



APPVIDEO_TDS

Convocatoria 2021-ENERO

49973734R | 48837197D

Antonio Martínez Fernández | Mariano Marín Ciller

Antoniomartinezfernandez17@gmail.com | mariano.marinc@um.es

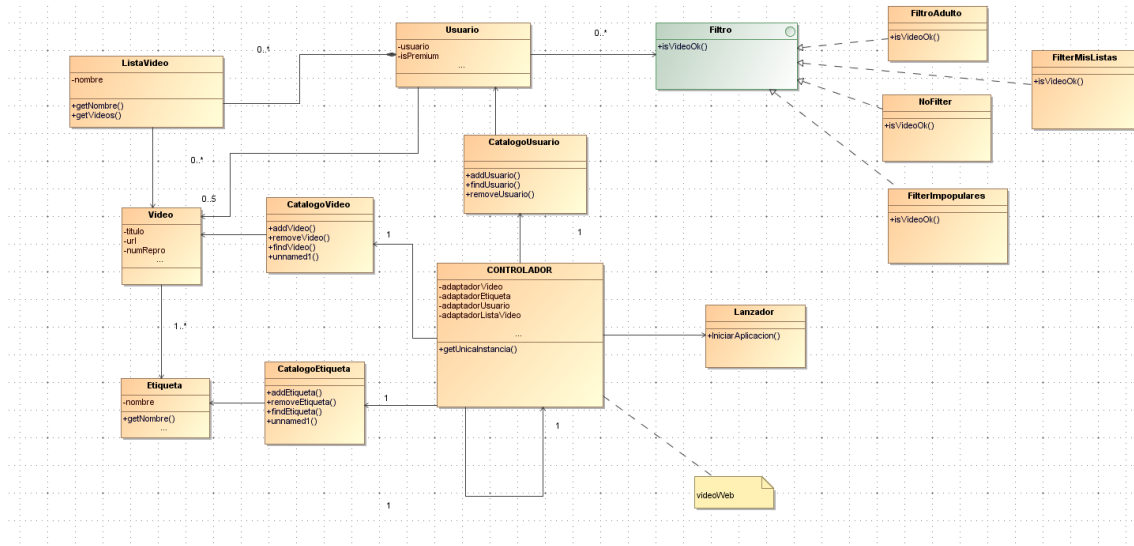
INDICE

1. Diagrama de Clases del Dominio
2. Diagrama interacción UML de registrar un video
3. Arquitectura aplicación y decisiones de diseño
4. Patrones empleados
5. Componentes usados
6. Test Unitarios
7. Manual de uso
8. Observaciones , valoración y tiempo

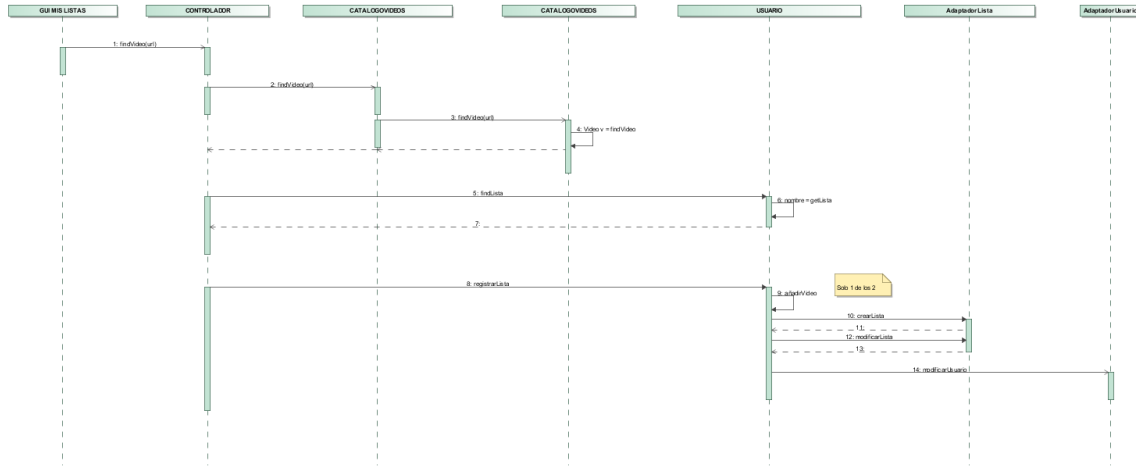
1 . Diagrama de Clases del Dominio



uml dominio.png



2 . Diagrama interacción UML de registrar un video



uml secuencia.png

3 . Arquitectura aplicación y decisiones de diseño

La aplicación ha sido desarrollada usando el patrón DAO que nos permite separar tanto la interfaz gráfica como el servicio de persistencia del esqueleto de la aplicación lo que supone un fácil y rapido cambio de estos si es necesario.

App Video consta de 4 paquetes más la clase Lanzador que inicia la aplicación.

PAQUETE CONTROLADOR

Solo contiene la clase Controlador , esta se encarga de centralizar y “enganchar” los diferentes adaptadores para el servicio de persistencia además de los catálogos locales , comunicarse con el servidor y con la interfaz gráfica sin realizar ninguna violación de los patrones que hemos estudiado en las clases de teoría.

Nos ofrece tanto funciones para acceder al servidor de persistencia para almacenamiento o carga de entidades como para obtener los datos en los catálogos locales. Controlador implementa el patrón singleton por lo cual solo habra una instancia de esta clase que se comprueba mediante introspección.

Cuando el lanzador inicia la aplicación se crea una instancia de Controlador donde se inicializan los catálogos locales con los datos del servidor de persistencia , accedido gracias a los diferentes adaptadores obtenidos desde la factoria DAO . Además consta de métodos centralizados como playVideo() que nos permite mantener un control sobre las reproducciones a un video para realizar las actualizaciones correspondientes.

