

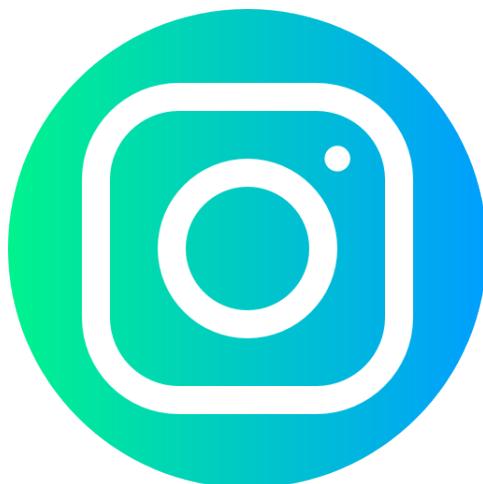
Tecnologías de Desarrollo de Software

PHOTO TDS

Francisco Arnaldo Boix Martínez - 52041159W

Juan Carlos Segura García - 21069703R

Enero 2023



Índice

1. Introducción	3
2. Diagrama de Clases.	3
3. Diagrama de secuencia para crear una foto.	3
4. Arquitectura de la aplicación.	4
5. Patrones de Diseño	4
5.1. Estrategia + Composite	5
5.2. DAO: Adapter + Abstract Factory	5
5.3. Singleton	6
5.4. Observer	6
5.5. Bridge	6
5.6. Composite	6
5.7. Decorator	6
6. Cambios sobre la propuesta	7
7. Decisiones de diseño	7
8. Componentes JavaBean.	8
9. Test unitarios	9
10. Manual de usuario	10
11. Repositorio GitHub.	17
12. Observaciones finales	18

1. Introducción

A lo largo de este documento pasamos a comentar los aspectos más relevantes de nuestra implementación de la aplicación PhotoTDS. Lo más importante es que ambos miembros del equipo de prácticas hemos intentado seguir los **principios SOLID**, así como favorecer la composición frente a la herencia.

2. Diagrama de Clases.

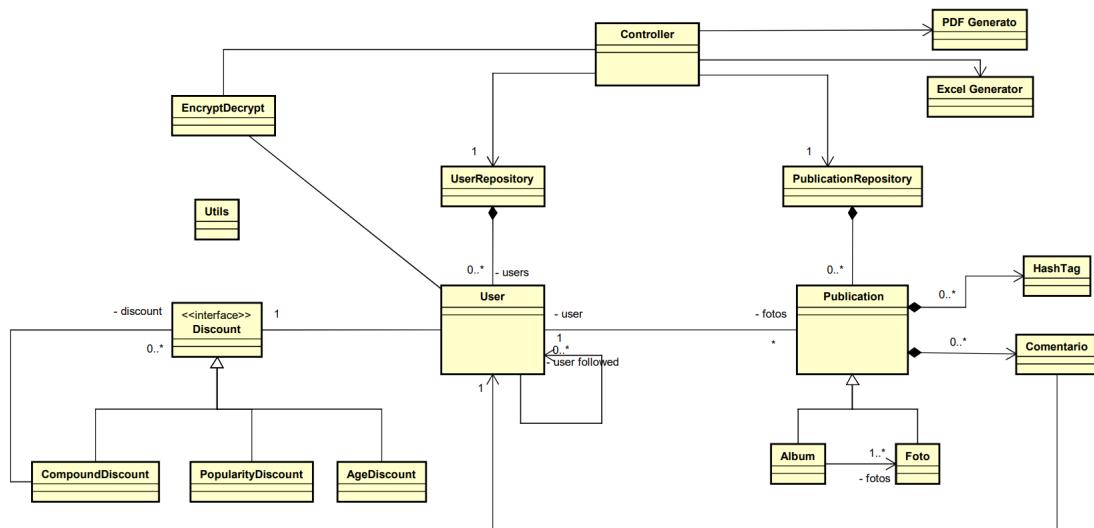


Figura 1: Diagrama de clases

La clase **Utils** no aparece conectada con nada porque ensucia el dibujo. Realmente se usa de manera global 7.

