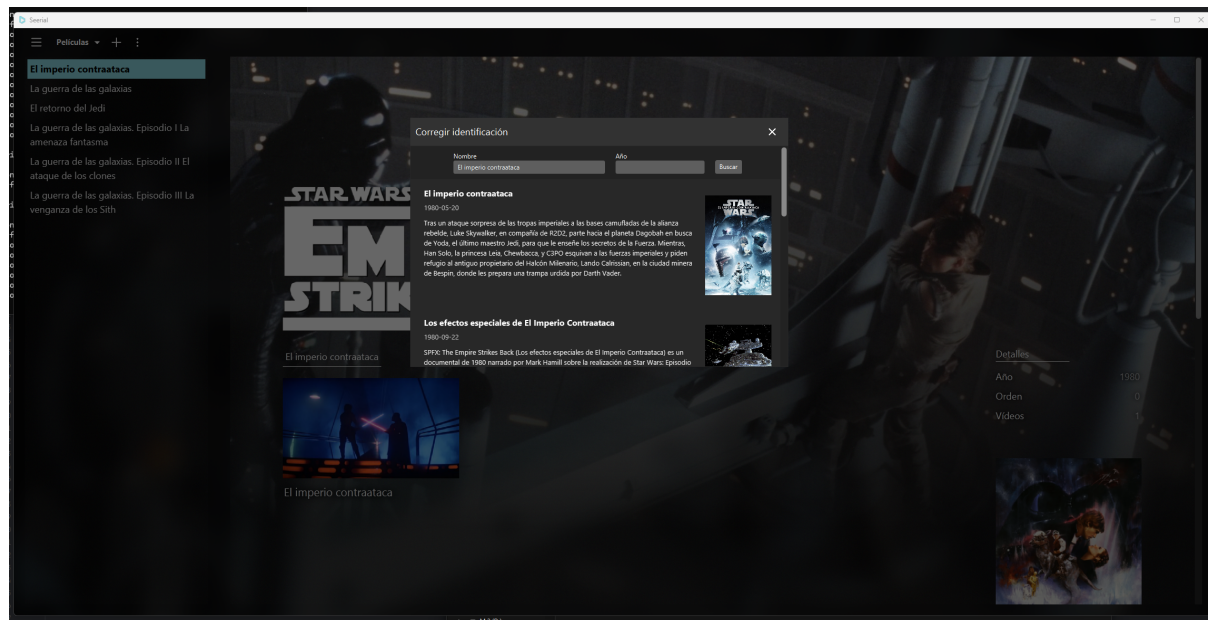


Figura 4.36: Test de corrección de identificación de películas: 2

Al igual que antes, se muestra la ventana con el buscador de, en este caso, películas (fig. 4.36).

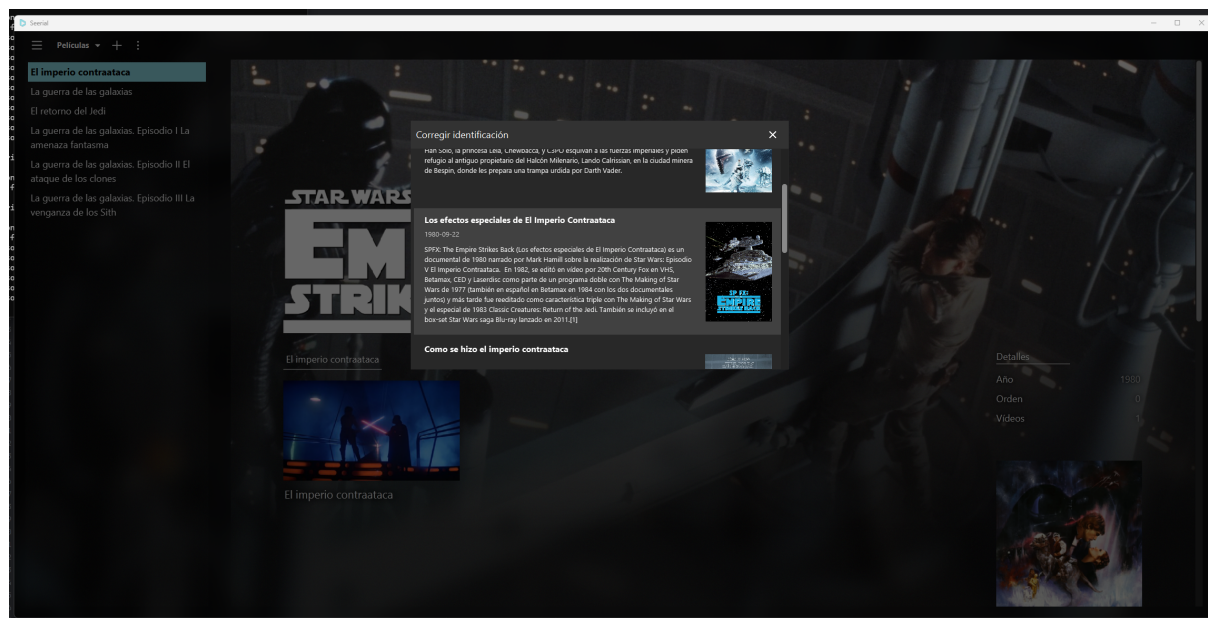


Figura 4.37: Test de corrección de identificación de películas: 3

Procedemos entonces a seleccionar una opción que no corresponda con la película actual, en este caso la segunda (fig. 4.37).

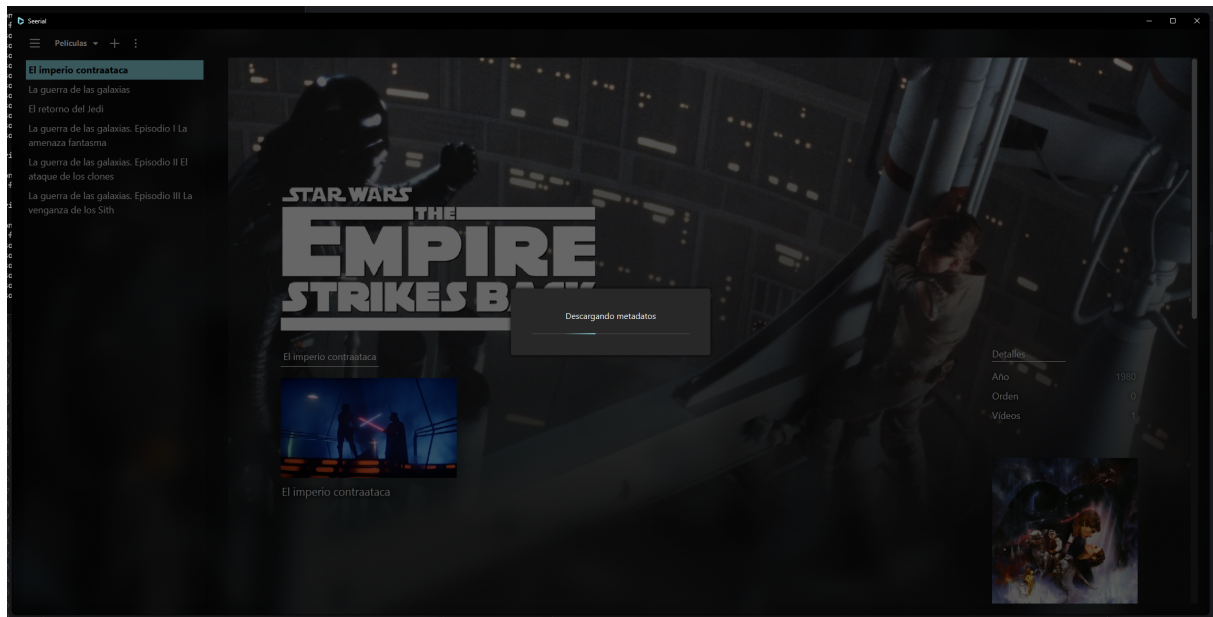


Figura 4.38: Test de corrección de identificación de películas: 4

Tras seleccionar la identificación, se cierra la ventana de búsqueda y se abre la ventana que nos indica que se están descargando los nuevos metadatos (fig. 4.38).

