TECNOLOGÍAS DE DESARROLLO DE SOFTWARE

AppVideo - PNG

Grupo 2.3
Junio 2022

Pedro Nicolás Gomariz
pedro.nicolasg@um.es

ÍNDICE

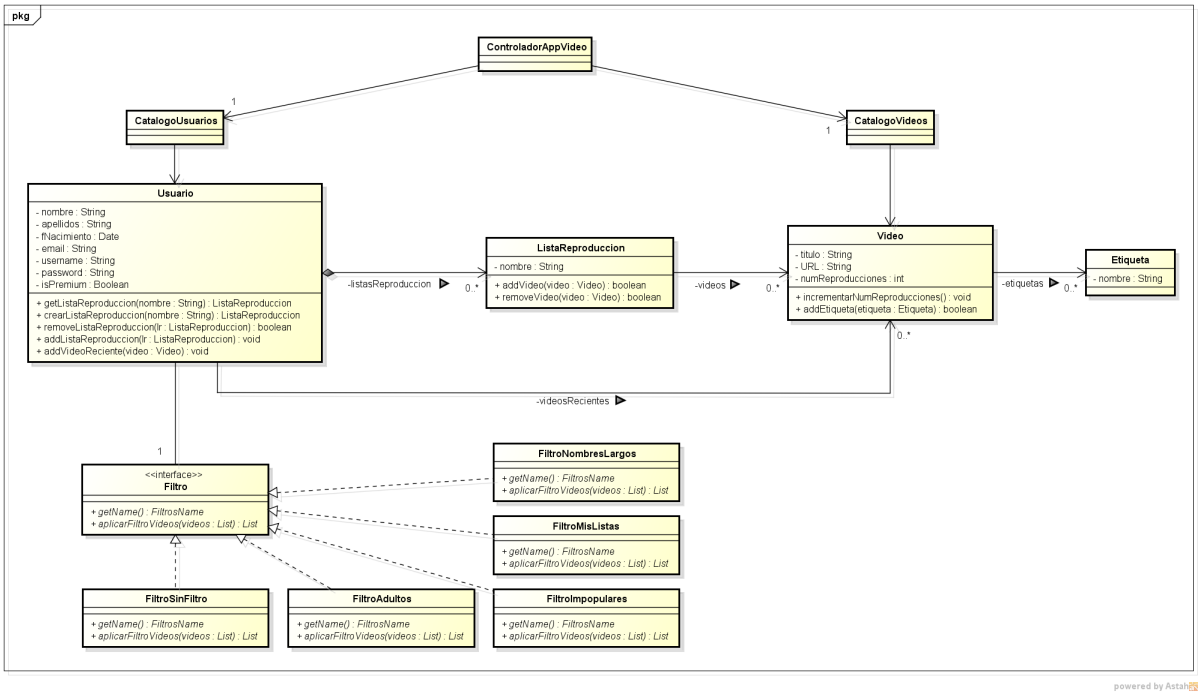
1.- INTRODUCCIÓN.

AppVideo es una aplicación de escritorio destinada a compartir videos, y basada en algunas de las aplicaciones web de contenido audiovisual más conocidas como son Youtube, Vimeo o DailyMotion.

El desarrollo de esta aplicación se plantea como parte de las prácticas de la asignatura Tecnologías de Desarrollo De Software, en el curso 2021-2022 con el objetivo de que los alumnos puedan aprender y poner en práctica conceptos como el desarrollo de una GUI empleando Swing para Java, conocer el modelo de separación en tres capas, aplicar patrones de desarrollo software, uso de las herramientas JUnit, Maven y Git, entre otras.

Para la realización de AppVideo, nos basaremos en la funcionalidad más básica de las aplicaciones mencionadas anteriormente para crear una aplicación de escritorio que nos permita buscar entre una lista de videos para poder visualizarlos, tener la posibilidad de crear y visualizar listas de reproducción, así como otras funcionalidades. Un usuario podrá convertirse en premium para obtener una funcionalidad adicional.

2.- DIAGRAMA DE CLASES DEL DOMINIO.



- En el diagrama solo se muestran los atributos y operaciones más relevantes, omitiendo así, por ejemplo, los métodos *get/set*, para evitar saturar el mismo. Del mismo modo sólo muestra las relaciones más importantes, por ejemplo, para la clase ControladorAppVideo solo se muestran las relaciones con los catálogos a pesar de que interactúa con casi todas las clases del dominio.

- Que un usuario pueda convertirse en premium o dejar de serlo implica que hay una distinción de roles en la app. La forma recomendada para representar esto es asociando

una jerarquía de roles a la clase Usuario. Puesto que para esta aplicación, disponemos únicamente de un rol se ha considerado emplear un atributo booleano que establece cuando un usuario es Premium.

- La implementación de Filtros se realiza aplicando el patrón Estrategia.

- Se define la clase FiltroSinFiltro que implementa la interfaz Filtro para representar que en ese momento dicho usuario no tiene ningún filtro aplicado.

- La lista de videos recientes de un Usuario se ha decidido que no se implementará como una videolist del usuario por defecto.

- Las etiquetas relacionadas con los videos se representarán mediante un objeto de la clase Etiqueta.

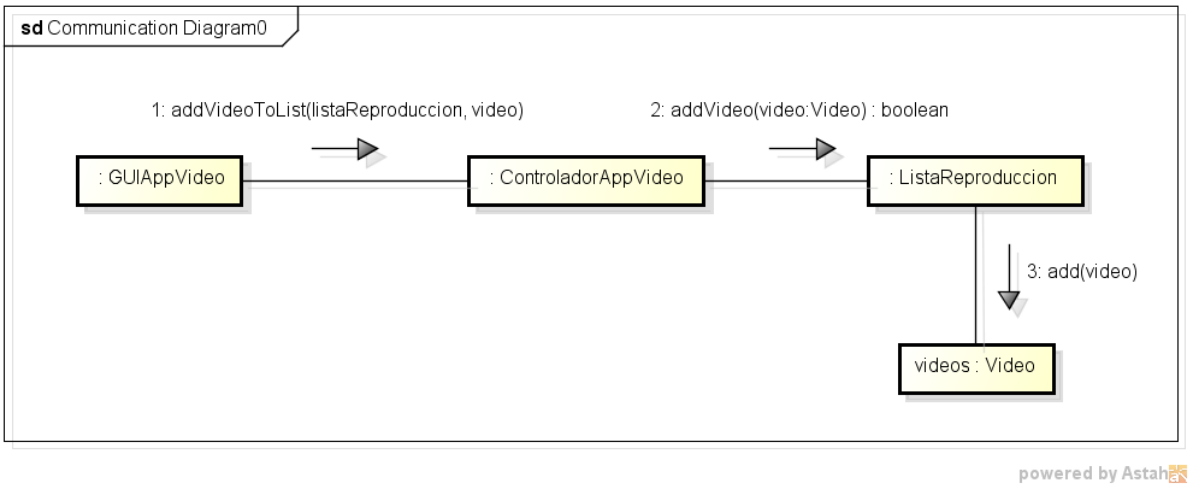
- La lista de videos top-ten de los usuarios premium será calculada, obteniéndose así cada vez que se solicite, a partir de obtener el conjunto de videos de la app y ordenar por número de visualizaciones.

