

# Tecnologías del desarrollo Software

Caso práctico: Programa de escritorio AppVideo



**Desarrolladores:**

Morad Abbou Aazaz

Francisco José Lopez

**Profesor:** Francisco Javier Bermúdez

**Grupo:** 2

Departamento de Informática y Sistemas

Facultad de Informática

Universidad de Murcia

Junio, 2019

# Contents

<b>1 Modelado y Diseño</b>	<b>1</b>
1.1 Casos de uso más relevantes . . . . .	1
1.2 Diagrama de clases . . . . .	1
1.3 Colaboración: Añadir video a lista . . . . .	2
<b>2 Arquitectura de la aplicación</b>	<b>3</b>
2.1 Vista . . . . .	3
2.2 Controlador . . . . .	9
2.3 Modelo de negocio . . . . .	9
2.3.1 Catálogos . . . . .	10
2.4 Servicio de persistencia . . . . .	10
2.4.1 Adaptadores para servicio de persistencia . . . . .	10
<b>3 Patrones de Diseño utilizados</b>	<b>12</b>
3.0.1 Singleton . . . . .	12
3.0.2 Factoria Abstracta . . . . .	12
3.0.3 Adaptador . . . . .	12
3.1 Estrategia . . . . .	12
<b>4 Componentes utilizados</b>	<b>13</b>
4.1 Buscador de videos (Java Bean) . . . . .	13
4.1.1 Creación del evento que define el suceso . . . . .	13
4.1.2 Creación de la Interfaz Listener . . . . .	13
4.1.3 Crear una lista de Listener . . . . .	14
4.1.4 Implementación del método <code>setArchivoVideo</code> . . . . .	14
4.1.5 Implementación del método <code>notificarCambio</code> . . . . .	14
4.2 Generación de PDF . . . . .	15
<b>5 Pruebas unitarias</b>	<b>16</b>
5.1 UsuarioTest . . . . .	16
5.1.1 VideoTest . . . . .	16
5.1.2 TestAdaptadorDAOTest . . . . .	17
<b>6 Manual de usuario</b>	<b>19</b>
6.1 Creación de una lista . . . . .	24
6.2 Reproducción de una lista . . . . .	25
6.3 Ventana de Perfil . . . . .	26
6.3.1 Panel de control para usuarios premium . . . . .	26
6.3.2 Edición de información personal . . . . .	27
6.3.3 Listas de usuario . . . . .	27
6.3.4 Eliminar Cuenta . . . . .	28
<b>7 Funcionalidad adicional</b>	<b>29</b>
7.1 Clases Adicionales . . . . .	29
7.1.1 Clase Notificacion . . . . .	29
7.1.2 Clase VideoWeb . . . . .	29
7.2 Gestión de imágenes de las miniaturas . . . . .	31

7.2.1	Mostrar las miniaturas ordenadas de los vídeos . . . . .	31
7.2.2	Tiempo de carga de las miniaturas . . . . .	31
7.2.3	Estructuras de almacenamiento de miniaturas . . . . .	32
7.3	Utilización de Maven . . . . .	33
<b>8</b>	<b>Problemas</b>	<b>33</b>
8.0.1	Problemás con la visualización de videos . . . . .	33
8.0.2	Visualización de miniaturas de Youtube . . . . .	33
<b>9</b>	<b>Conclusiones y resultado</b>	<b>33</b>
<b>10</b>	<b>Referencias</b>	<b>34</b>

# 1 Modelado y Diseño

## 1.1 Casos de uso más relevantes

Actor	Caso de Uso	Descripción breve
Usuario	Registarse	Un usuario puede registrarse con su nombre, apellidos, fecha de nacimiento, email, nombre de usuario y contraseña.
Usuario registrado	Login	Un usuario entra al sistema con su nombre usuario y contraseña. El usuario debe estar registrado.
Usuario registrado	Buscar video	Un usuario registrado busca un video o varios videos en el catálogo de videos.
Usuario registrado	Reproducir video	Un usuario registrado puede reproducir.
Usuario registrado	Pausar video	Un usuario registrado pausa un video en reproducción.
Usuario registrado	Añadir etiquetas	Un usuario registrado añade etiquetas a un video.
Usuario registrado	Mostrar playlist	Un usuario registrado muestra una lista de reproducción afín.
Usuario registrado	Crear playlist	Un usuario registrado crea una lista de reproducción.
Usuario registrado	Añadir video a playlist	Un usuario registrado añade un video a una de sus listas de reproducción.
Usuario registrado	Eliminar video a playlist	Un usuario registrado elimina un video existente en una lista de reproducción afín determinada.
Usuario registrado	Reproducir playlist	Un usuario registrado reproduce los videos de una determinada lista de reproducción.
Usuario registrado	Contratar premium	Un usuario registrado puede convertirse un usuario premium.
Usuario premium	Mostar playlist "Mas vistos"	Un usuario premium puede mostrar y reproducir una lista que contiene los videos más vistos
Usuario registrado	Felicitar cumpleaños	El sistema muestra una felicitación a un usuario premium
Usuario registrado	Aplicar filtros	Un usuario premium puede aplicar distintos filtros en su búsqueda
Usuario registrado	Generar PDF	Un usuario premium puede generar un pdf con sus listas