Y finalmente contiene metodos de apoyo de obtención de datos o comunicación con otras capas para no violar los patrones antes mencionados de teoría además de metodos para comunicarse con el cargadorVideo implementado del tipo JavaBean para obtener los diferentes videos a partir de xml y recibir como único listener en caso de que se haya producido la carga.

PAQUETE DOMINIO

Nos encontramos un paquete que almacena los catálogos locales de videos, usuarios y etiquetas junto con las clases que definen las diferentes entidades que vamos a poder almacenar en el servidor como Usuario, Video etc. También incluye los filtros (clase abstracta y clases hijas) implementados con el patron estrategia (creados según su nombre de clase y con un método común).

PAQUETE GUI

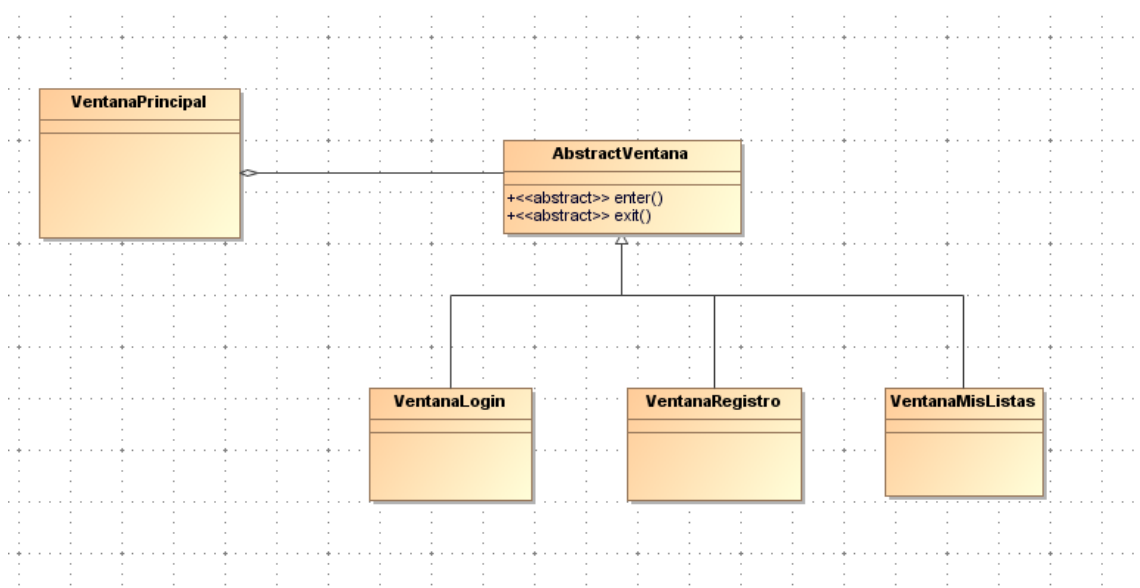
En este paquete hemos definido todas las interfaces gráficas de la aplicación que siguen la siguiente estructura.

VentanaPrincipal es un frame que contiene en su parte superior los diferentes menus para acceder a toda la funcionalidad y en la parte inferior y central se deja espacio para los diferentes paneles que irán cambiado con respecto a la opción que elijamos. Es llamada desde el lanzador al iniciar la aplicación.

Para el acceso a las demás ventanas hemos decidido usar CardLayout y dado que la implementación de este tipo no permite acceder a que card se esta mostrando actualmente , mediante un atributo cardactual conseguimos tratar la ventana de acuerdo a si se esta mostrando o no .

Las “subventanas” que se muestran detro del marco del frame general son gestionadas implementando las interfaz Iwindows que en el momento que son llamadas a ser ventana principal o dejan de serlo realizan una serie de operaciones para mostrar su contenido u ocultarlo según toque.

Se podría haber usado un patron de diseño para evitar el uso de cases y comprobaciones instance of de las instancias de los diferentes tipos de paneles de manera que simplemente tuvieramos que realizar un ventanaactual.exit() o ventanaactual.enter() en vez de comprobar su tipo dinámico. Esto se podria haber conseguido con una clase abstracta y con el método plantilla exit y enter que tendrian que implementar las instancias de la clase.



```
ventana = new VentanaLogin();
```

```
AbstractVentana ventanaActual = ventana;
```

```
ventana.enter();
```

PAQUETE PERSISTENCIA

Contiene todos los adaptadores para entidades que se van a registrar en el servidor (Listas, Usuarios, Video, Etiquetas que nos permiten tener una interfaz generalizada sin importar el tipo de persistencia que usemos además nos permitiría cambiar rápidamente en caso de que quisieramos cambiar los algoritmos detrás de la obtención de datos del servidor o el propio servidor.

Por otro lado tenemos FactoriaDAO y TDSFactoriaDAO que son necesarios para implementar una abstract factory y su implementación concreta que contiene además métodos para la obtención de los adaptadores que vamos a usar para acceder al servicio de persistencia.

Finalmente tenemos las implementaciones concretas para el acceso de datos del servidor de persistencia . Para cada tipo de Entidad se ha creado una clase que implemente su correspondiente adaptador y se han implementado dentro los métodos de creación , modificación , eliminación y obtención (tanto de una entidad como de todas) independientemente de si era requerida para la funcionalidad de la práctica o no , pensando en una mayor escalabilidad del proyecto.

Los métodos auxiliares de tratado de cadenas los hemos incluido dentro de sus clases correspondientes pero para mayor portabilidad podriamos haber creado una clase solo para contener todas las operaciones auxiliares que se podrían necesitar si la aplicación crece.