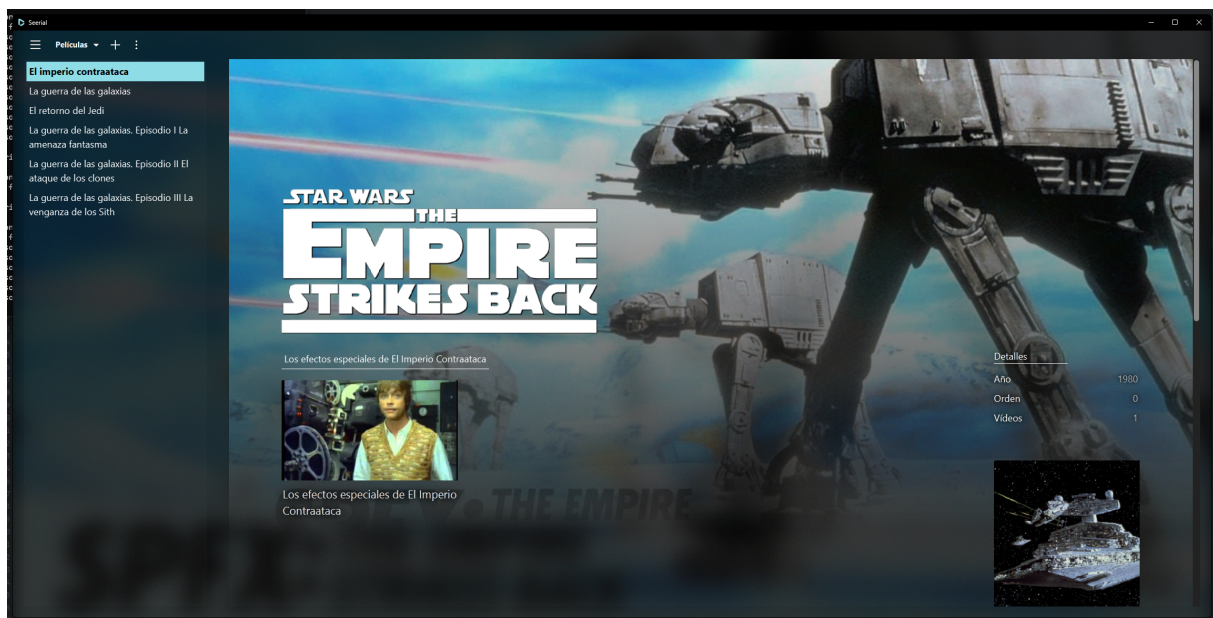


Figura 4.39: Test de corrección de identificación de películas: 5

Una vez termina el proceso (fig. 4.39), podemos comprobar que la identificación se ha aplicado correctamente, por lo que la prueba no ha detectado ningún error.

4.7.2.3. Prueba de análisis de archivos: series

El objetivo de esta prueba es comprobar el desempeño del análisis de archivos. Para ello, procederemos a crear una nueva biblioteca de series.

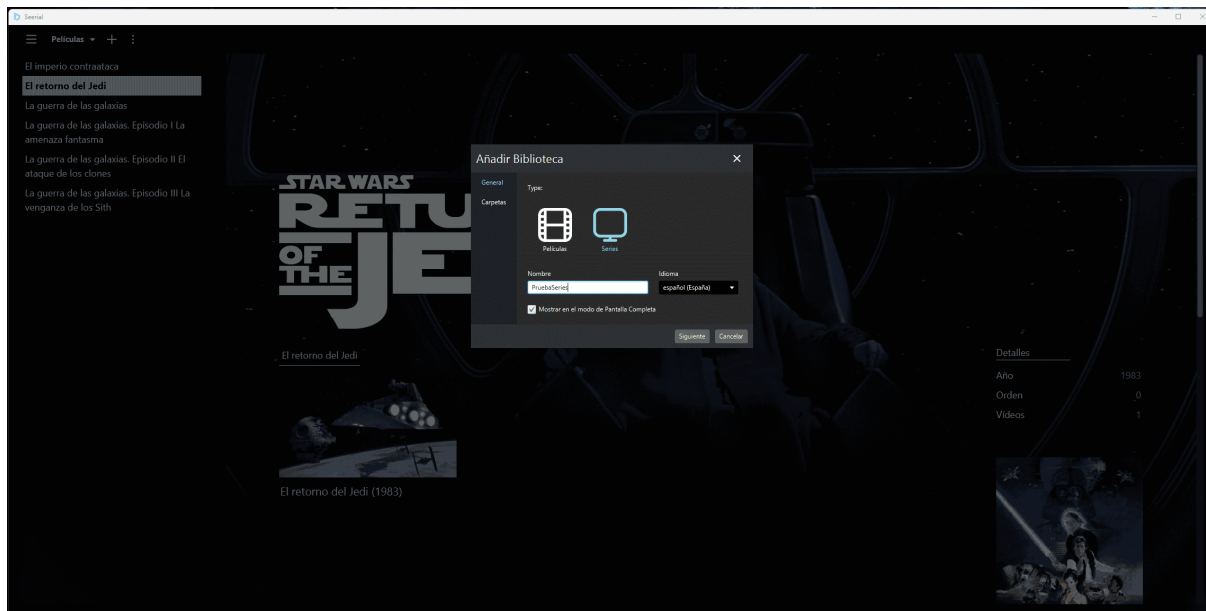


Figura 4.40: Test de análisis de archivos para series: 1

Comenzaremos creando una nueva biblioteca de tipo "Series"(fig. 4.40).

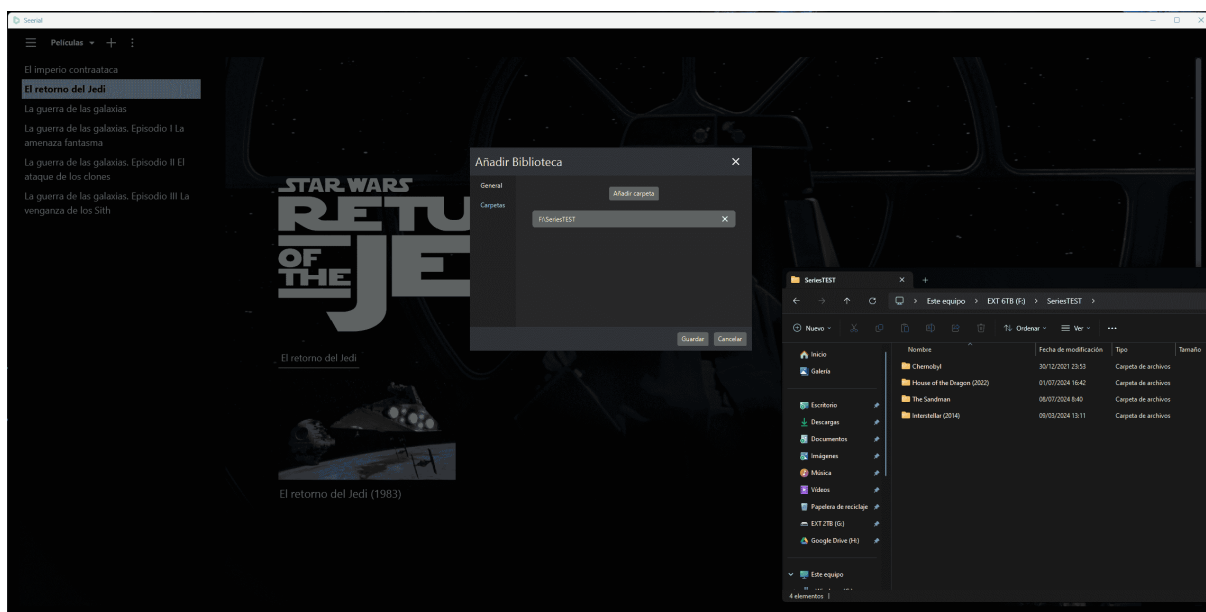


Figura 4.41: Test de análisis de archivos para series: 2

Esta biblioteca tendrá una carpeta raíz con el contenido que utilizaremos para la prueba (fig. 4.41). Esta carpeta contendrá una lista de series y una película, con el fin de comprobar el comportamiento del análisis de archivos.

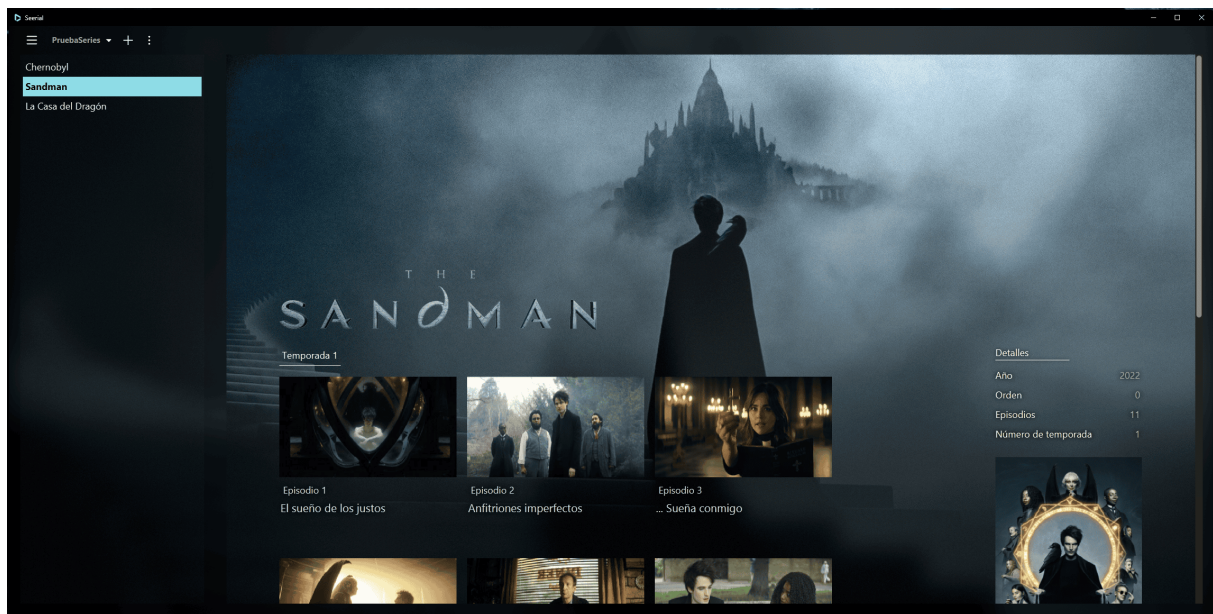


Figura 4.42: Test de análisis de archivos para series: 3

Al crear la biblioteca, se van agregando una a una las diferentes series de prueba. Y como podemos observar en (fig. 4.42), la película '*Interestellar*' no ha sido agregada, ya que no es una serie.