3. Diagrama de secuencia para crear una foto.

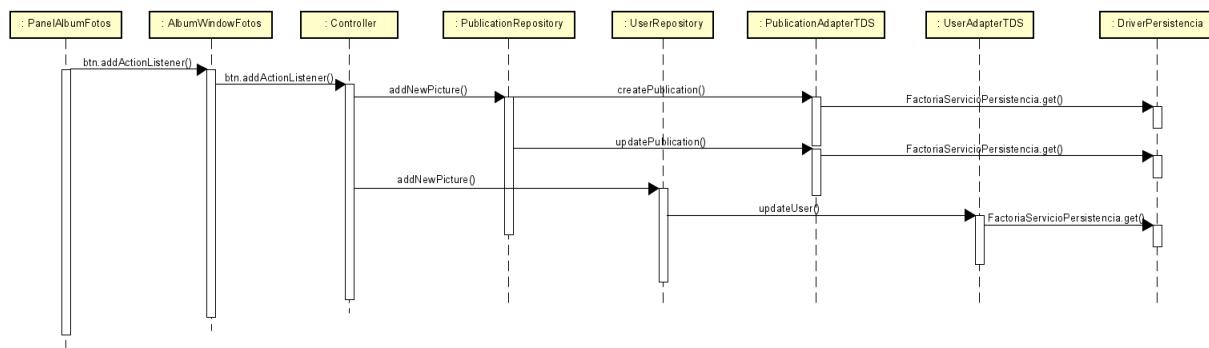


Figura 2: Diagrama de secuencia

4. Arquitectura de la aplicación.

Para desarrollar la aplicación hemos seguido una arquitectura de 3 capas. Concretamente hemos hecho uso de la tan famosa MVC. Cuando se nos presentaron los requisitos de la aplicación barajamos la opción de seguir una arquitectura distinta, como por ejemplo arquitectura hexagonal (1) y hacer algo diferente, sin embargo optamos por seguir la arquitectura propuesta dentro de la asignatura para seguir con las recomendaciones de los profesores.

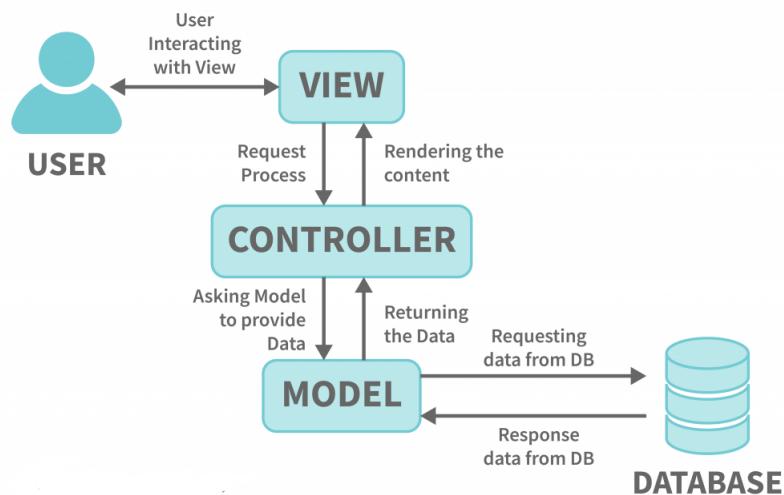


Figura 3: Arquitectura de la aplicación

Este patrón arquitectónico se sustenta en la separación en 3 capas:

- **Interacción con el usuario (Vista):** Esta es la capa que el usuario es capaz de ver para interactuar con nuestra aplicación. Dentro de la misma se incluyen las ventanas, paneles y demás elementos visuales.
- **Controlador:** Ofrece una fachada para que la capa superior (vista) pueda interactuar con elementos del dominio, pero sin conocer toda su lógica interna. Para que el controlador pueda ofrecer una fachada totalmente funcional, ha de conocer también la persistencia.
- **Modelo:** Dentro de esta capa es donde se esconden todas las entrañas del modelo de la aplicación, en nuestro caso **PhotoTDS**.

Como podemos intuir, esta forma de organizar nuestra aplicación nos permite cumplir con el principio **separación modelo-vista**.

5. Patrones de Diseño

Para crear un software con cierta calidad debemos hacer uso de algún patrón de diseño. En concreto dentro de nuestra propuesta para la aplicación **PhotoTDS** hemos hecho uso de los siguientes patrones.

5.1. Estrategia + Composite

Usamos estos patrones para implementar la funcionalidad de los descuentos. Atendiendo al dominio de nuestra aplicación, tiene sentido puesto que una misma persona puede ser beneficiaria de varios tipos de descuentos.

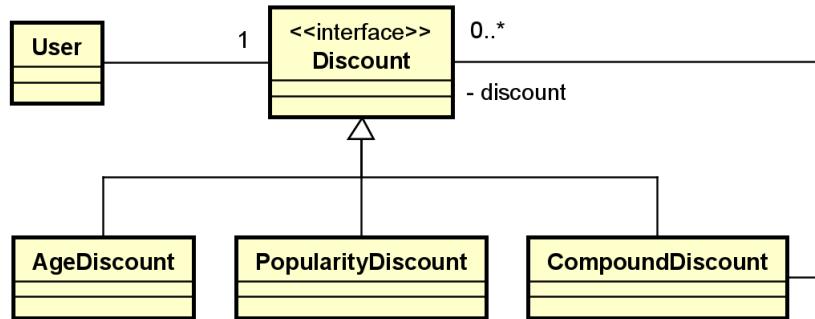


Figura 4: Estrategia + Composite

5.2. DAO: Adapter + Abstract Factory

Para hacer que **PhotoTDS** sea independiente de una tecnología de base de datos concreta, hemos implementado el patrón DAO. Este patrón, en realidad está compuesto por dos patrones: Adaptador y Factoría Abstracta.