4 . Patrones empleados

Aunque hemos tocado este tema en el apartado anterior por encima vamos a proceder a listar los patrones que hemos usado para el desarrollo de la práctica.

Para la estructura global de la aplicación hemos usado el patrón Dao que se compone de diferentes patrones . Gracias a este patrón podemos separar la aplicación y otras capas de la capa de persistencia del propio servidor. Esta formado por los patrones Adapter , Abstract Factory y Singleton.

Los adaptadores han sido usados para el acceso al servidor de persistencia para las diferentes entidades .

El patrón singleton ha estado presente en varias clases ; Controlador , Factoria Dao , instancias de los adaptadores y de los catálogos . Este es usado para asegurarnos de que solo hay una instancia de dicha clase a la vez en tiempo de ejecución.

El patrón Strategy los hemos creído conveniente implementar para los diferentes filtros que puede usar el usuario . Nos permite en tiempo de ejecución cambiar de algoritmo dependiendo de la instancia que lo llame en tiempo de ejecución. Se implementa con un método plantilla para especificar lo común y un método abstracto que deben implementar las subinstancias con el proceso de filtrado de si un video es válido o no de acuerdo con el filtro.

Por otro lado en el componente se ha usado el patrón MDE , basado en el patrón Observer nos permite seleccionar un objeto el cual queremos que cuando algo cambie en él se notifique a una serie de listeners que estan suscritos a él. En el caso de nuestro componente existen los metodos dettach y attach para añadir los listeners , update para notificar a los listeners , y en listener (CONTROLADOR) una funcion enteradoCambios() que realiza lo pertinente cuando es notificado de un cambio.

5 . Componentes usados

En nuestra aplicación AppVideo hemos usado dos componentes JavaBeans : Luz y CargadorVideo .

El primero lo implementamos durante las clases de prácticas por lo que nos extenderemos poco con él . Lo hemos usado para la obtención del archivo xml con los videos a cargar gracias a una FileChooser de java , cuando se han cargado el componente recibe el evento y cambia de color a amarillo.

Por otro lado tenemos el Componente CargadorVideo , el cual , tocando un poco el tema de tiempo destinado a la práctica o problemas que hayan podido surgir nos ha dado bastantes dolores de cabeza. Al hacer la exportación del jar para incluirlo en el proyecto principal nos daba alguna clase de error debido a la jerarquía de clases de ambos proyectos que al parecer eran incompatibles. Hemos usado el método usando el manifest.mf pero con el export default de eclipse y rebuildando el proyecto conseguimos que funcionara.

Este componente contiene la clase VideoListene que hemos tenido que implementar en Controlador y que define que hará este una vez se le haya notificado el evento.

Se compone también de diferentes clases proporcionadas por los profesores de la asignatura que convertían los archivos xml en un objeto de java , que luego teníamos que transformar en el nuestro propio del proyecto.

VideosEvent extiende la interfaz EventObject y es el evento que se enviara y que contiene los videos transformados.

Finalmente en la clase principal Cargador de videos hemos implementado los métodos attach , dettach y update que como hemos explicado en arquitectura de la aplicación se encargan de añadir , quitar listeners del objeto y notificar a estos con el objeto evento de la clase VideosEvent.

6 . Test Unitarios

En la especificación de la práctica se pedía la implementación de pruebas unitarias para al menos una clase del dominio por lo que nosotros hemos elegido la clase `CatalogoVideos` ya que es la que más funciones tiene y nos ha parecido más representativa para realizarle las pruebas.

Hemos implementado un método con `@BeforeClass` que se encarga de crear los videos e inicializar los componentes necesarios .

Después hemos creado un método `@Before` para comprobar que el insertar se realizaba correctamente gracias a un `assertTrue` de para cada video si estaba contenido en la lista que esperábamos.

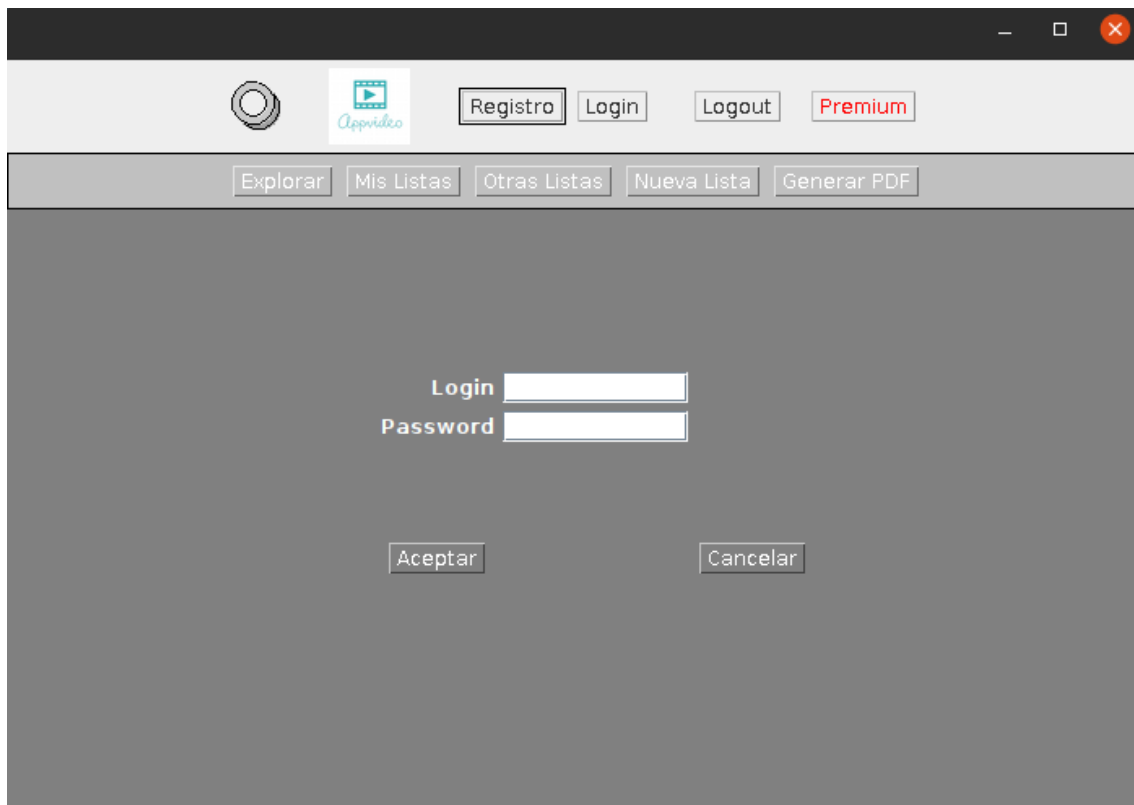
Y finalmente los métodos `@Test` para eliminar un video (comprobando que no se encontraba después con `assertFalse` del `contains` , obtener los 10 mejores videos reproduciendo según su orden un número de veces y comprobando que la lista que devuelve estaba en el orden adecuado además del método `modify video` que comprueba si los atributos se han cambiado correctamente y el método `testFilter video` que aplicando restricciones de titulo y etiquetas hemos conseguido una colección de videos igual a la esperada.