Esta prueba nos permite también comprobar si se han creado correctamente las temporadas de una serie, así como si se han agregado todos los episodios. Además, en el caso de la película, esta no es agregada debido a que no se ha identificado una serie con dicho nombre. Este comportamiento es el esperado en el caso de las series, ya que una serie sin metadatos no será almacenada en la aplicación.

4.7.2.4. Prueba de análisis de archivos: películas

Esta prueba es muy similar a la anterior, pero el comportamiento a comprobar es ligeramente diferente. En este caso, si un contenido no tiene metadatos en la base de datos, se creará igualmente.

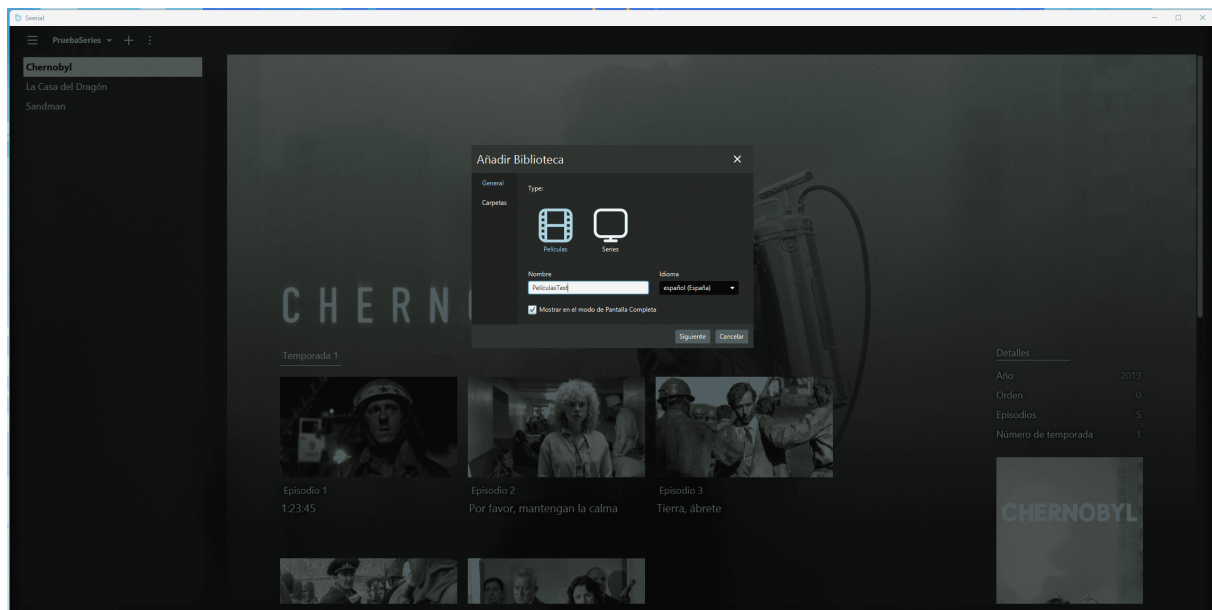


Figura 4.43: Test de análisis de archivos para películas: 1

Comenzamos creando la biblioteca, pero esta vez del tipo "Películas"(fig. 4.43).

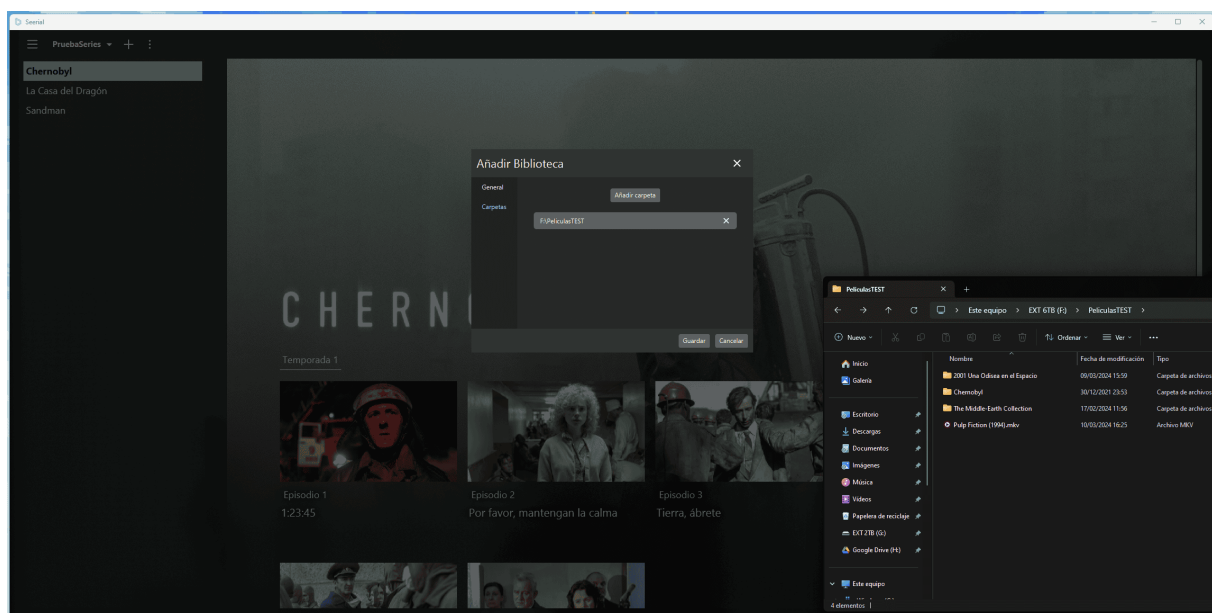


Figura 4.44: Test de análisis de archivos para películas: 2

Agregamos la carpeta de prueba, al igual que en la prueba anterior, pero en este caso con varias películas y una serie (fig. 4.44). Además, podemos comprobar de paso las distintas estructuras de archivos posibles para el contenido de esta biblioteca; es decir, probaremos con una película sin carpeta, otra dentro de una carpeta con el mismo nombre, y por último, una colección de películas.

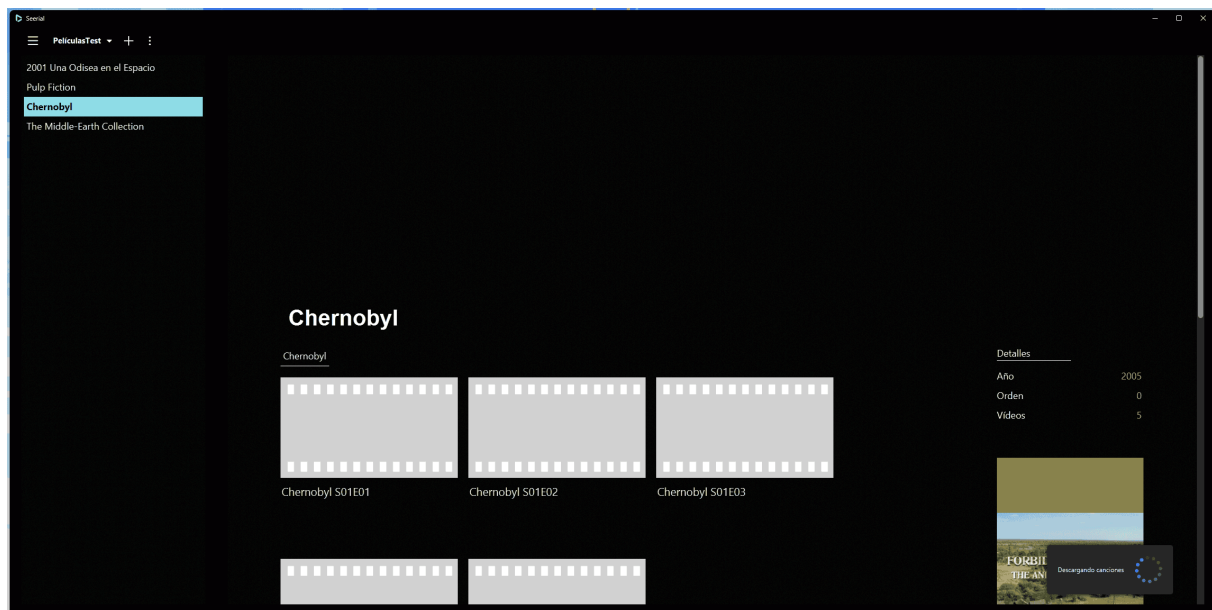


Figura 4.45: Test de análisis de archivos para películas: 3

Al crear la biblioteca, se agregan una a una las películas, al igual que la serie que hemos añadido para realizar la prueba (fig. 4.45).



Figura 4.46: Test de análisis de archivos para películas: 4

Como podemos observar, la serie se ha añadido como si se tratara de una película (fig. 4.46). Podríamos pensar que esto se debe a que el buscador de metadatos ha encontrado resultados al buscar *Chernobyl*. Sin embargo, aún sin encontrar resultados, se habría añadido, aunque sin imagen de póster.