## 1.2 Diagrama de clases

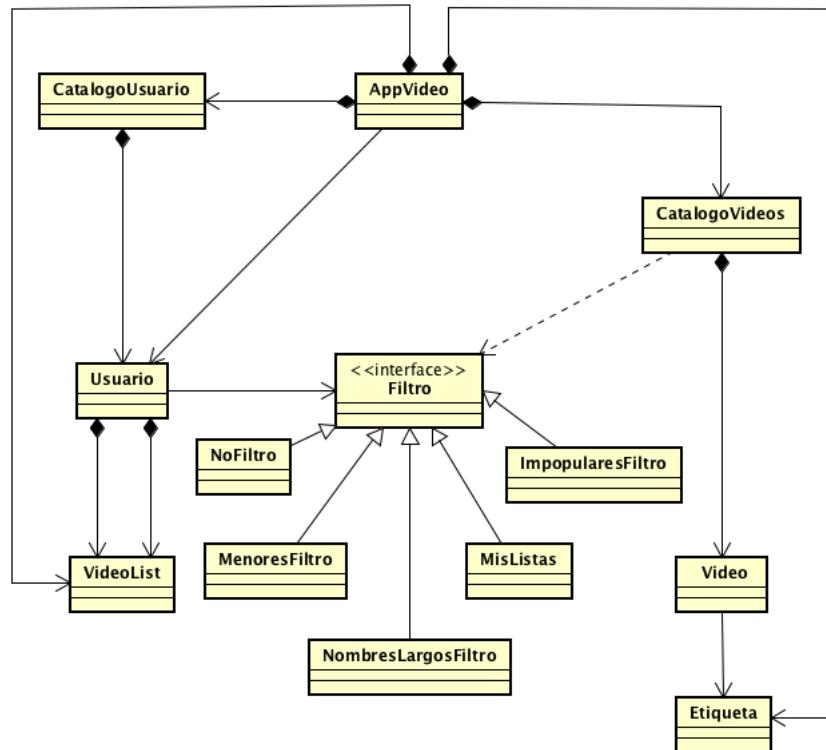


Figure 1: Añadir un video a una lista

### 1.3 Colaboración: Añadir video a lista

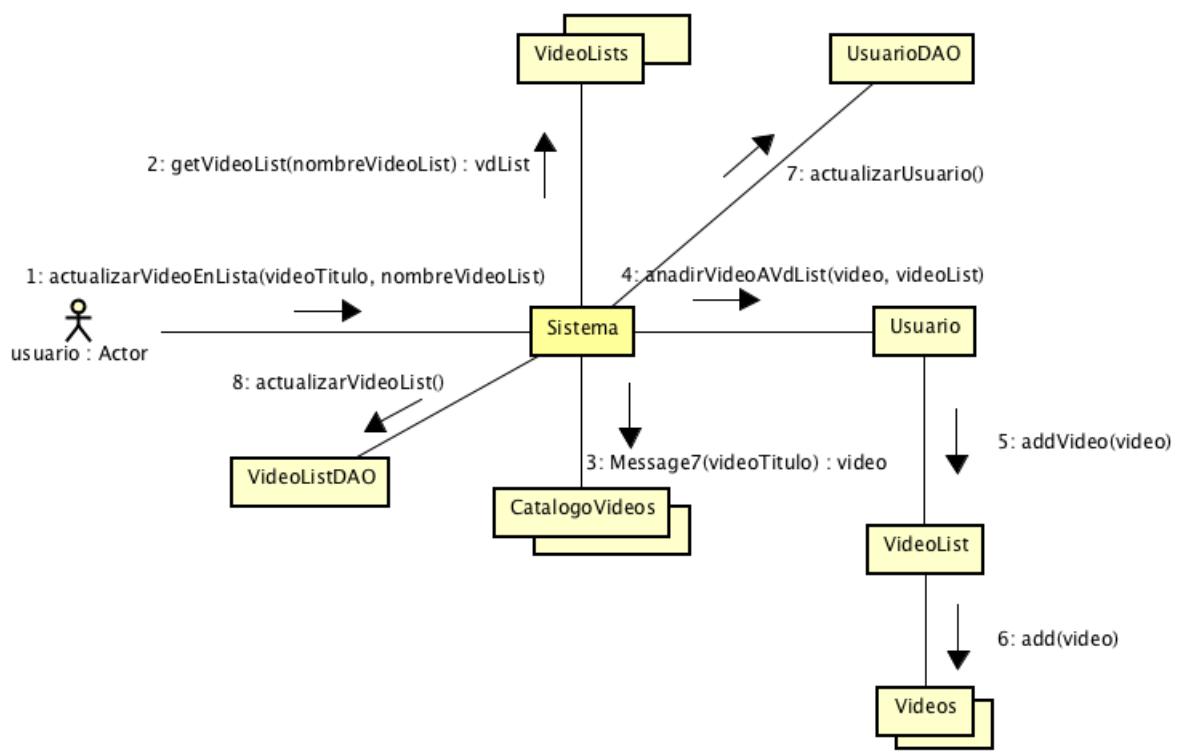


Figure 2: Añadir un video a una lista

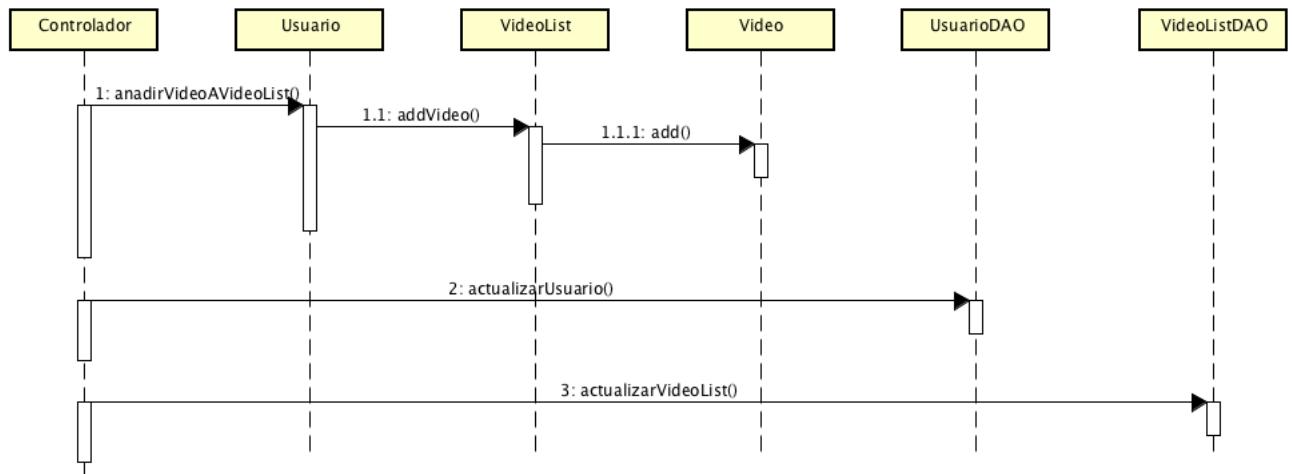


Figure 3: Añadir un video a una lista

## 2 Arquitectura de la aplicación

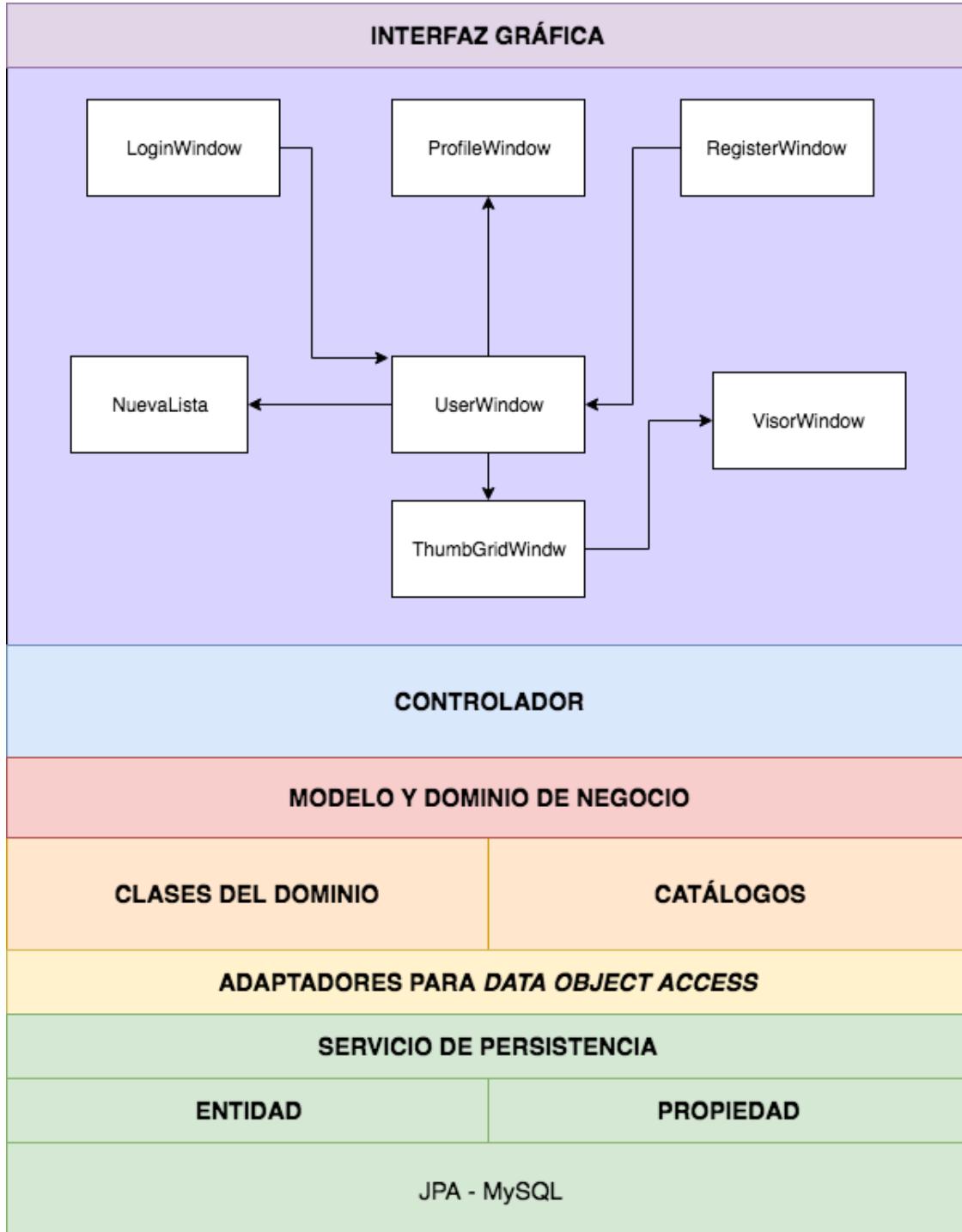


Figure 4: Caption

### 2.1 Vista

Para nuestra capa de presentación hemos utilizado la librería JavaFX. JavaFX es una familia de productos y artefactos de Oracle Corporation para la creación de aplicaciones avanzadas que hagan uso de Internet y componentes multimedia.

Una ventana en JavaFX es representada por un fichero FXML. FXML es un lenguaje basado en XML que construye el esqueleto de la interfaz de usuario para la lógica de la aplicación del código. La interfaz gráfica JavaFX ofrece un moderna y refrescante perspectiva de APIs que pueden usarse para crear aplicaciones móviles y de escritorio en la JVM. Similarmente como lo hizo su predecesor, Swing, JavaFX proporciona un conjunto de widgets estándar, así como los medios para extender este conjunto de widgets con componentes personalizados y nuevo comportamiento.

JavaFX también agrega nuevas capacidades como enlaces de propiedades (bindings), soporte de estilos a través de CSS y un formato de descripción de UI llamado FXML. Como su nombre lo indica, FXML es un formato basado en XML que permite a los desarrolladores definir interfaces de usuario de manera declarativa, en lugar de definir interfaces de manera programática, es decir, mediante el uso directo de las API de JavaFX.

El siguiente fragmento de código define una cuadrícula en la que seis elementos son posicionados en un diseño de dos columnas:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <?import javafx.scene.control.Button?>
3  <?import javafx.scene.control.Label?>
4  <?import javafx.scene.control.PasswordField?>
5  <?import javafx.scene.control.TextField?>
6  <?import javafx.scene.layout.GridPane?>
7  <GridPane fx:controller="Vistas.sample.LoginWindowController">
8      <Label text="Username:" GridPane.columnIndex="0"
9          GridPane.rowIndex="0"/>
10     <TextField           GridPane.columnIndex="1"
11         GridPane.rowIndex="0"/>
12     <Label text="Password:" GridPane.columnIndex="0"
13         GridPane.rowIndex="1"/>
14     <PasswordField       GridPane.columnIndex="1"
15         GridPane.rowIndex="1"/>
16     <Button text="Sign in" GridPane.columnIndex="0"
17         GridPane.rowIndex="2"/>
18 </GridPane>
```



Figure 5: Ventana Login resultante

Sin embargo, las vistas en formato FXML carecen de cualquier específico en la aplicación, y eso es bueno porque mantiene las responsabilidades separadas. Los elementos de vista deberían ser responsables de definir cómo se ve la UI, pero no cómo se comporta la misma; Esta es la responsabilidad de la parte del **controlador**. En términos de FXML, un controlador es un objeto que participa en el diseño de la interfaz y puede reaccionar

a eventos activados por elementos de UI definidos en la vista asociada. A diferencia del punto de entrada principal de una aplicación JavaFX, donde es requerido extender una clase específica (`javafx.application.Application`), una clase de controlador puede definir su propia jerarquía sin necesidad de extender o implementar un tipo particular relacionado con JavaFX; Esto te da rango libre para desarrollar tus propios tipos y jerarquías de controladores según sea necesario. En este caso, el controlador se encarga de inicializar y gestionar los componentes gráfico de una vista en JavaFX.

La ruta del controlador deberá ser declarado en el panel principal de la marca.

```
1 | <GridPane fx:controller="Vistas.sample.LoginWindowController">
```

Para que un componente gráfico en FXML pueda ser manipulado por el controlador, se debe asignarle un identificador `fx:id = "identificador"` dentro del la marca. Además, un fichero FXML se pueden hacer llamadas a métodos dentro del controlador cuando ocurra algun evento. Tan solo hay que saber qué evento y método será invocado. El nombre del método se pondrá tras una almohadilla. `onMouseClicked = "metodo"`

```
1 | <GridPane fx:controller="Vistas.sample.ControladorFXML">
2 |   <Label text="Username:"/>
3 |   <TextField fx:id="usernameField"/>
4 |   <Label text="Password:" GridPane.columnIndex="0"
5 |     GridPane.rowIndex="1"/>
6 |   <PasswordField fx:id="passwordField"/>
7 |   <Button text="Sign in" onMouseClicked="#signIn"/>
8 | </GridPane>
```

```
8 | public class ControladorFXML {
9 |     @FXML
10 |     private TextField usernameField;
11 |     @FXML
12 |     private PasswordField passwordField;
13 |     .....
14 |     public void signIn(MouseEvent mouseEvent) {
15 |         login(usernameField.getText(), passwordField.getText());
16 |     }
17 | }
```

No debemos confundirlo con el controlador del esquema Modelo-Vista-Controlador. Cada fichero FXML tiene su propio controlador que se encarga de mostrar la información necesaria en la vista y comunicarse con las demás clases ajenas a la vista.

En el programa se ha diseñado la arquitectura de ventanaje.

En esta aplicación de escritorio se ha optado por el diseño e implementación de la vista con JavaFX. Se han implementado un total de 7 ficheros FXML con sus respectivos controladores y se les han aplicado una serie de estilos gráficos CSS. Las vistas que han sido creadas son:

- **LoginWindow.fxml**: Es la vista principal que muestra la ventana de Login. Dicho fichero tiene un controlador **LoginWindowController** que se encarga de recoger la información de nombre de usuario y comunicarse con el controlador principal del programa.  
Un método destacado del controlador de esta vista es `login()`.