Con esto habríamos probado toda la funcionalidad de la clase `CatalogoVideos` que no incluía el servidor de persistencia.

7 . Manual de uso

CON EL SERVIDOR LANZADO CON “ JAVA -JAR SERVIDOR DE PERSISTENCIA “

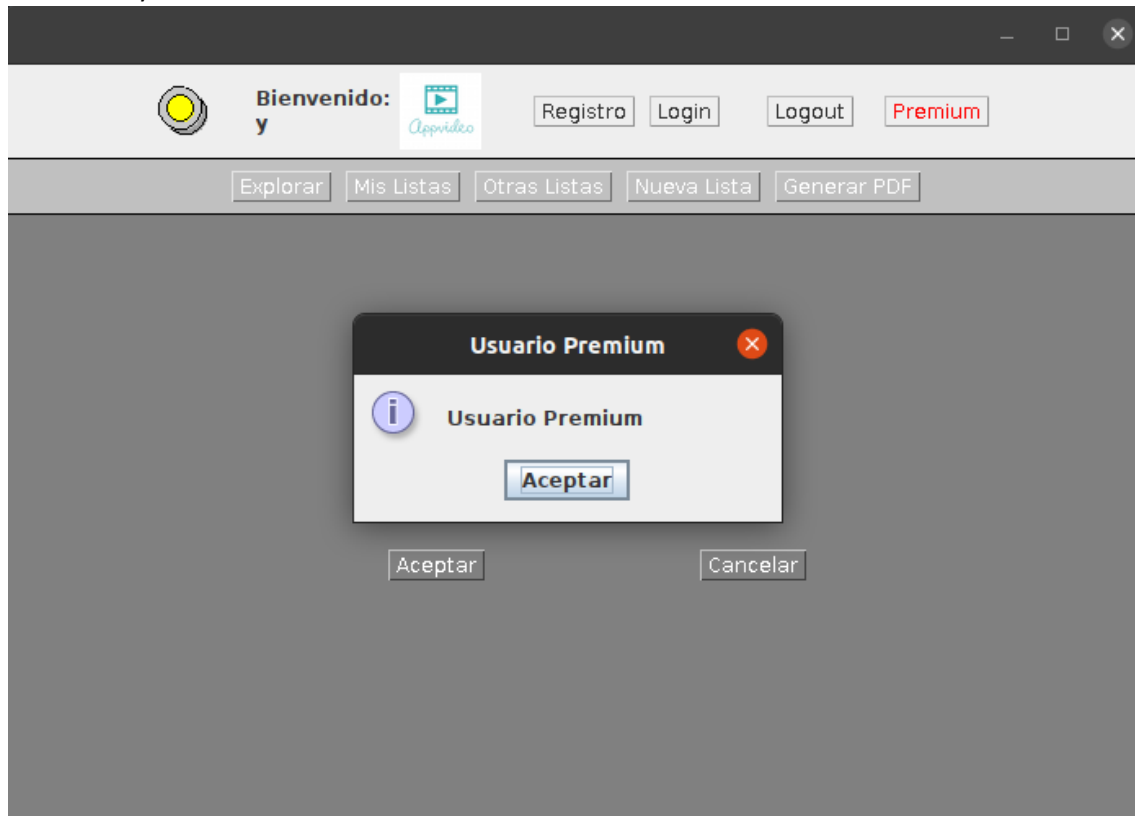
EJECUTAMOS LA CLASE LANZADOR y obtendremos lo siguiente :



The screenshot shows a web browser window with the AppVideo application. The interface includes a top navigation bar with a logo, the AppVideo name, and buttons for Registro, Login, Logout, and Premium. Below this is a secondary navigation bar with buttons for Explorar, Mis Listas, Otras Listas, Nueva Lista, and Generar PDF. The main content area is a dark gray rectangle containing a login form with fields for Login and Password, and buttons for Aceptar and Cancelar.