Además, tanto las películas individuales como la colección se han añadido correctamente.

El comportamiento relacionado con añadir contenido independientemente de los resultados de búsqueda se debe a que hay películas, conciertos y documentales que no aparecen en la base de datos, por lo que la decisión que se ha tomado es la de permitir al usuario agregar ese contenido y personalizarlo manualmente.

4.7.2.5. Prueba de preferencia de idioma en pistas de audio y subtítulos

Esta prueba se basa en seleccionar idiomas preferentes en el menú de configuración y comprobar con diferentes vídeos si las pistas preferentes son seleccionadas automáticamente. Dado que esta preferencia sólo es válida para el reproductor integrado, la prueba se realizará en el modo pantalla completa.

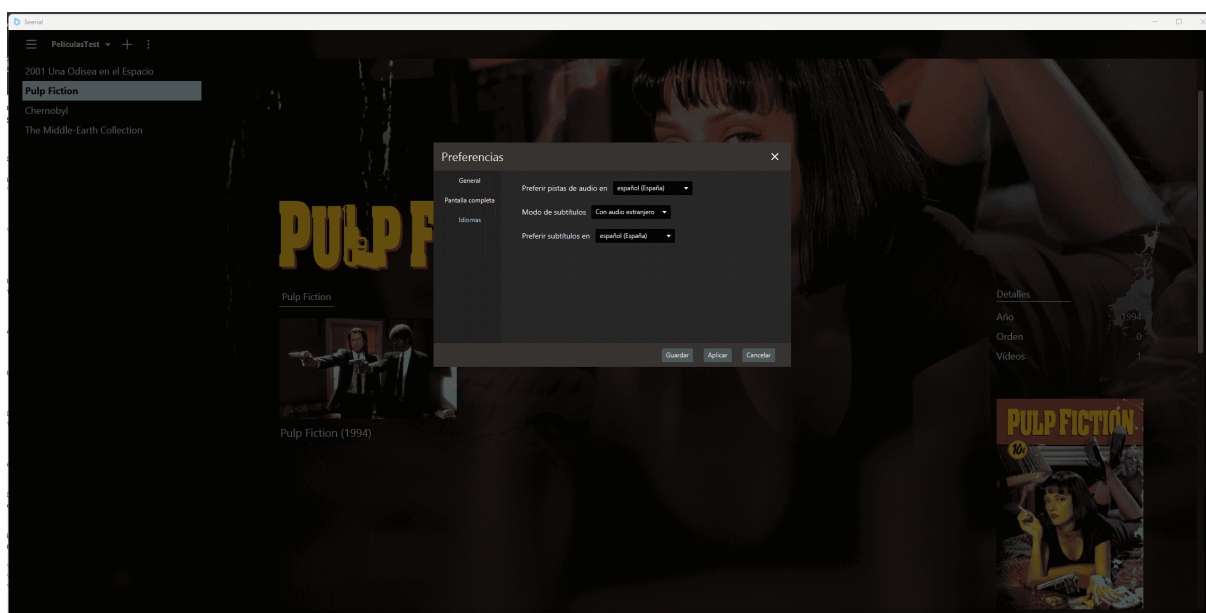


Figura 4.47: Test de preferencia de idioma: 1

Comenzamos seleccionando las pistas en el menú de configuración, accesible mediante el botón de la esquina superior izquierda del modo escritorio (fig. 4.47).

Las opciones son:

- Preferir pista de audio en: idioma preferido para la pista de audio.
- Modo de subtítulos: muestra los subtítulos de manera manual, cuando la pista de audio está en un idioma distinto al preferido, o siempre.
- Preferir subtítulos en: idioma preferido para la pista de subtítulos.

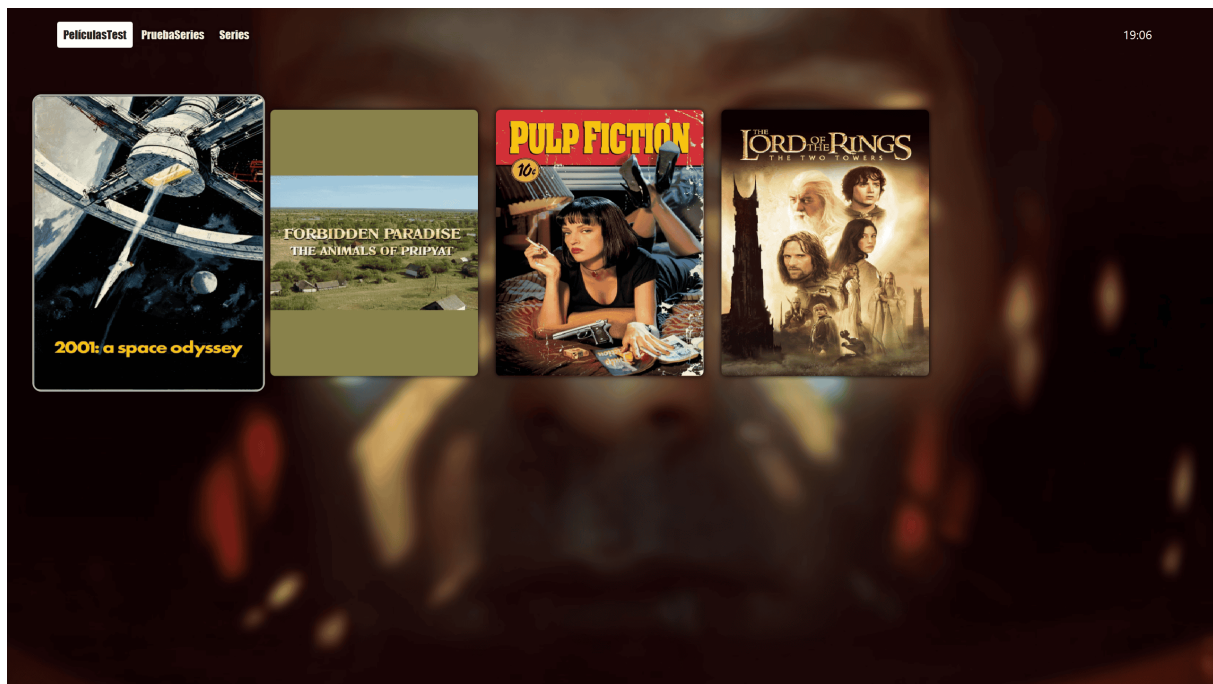


Figura 4.48: Test de preferencia de idioma: 2

Una vez seleccionada la configuración de prueba, cambiaremos al modo pantalla completa. Esto se puede realizar pulsando *F11* o mediante el menú contextual de la esquina superior izquierda del modo escritorio. En este modo, seleccionaremos una película, en este caso '*2001: Una Odisea en el Espacio*' (fig. 4.48).



Figura 4.49: Test de preferencia de idioma: 3

Al seleccionar la película, se mostrará la vista de temporada, que contiene la información de la película en cuestión (fig. 4.49). A continuación, pulsaremos el botón para ver la película.

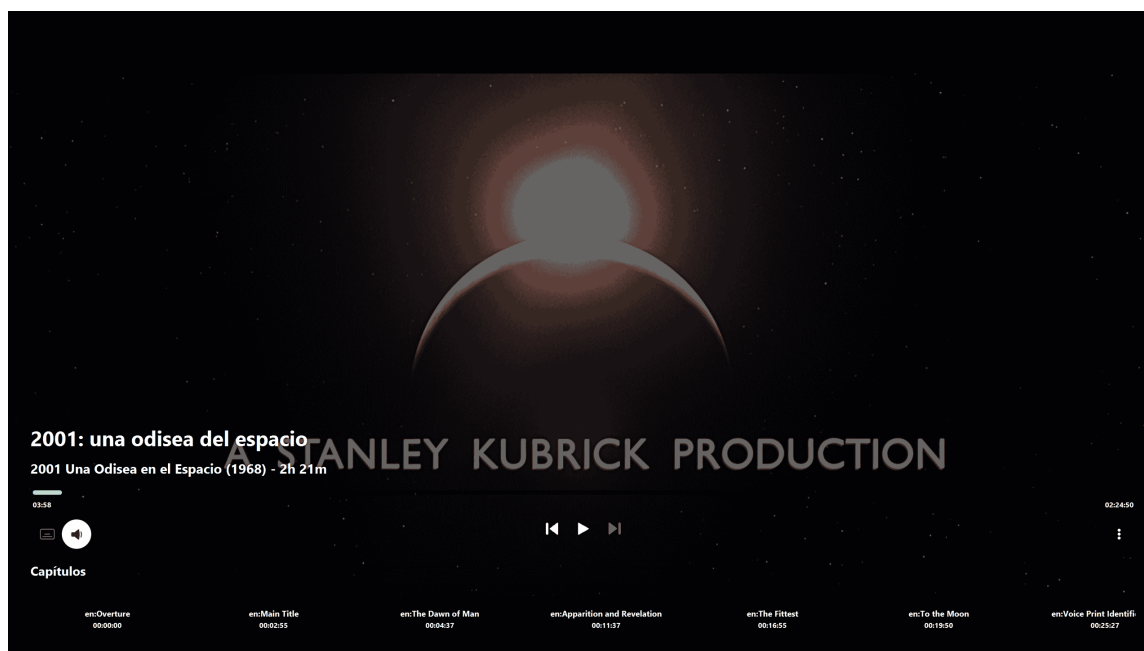


Figura 4.50: Test de preferencia de idioma: 4

Una vez abierto el reproductor, pararemos la reproducción con la tecla *Espacio* y, moviéndonos con el teclado, pulsaremos *Intro* sobre el botón con icono de altavoz para abrir el menú de pistas de audio (fig. 4.50).