```

18  private void login(ActionEvent event) {
19      String user = userField.getText();
20      String passwd = passwdField.getText();
21      textoPassword.setText("");
22      boolean logeado = controlador.login(user, passwd);
23      if (logeado) {
24          textoPassword.setText("Cargando...");
25          System.out.println("El usuario " + user + " esta
26                             logeado");
27          Usuario usuarioActual = controlador.getUsuarioActual();
28          if (usuarioActual == null) {
29              System.err.println("El usuario " + user + " no existe");
30          } else {
31              try {
32                  gotoUserWindow(event, this.controlador);
33              } catch (IOException e) {}
34          }
35      } else {
36          textoPassword.setStyle("-fx-stroke: red;");
37          textoPassword.setText("Usuario o contraseña
38                             incorrectos");
39      }
}

```

Como se puede ver, este método recoge las cadenas de caracteres que representan al nombre de usuario y contraseña, y llaman al controlador de la aplicación pasándole esos Strings.

- **UserWindow.fxml:** Es la ventana principal de usuario. Es la vista más importante del programa y la principal para cada usuario logeado. Esta vista tiene un controlador llamado UserWindowController, que es el controlador de vistas más importante porque se encarga de la mayor parte de la gestión de los componentes gráficos y llamadas a las otras vistas. Esta vista tiene como *layout* principal un BorderPane, que es equivalente a BorderLayout en Java Swing. Dicho panel está constituido por 5 partes: Center, Top, Left, Right y Bottom, cada una con sus respectivas funciones:
  - Center: Este panel albergará varias ventanas de diferente tipo. Más adelante se comentará cuales son.
  - Left: Se utiliza para mostrar los botones que representan las listas de un usuario.
  - Right: Muestra las etiquetas disponibles para etiquetar un vídeo.
  - Bottom: Esta parte se utiliza para la búsqueda de vídeos dentro del programa y la carga de nuevos vídeos a la base de datos utilizando un componente JavaBean.
  - Top: Muestra el nombre de usuario y le felicita en su día de cumpleaños en caso de que sea premium.
- **ThumbGridWindow:** Ventana en el que estan contenidas las miniaturas de los vídeos. Esta vista está incrustada en la parte central del BorderPane de UserWin-

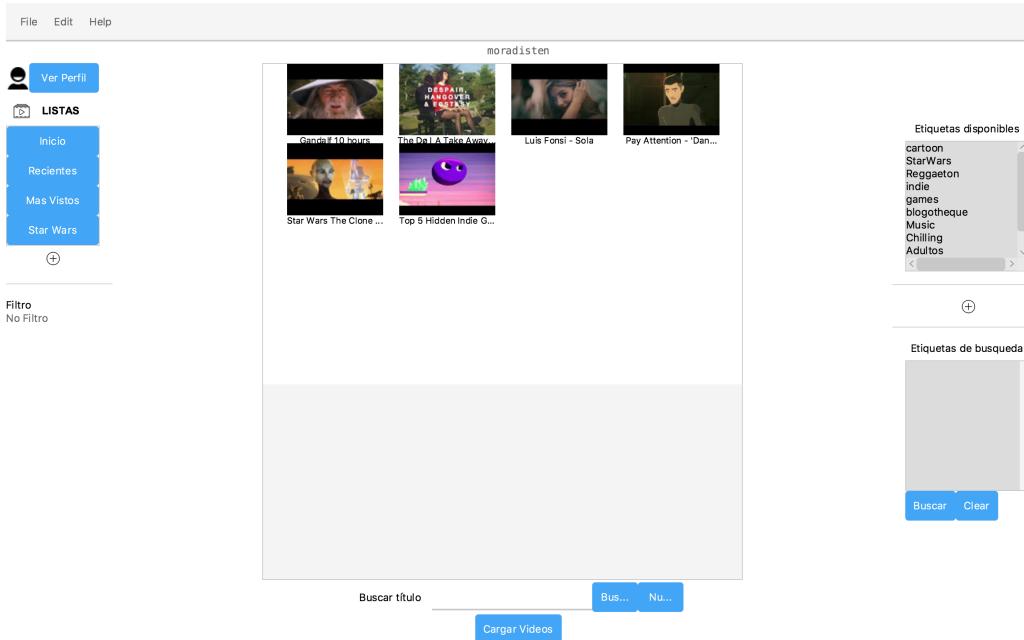


Figure 6: UserWindow

dow. Su controlador se encarga de crear una miniatura de todos los videos en la BBDD y almacenarlos en un *HashMap* de imágenes que toman una *url* como clave. El motivo del uso de un *HashMap* es el consumo de tiempo que supone la creación de una imagen en miniatura para cada url. Por lo que al crearla se almacena en un *HashMap* para mejorar el rendimiento y la velocidad de acceso a una miniatura.

- **RegisterWindow:** Ventana de registro para un usuario. Tras registrarse, se redireccionará a la ventana principal de usuario

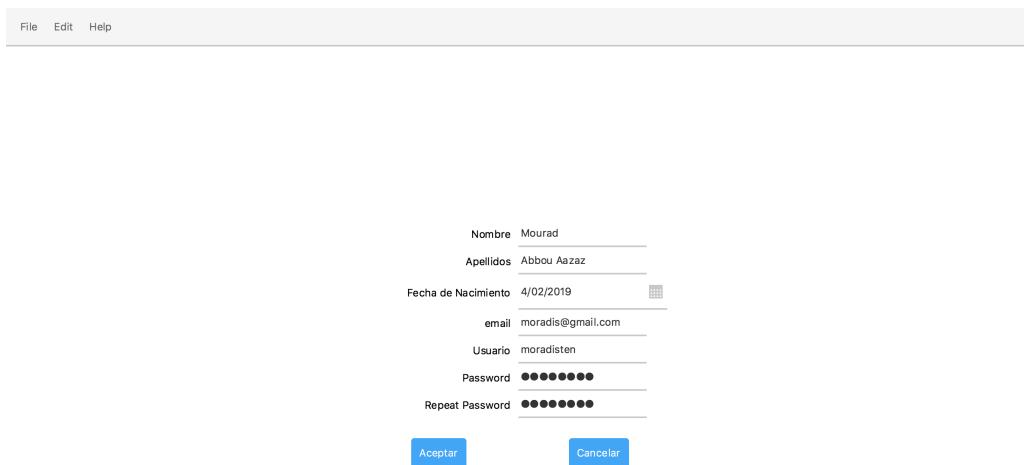


Figure 7: Ventana de registro

- **ProfileWindow** : Esta ventana representa el panel de control y ajustes del usuario, así como la ventana donde el usuario puede modificar su información, eliminar o modificar sus listas, o contratar una suscripción premium. Tambien se ubica en el centro del BorderPane.

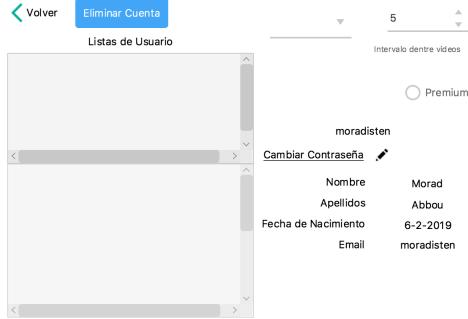


Figure 8: Ventana de perfil

En la parte izquierda, se mostrarán las listas creadas por el usuario y se podrán modificar o eliminar. En la parte inferior derecha se muestran los datos del usuario, y se podrán modificar, así como cambiar la contraseña. El nombre de usuario **no es modificable**. El resto de componentes en esta vista se explican en el manual de usuario.

- **VisorWindow** : Esta ventana se utiliza para visualizar un vídeo. En esta ventana tambien se podrán añadir etiquetas al vídeo actual que se está visualizando. También se ubica en el centro del BorderPane

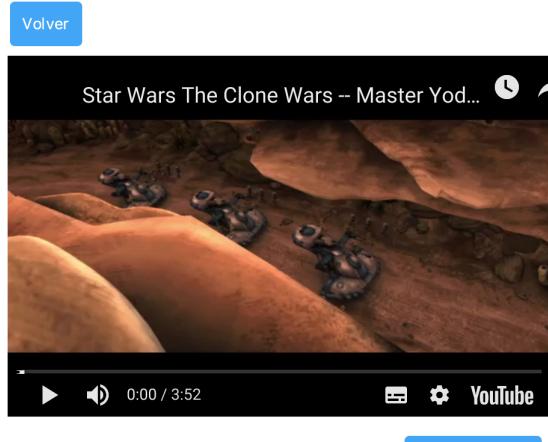


Figure 9: Caption

- **NuevaLista** : Esta vista se utiliza para crear nuevas listas, y también se posiciona en el centro del BorderPane. Cabe la posibilidad de añadirle videos en el momento de la creación.

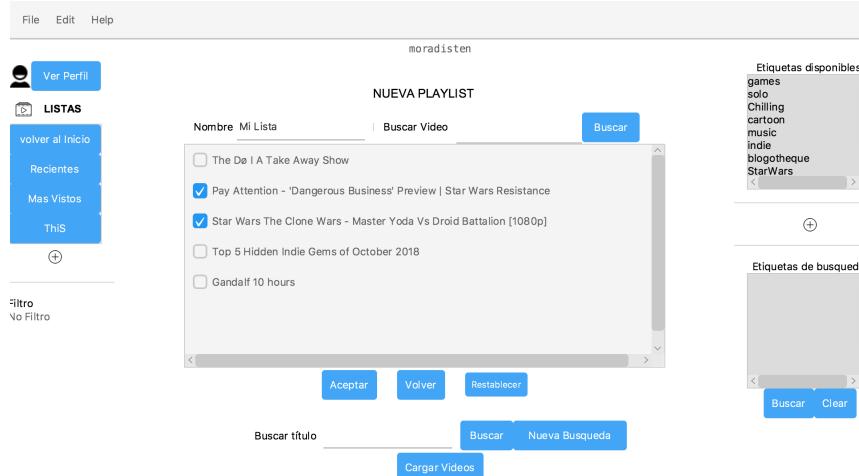


Figure 10: ventana de creación de una lista

## 2.2 Controlador

El controlador del programa es el que realiza la separación del *modelo–vista–controlador* y abstrae la lógica de negocio de las vistas. El controlador actúa como fachada para las vistas y se encarga de proporcionar los métodos de creación de objetos, así como de la intercambio de la información y solicitudes entre vista y lógica de negocio. De esta forma, las vistas se encargan únicamente a mostrar información y componentes gráficos.

```

40
41 public class Controlador implements VideosListener,
42     IBuscadorVideos {
43
44     private static Controlador instanciaUnica;
45     private FactoriaDAO factoriaDAO;
46     private CatalogoUsuarios catalogoUsuarios;
47     private CatalogoVideos catalogoVideos;
48     private AdaptadorUsuarioDAO adaptadorUsuario;
49     private AdaptadorVideoListDAO adaptadorVideoList;
50     private AdaptadorVideoDAO adaptadorVideo;
51     private Usuario usuarioActual;
52     private boolean logeado;
53     private BuscadorVideos buscadorVideos;
54     PoolEtiqueta poolEtiqueta;
55     private boolean playing;
      .....

```

## 2.3 Modelo de negocio

En la capa de modelo de negocio encontramos las clases que corresponden con la lógica de negocio y que corresponden con el diagrama de clases.

### 2.3.1 Catálogos

Los catálogos implementados ayudan a tener una visión de alto nivel sobre los objetos almacenados y encapsulan todas la operaciones de consulta sobre el objeto. Esto ayuda a que las consultas no se hagan directamente en el servicio de persistencia para mejorar el acceso y la velocidad. El controlador será el encargado de cargar los objetos en los catálogos al iniciar el programa. Los catálogos propuestos por la práctica de asignatura son *CatalogoUsuario* y *CatalogoVideos*.

## 2.4 Servicio de persistencia

En la capa de servicio de persistencia, se ha utilizado una base de datos proporcionada por el profesorado de la asignatura. Para clases relacionadas con el acceso al servicio de persistencia. Como servicio de persistencia de las aplicaciones orientadas a objetos se usan comúnmente frameworks que proporcionan la funcionalidad relational mappers como JPA/Hibernate , y que además se encargan de tareas como el manejo de transacciones o de una caché. En nuestro caso utilizamos un servicio de persistencia de objetos creado para la asignatura, dado que el uso de un framework objeto-relacional cae fuera del ámbito de esta asignatura y el uso de serialización de objetos no sería muy realista. No obstante, como se explicará más adelante, se aplicará el patrón DAO (Data Access Objects) para conseguir que la aplicación sea independiente del servicio de persistencia usado. El servicio de persistencia será explicado en un documento aparte. Existen dos clases para la base de datos: Entidad y Propiedad. La entidad corresponde al objeto que se almacena en la BBDD que corresponde con el objeto de una clase del domino. Cada entidad tiene una o varias propiedades que corresponden con los atributos de un objeto.

### 2.4.1 Adaptadores para servicio de persistencia

Los adaptadores son clases que implementan una interfaz y proporcionan un acceso a las entidades de la BBDD de las clases persistidas. De esta manera, el resto de la aplicación se abstrae de la implementación de conexiones con la BBDD. Las clases persistidas son *Usuario*, *Video* y *VideoList*.

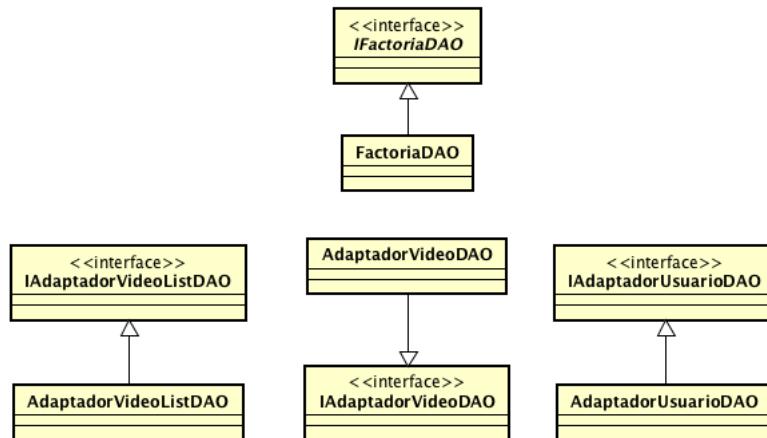


Figure 11: Caption

Supuesto que un objetivo de la aplicación es conseguir independencia del servicio de persistencia utilizado se utiliza el patrón DAO, estudiado en clase, que se basa en el empleo

del patrón Adaptador y Factoría Abstracta. Para cada entidad persistente (en nuestro caso `Usuario`, `Video` y `VideoList`) se crean adaptadores que implementan una interfaz DAO: `AdaptadorUsuarioDAO`, `AdaptadorVideoDAO` y `AdaptadorVideoListDAO`, como muestra el diagrama de clases de abajo. Para crear un adaptador concreto, por ejemplo `AdaptadorUsuarioDAO` en nuestro caso, se aplica una factoría abstracta dado que tenemos una familia de productos: la familia es el servicio de persistencia y los productos son los adaptadores, uno para cada entidad. La factoría se implementa como un *singleton*. Por tanto, nuestra capa de persistencia estará formada por los adaptadores para el servicio de persistencia proporcionado y las clases de la factoría abstracta. Comenzaremos mostrando la clase raíz de la jerarquía de factorías abstractas.