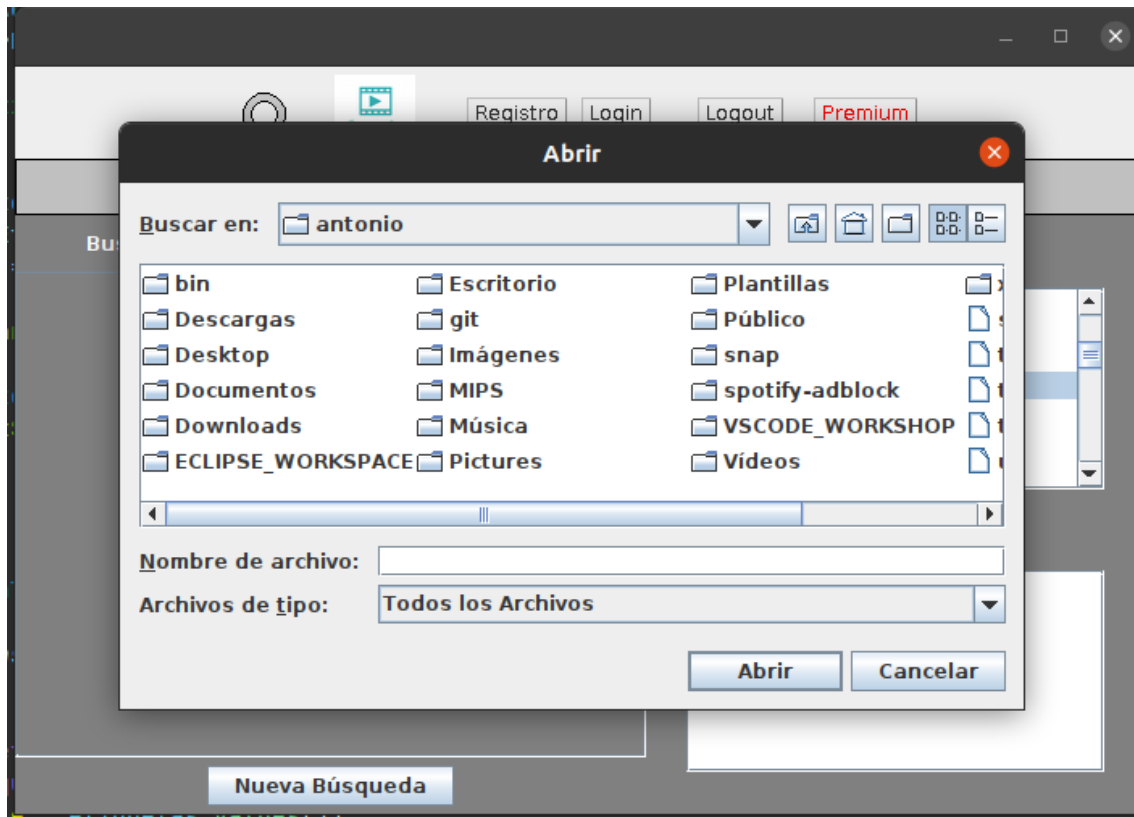
En este punto solo podemos acceder a Registro y Login y una vez lo hagamos se nos dará la bienvenida. En caso de querer que el usuario se convierte en premium simplemente clickamos

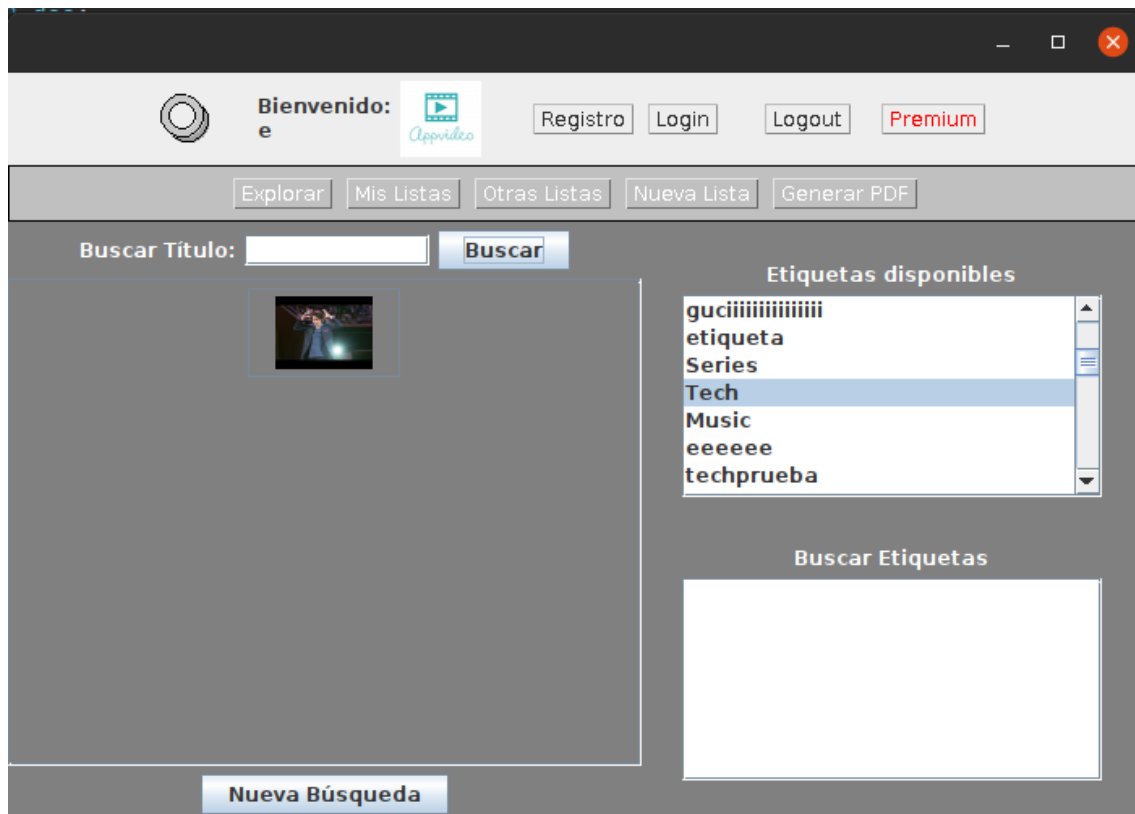
PREMIUM y nos convertiremos.