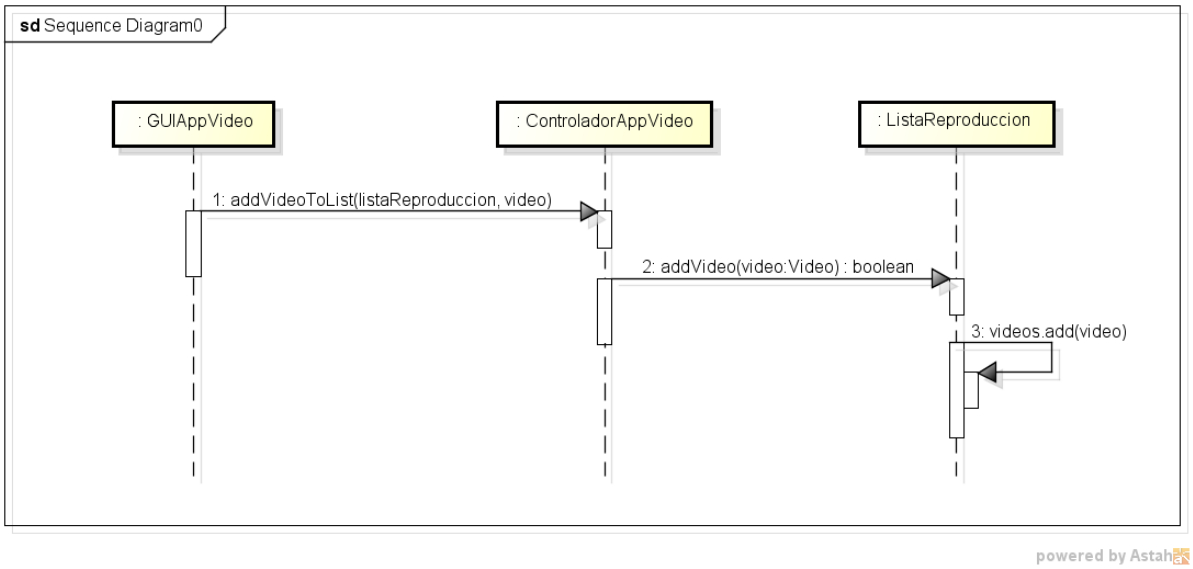
- Las clases persistentes serán Usuario, Video, ListaReproduccion y Etiqueta.

3.- DIAGRAMA DE INTERACCIÓN UML.

A continuación se mostrará un diagrama de secuencia y comunicación el cual representa el hecho de que un Usuario del sistema añada un Video a una ListaReproduccion.

Operación	**addVideoToList(listaReproduccion : ListaReproduccion, video : Video)**
Referencias	CDU 1. Registrar Video en ListaReproduccion.
Controlador	ControladorAppVideo
Precondiciones	
● Existe un Usuario **usuario** registrado en el sistema el cual posee la ListaReproduccion **listaReproduccion**. ● Existe un Video **video** previamente registrado en el sistema.	
Postcondiciones	
● Se asocia **video** con **listaReproduccion**.	





4.- ARQUITECTURA DE LA APLICACIÓN.

AppVideo se organiza de acuerdo al modelo estudiado en la parte teórica de la asignatura denominado ***modelo de tres capas***, de forma que cada una de las capas reúne clases relacionadas con un mismo aspecto del sistema (GUI, dominio o almacenamiento) evitando así el acoplamiento, lo que favorece el cambio, la reutilización y el desarrollo en paralelo.

- **Capa de presentación:** En esta capa se implementan las funcionalidades relacionadas con la interfaz de usuario (GUI). Para la creación de las mismas se hace uso de Swing, una biblioteca gráfica para Java. Esta capa hará uso del controlador de AppVideo que actuará de fachada entre esta misma capa y la de dominio.

- **Capa de dominio (Negocio):** En esta capa se realiza la implementación de la logica y reglas de negocio. Al separar esto mismo y evitar un acoplamiento con el resto de capas se permite que para esta misma aplicación se puedan construir distintas interfaces gráficas o se puedan emplear distintos servicios de persistencia/bases de datos. En esta capa se añaden catálogos (de Usuarios y Videos) para tener en

memoria los objetos de dichas clases y que la operación de recuperación sea más eficiente en tiempo de ejecución.

- **Servicios técnicos:** en esta capa se incluyen los servicios y frameworks de persistencia, los cuales se usan para el almacenamientos de los datos de la aplicación. En nuestro caso se emplea el servicio de persistencia H2 desarrollado y facilitado por los profesores de la asignatura Tecnologías de Desarrollo De Software.

Si observamos la estructura de paquetes de nuestro proyecto podemos encontrar los siguientes:

- **lanzador:** Contiene la clase con el método *main*, encargado de lanzar la aplicación.
- **vista:** Contiene las clases relacionadas con la capa de presentación.
- **controlador:** Contiene la clase relacionada con el controlador de la aplicación.
- **modelo:** Contiene las clases relacionadas con la capa del dominio.
- **persistencia:** Contiene las clases relacionadas con la persistencia de los datos.

5.- PATRONES DE DISEÑO UTILIZADOS.

5.1.- PATRONES GRASP.

Los patrones GRASP en programación orientada a objetos describen los principios básicos de asignación de responsabilidades a clases. A continuación se indican cuales se han empleado con un claro ejemplo de uso de la aplicación AppVideo.

- **Controlador:** Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es el controlador quien recibe los datos del usuario y quien los envía a las distintas clases según el método llamado. En mi caso tengo ControladorAppVideo encargado de hacer de fachada entre la presentación y el dominio. Tendrá al menos un método para cada operación que el sistema debe realizar. Permite aumentar la cohesión y disminuir el acoplamiento.