```

56 public abstract class IFactoryDAO {
57
58     private static FactoriaDAO unicaInstancia;
59
60     public static final String DAO_TDS = "persistencia.
61         TDSFactoryDAO";
62
63     public static FactoriaDAO getUnicaInstancia(String tipo)
64         throws DAOException {
65         if (unicaInstancia == null) {
66             try {
67                 unicaInstancia = new FactoriaDAO(); //(
68                     TDSFactoryDAO) Class.forName(tipo).
69                     newInstance();
70             } catch (Exception e) {
71                 throw new DAOException(e.getMessage());
72             }
73         }
74         return unicaInstancia;
75     }
76
77     public static FactoriaDAO getUnicaInstancia() throws
78         DAOException {
79         if (unicaInstancia == null) return getUnicaInstancia(
80             FactoriaDAO.DAO_TDS);
81         else return unicaInstancia;
82     }
83
84     protected IFactoryDAO() {}
85
86     public abstract IAdaptadorUsuarioDAO getUsuarioDAO();
87     public abstract IAdaptadorVideoListDAO getVideoListDAO();
88     public abstract IAdaptadorVideoDAO getVideoDAO();
89 }
```

### 3 Patrones de Diseño utilizados

Los patrones de diseño utilizados para este programa son fundamentalmente la factoría abstracta y el patrón Adaptador para el servicio de persistencia.

A continuación, los patrones creados son:

#### 3.0.1 Singleton

El patrón *Singleton* es implementado por el Controlador de la aplicación, y los adaptadores de los objetos persistidos. También lo implementa el controlador gráfico de *UserWindow*.

#### 3.0.2 Factoria Abstracta

Se implementa este patrón en la persistencia para crear los objetos adaptadores DAO para acceder al servicio de persistencia.

#### 3.0.3 Adaptador

Implementado por las clases de los adaptadores de las clases persistidas, *AdaptadorUsuarioDAO*, *AdaptadorVideoDAO* y *AdaptadorVideoListDAO*. Las clases adaptadoras implementan una interfaz que conoce a sus clientes y proporciona el acceso a los objetos de la base de datos.

### 3.1 Estrategia

Utilizamos el patrón estrategia para los distintos comportamientos y restricciones en los filtros. La clase *Filtro* tiene varias clases hijas que implementan su propia restricción para los videos.

## 4 Componentes utilizados

### 4.1 Buscador de videos (Java Bean)

Hemos implementado un componente JavaBean cuya función era seleccionar un archivo XML y *parsear* la información de nuevos videos para añadirlos en la aplicación. Supongamos una clase **BuscadorCanciones** que tiene un método **setArchivoVideo**. Para notificar un cambio en dicha clase creamos una clase **ArchivoVideosEvent** que represente el cambio en esa propiedad.

```

1 public class ArchivoVideosEvent extends EventObject {
2
3     private static final long serialVersionUID = 1L;
4     protected Videos nuevoVideo;
5     public ArchivoVideosEvent(BuscadorVideos finder, Videos
6         nuevo) {
7         super(finder);
8         nuevoVideo = nuevo;
9     }
10    public Videos getNuevoVideo() {return nuevoVideo;}
11 }
```

A continuación hemos tomado los siguientes pasos:

#### 4.1.1 Creación del evento que define el suceso

Creamos una clase Evento ArchivoEvent.

```

1 public class ArchivoVideosEvent extends EventObject {
2
3     private static final long serialVersionUID = 1L;
4     protected Videos nuevoVideo;
5
6     public ArchivoVideosEvent(BuscadorVideos finder, Videos
7         nuevo) {
8         super(finder);
9         nuevoVideo = nuevo;
10    }
11
12    public Videos getNuevoVideo() {
13        return nuevoVideo;
14    }
15 }
```

#### 4.1.2 Creación de la Interfaz Listener

Creamos una interfaz que la implementarán los objetos interesados en ser notificados del cambio. En nuestro caso, creado la interfaz **VideosListener** y extiende de **EventListener**

```

1
2 public interface VideosListener extends EventListener{
3     public void nuevosVideos(ArchivoVideosEvent event);
```

4 }

#### 4.1.3 Crear una lista de Listener

Creamos una lista de **Listener** en el objeto que produce los sucesos, **BuscadorVideos**, y los métodos para añadir y eliminar objetos de esa lista.

```

1 public class BuscadorVideos {
2     private LinkedList<VideosListener> videosListeners;
3     private String archivoVideo;
4
5     public BuscadorVideos() {..}
6 }
7

```

Definimos los dos métodos que añaden objetos **listeners** a la lista que teníamos en **BuscadorVideos**.

```

1 ....
2     public synchronized void anadirVideoListener(VideosListener
3         listener) {
4         videosListeners.add(listener);
5     }
6
7     public synchronized void removeVideoListener(VideosListener
8         listener) {
9         videosListeners.remove(listener);
10    }
11

```

#### 4.1.4 Implementación del método setArchivoVideo

```

1 public void setArchivoVideo(String nuevoVideo) {
2     this.archivoVideo = nuevoVideo;
3     Videos video = CargadorVideos.cargarVideos(archivoVideo);
4     ArchivoVideosEvent event = new ArchivoVideosEvent(this,
5             video);
6     notificarCambio(event);
7 }
8

```

#### 4.1.5 Implementación del método notificarCambio

Con el método **notificarCambio** notificamos el cambio de la propiedad **Video** a los **listeners**. La utilización de la palabra clave **synchronized** evita que varios hilos se ejecuten en exclusión mutua.

```
1 private void notificarCambio(ArchivoVideosEvent event) {  
2     LinkedList<VideosListener> lista;  
3     synchronized (this) {  
4         lista = new LinkedList<VideosListener>(videosListeners);  
5     }  
6     for (VideosListener vdListener : lista) {  
7         vdListener.nuevosVideos(event);  
8     }  
9 }
```

## 4.2 Generación de PDF

Se utiliza el componente iText-5.0.5 para eliminar generar ficheros PDF con las listas del usuario.

## 5 Pruebas unitarias

### 5.1 UsuarioTest

```
1 public class UsuarioTest {  
2  
3     Usuario usuario;  
4     Video video1, video2;  
5     VideoList videoList;  
6     String ruta1 = "https://www.youtube.com/watch?v=gd1c885zg04"  
7         ;  
8     String ruta2 = "https://www.youtube.com/watch?v=WM28K5do8_k"  
9         ;  
10  
11     @Before  
12     public void initialize() {...}  
13  
14     @Test  
15     public void testAddCancionAListaInexistente() {...}  
16  
17     @Test  
18     public void testAnadirVideoALista() {...}  
19  
20     @Test  
21     public void testAnadirVideoAReciente() {...}  
22  
23     @Test  
24     public void testContieneLista() {...}  
25  
26     @Test  
27     public void testNoContieneLista() {...}  
28  
29     @Test  
30     public void testTiene23anyos() {...}  
31  
32     @Test  
33     public void testEsMayorEdad() {...}  
34  
35     @Test  
36     public void testFiltro() {...}  
37  
38     @Test  
39     public void cambiarFiltroImpopularesSinPremium() {...}  
40  
41     @Test  
42     public void cambiarFiltroConPremium() {...}  
43 }
```

#### 5.1.1 VideoTest

```

1 public class VideoTest {
2
3     Video video1;
4     Video video2;
5     Video video3;
6     Video video4;
7     Video video5;
8     Video video6;
9     Video video7;
10
11    String ruta1 = "https://www.youtube.com/watch?v=gd1c885zg04"
12        ;
13    String ruta2 = "https://www.youtube.com/watch?v=WM28K5do8_k"
14        ;
15    String ruta3 = "https://www.youtube.com/watch?v=iY4TPWa2vEc"
16        ;
17    String ruta4 = "https://www.youtube.com/watch?v=n18g4bRJDCY"
18        ;
19    String ruta5 = "https://www.youtube.com/watch?v=SBKMVUnEYVc"
20        ;
21    String ruta6 = "https://www.youtube.com/watch?v=i3mlkS13fM4"
22        ;
23    String ruta7 = "https://www.youtube.com/watch?v=QCyIY10KBnk"
24        ;
25
26    VideoWeb videoWeb = VideoWeb.getUnicaInstancia();
27
28    @Before
29    public void initialize() {...}
30
31    @Test
32    public void testTitulos() {...}
33
34    @Test
35    public void testGetRuta() {...}
36
37    @Test
38    public void testGetEtiquetas() {...}
39
40    @Test
41    public void testReproducciones() {...}
42
43 }

```

### 5.1.2 TestAdaptadorDAOTest

```

1 public class TestAdaptadoresDAO {
2
3
4     private FactoriaDAO factoriaDAO;

```

```
5     private IAdaptadorUsuarioDAO adaptadorUsuarioDAO;
6     private IAdaptadorVideoDAO adaptadorVideoDAO;
7     private IAdaptadorVideoListDAO adaptadorVideoListDAO;
8     private Usuario usuario1, usuario2;
9     private Video video1, video2;
10    private VideoList videoList;
11
12    @Before
13    public void inicializarDatos () throws DAOException {...}
14
15    private void imprimirUsuario(Usuario usuario) {...}
16
17    private void imprimirVideo(Video video) {...}
18
19    private void imprimirVideoList(VideoList videoList) {...}
20
21    @After
22    public void imprimirResultadosDelaBBDD() {...}
23
24    @Test
25    public void recuperarUsuario() {...}
26 }
```

## 6 Manual de usuario

Primeramente, arrancamos el servicio de persistencia. Para eso debemos ir a la carpeta que contenga el fichero ejecutable **ServidorPersistencia.jar** para arrancar la base de datos.