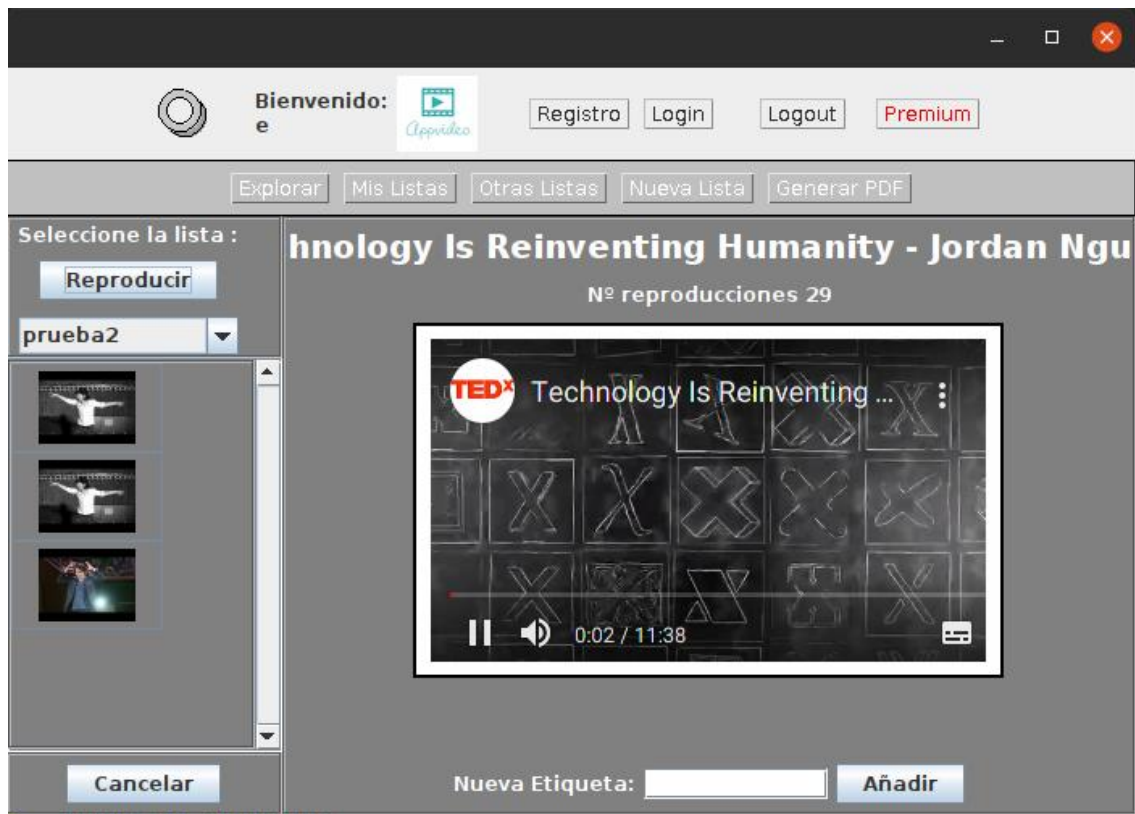
The screenshot shows the same web application interface as above, but with a registration form displayed in the main content area. The form consists of the following fields and labels: '*Nombre:', '*Apellidos:', '*Fecha nacimiento:', 'email:', '*Usuario:', '*Contraseña:', and '*Repetir contraseña:'. Each label is followed by a text input field. The 'Fecha nacimiento:' field includes a small calendar icon. At the bottom of the form are two buttons: 'Registrar' and 'Cancelar'.

Pulsando el botón circular de Luz podremos mediante un FileChooser abrir el archivo xml con la lista de videos.