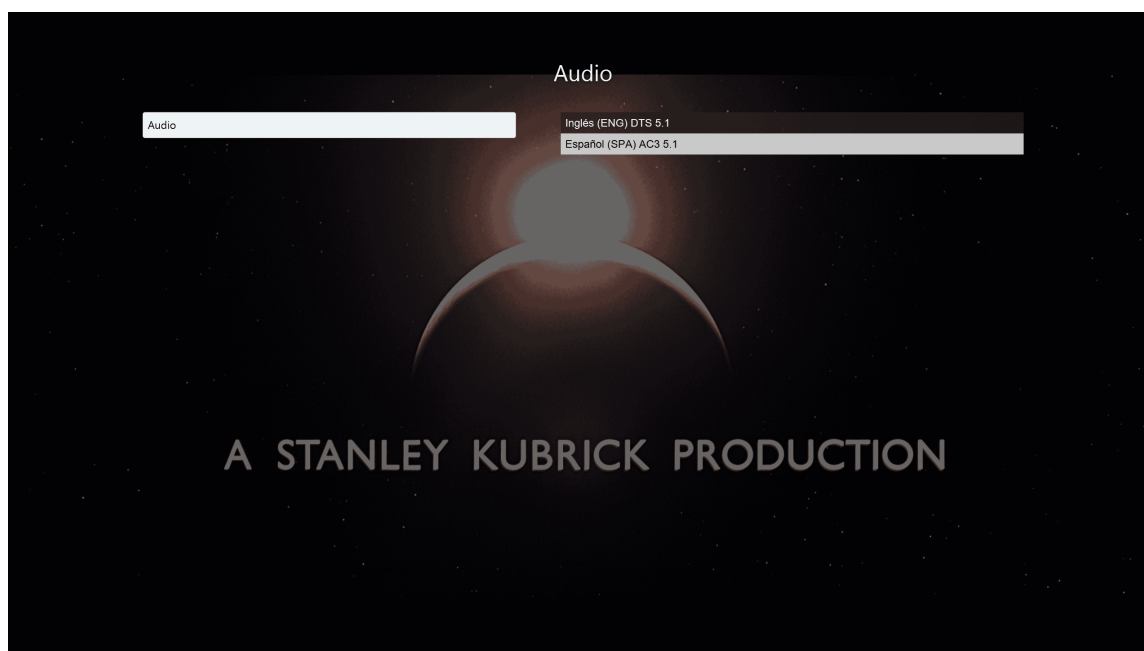


Figura 4.51: Test de preferencia de idioma: 5

Como podemos observar, la pista seleccionada automáticamente ha sido la que corresponde con el idioma preferido (fig. 4.51). Si realizamos la prueba con un vídeo que no tiene ninguna pista en dicho idioma, se seleccionará la primera pista automáticamente. Para salir de este menú, pulsaremos el botón *Delete* o *Escape*.

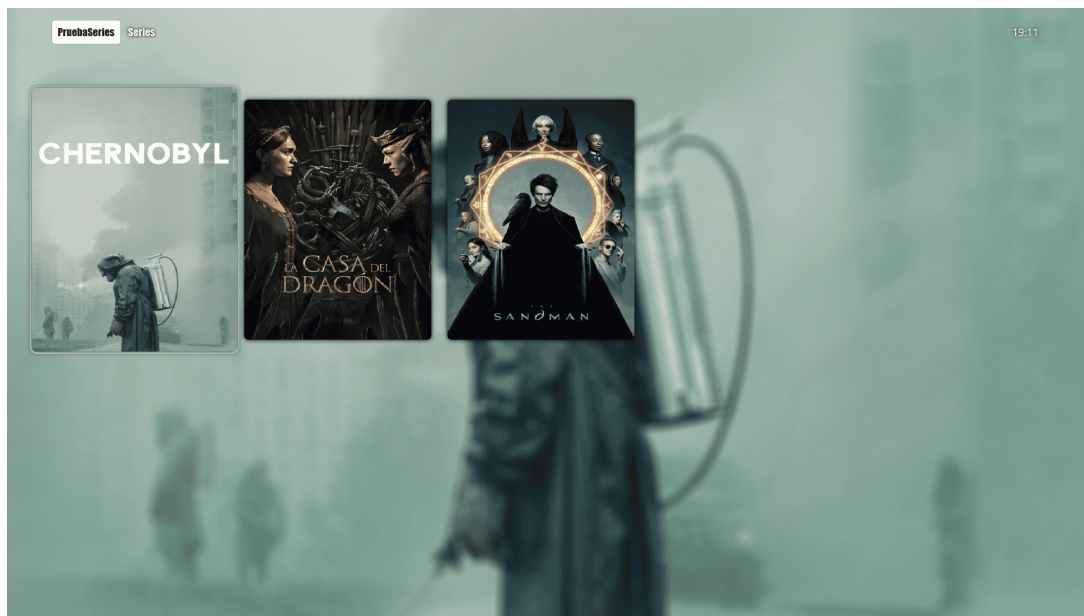


Figura 4.52: Test de preferencia de idioma: 6

Ahora realizaremos la misma comprobación con un vídeo que no tiene audio en el idioma preferido. En este caso, la pista de audio seleccionada es la primera. Para ello, utilizaremos la serie 'Chernobyl', cuyo audio es únicamente en inglés (fig. 4.52).

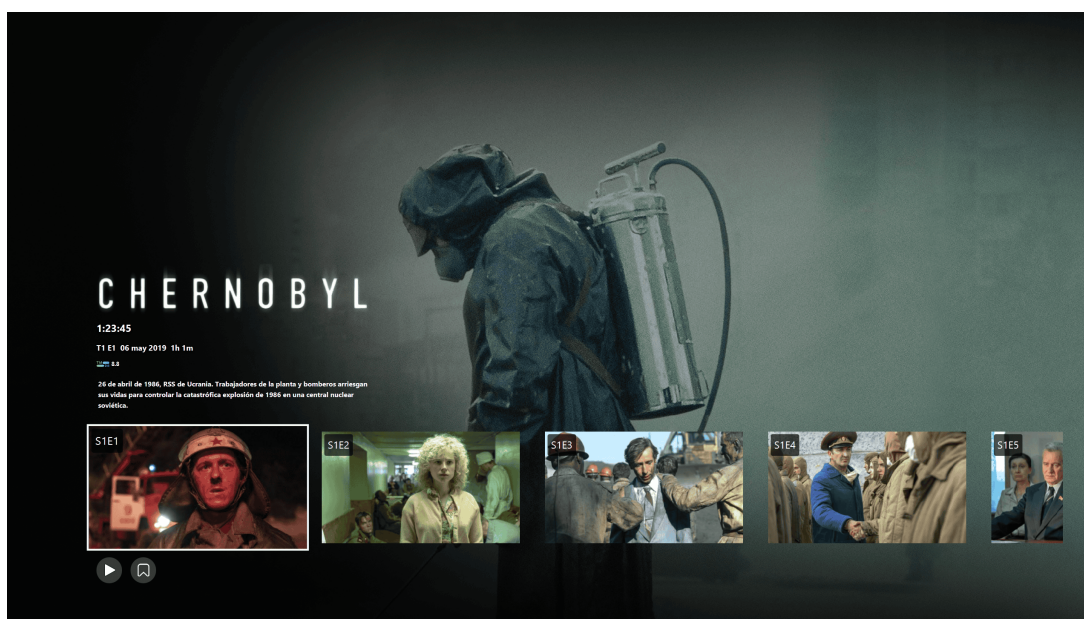


Figura 4.53: Test de preferencia de idioma: 7

Una vez en la vista de temporada (fig. 4.53), seleccionamos uno de los episodios para reproducirlo y comprobar las pistas seleccionadas, tal y como lo hemos hecho con la película.