```
1 $ java -jar Servidorpersistencia.jar
2 Objetos distribuidos listos : FactoriaServicioPersistencia
3 Presionar ENTER para salir (los objetos distribuidos ser n
   destruidos) ...
```

Una vez arrancado, ejecutamos el programa.  
Nos encontraremos con la pantalla principal, donde nos aparecerán dos campos de texto para introducir un usuario y contraseña.

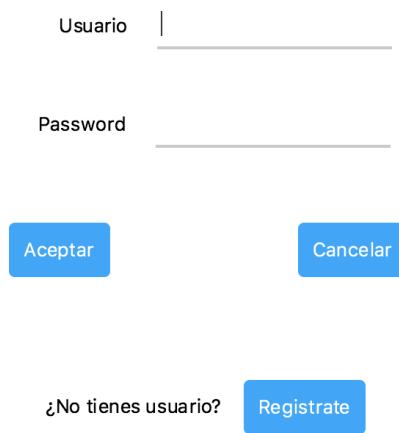


Figure 12: Ventana Login

En caso de no estar registrados en el sistema, pinchamos en el botón "Regístrate". Eso nos llevará a una ventana de registro. Dentro de dicha ventana, rellenaremos nuestro datos. No podemos poner un nombre de usuario ya existente. Una vez, hecho, pulsamos aceptar.

Nombre	Morad
Apellidos	Abbou
Fecha de Nacimiento	24/04/1996
email	mourad.abbou@um.es
Usuario	moradisten
Password	●●●●
Repeat Password	●●●●
<input style="width: 100px; margin-right: 20px;" type="button" value="Aceptar"/> <input style="width: 100px;" type="button" value="Cancelar"/>	

Figure 13: Ventana de registro

Nombre	Morad
Apellidos	Abbou
Fecha de Nacimiento	24/04/1996
email	
Usuario	
Password	
Repeat Password	
<input style="width: 100px; margin-right: 20px;" type="button" value="Aceptar"/>	

Abril 1996

Lun	Mar	Mié	Jue	Vie	Sáb	Dom
14	1	2	3	4	5	6
15	8	9	10	11	12	13
16	15	16	17	18	19	20
17	22	23	24	25	26	27
18	29	30	1	2	3	4
19	6	7	8	9	10	11
						12

Figure 14: Importante utilizar el DatePicker para seleccionar las fechas

Nombre	Francisco José
Apellidos	Lopez
Fecha de Nacimiento	12/02/1992
email	fjlopez@um.es
Usuario	moradisten
Password	●●●●
Repeat Password	●●●●
<input style="width: 100px; margin-right: 20px;" type="button" value="Aceptar"/> <input style="width: 100px;" type="button" value="Cancelar"/>	

El usuario moradisten ya existe

Figure 15: Tenemos que cambiar de nombre de usuario

Tras registrarnos, aparecerá la ventana principal de usuario.

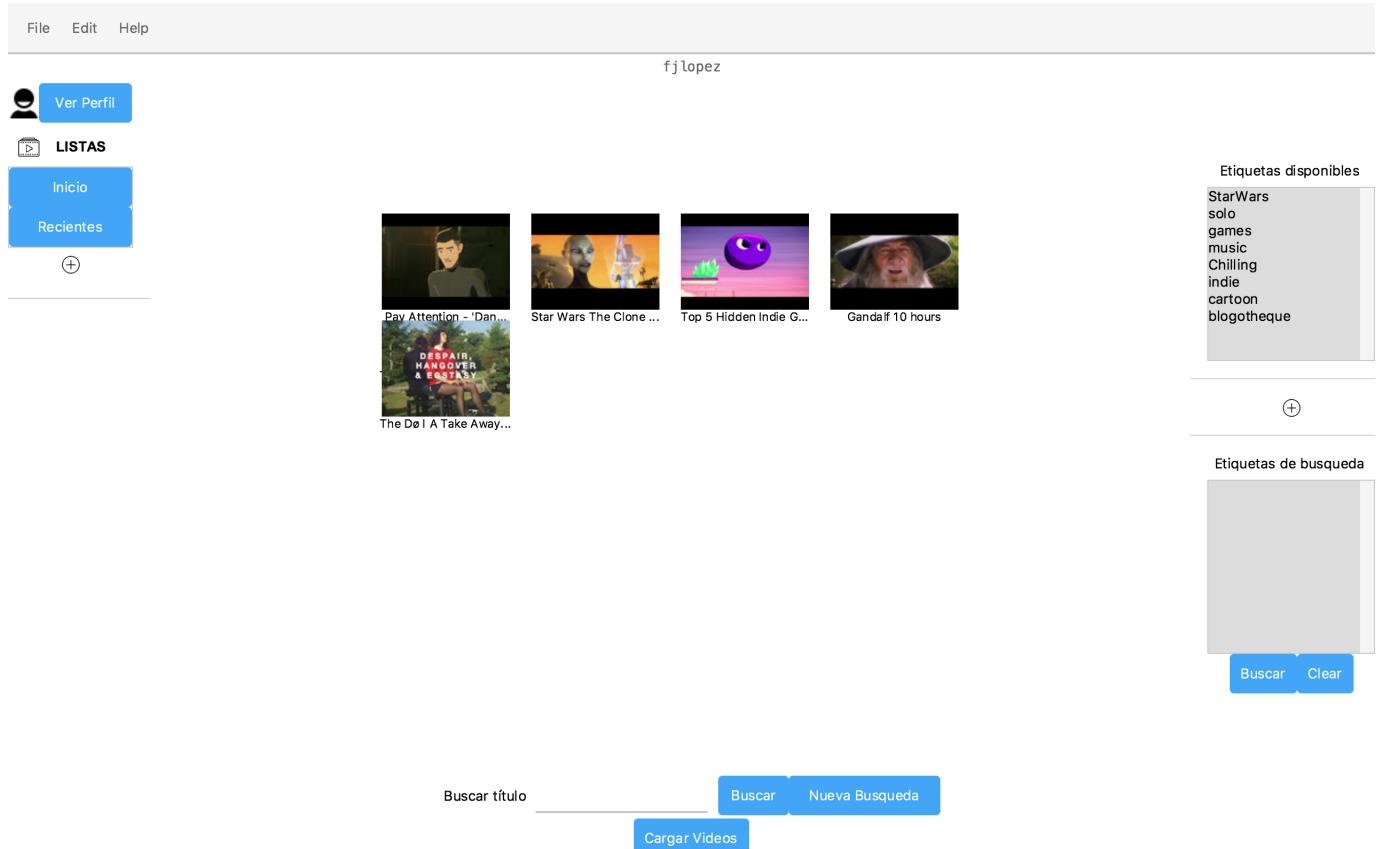


Figure 16: Ventana principal de usuario

En la parte central nos encontramos con los vídeos que hay en el sistema representados por sus miniaturas. En la parte izquierda, nos encontramos con el botón "**Ver Perfil**" y más abajo una serie de botones que corresponden con las listas del usuario.

El botón "Inicio" solo se utiliza para volver a las miniaturas de todos los vídeos del sistema. El botón "Recientes" corresponde a los últimos vídeos que se han visto en la sesión. más abajo, nos encontramos con el botón "+" para crear una nueva lista de vídeos.

En la parte derecha nos encontramos con las etiquetas disponibles del sistema, con la posibilidad de añadir etiquetas nuevas y más abajo una caja con las etiquetas de búsqueda. En la parte inferior nos encontramos con dos opciones:

- **Búsqueda de vídeos.** La búsqueda puede ser:

- Por título. Tan solo hace falta escribir el nombre en la campo de texto y pulsar al botón "Buscar"
- Nueva Búsqueda. Se utiliza para hacer una búsqueda conjunta de etiquetas y título.

- **Cargar vídeos.** Utiliza el componente Java Bean y su función es añadir nuevos vídeos a través de un fichero XML.

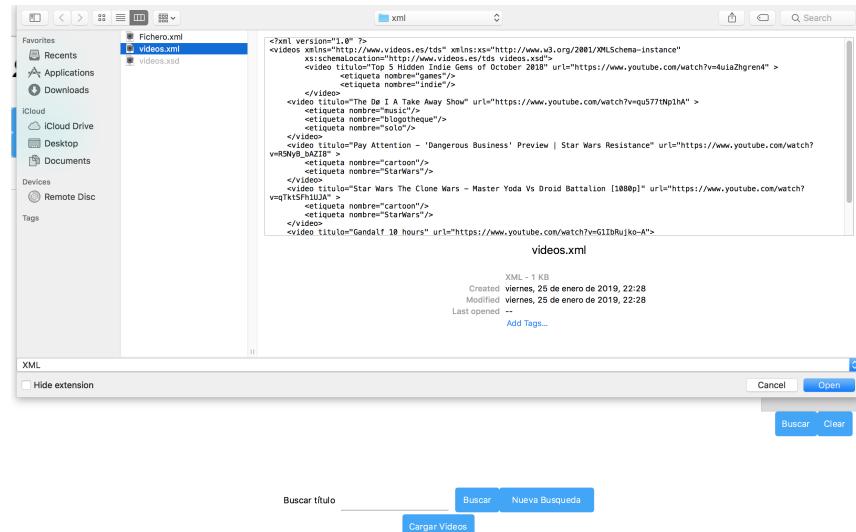


Figure 17: Caption