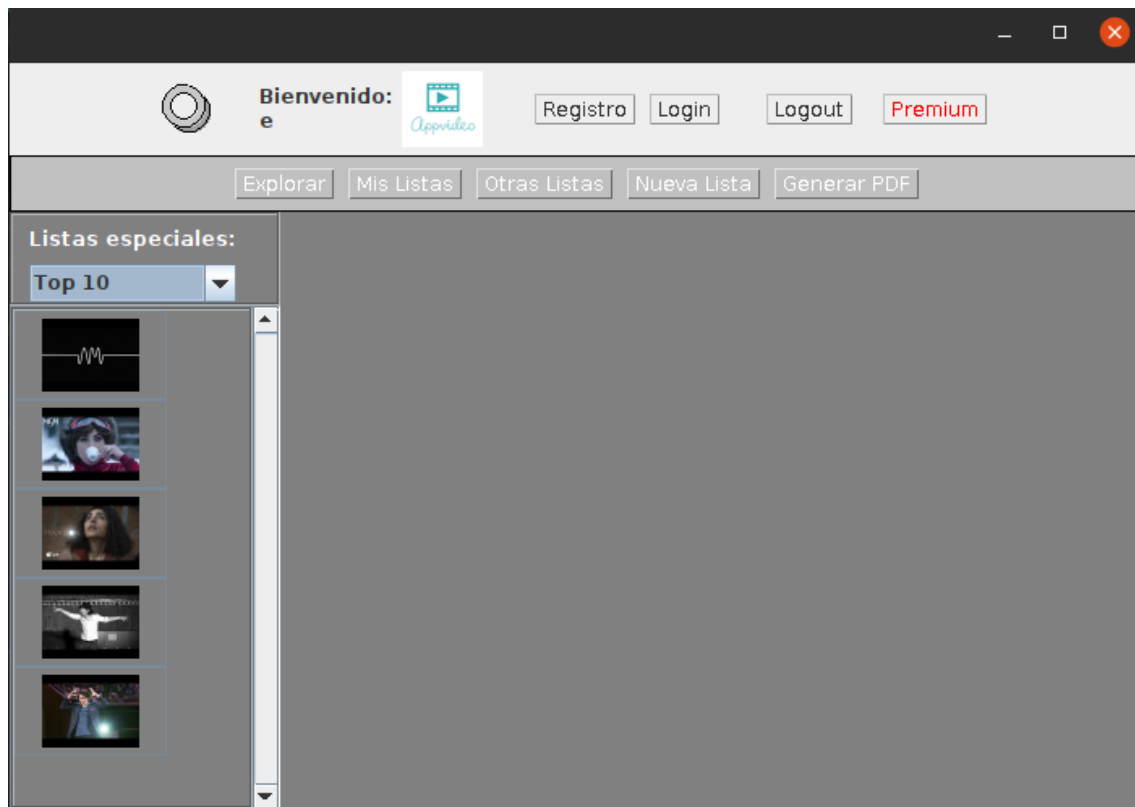




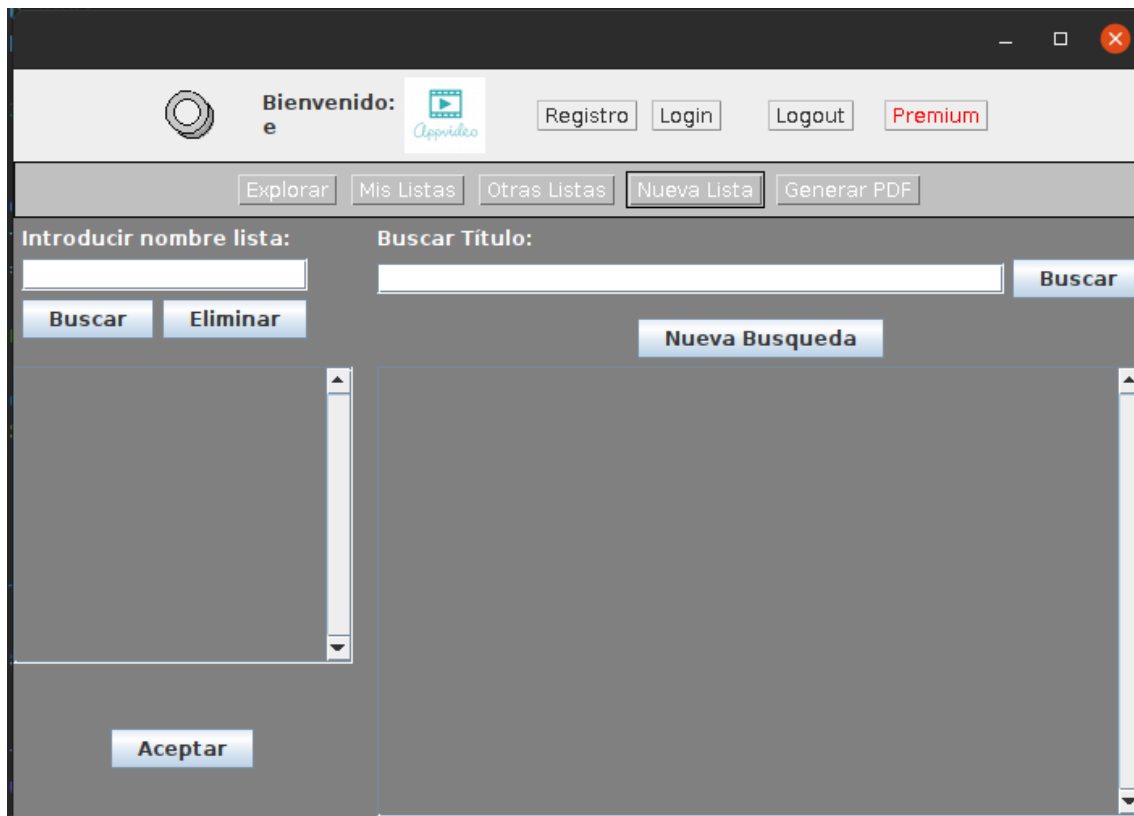
En la pestaña explorar podemos filtrar por etiqueta y por titulo los videos que haya actualmente en la base de datos . Si hacemos doble click en la etiqueta se nos añadirá a la lista de abajo . Finalmente si queremos resetear la búsqueda pulsamos Nueva Búsqueda.



En la pestaña Mis listas podemos acceder a nuestras listas creadas en NuevaLista y añadir las etiquetas que queremos a los videos que se estén reproduciendo.



En la pestaña de Otras Listas encontraremos la lista TOP 10 y los recientes (5 ultimos videos reproducidos por el usuario actual)



En Nueva lista podemos introducir el nombre de una lista nueva o a partir de una creada añadir videos seleccionandolos del lado derecho . Si queremos filtrar usaremos la barra de buscar titulo y cuando este todo a nuestro gusto clickaremos aceptar y se aplicarán los cambios.