- **Experto:** Este patrón nos indica que debemos asignar una responsabilidad a la clase que contiene la información necesaria para realizarla, de esta forma evitamos tener una clase “dios” (controlador). Por ejemplo, cuando se crea una ListaReproduccion o se añade un Video a una de estas en lugar de solicitar las ya existentes y realizar las comprobaciones para añadir o no, esto se delega en un método de la propia clase que es el que tiene la información para realizarlo.

- **Creador:** Este patrón nos ayuda a identificar quién debe ser el responsable de crear la instancia de un determinado objeto. Por ejemplo, no tendría sentido que ControladorAppVideo sea el encargado de crear un objeto de tipo ListaReproduccion, sino más bien esta acción debe ser delegada y desempeñada por la clase Usuario ya que esta clase además de contener la información necesaria para su creación, posteriormente a crearla almacenará dicha instancia en una estructura de datos. Permite bajo acoplamiento, lo cual supone facilidad de mantenimiento y reutilización.

- **Bajo Acoplamiento:** Se trata al máximo de evitar dependencias innecesarias entre clases. Además aplicando de forma correcta otros patrones mencionados anteriormente también se favorece esto mismo.

- **Alta Cohesión:** Se aplica tratando que la legibilidad y reusabilidad del código sea bastante alta, a la vez que la complejidad se mantiene. Por ejemplo, en el caso de la interfaz gráfica se trata de construir numerosos paneles y métodos de forma que pasándole un único parámetro con cierta información se puedan emplear en distintos sitios del código.

5.2.- PATRONES DE DISEÑO.

- **Patrón Singleton:** Patrón de diseño creacional que nos permite asegurarnos de que una clase tenga una única instancia, a la vez que proporciona un punto de acceso global a dicha instancia. En nuestro caso lo empleamos en numerosos puntos de la app, como pueden ser los adaptadores de la persistencia, los catálogos, el controlador y algunos paneles como el reproductor de la GUI.

- **Patrón DAO (Data Access Object):** Pretende independizar la aplicación de la forma de acceder a la base de datos, o cualquier otro tipo de repositorio de datos. Para ello se centraliza el código relativo al acceso al repositorio de datos en las clases llamadas DAO. En nuestro caso se emplea para desacoplar la capa de almacenamiento del resto de la aplicación, para ello se han creado 4 interfaces DAO (IAdaptadorUsuarioDAO, IAdaptadorVideoDAO, IAdaptadorListaReproduccionDAO y IAdaptadorEtiquetaDAO) que contienen los métodos CRUD para los distintos objetos persistentes.

- **Patrón Adaptador:** Patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles. En nuestro caso se aplica en las implementaciones de las interfaces anteriormente comentadas, ya que cuando trabajamos con datos de la BD, estos están almacenados en tipos de objetos que nuestro modelo no entiende, por lo que dichas clases se encargan de transformar estos datos a objetos del dominio de nuestra aplicación.

- **Patrón Estrategia:** Este patrón ayuda a definir diferentes comportamientos o funcionalidades que pueden ser cambiadas en tiempo de ejecución. En mi caso lo empleo para implementar la posibilidad de que un usuario pueda seleccionar entre distintos filtros a aplicar en la búsqueda de sus videos. Para ello se define una interfaz la cual se encarga de definir cuál será el comportamiento en el resto de clases, por otro lado las clases que heredan de esta interfaz serán los distintos filtros que son los que implementan los distintos métodos de distinta forma dependiendo de qué videos deba filtrar.

6.- COMPONENTES UTILIZADOS.

Para este proyecto se han empleado tres tipos distintos de componentes:

- **Componente de Terceras Partes:** Se emplea el componente *"JCalendar"* para introducir la fecha de nacimiento en el panel de registro de un nuevo usuario. Para su uso puede descargar el .jar o bien como estamos empleando Maven en el proyecto añadir las dependencias en el fichero *pom.xml*, en mi caso se opta por esta segunda opción. https://mvnrepository.com/artifact/com.toedter/jcalendar/1.4. Posteriormente se añadirá a la paleta de componentes de Windows Builder para poder usarlo desde ahí.

- **Componente desarrollado en clase:** Los profesores de la asignatura proporcionan el código del componente *"Luz"* para el que se obtiene el .jar que posteriormente incluiremos en la carpeta de librerías de AppVideo, además lo añadiremos al Build Path y a la paleta de componentes de Windows Builder. Este componente será empleado en nuestro proyecto como botón para disparar el siguiente componente que se va a explicar.

- **Componente propio:** Los profesores de la asignatura proporcionan la especificación de un componente JavaBean que se deberá implementar (CargadorVideos). Para ello se proporcionan las clases necesarias para extraer de un fichero XML una lista de videos, con sus URLs, títulos y etiquetas y parsearlo a una lista de objetos de la clase Video (no la del dominio AppVideo). Dada esta funcionalidad se pide implementar una interfaz que extiende de EventListener, y otra que extienda de EventObject para conseguir capturar los eventos cuando se ejecute un método. Esto nos permitirá definir un método en ControladorAppVideo al que pasarle un fichero XML, y mediante la implementación del manejador de eventos, éste será capaz de transformar dichos vídeos a objetos de su dominio para añadirlos al sistema y que estén disponibles para todos los usuarios.

7.- TESTS UNITARIOS IMPLEMENTADOS (JUnit).

JUnit es un framework que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el

valor de retorno esperado. En nuestro caso estaremos haciendo uso de JUnit 4, para poder realizar uso de las anotaciones *@Before*, *@After*, *@Test*, etc…

En mi caso se realizan test unitarios a las clases del dominio Video y ListaReproduccion, situados en */src/test/java/appvideo/modelo*.

En esta memoria se explicarán los test implementados a la clase ListaReproduccion, ya que los implementados para la clase video se realizan de igual forma.