```

1  <videos xmlns="http://www.videos.es/tds" xmlns:xs="http://
   www.w3.org/2001/XMLSchema-instance" xs:schemaLocation="
   http://www.videos.es/tds videos.xsd">
2    <video titulo="Luis Fonsi - Sola" url="https://
      www.youtube.com/watch?v=sK1A0A3ufpc" >
3      <etiqueta nombre="Reggaeton"/>
4      <etiqueta nombre="Music"/>
5    </video>
6    <video titulo="Adverticement2" url="https://
      www.youtube.com/watch?v=utUPth77L_o" >
7      <etiqueta nombre="Apple"/>
8      <etiqueta nombre="Commercial"/>
9    </video>
10   </videos>

```

Volviendo a la página de inicio, si pulsamos a la miniatura, nos reproducirá el vídeo.

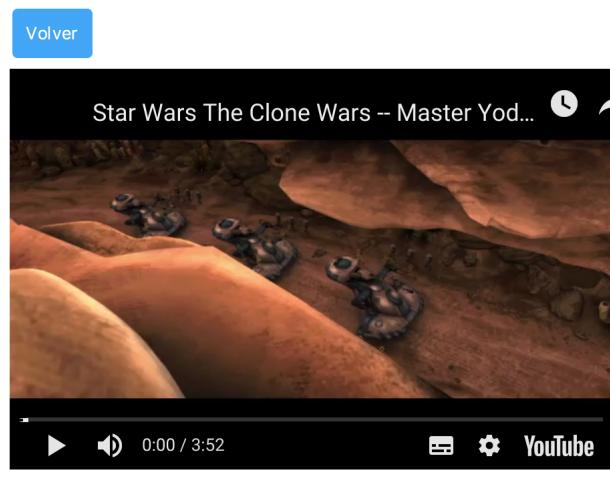
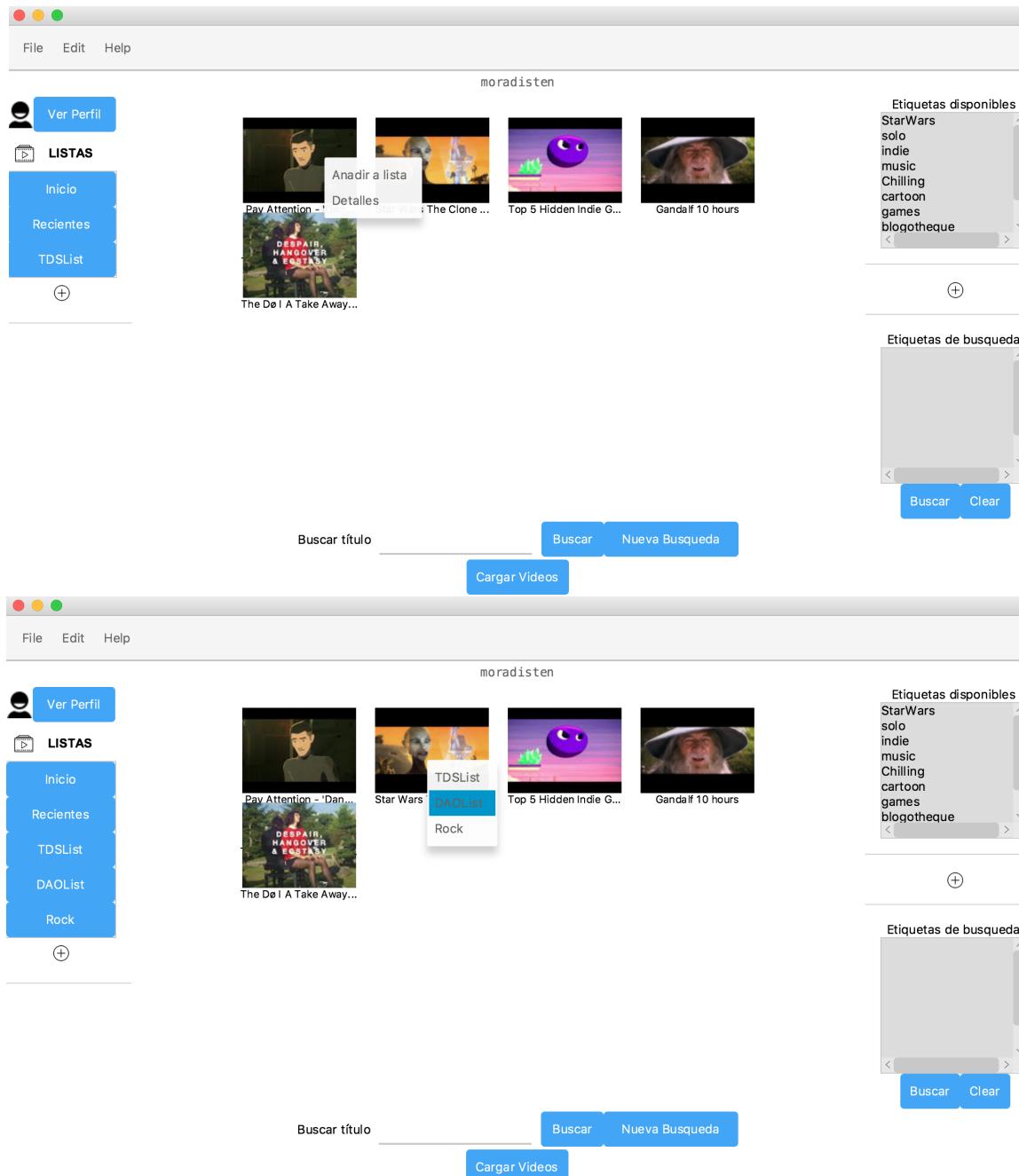


Figure 18: Reproducción del vídeo

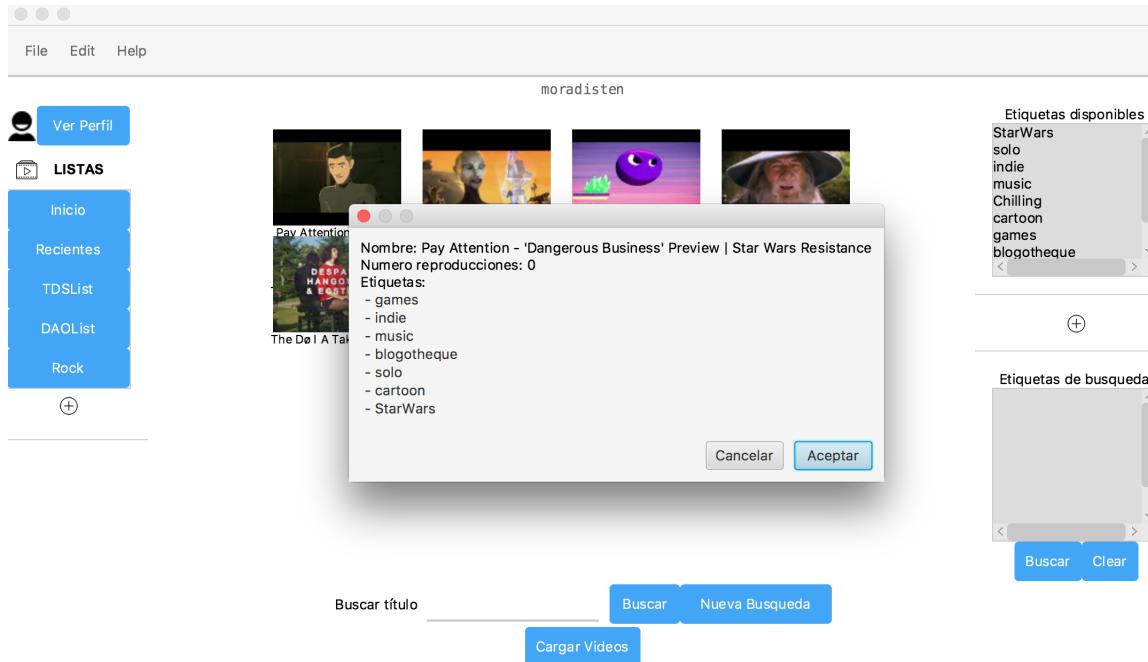
En la ventana de visualización nos aparece el número de reproducciones y un botón para añadirle nuevas etiquetas al vídeo. Para volver al inicio, tan solo hace falta pinchar en el botón "**volver**".

Si pinchamos con el botón de clic derecho sobre una miniatura, nos dará dos opciones:

- **Añadir a una lista.** Si pinchamos esa opción nos aparecerá un listado de listas de vídeos del usuario.

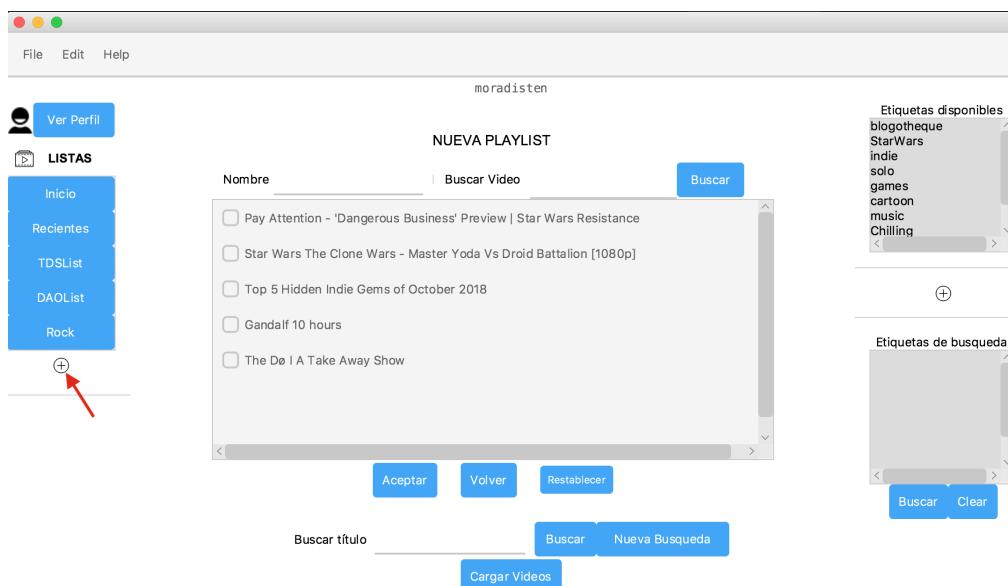


- **Detalles.** Muestra los detalles del vídeo.



## 6.1 Creación de una lista

Pinchamos en el botón "+" que se encuentra en la parte inferior derecha, debajo de los botones de las listas.



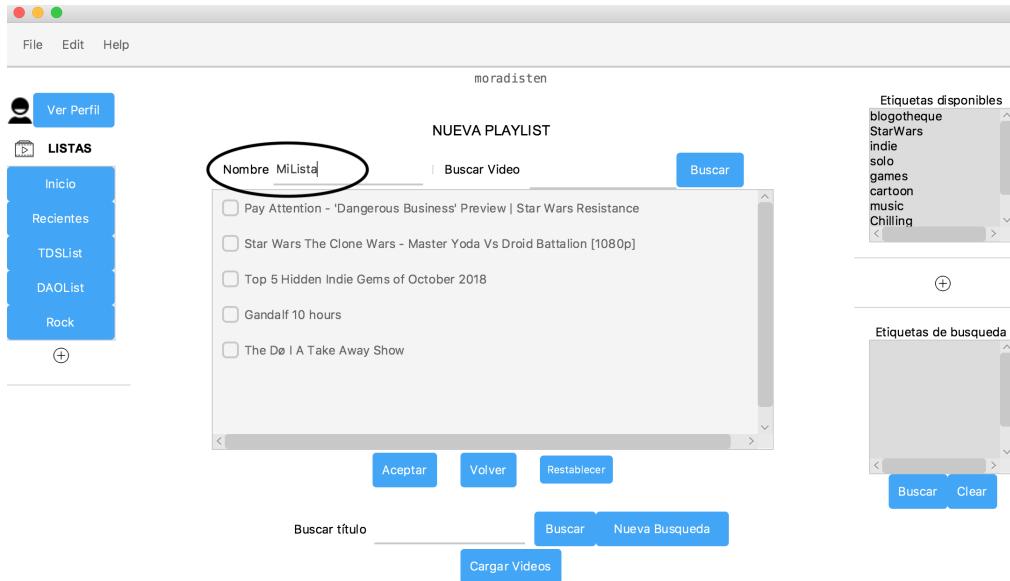


Figure 19: Escribimos el nombre

Podemos buscar un vídeo concreto con el buscador.

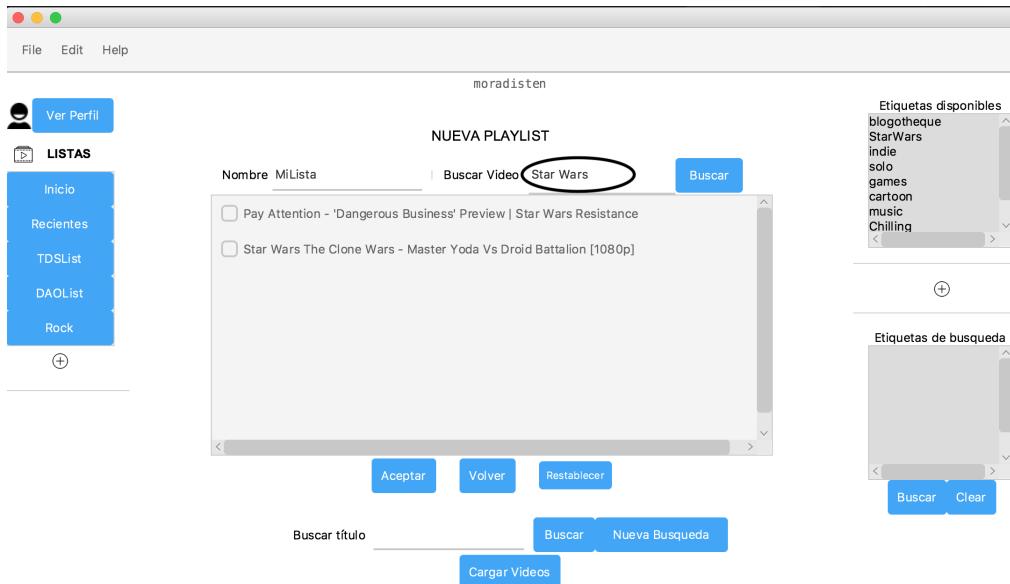


Figure 20: Búsqueda de vídeos para crear una lista

Cabe destacar que un usuario *premium* tiene una lista adicional de los videos más vistos.

## 6.2 Reproducción de una lista

Para reproducir una lista entera, se debe pulsar con el botón del click derecho a la lista. A continuación, aparecerá el ítem "Reproducir lista" y pinchamos en él.

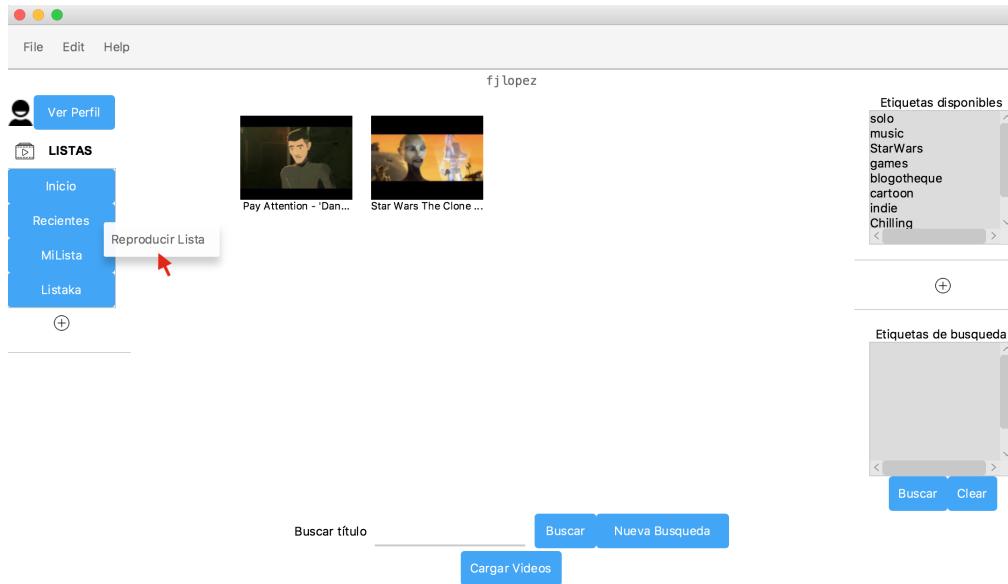
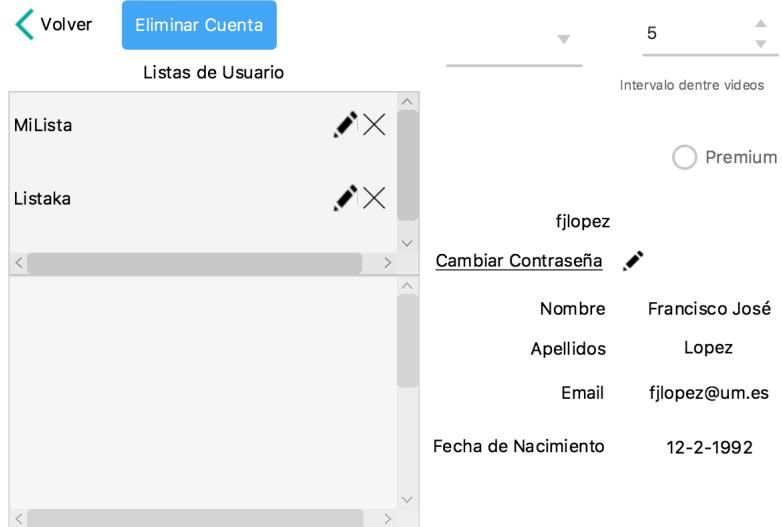


Figure 21: Caption

### 6.3 Ventana de Perfil

Dentro de la ventana principal de usuario, en la parte derecha, encima del apartado de las listas, pinchamos en el botón "**Ver Perfil**", y nos redireccionará a la ventana de perfil y el panel de control.



Dentro de la ventana del perfil, podemos destacar 4 partes:

#### 6.3.1 Panel de control para usuarios premium

En la parte superior derecha, un usuario puede hacerse usuario *premium*. Tan solo hace falta pulsar al *checkbox*.

Las ventajas que aporta una cuenta premium son:

- **Filtros.** Un usuario premium puede aplicar distintos filtros de búsqueda.

- **Generar PDF.** Permite generar un fichero PDF con las listas del usuario.