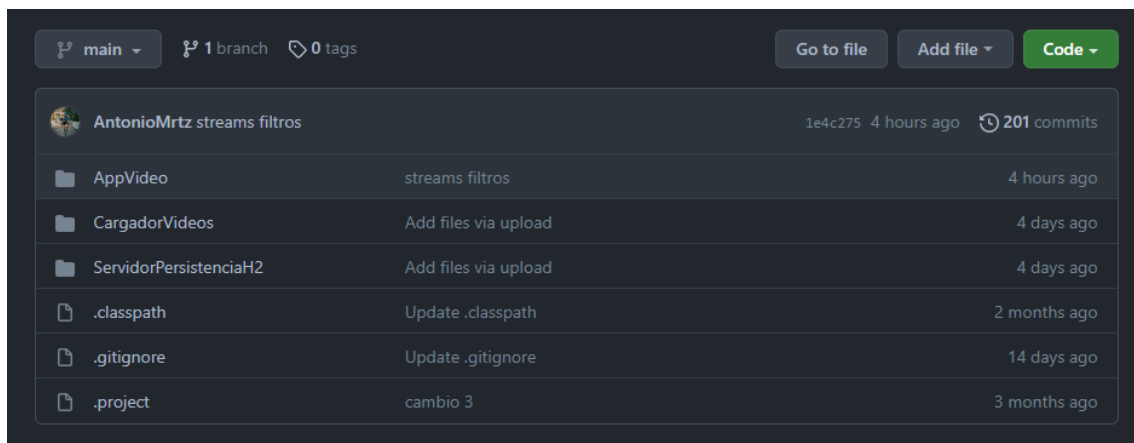
Si clicamos en Generar PDF se nos creará un pdf con las listas del usuario en el directorio actual del proyecto.

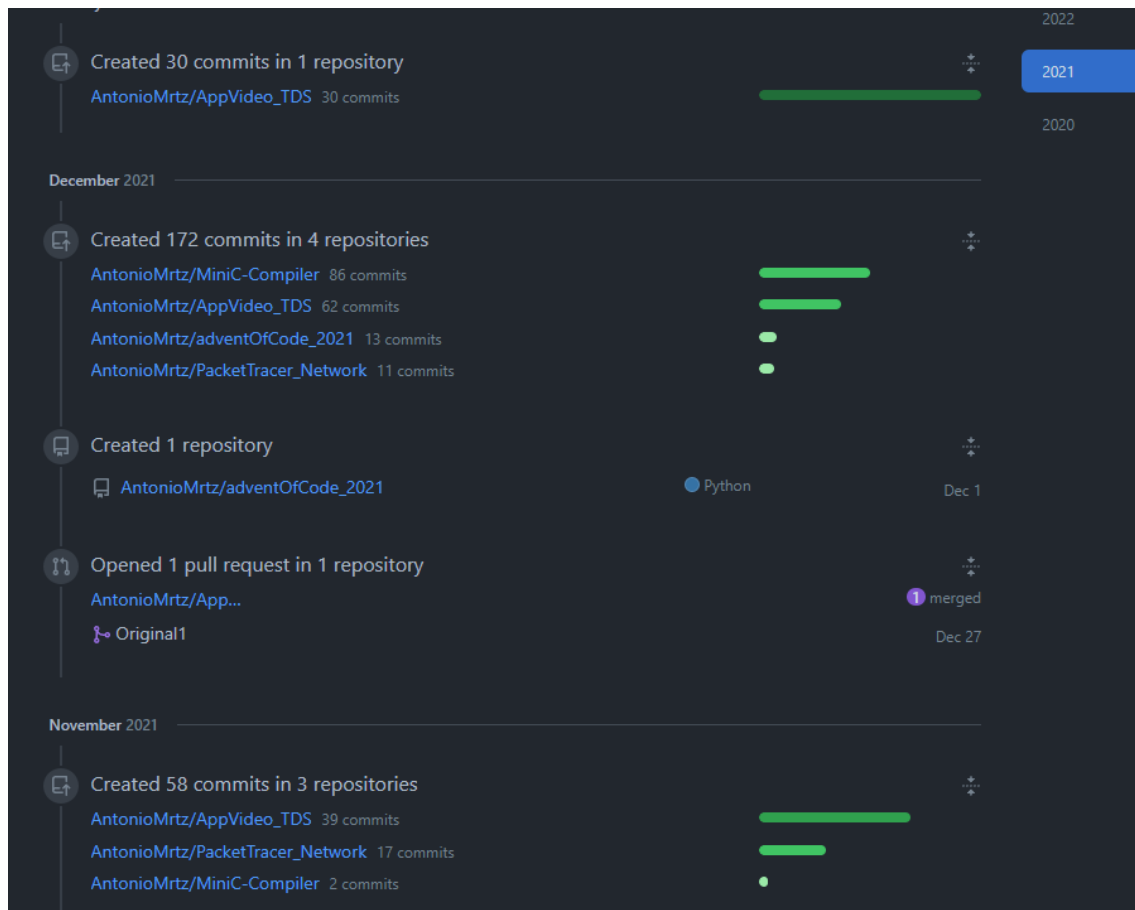
Finalmente podemos cerrar sesión pulsando log out.

8. OBSERVACIONES , VALORACIÓN Y TIEMPO

Para finalizar esta documentaciión me gustaria ofrecer un balance y una opinión sobre la práctica y la realización de la misma.

En cuanto a tiempo hemos intentado llevarla al dia sesión a sesión pero debido a la carga de trabajo del cuatrimestre antes de vacaciones solo pudimos realizar la interfaz gráfica y la estructura (esqueleto) de lo que serían las demás clases del proyecto. El calculo del tiempo exacto no sabría decirlo pero adjunto un pantallazo de los commits de github y el mapa de actividad del proyecto.





Ahí podemos observar el número de commits según el mes para el proyecto AppVideoTDS.

Por otro lado la práctica me ha parecido entretenida de realizar pero la cantidad de cosas que podían fallar o dar errores por cosas externas a programar era muy grande . Hemos tenido que solucionar muchos problemas con javafx debido a que no reconocia las subclases de esa librería y en ubuntu el solucionar problemas se nos ha complicado bastante.