En primer lugar debemos declarar los elementos que vamos a utilizar, posteriormente los inicializamos, para ello emplearemos la notación *@Before* para indicar que ese método se debe ejecutar antes que el resto de test.

```
private ListaReproduccion listaReproduccion;
private Video video;

@Before
public void testListaReproduccion() {

    listaReproduccion = new ListaReproduccion("Series");
    assertNotNull(listaReproduccion);

    listaReproduccion.setCodigo(1);

    video = new Video("Invasion", "https://www.youtube.com/watch?v=DlaHxL3mHAU");
    video.addEtiqueta(new Etiqueta("Series"));
    video.addEtiqueta(new Etiqueta("AppleTV"));
}
```

A continuación se realizan una serie de test, empleando la notación *@Test*, en primer lugar sobre los métodos get/set de las clases para comprobar que los objetos se inicializan correctamente a través del constructor. Para ello se hace uso de la instrucción *assertEquals*.

```
@Test
public void testGetCodigo() {
    assertEquals(listaReproduccion.getCodigo(), 1);
}

@Test
public void testSetCodigo() {
    listaReproduccion.setCodigo(2);
    assertEquals(listaReproduccion.getCodigo(), 2);
}

@Test
public void testGetNombre() {
    assertEquals(listaReproduccion.getNombre(), "Series");
}
```

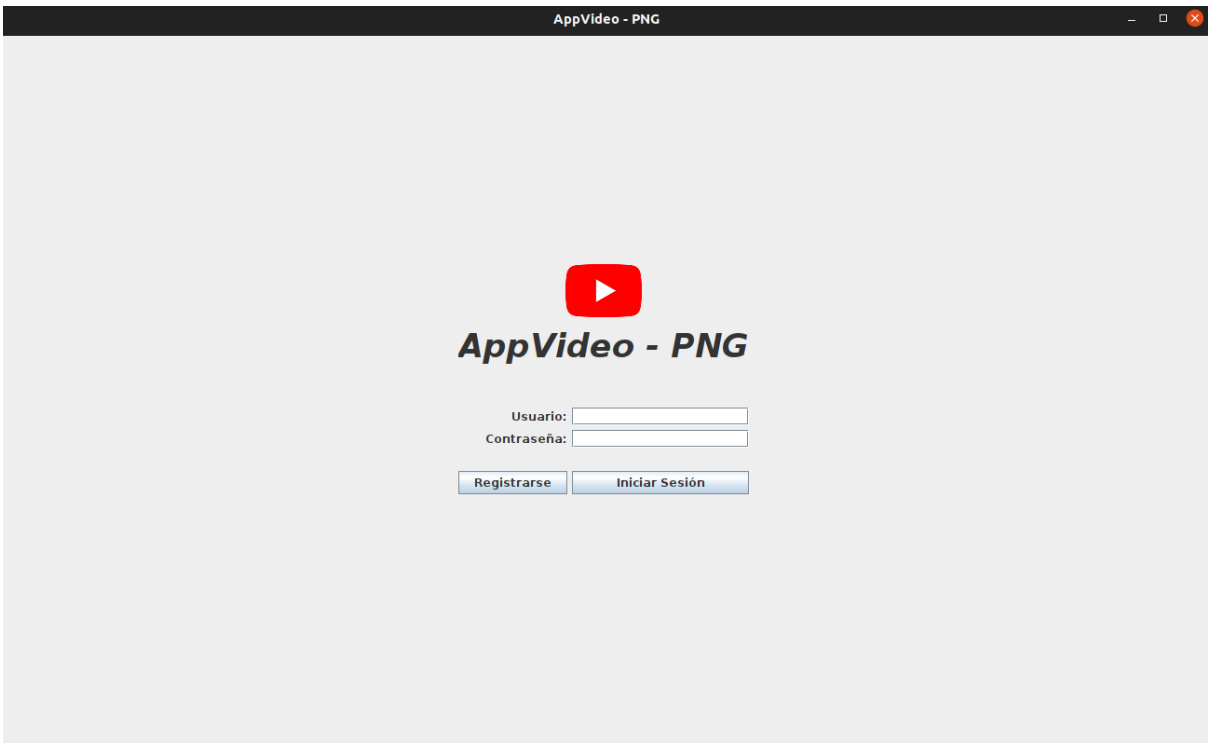
Por último, se incluyen unos test para comprobar la gestión de los videos de la ListaReproduccion. En primer lugar comprobamos que cuando creamos una lista de videos, la lista se inicializa correctamente y es una lista vacía. Posteriormente realizaremos otro test para añadir un nuevo video a la lista y después volvemos a intentar a añadirlo y comprobamos que ya no es posible (gestión de duplicados). Para finalizar se realiza un test donde se comprueba que efectivamente la operación de eliminar un video de la lista es correcta y en el caso de que el video no esté contenido en la lista nos de un error.

```
@Test
public void testGetVideos() {
    assertTrue(listaReproduccion.getVideos().isEmpty());
}

@Test
public void testAddVideo() {
    assertTrue(listaReproduccion.addVideo(video));
    assertFalse(listaReproduccion.addVideo(video));
    listaReproduccion.removeVideo(video);
}

@Test
public void testRemoveVideo() {
    listaReproduccion.addVideo(video);
    assertTrue(listaReproduccion.removeVideo(video));
    assertFalse(listaReproduccion.removeVideo(video));
}
```

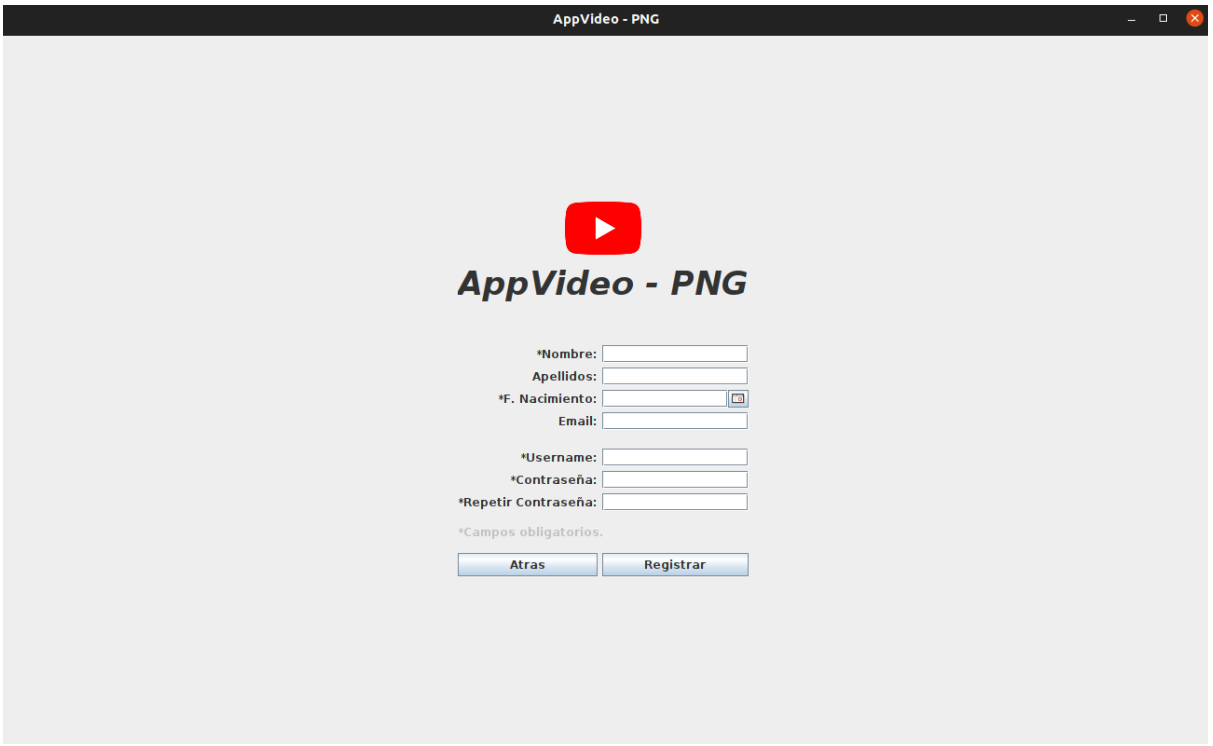
8.- MANUAL DE USUARIO.