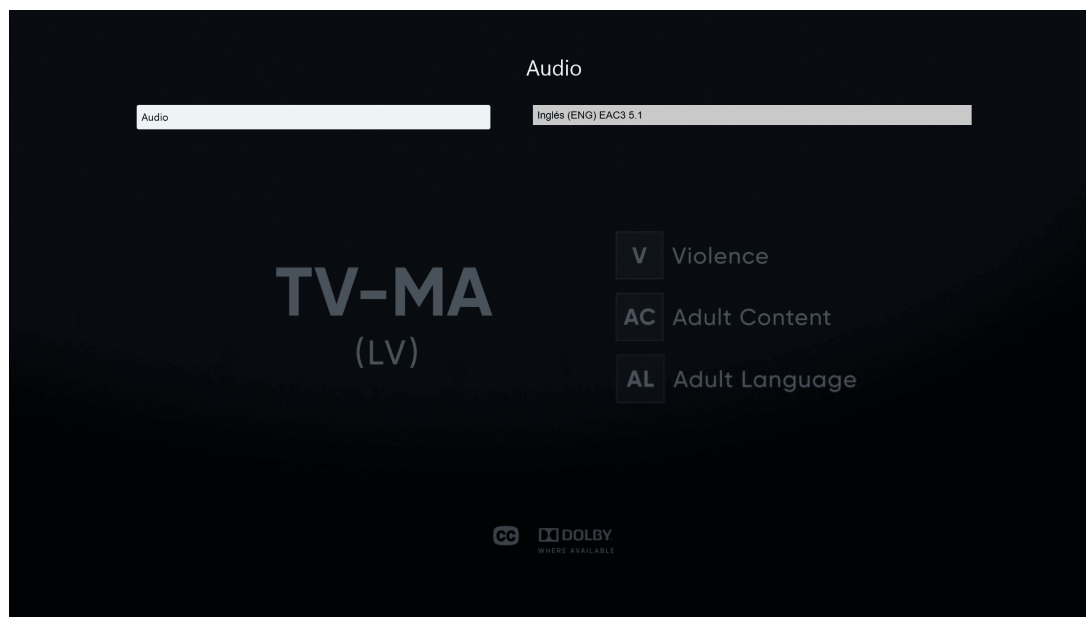


Figura 4.54: Test de preferencia de idioma: 8

Dado que sólo hay una pista de audio, esta es seleccionada por defecto (fig. 4.54).

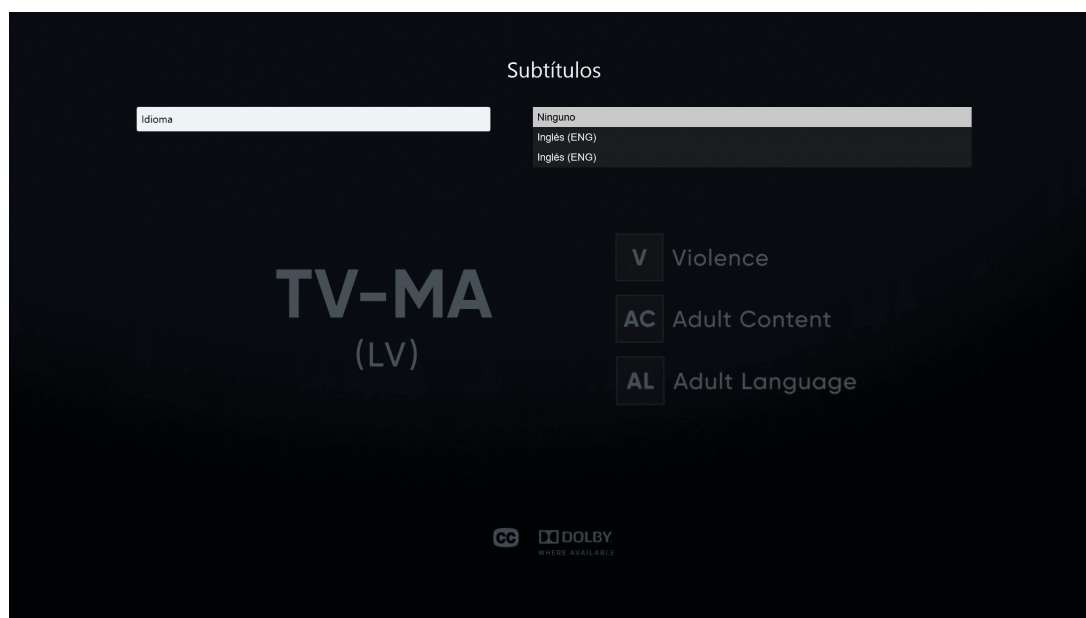


Figura 4.55: Test de preferencia de idioma: 9

Al comprobar las pistas de subtítulos, vemos que ninguna es seleccionada (fig. 4.55). Esto ocurre porque, aunque se ha seleccionado la opción de '*Mostrar con audio extranjero*', no hay ninguna pista de subtítulos en el idioma preferido.

La prueba de preferencia de idiomas es extensa, y necesita de contenido con diferentes configuraciones de pistas para comprobar que se comporta correctamente bajo cualquier circunstancia, por lo que en esta sección se ha mostrado únicamente un breve ejemplo.

Se han realizado numerosas pruebas para esta función, dado que el comportamiento esperado no se ha conseguido desde el principio. Esto se debe a que las pistas de audio y subtítulos almacenan el idioma de manera diferente a la clase `Locale` de Java, por lo que el código no funcionaba al principio, ya que no detectaba el idioma preferido.

Para resolver este problema, se ha hecho uso de un mapa de codificación, donde el código del idioma correspondiente a la clase `Locale` se almacena junto con el código de idioma que utilizan las pistas de audio y subtítulos.

```
1 private static final Map<String, String> languageCodeMap = new HashMap<>();
2 static {
3     languageCodeMap.put("es", "spa");
4     languageCodeMap.put("en", "eng");
5     languageCodeMap.put("fr", "fre");
6     languageCodeMap.put("de", "ger");
7     languageCodeMap.put("it", "ita");
8     languageCodeMap.put("ja", "jpn");
9     languageCodeMap.put("zh", "chi");
10    languageCodeMap.put("ko", "kor");
11    languageCodeMap.put("hi", "hin");
12    languageCodeMap.put("ar", "ara");
13 }
```

Listado 4.27: Solución al problema de preferencia de idiomas

4.7.2.6. Prueba de eliminación de contenido

Para esta prueba, eliminaremos una biblioteca al completo. La función encargada de eliminar una biblioteca hace uso de las funciones para eliminar episodio, temporada y serie, por lo que probaremos todas estas a la vez.



Figura 4.56: Test de eliminación de contenido: 1

Comenzaremos seleccionando la biblioteca a eliminar, y pulsaremos el botón de opciones de la parte superior, junto al de añadir biblioteca. Una vez se muestra el menú contextual de la biblioteca (fig. 4.56), seleccionamos la opción de eliminar.

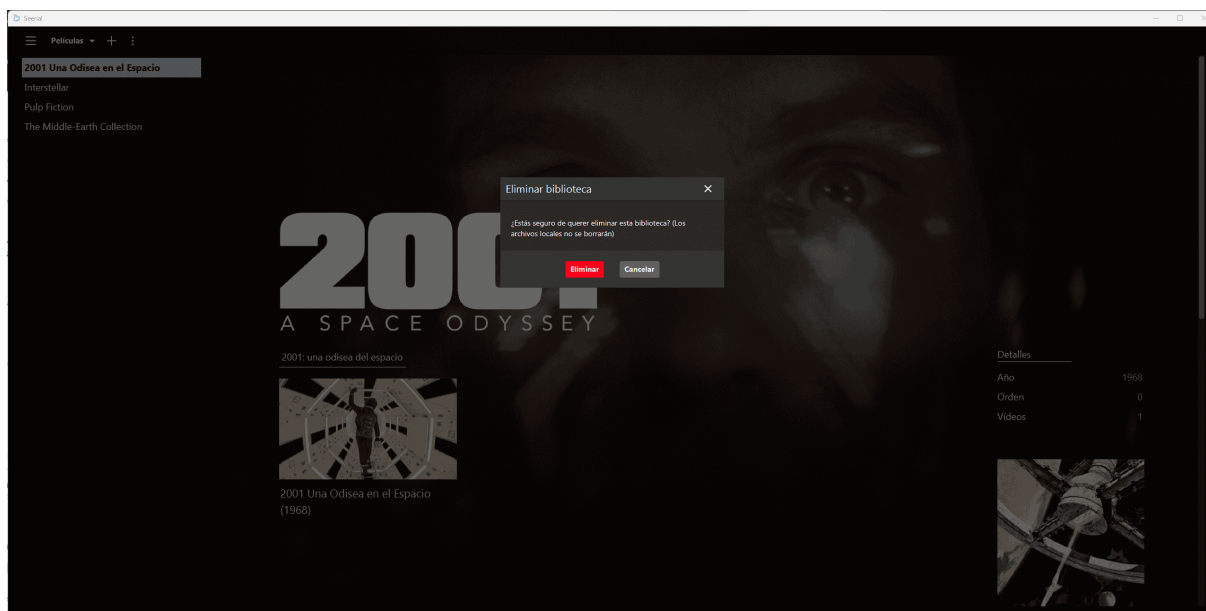


Figura 4.57: Test de eliminación de contenido: 2

Dado que eliminar contenido borra las imágenes, música y vídeos del mismo; es prudente mostrar una ventana de confirmación para asegurarnos de que el usuario realmente desea eliminar el contenido (fig. 4.57).



Figura 4.58: Test de eliminación de contenido: 3

Una vez eliminada la biblioteca, la aplicación seleccionará automáticamente la primera biblioteca de la lista, o ninguna en caso de no haber más bibliotecas (fig. 4.58).

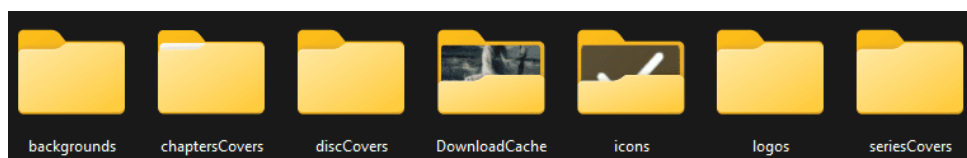


Figura 4.59: Test de eliminación de contenido: 4

Para comprobar que los archivos de la biblioteca se han eliminado satisfactoriamente, hemos de comprobar una a una las carpetas de imágenes, música y vídeos (fig. 4.59).