Otra opción que se puede configurar es el intervalo en segundos que tiene que pasar para que se reproduzca el siguiente vídeo en la reproducción de una lista.

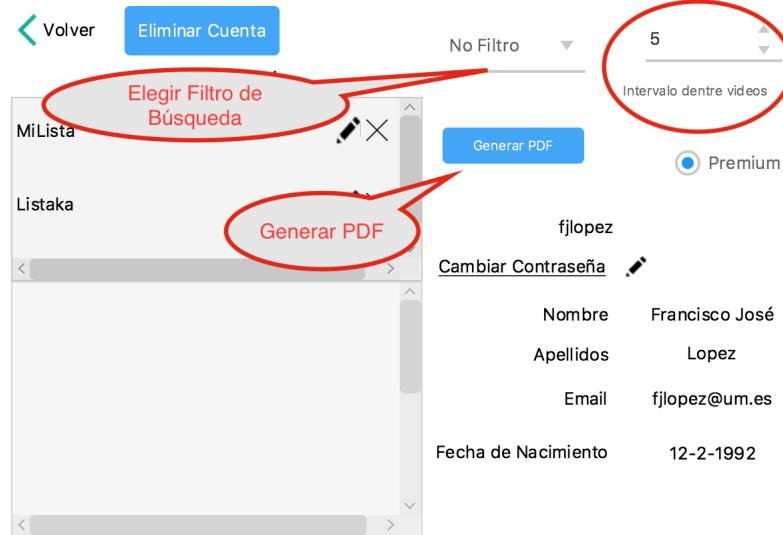
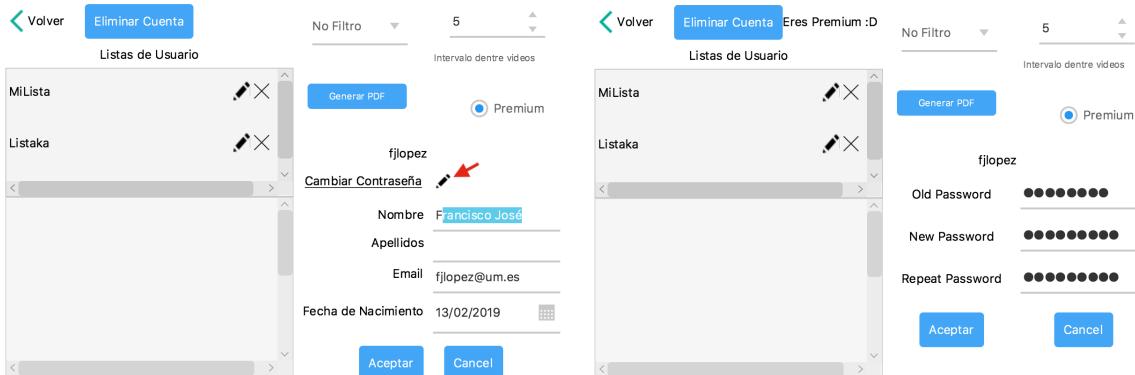


Figure 22: Usuario Premium contratado

### 6.3.2 Edición de información personal

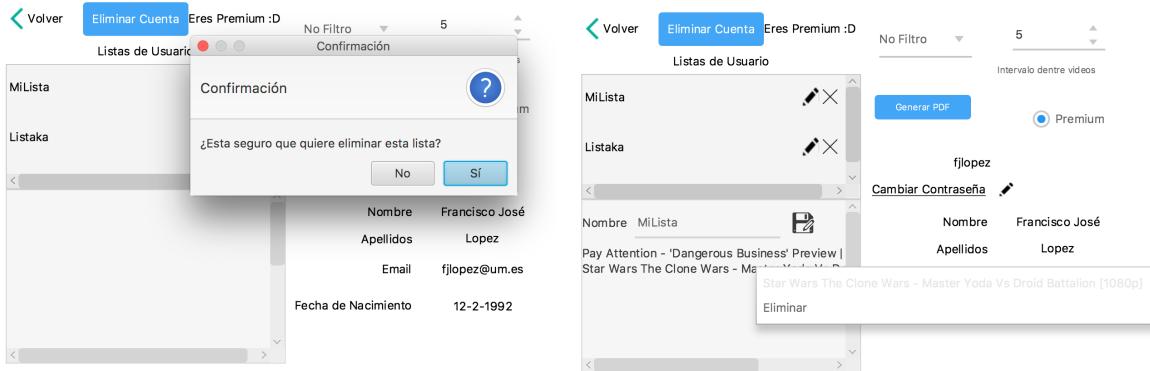
Debajo del panel de control, se nos muestra la información personal del usuario. Podemos modificar la información personal pinchando en el ícono del lápiz entre el panel control y el panel de información. Podremos modificar nuestra información personal y guardarla. También podemos modificar la contraseña con pinchando en "Cambiar contraseña".



### 6.3.3 Listas de usuario

En la parte izquierda, se muestran las listas creadas por el usuario. Cada lista se puede eliminar pulsando en el ícono de "X" o editarla pulsando en el ícono del lápiz. En el panel de edición de lista, en la parte de abajo, podemos modificar el nombre y eliminar un video de una lista pulsando el botón de click derecho sobre el nombre del vídeo.

Tras finalizar la edición, pinchamos en el ícono de guardar.



#### 6.3.4 Eliminar Cuenta

En la parte superior izquierda podremos eliminar nuestra cuenta pinchando en el botón "**Eliminar Cuenta**". Se nos pedirá confirmación poniendo nuestra contraseña y esta acción es irreversible.

## 7 Funcionalidad adicional

Dentro del proyecto, se ha añadido funcionalidad adicional como la inclusión de nuevas clases de apoyo, algoritmos para solucionar problemas o *plugins*.

### 7.1 Clases Adicionales

#### 7.1.1 Clase Notificacion

Esta clase se utiliza solo para mostrar *pop-ups* como forma de notificar algun cambio o error. Entre sus funciones más destacadas se encuentran:

- `voidFieldsError()`. Notifica un error cuando los campos del registro están vacíos
- `listNotFoundError`. Notifica un error de por no encontrar una lista determinada
- `passwdMatchError()`. Notifica un error cuando una contraseña y su repetición no coinciden.
- `confirmationQuestion()`. Pop-up de confirmación de una acción determinada. Por ejemplo, confirmación de contratación de cuenta *premium*.
- `showVideoDetails()`. Muestra los detalles de un video determinado.
- `deleteAcountQuestion()`. Muestra un Pop-up para confirmar la eliminación de una cuenta.

#### 7.1.2 Clase VideoWeb

Al hacer la parte de las vistas con JavaFX, nos encontramos con un problema. El profesorado proporcionó un componente para la visualización de vídeos de Youtube, pero solo funcionaba en Java Swing. Por lo nos hemos visto obligados a crear nuestro propio componente de visualización de vídeos de Youtube. Esta clase implementa un patrón *Singleton*.