En primer lugar debemos ejecutar la BD, para ello entramos al directorio *“ServidorPersistenciaH2”* y ejecutamos en una terminal el comando *“java -jar ServidorPersistenciaH2.jar”*.

Posteriormente lanzaremos AppVideo, entonces nos lanzará una ventana de login donde podremos iniciar sesión si ya estamos registrados, en caso contrario haciendo clic al botón “Registrarse” podremos acceder a la ventana de registro.

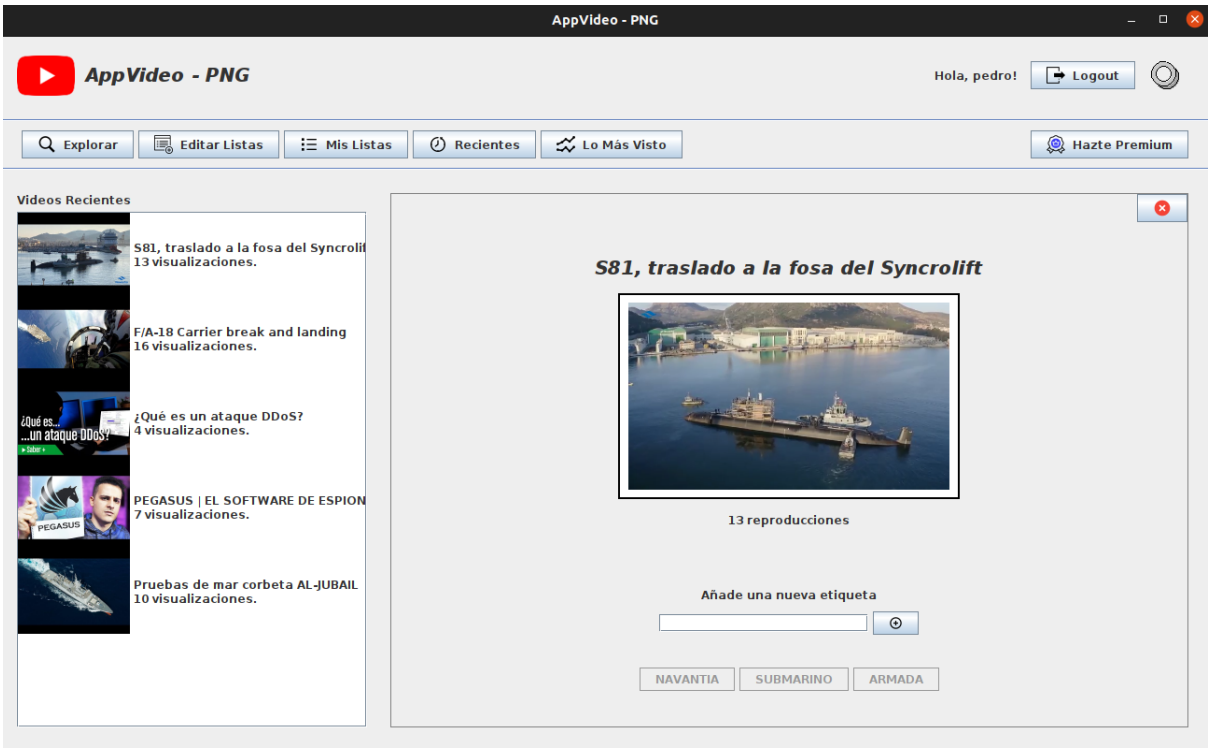


En esta ventana podremos crear un nuevo usuario rellenando los campos obligatorios (marcados con *) y haciendo clic en el botón “Registrar”. Después de esto seremos redirigidos al panel login donde podremos iniciar sesión.