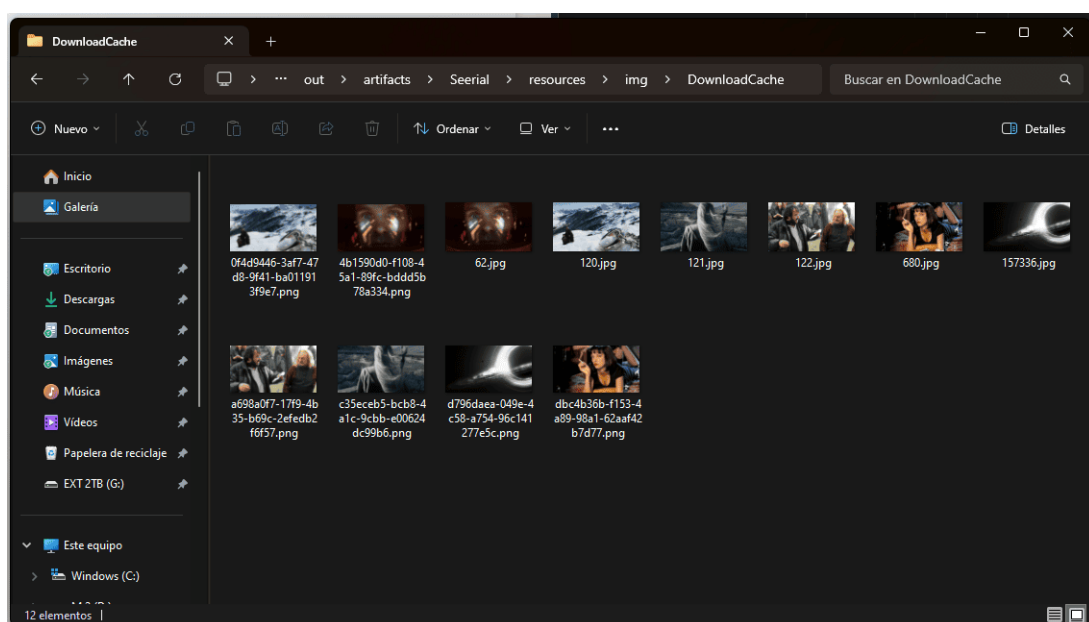


Figura 4.60: Test de eliminación de contenido: 5

Revisando las carpetas, se ha detectado una carpeta con contenido que debería estar vacía (fig. 4.60). Esta carpeta es *DownloadCache*, que almacena las imágenes descargadas del contenido mientras se está añadiendo o editando. Estos archivos deberían ser eliminados al terminar el análisis de archivos, al igual que cuando se guarda la edición de una temporada.

Para resolver este problema, hemos de crear una función que elimine el contenido de dicha carpeta y llamarla al terminar la búsqueda de archivos, y al guardar la edición de una temporada. Realizar la prueba es tan sencillo como añadir una biblioteca y, acto seguido, comprobar si la carpeta *DownloadCache* está vacía.

```
1 public void clearImageCache(){
2     try{
3         FileUtils.cleanDirectory(new File("resources/img/DownloadCache"));
4     } catch (IOException e) {
5         throw new RuntimeException(e);
6     }
7 }
```

Listado 4.28: Solución al problema del caché de imágenes descargadas

Esta prueba es de vital importancia, pues la correcta eliminación del contenido asegura que los datos de la aplicación ocupen el espacio necesario y no más.

4.7.2.7. Prueba de movimiento de scroll

El movimiento de scroll en el modo pantalla completa aplica a la pantalla principal, con el grid de series, al igual que a la vista de temporada, con la lista de episodios. Este movimiento se realizó en primera instancia por medio de una clase de JavaFX llamada **ScrollPane**. Esta clase genera una sección cuyo contenido se almacena a modo de scroll, permitiendo un desplazamiento 'infinito'. La clase ScrollPane se ha utilizado en el modo escritorio, y funciona perfectamente, pero para el modo pantalla completa no era óptimo. Esto se debe a que está pensada para su uso mediante ratón y, dado que la movilidad en este modo es mediante teclado, el movimiento resultante es funcional, pero no propio de una aplicación moderna. Dicho movimiento es instantáneo, y no permite añadir animaciones o transiciones para mejorarlo visualmente. Además, el elemento seleccionado al alcanzar el límite horizontal o vertical es siempre el último en pantalla, y esto no se puede cambiar.

Dado que la clase ScrollPane no incluía la funcionalidad requerida, se optó por recrear la funcionalidad del ScrollPane mediante la modificación de la posición de los elementos en pantalla; pudiendo así crear una animación de movimiento visualmente

atractiva, así como mantener el elemento seleccionado en el centro de la pantalla para facilitar al usuario la visualización del mismo.

Para probar esta funcionalidad es necesario utilizar la aplicación y moverse entre los episodios de una serie, o los vídeos de una película. En este caso, probaremos con una serie.



Figura 4.61: Test de movimiento de scroll: 1



Figura 4.62: Test de movimiento de scroll: 2

Para comenzar, nos desplazamos hasta el episodio que corresponde con el centro de la pantalla (fig. 4.61 y 4.62).



Figura 4.63: Test de movimiento de scroll: 3

Al continuar el desplazamiento, el siguiente episodio pasa a ocupar el centro de la pantalla (fig. 4.63).



Figura 4.64: Test de movimiento de scroll: 4

Al llegar al final (fig. 4.64), dado que los últimos episodios están dentro de la vista, no es necesario continuar moviéndolos, por lo que permanecerán en su posición.



Figura 4.65: Test de movimiento de scroll: 5

Para finalizar la prueba, realizamos el mismo recorrido de forma inversa, comprobando que el comportamiento es idéntico (fig. 4.65).

Capítulo 5

CONCLUSIÓN

5.1. Conclusiones

El desarrollo de la aplicación ha sido todo un reto. Aunque tener experiencia con el lenguaje Java ha sido beneficioso, un proyecto de esta magnitud requiere un extenso proceso de aprendizaje en el uso de un framework de desarrollo.

Ahora que el desarrollo principal está terminado, puedo afirmar que, si bien JavaFX es un framework de desarrollo muy potente, no está lo suficientemente modernizado en comparación con otros frameworks como .NET o posiblemente Flutter. El desarrollo de JavaFX parece haber quedado algo abandonado desde hace tiempo, lo cual se refleja en la falta de algunas características modernas presentes en otros frameworks, así como en una comunidad de soporte menos activa.

Además, esta experiencia me ha enseñado que la elección de un framework debe considerar no solo la capacidad técnica actual, sino también el soporte y la evolución futura del mismo. En retrospectiva, frameworks como .NET, con su fuerte soporte y constante evolución, o Flutter, conocido por su rendimiento y capacidad para crear aplicaciones nativas en múltiples plataformas, podrían haber ofrecido ventajas significativas.

5.2. Posibles desarrollos futuros

La aplicación cumple con los requisitos iniciales y tiene un funcionamiento óptimo. Sin embargo, siempre existe la posibilidad de fallo, por lo que continuar haciendo pruebas para mejorar la robustez, así como revisar el diseño en busca de mejoras, es un buen curso de acción a futuro.

Además de solucionar errores, también interesa implementar nueva funcionalidad, como podría ser:

- Búsqueda y descarga de subtítulos utilizando la API de OpenSubtitles [34].
- Implementar un inicio alternativo en el modo pantalla completa, que permita visualizar la lista de episodios y películas que se están viendo actualmente, para permitir una reproducción más rápida y mantener el estado de visualización en la pantalla de inicio.
- Resumir la implementación de la generación de miniaturas para los capítulos de un vídeo. Esta característica se consiguió implementar, pero se deshabilitó debido a que tardaba demasiado tiempo en generar las miniaturas.
- Incluir más información de los metadatos del contenido en el modo escritorio.
- Implementar una versión del reproductor para el modo escritorio, para así poder registrar los datos de visualización del usuario en este modo.
- Permitir la reproducción de la música/vídeo de fondo en el modo escritorio, proporcionando al usuario la opción de activar o desactivar esta funcionalidad en el menú de ajustes.
- Incluir Java y JavaFX en el propio ejecutable para facilitar la ejecución de la aplicación en Windows.

Capítulo 6

ANEXOS

6.1. Dependencias necesarias para ejecutar la aplicación

La aplicación estará disponible en una *release* en el Github del proyecto [35]. Se distribuirá en formatos JAR y EXE para facilitar la ejecución en diferentes plataformas. Estos archivos son portables, por lo que no es necesario instalar la aplicación. Además, se proporcionan dos ejecutables diferentes, uno para cada modo de la aplicación; aunque es posible alternar entre modos dentro de la misma.

Las dependencias necesarias son:

- Java 8.
- JavaFX.
- Python 3.
- Python Pytube (instalable mediante 'pip install pytube').

6.1.1. Windows

Para ejecutar la aplicación en Windows, podemos utilizar tanto el archivo JAR como el ejecutable del modo Escritorio o Pantalla Completa.