```

1  private static final String cabeceraURLYoutube = "https://
2   www.youtube.com/watch?v=";
3  private static VideoWeb unicaInstancia;
4  private String reproducir;
5  private Pane panel;
6  private static boolean activo;
7  private static WebView webView;
8  private boolean iniciadoVideo;
9
10 public static VideoWeb getUnicaInstancia() {...}
11 private VideoWeb() {...}

```

Esta clase es utilizada en los controladores `ThumbGridController` y `VisorController` para sus vistas.

Sus funciones son similares al del componente proporcionado por el profesorado, las cuales son:

- **Reproducción de videos.** Se puede reproducir un video de Youtube con el método `playVideo()`.

```

12 public void playVideo(String url) {
13
14     if (!url.startsWith(cabeceraURLYoutube)) {
15         System.out.println("Error de url de youtube: debe
16             empezar por < https://www.youtube.com/watch?v= >" +
17             );
18         System.exit(1);
19     }
20     if (url.length() != 43) {
21         System.out.println("Error: La URL debe ser de 43
22             caracteres");
23         System.exit(1);
24     }
25     if (!activo) {
26         String lineaAReemplazar = "watch?v=";
27         String lineaParaReemplazar = "embed/";
28         String urlEmbedded = url.replace(lineaAReemplazar,
29             lineaParaReemplazar);
30         this.reproducir = "<iframe" +
31             " frameborder=\"0\"" +
32             " scrolling=\"no\" marginheight=\"0\" " +
33             " marginwidth=\"0\" " +
34             " width=\"456\" " +
35             " height=\"300\" " +
36             " type=\"text/html\" " +
37             " src=\"" + urlEmbedded + "?autoplay=1&fs=0&
38                 amp;iv_load_policy=3&amp;showinfo=0&rel
39                 =0&amp;cc_load_policy=0&amp;start=0&end
40                 =0&amp;origin=undefined\>" +
41                 "</iframe>";
42         webView.getEngine().loadContent(reproducir);
43         activo = true;
44     }
45 }
```

- **Cancelación de un video.**

```

39 public void cancel() {
40     this.activo = false;
41     webView.getEngine().load(null);
42 }
```

- **Obtener una miniatura de un vídeo.** Proporciona una miniatura del vídeo de Youtube con su url.

`ImageView img = VideoWeb.getThumb("youtube.com=/watch?v=I4oJK9dSkj1")`

- **Obtener una imagen de un vídeo.**

`ImageView img = VideoWeb.getImage("youtube.com=/watch?v=I4oJK9dSkj1")`

- **Licencia de uso.** Este componente de uso libre y se invoca al método `getVersion()`.  
*Version 1.2 – Author : Morad Abbou Azaz – (c) 2018 – 2019 TDS*

## 7.2 Gestión de imágenes de las miniaturas

A continuación, se muestran los problemas que se han tenido con el manejo de las miniaturas de los vídeos.

### 7.2.1 Mostrar las miniaturas ordenadas de los vídeos

Uno de los mayores problemas que se nos presentó fue la visualización de las miniaturas de los vídeos.

El primer problema fue la colocación de las miniaturas dentro del GridPane (GridBagLayout en JavaSwing), sobretodo cuando se añadían nuevos vídeos o se hacía una búsqueda. El problema principal era el solapamiento de miniaturas.

Para solventar este problema, se ideó un algoritmo sencillo de ordenación:

```

43 mientras (contador_imagenes < numero_videos_total)
44     hacer
45         pos_columna := contador_imagenes mod NUM_MAX_COLUMNAS;
46         si (ImagenActual no nula)
47             hacer
48                 insertar(pos_columna, pos_fila, ImagenActual)
49             fin_si
50         contador_imagenes := contador_imagenes + 1;
51         si (contador_imagenes mod NUM_MAX_COLUMNAS == 0)
52             hacer
53                 pos_fila := pos_fila + 1;
54             fin_si
55 fin_mientras

```

### 7.2.2 Tiempo de carga de las miniaturas

El principal problema que nos encontramos al mostrar una miniatura es que el proceso de carga era muy lento. Al iniciar el programa, el tiempo de carga era bastante notorio y se incrementaba conforme se añadían nuevos.

Uno de los métodos del componente `VideoWeb` era `getThumb(url)` que retornaba una imagen que representaba la miniatura de un video de Youtbe dada su URL. Este método resultó ser muy ineficiente.

El tiempo de que tardaba el programa en cargar las miniaturas desde que se hace un *login* era de 5.3421 segundos de media.

Esto resultaba ser bastante incómodo para la experiencia de usuario. Una solución que se le dio fue la implementación de un algoritmo.

Este algoritmo se encarga de descargar una miniatura y almacenarla en la carpeta `/thumbs` con el nombre del ID del vídeo en Youtube.

Por ejemplo, si tenemos la URL `https://www.youtube.com/watch?v=4uiaZhgren4`, el programa descargaría la miniatura utilizando esa URL y la almacenaría bajo el nombre de `4uiaZhgren4.jpg`.

Para evitar una descarga continua de la misma miniatura, hemos creado un hashMap en `ThumbGridController` llamado `rutas` en donde almacena la ruta de fichero de la imagen,

tomando como clave la URL del vídeo al que pertenece.

```
rutas = new HashMap<String, String>();
clave: URL del video de Youtube
valor: Ruta de la imagen descargada en el proyecto.
```

El esqueleto del algoritmo en cuestión es:

```

1 rutas = new HashMap<String, String>();
2 ...
3
4 Imagen get_imagen(url : String)
5 begin
6     String IDVideo := getIDVideo(url);
7     String ruta_imagen_fichero := "/thumbs" + IDVideo + ".jpg"
8     if (rutas.vacia == true)
9     then
10         rutas.put(url, ruta_imagen_fichero);
11         Imagen imagen := crear_imagen_ruta(url);
12         Fichero F := imagen.to_fichero;
13         almacenar_fichero_en(F, ruta_imagen_fichero);
14         return Imagen;
15     else
16         Fichero F = abrir_fichero(rutas.get(url));
17         Imagen imagen := F.get_imagen();
18         return imagen;
19     end
20 end

```

Con este algoritmo se reduce significativamente el tiempo de carga, ya que la imagen se almacena en local y no hay que descargarla cada vez que se arranca el programa.

### 7.2.3 Estructuras de almacenamiento de miniaturas

Las estructuras de datos que se han implementado para mejorar el rendimiento y el consumo de recursos para el manejo de miniaturas es:

- **mapaImagenes**. Es un `HashMap` cuya clave es un `String` que contiene la URL y su valor es un objeto de tipo `ImageView` (Componente gráfico de representación de una imagen en Java Swing). Con esto nos aseguramos no tener que crear el mismo objetos `ImageView` varias veces para una misma URL, sino que se toma del `HashMap`.
- **conjuntoURL**. Es un conjunto que almacena las URLs que se van mostrar en ese momento. Por ejemplo, si estamos dentro de una `playList`, el Set contendrá las URLs de los videos pertenecientes a esa lista.

En definitiva, `conjuntoURL` se utiliza para almacenar las URLs que se van a mostrar. Para acceder a las miniaturas de esas URLs utilizamos el mapa `mapaImagenes` que nos devuelve un objeto `ImageView`. Si una URL no tiene una imagen asignada, se descargará esa imagen y se insertará en la carpeta `/thumbs/` con el ID del video como nombre para mejorar su identificación. Después se inserta en `mapaImagenes`.

## 7.3 Utilización de Maven

En este proyecto se ha utilizado un módulo de maven, para añadir dependencias de determinados repositorios, tales como los *plug-ins* de generación de PDF.

# 8 Problemas

## 8.0.1 Problemás con la visualización de videos

Un problema que presenta JavaFX es su inmensa inestabilidad con la visualización de vídeos embedidos de Youtube. Al intentar visualizar un vídeo, este se reproduce correctamente pero de repente toda la máquina virtual de Java (JVM) se cierra repentinamente notificando un error. Tras buscar información, se concluyó que JavaFX presenta graves problemás de estabilidad con ciertas versiones del JDK en general, además del entorno en donde se desarrolla. Este problema no es a causa de la mala programación del equipo, sino que aparece documentado como un *bug* que presentann los FrameWorks de JavaFX.

Enlaces de interés:

- StackOverFlow: Playing a video in JavaFX
- OpenJDK: Bug playing embedded Youtube video

## 8.0.2 Visualización de miniaturas de Youtube

Como se ha comentado anteriormente, la gestión de las miniaturas de los vídeos de Youtube en cuadrículas y su visualización al pinchar en una lista ha sido uno de los mayores obstáculos del proyecto. Los problemás que se nos presentaban eran:

- La miniatura se tenía que descargar directamente de Internet y consumía mucho tiempo
- Muchas miniaturas se solapaban debido al mal cálculo de filas y columnas
- Habían problemás al refrescar las vistas y quedaban miniaturas que no debían aparecer.
- Al eliminar un vídeo de una lista, seguía apareciendo su miniatura.
- Aparecían las mismas miniaturas para distintas URLs.

# 9 Conclusiones y resultado

Al ser repetidores, hemos tenido la idea de un proyecto ambicioso y queríamos avanzar con nuevas funcionalidades y dejar una aplicación mejorada. Este proyecto nos ha llevado muchísimo más tiempo del que se suele necesitar para la asignatura. Gracias a los conocimientos adquiridos en las asignaturas de Procesos del Desarrollo Software y Gestión de Proyectos, pudimos hacer una estimación del coste en tiempo y hacer un diseño que sencillo para mejorar la escalabilidad del proyecto. Agradecemos al profesor Bermúdez de darnos la libertad para desarrollar el proyecto a nuestro gusto siempre que se respeten los requisitos. El problema que se nos presentaba era que cuanto más se añadían nuevas funcionalidades, más errores iban apareciendo y más gestiones se tenían que hacer. Y es que

en la ingeniería de Software, el 80% del tiempo se destina a arreglar y encontrar *bugs*. Nos quedamos satisfechos del trabajo realizado y esperamos dejar nuestro código disponible en GitHub después de acabar el curso. Este proyecto también nos ha enseñado la importancia de las disciplinas de la ingeniería del Software pues a lo largo del proyecto hemos tenido que reestructurar ciertas partes de la arquitectura de la aplicación ya que, o bien eran inviables, o bien eran muy inefficientes. En conclusión, hemos disfrutado con la práctica de esta asignatura y esperemos que la experiencia nos ayude en el futuro. Cabe destacar que nadie nos ayudó con JavaFX por lo que tuvimos que investigar por nuestra cuenta.

A continuación, mostramos los resultados de las horas que hemos invertido:

Tareas	Horas(Aprox.)
Implementación de la persistencia	25
Implementación del ventanaje básico	30
Implementación del modelo	18
Desarrollo de la ventana UserWindow	22
Desarrollo de la ventana ProfileWindow	17
Desarrollo de la ventana ThumbGrid	26
Implementación del Controlador Principal	11
Solucion de errores Controlador-Vista	24
Solucionar errores Controlador-modelo	13
Errores varios	34
<b>TOTAL</b>	<b>200</b>

## 10 Referencias

- <https://openjfx.io>
- Oracle JavaFX
- DAO Patterns
- Maven Module