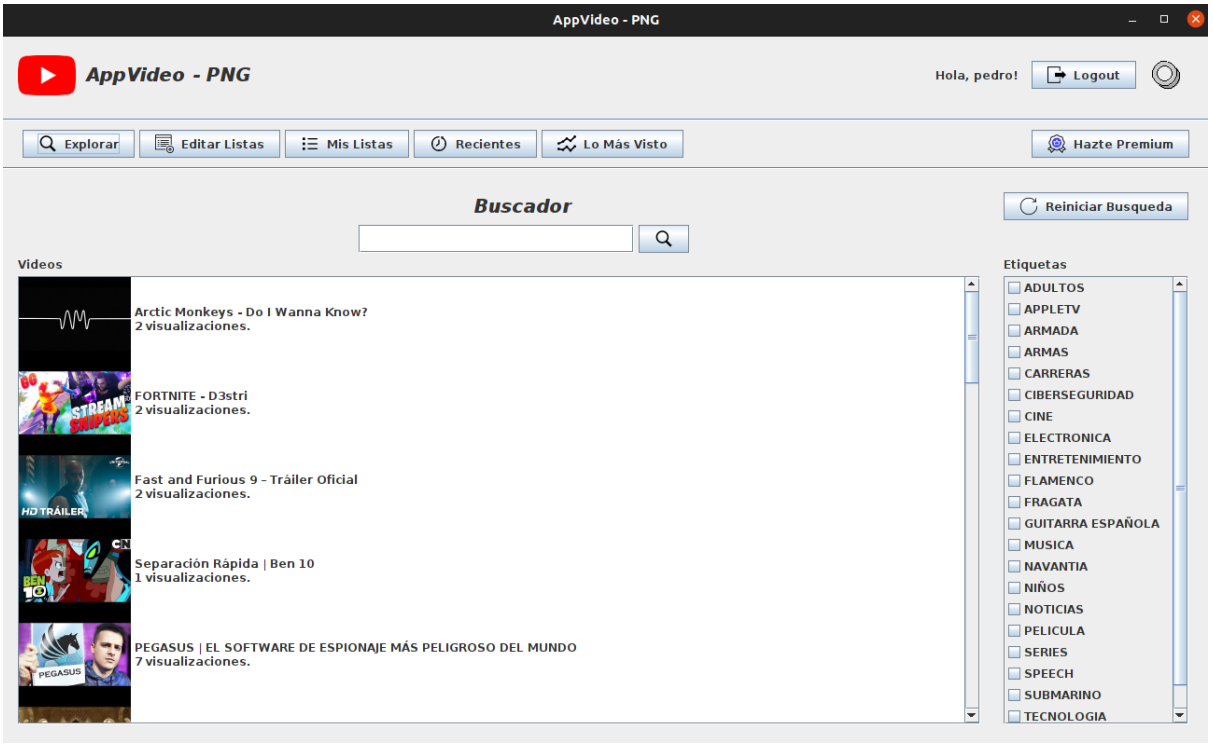


Cada vez que el usuario accede al sistema es dirigido al panel de recientes, si es la primera vez que accede al sistema, la lista de videos recientes estara vacia, en caso contrario aparecerán ordenados los últimos 5 videos que el usuario visualizó, además haciendo doble

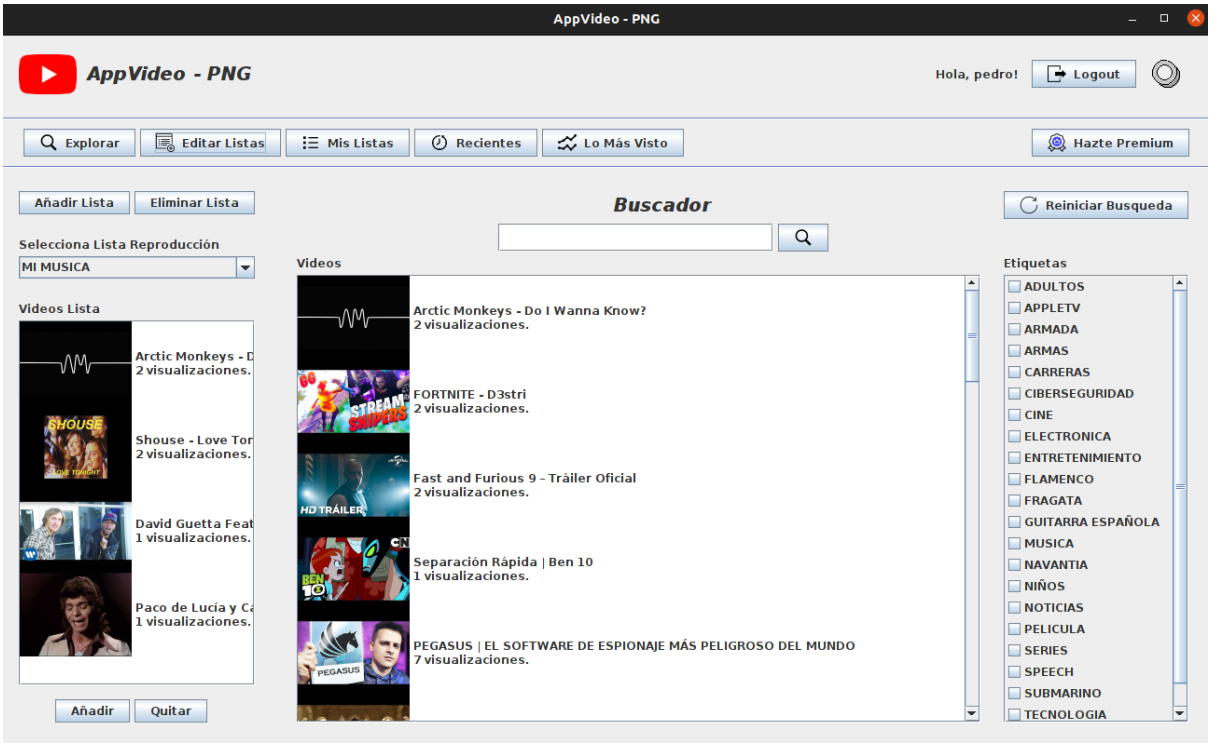
clic en alguno de los videos podrá visualizarlo, ver sus datos asociados y añadirle una nueva etiqueta.



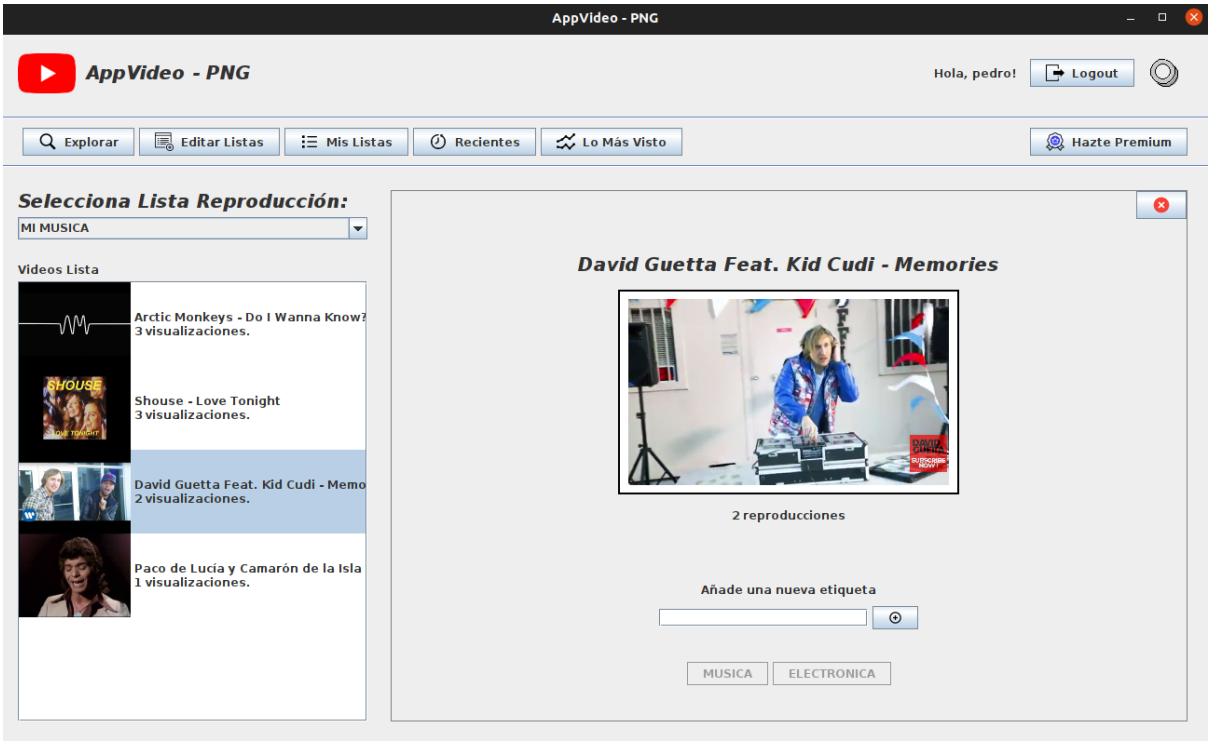
Haciendo clic en el botón “Explorar” de la barra de navegación el sistema redirigirá al usuario al panel de explorar donde podrá buscar entre todos los videos registrados en el sistema. Para realizar búsquedas más concretas el usuario podrá filtrar por texto o por etiquetas. Haciendo doble clic en alguno de los videos este comenzará a reproducirse, si el usuario se dirige a alguna ventana donde se encuentre el panel reproductor (Mis Listas, Recientes, Lo Más Visto), podrá visualizarlo.