6.1.2. Linux

Para ejecutar la aplicación en Linux, hemos de utilizar cualquiera de los archivos JAR. Sin embargo, es importante destacar que la ejecución de archivos JAR en Linux requiere la instalación manual de Java y la configuración adecuada para la ejecución de estos. Alternativamente, los usuarios pueden ejecutar la aplicación mediante la terminal usando el comando 'java -jar archivo.jar'.

6.2. Estructura de carpetas y nombres

En esta última sección, veremos cómo se han de estructurar los archivos para que la detección de contenido funcione correctamente. Esta estructura depende del contenido, ya que es distinta para series y películas.

6.2.1. Series

```
1 CarpetaBiblioteca
2 +---Serie1
3 |   +---S01
4 |   |   episodio 1.ext
5 |   |   episodio 2.ext
6 |   |   ...
7 +---Serie2
8 |   episodio 1.ext
9 |   episodio 2.ext
```

Listado 6.1: Estructura de carpetas Serie

*ext = extensión de archivo

Las series se detectan por su carpeta raíz, buscando su nombre y analizando todos los archivos de vídeo dentro de esa carpeta. Debido a esto, el nombre de la carpeta debe ser el nombre de la serie; y es mejor si el nombre incluye el año de su primera emisión, aunque el año es opcional y no necesita estar entre paréntesis.

```
1 Juego de Tronos (2011)
```

Listado 6.2: Ejemplo de una carpeta de una serie

Los archivos de vídeo dentro de la carpeta se procesan por igual y no se tienen en cuenta las carpetas internas, pero el nombre de los archivos de vídeo es clave para

que el algoritmo encuentre la coincidencia exacta. El nombre del archivo de vídeo debe tener uno de estos formatos:

```
1 S02E12.mkv
2 24.mkv
```

Listado 6.3: Formatos del nombre de un episodio

Todo lo adicional es opcional, como el nombre de la serie o el nombre del episodio. Solo necesita tener el formato de temporada/episodio o el formato de episodio absoluto. Para el primero, la S/E puede estar en minúscula (s/e) y los números pueden estar en el formato (01) o (1).

6.2.2. Películas

Las películas, documentales y conciertos se procesan de la misma manera. Esto se hace para agregar conciertos con múltiples archivos de vídeo o películas con vídeos adicionales.

```
1 CarpetaBiblioteca
2 +---película1.ext
3 |
4 +---Película2
5 |   +---película2.ext
6 |
7 +---Colección1
8 |   +---Película3
9 |     película3.ext
10 |     extras.ext
11 |     ...
12 |   +---Película4
13 |     película4.ext
```

Listado 6.4: Estructura de carpetas Película

Las películas/conciertos se pueden organizar solos o por colecciones. Las colecciones se crean automáticamente, basándose en la estructura de carpetas, por lo que el nombre de la colección será el nombre de la carpeta raíz de la colección.

Las colecciones solo se crean cuando hay carpetas dentro de la carpeta raíz (carpeta de la colección) y hay archivos de vídeo dentro de ellas.

Las películas/conciertos se detectan por el nombre de su carpeta, por lo que se recomienda incluir siempre el año junto con el título. Es irrelevante si el año se encuentra rodeado de paréntesis o no, pues estos se eliminan a la hora de detectar el texto.

```
1 El Caballero Oscuro (2008) //Solo archivo
2
3 El Caballero Oscuro (2008)
4 +---El Caballero Oscuro (2008).mkv //Carpeta + archivo
5
6 Trilogía de El Caballero Oscuro
7 +---Batman Begins (2005) //Colección + carpeta + archivo
8 | +---Batman Begins (2005).mkv
9 +---El Caballero Oscuro (2008)
10 | +---El Caballero Oscuro (2008).mkv
```

Listado 6.5: Ejemplos de películas

Si una carpeta contiene varios archivos de vídeo, se catalogará como una película con extras, lo que será lo mismo que un concierto con varios archivos de vídeo.

Bibliografía

- [1] TMDb. Web oficial de themoviedb, 2024. URL <https://www.themoviedb.org/>. comprobado en 2024-06-23.
- [2] Plex. Web oficial del servicio plex, 2024. URL <https://www.plex.tv/es/>. comprobado en 2024-06-23.
- [3] TheTVDB. Web oficial de thetvdb, 2024. URL <https://thetvdb.com/>. comprobado en 2024-06-23.
- [4] XBMC. Web oficial del servicio kodi, 2024. URL <https://kodi.tv/>. comprobado en 2024-06-23.
- [5] Jellyfin. Web oficial del servicio jellyfin, 2024. URL <https://jellyfin.org/>. comprobado en 2024-06-23.
- [6] Microsoft. Web oficial del framework microsoft .net, 2024. URL <https://dotnet.microsoft.com/es-es/>. comprobado en 2024-06-23.
- [7] Oracle. Web oficial del framework javafx, 2024. URL <https://openjfx.io/>. comprobado en 2024-06-23.
- [8] Electron. Web oficial del framework electron, 2024. URL <https://www.electronjs.org/>. comprobado en 2024-06-23.
- [9] Google. Web oficial del framework flutter, 2024. URL <https://flutter.dev/>. comprobado en 2024-06-23.
- [10] JetBrains. Web oficial del ide intellij idea, 2024. URL <https://www.jetbrains.com/es-es/idea/>. comprobado en 2024-06-23.
- [11] Gluon. Web oficial de la herramienta scenebuilder, 2024. URL <https://gluonhq.com/products/scene-builder/>. comprobado en 2024-06-23.
- [12] Apache. Web oficial del gestor de proyectos apache maven, 2024. URL <https://maven.apache.org/>. comprobado en 2024-06-23.

- [13] Git. Web oficial de git, 2024. URL <https://git-scm.com/>. comprobado en 2024-06-23.
- [14] Microsoft. Web oficial de la plataforma de control de versiones github, 2024. URL <https://github.com/>. comprobado en 2024-06-23.
- [15] Gaurav Kumar and Pradeep Kumar Bhatia. Comparative analysis of software engineering models from traditional to modern methodologies. In *2014 Fourth International Conference on Advanced Computing Communication Technologies*, pages 189–196, 2014. doi: 10.1109/ACCT.2014.73.
- [16] Julia Martins. ¿qué es la metodología kanban y cómo funciona?, 2024. URL <https://asana.com/es/resources/what-is-kanban>. comprobado en 2024-06-23.
- [17] Trello. Web oficial de la herramienta de gestión de proyectos trello, 2024. URL <https://trello.com/es>. comprobado en 2024-06-23.
- [18] TMDb. Web oficial de la api de themoviedb, 2024. URL <https://developer.themoviedb.org/reference/intro/getting-started>. comprobado en 2024-06-23.
- [19] Python Software Foundation. Web oficial de python, 2024. URL <https://www.python.org/>. comprobado en 2024-06-23.
- [20] FFMPEG. Web oficial de la herramienta ffmpeg, 2024. URL <https://ffmpeg.org/>. comprobado en 2024-06-23.
- [21] Mozilla. Explicación de mvc (modelo-vista-controlador), 2024. URL <https://developer.mozilla.org/es/docs/Glossary/MVC>. comprobado en 2024-06-23.
- [22] Kishori Sharan. *Learn JavaFX 8: Building User Experience and Interfaces with Java 8*. Apress, New York, NY, 1st edition, 2015. ISBN 978-1484211434.
- [23] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Boston, MA, 1st edition, 1994. ISBN 978-0201633610.
- [24] Wikipedia. Wikipedia: Json, 2024. URL <https://en.wikipedia.org/wiki/JSON>. comprobado en 2024-06-23.
- [25] Baeldung. A guide to the resourcebundle, 2024. URL <https://www.baeldung.com/java-resourcebundle>. comprobado en 2024-06-23.

- [26] OpenAI. Web oficial de chatgpt, 2024. URL <https://chatgpt.com/>. comprobado en 2024-06-23.
- [27] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O'Reilly Media, Sebastopol, CA, 3rd edition, 2006. ISBN 978-0596528126.
- [28] Oracle. Reproductor mediaPlayer, 2024. URL <https://docs.oracle.com/javafx/2/api/javafx/scene/media/MediaPlayer.html>. comprobado en 2024-06-23.
- [29] VideoLAN. Web oficial de la librería de vlc, 2024. URL <https://www.videolan.org/vlc/libvlc.html>. comprobado en 2024-06-23.
- [30] Caprica. Github de la librería de vlc para javafx, 2024. URL <https://github.com/caprica/vlcj-javafx>. comprobado en 2024-06-23.
- [31] MPV. Web oficial del reproductor mpv, 2024. URL <https://mpv.io/>. comprobado en 2024-06-23.
- [32] JNA. Github de java native access, 2024. URL <https://github.com/java-native-access/jna>. comprobado en 2024-06-23.
- [33] MPV. Manual del reproductor mpv, 2024. URL <https://mpv.io/manual/master/>. comprobado en 2024-06-23.
- [34] OpenSubtitles. Api de opensubtitles, 2024. URL <https://opensubtitles.stoplight.io/docs/opensubtitles-api/e3750fd63a100-getting-started>. comprobado en 2024-06-23.
- [35] Juan Llaveró Company. Github del proyecto seerial, 2024. URL <https://github.com/juanllaveró/Seerial>. comprobado en 2024-06-23.