Haciendo clic en el botón “Editar Listas” de la barra de navegación el sistema redirigirá al usuario al panel de editar listas, donde encontraremos dos paneles, uno de explorar como el explicado anteriormente y otro que nos permitirá crear o eliminar una nueva lista de reproducción. Además nos permitirá seleccionar entre alguna de las creadas para añadir o quitar videos que busquemos en el panel de explorar.

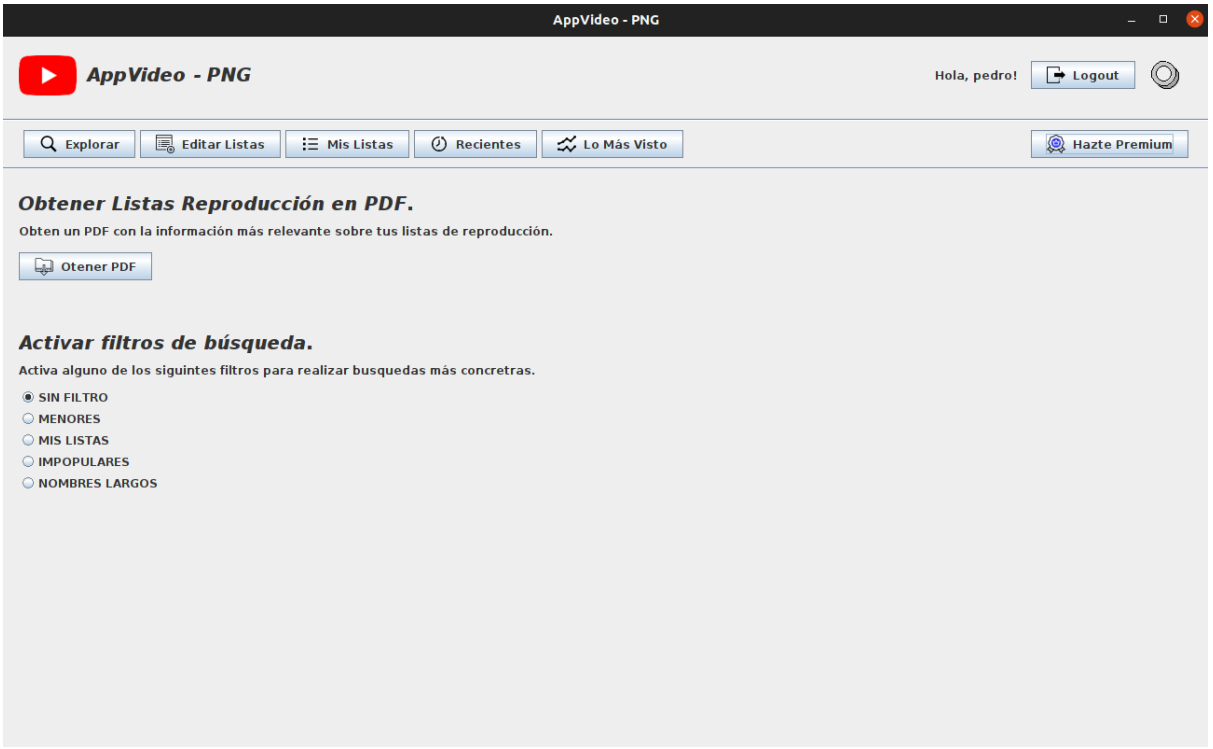


Haciendo clic en el botón “Mis Listas” de la barra de navegación el sistema redirigirá al usuario al panel de mis listas, el cual nos permitirá seleccionar una entre las distintas listas de reproducción del usuario para ver los videos que contiene y haciendo doble clic sobre uno de los videos tendra la posibilidad de visualizarlo en el panel reproductor de la derecha.



Haciendo clic en el botón “Hazte Premium” de la barra de navegación el sistema redirigirá al usuario al panel premium, donde inicialmente se mostrarán las ventajas de convertirse en un usuario de este tipo. Una vez realizado el pago de la suscripción haciendo clic en el botón “Pagar” ya tendremos acceso a la funcionalidades premium que más adelante comentaremos.

Una vez convertido el usuario en premium tendrá acceso al panel premium donde tendrá la posibilidad de hacer clic en el botón “Obtener PDF” para obtener un PDF donde podrá observar sus distintas videolist, junto a los videos que compone cada una y sus visualizaciones correspondientes. Además tendrá la posibilidad de activar algún filtro de búsqueda el cual le permitirá realizar búsquedas más concretas en los paneles explorar explicados anteriormente. El filtro “Menores” eliminará de las búsquedas aquellos videos que contengan la etiqueta Adultos, por otro lado el filtro “Mis Listas” eliminará de la búsqueda aquellos videos que esten incluidos en alguna videolist del usuario, el filtro “Impopulares” eliminará de la búsqueda aquellos videos que han sido reproducidos menos de 5 veces, por último, el filtro “Nombres Largos” eliminará de la búsqueda aquellos videos los cuales tengan más de 16 caracteres de título.



Ejemplo de PDF con información sobre las videolist de un usuario premium.

pedro_listasVideos.pdf

APP Video - PNG

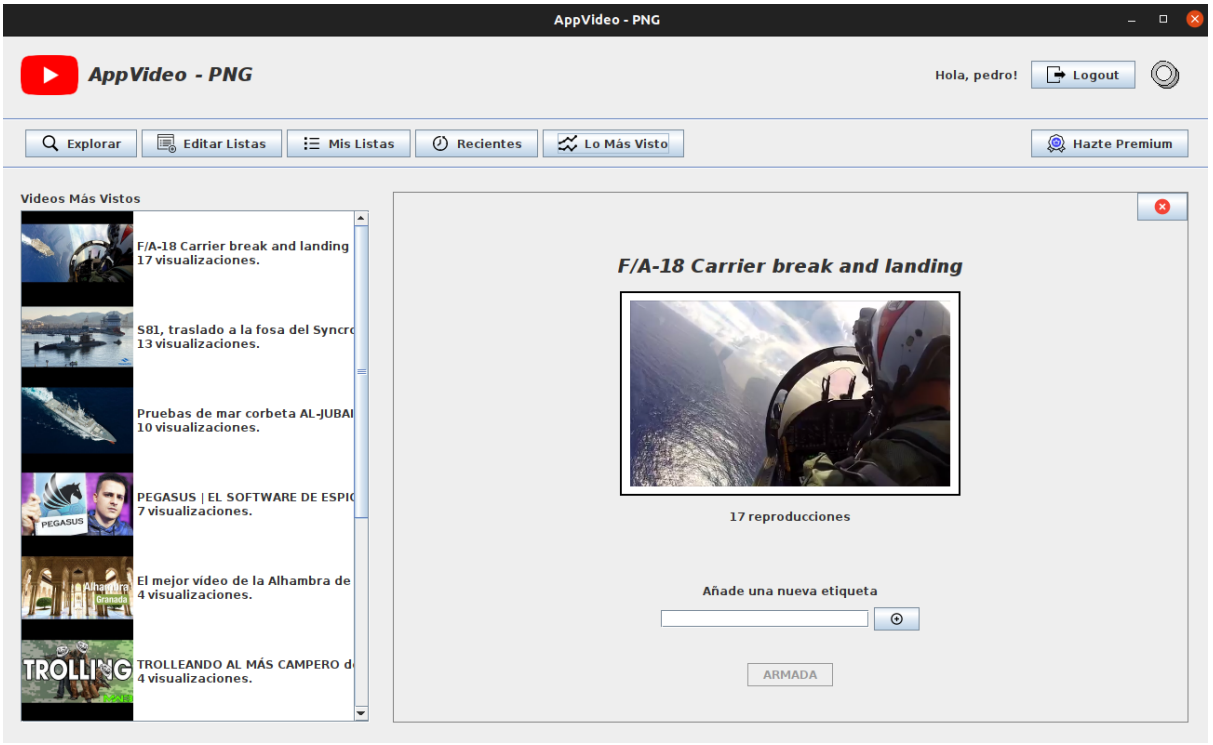
Listas de Reproducción del usuario 'pedro'.

EJERCITO
S81, traslado a la fosa del Syncrolift - 13 reproducciones.
Pruebas de mar corbeta AL-JUBAIL - 10 reproducciones.
F/A-18 Carrier break and landing - 17 reproducciones.

CIBERSEGURIDAD
PEGASUS | EL SOFTWARE DE ESPIONAJE MÁS PELIGROSO DEL MUNDO - 7 reproducciones.
¿Qué es un ataque DDoS? - 4 reproducciones.

GAMEPLAYS
FORTNITE - D3stri - 2 reproducciones.
TROLLEANDO AL MÁS CAMPERO de Modern Warfare 3 - 4 reproducciones.

Además, un usuario premium podrá hacer clic en el botón “Lo Más Visto” del panel de navegación, para que sea redirigido a un panel donde podrá ver la lista top-ten de videos de AppVideo, además haciendo doble clic en alguno de ellos tendrá la posibilidad de visualizarlo en el panel reproductor.



Por último, haciendo clic en el componente Luz tenemos la opción de abrir un JFileChooser que nos permitirá seleccionar un XML con información de videos. Esto nos permitirá cargar nuevos videos que aún no están registrados en el sistema de AppVideo, dando a los usuarios la posibilidad de poder visualizarlos. Ejemplo de fichero XML:

```
<?xml version="1.0" ?>
<videos xmlns="http://www.tds.es/videos"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
    xs:schemaLocation="http://www.tds.es/videos videos.xsd">

    <video titulo="Invasion">
        <URL><![CDATA[https://www.youtube.com/watch?v=DlaHxL3mHAU]]></URL>
        <etiqueta>Series</etiqueta>
        <etiqueta>AppleTV</etiqueta>
    </video>

    <video titulo="Technology Is Reinventing Humanity - Jordan Nguyen">
        <URL><![CDATA[https://www.youtube.com/watch?v=VxuWIXdYYyQ]]></URL>
        <etiqueta>Tecnologia</etiqueta>
        <etiqueta>Speech</etiqueta>
    </video>
</videos>
```

9.- OBSERVACIONES FINALES.

Como conclusión a este proyecto decir que me ha parecido muy interesante ya que me ha permitido aprender a realizar una aplicación totalmente funcional y de gran envergadura ya que hasta ahora no se había realizado ningún proyecto tan completo en el grado. Además

realizar todo esto me ha permitido aprender a usar Swing, aprender y poner en práctica muy buenas prácticas de programación, así como el uso de patrones y arquitectura en tres capas para obtener un código correcto, legible y reutilizable, además aprender sobre el uso/desarrollo de componentes, el uso de herramientas de desarrollo como Git y Maven y también el desarrollo de test mediante JUnit.

Con respecto al proyecto yo mismo decidí realizarlo en solitario este segundo trimestre comenzando el 13 de Febrero y finalizando el 6 de Mayo, por lo que estimo que en cuanto a carga de trabajo habré dedicado unas 110/130h a la realización del mismo. Decir que al principio fue algo costoso, ya que no tenía ni idea sobre cómo estructurarlo y las tecnologías a emplear, pero conforme vas avanzando la curva de aprendizaje se va suavizando y va resultando más ameno y satisfactorio continuar con la realización del mismo.