El patrón adaptador nos permite abstraernos de una tecnología de base de datos concreta, y la factoría nos permite crear la familia de clases asociadas a una tecnología de persistencia concreta. Dentro de nuestra implementación el diagrama de clases sería el siguiente:

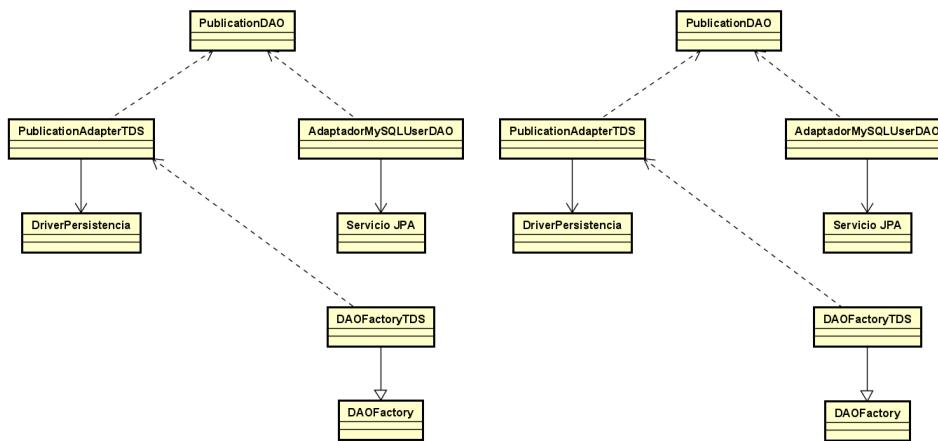


Figura 5: Adapter + Abstract Factory

5.3. Singleton

El uso del patrón es muy común, y en nuestra aplicación no podía ser menos. Usamos este patrón cuando necesitamos una instancia **global** de una clase. Para evitar posibles errores de programación lo forzamos aplicando **Singleton**. Ejemplos donde podemos encontrar el patrón Singleton dentro de nuestra implementación de PhotoTDS son: Repositorios, Adaptadores, Factorías... etc.

5.4. Observer

Este patrón lo usamos directa e indirectamente. De manera indirecta, **Swing** utiliza el **Modelo de Delegación De Eventos** para implementar los **Action Listener** de los botones (entre otros).

En nuestro caso, de manera directa lo hemos tenido que aplicar con el uso del **Cargador de Fotos**. Comentaremos este aspecto con más detalle en la Sección 8.

5.5. Bridge

Aunque este patrón no lo usamos directamente dentro de nuestro proyecto, **Swing** lo usa por debajo. Por ejemplo, la clase **JButton** contiene atributo *implementación*, identificativo del uso de un Bridge.

5.6. Composite

Este patrón nuevamente se usa de manera indirecta por **Swing**. En concreto tiene que ver con la clase **Component** de Swing. Todos los componentes que ponemos sobre las ventanas, pueden verse compuestos a su vez por más componentes.

5.7. Decorator

Por último nos encontramos con el Decorador nuevamente usado de manera indirecta por Swing. En concreto podemos encontrarlo en los paneles con **Scroll**. Una forma de crear un panel (**JPanel**) con scroll, es decorarlo con un **JScrollPane**.

```
1  public static void main(final String[] args) {
2      JPanel panel = new JPanel();
3      panel.setBorder(BorderFactory.createLineBorder(Color.red));
4      panel.setPreferredSize(new Dimension(800, 600));
5
6      final JScrollPane scroll = new JScrollPane(panel); //Decorar
7      scroll.setVisible(true);
8  }
```

6. Cambios sobre la propuesta

- **Titulos en publicaciones:** No es un cambio trascendental, pero decidimos ponerle un **título** a las **publicaciones**. De hecho, hemos tomado la decisión de que dos publicaciones **no** puedan tener el **mismo** título, al igual que no puede hacer dos usuarios con el mismo nombre de usuario.
- **Implementación de las notificaciones:** En vez de guardar una lista con las notificaciones de cada usuario, o implementar el patrón **observador**, hicimos caso a las indicaciones mencionadas en las clases prácticas de la asignatura, donde en vez de tener un sistema de notificaciones, mostramos las publicaciones personalizadas para cada usuario atendiendo a la última hora de su login.
Básicamente mostramos las fotos de los usuarios que **sigue**, que se hayan subido **después** de su **último login**. El resto no se muestran porque consideramos que el usuario ya las ha visualizado.

7. Decisiones de diseño

- **Diferenciar publicaciones a nivel de persistencia en vez de repositorio:** Como podemos comprobar en el Diagrama de Clases de la Sección 2, tanto las **Fotos**, como los **Álbumes** son tipos de **Publicación**. Para poder la persistencia de los mismos y respetar la jerarquía de clases, contamos con dos opciones:
 - Hacer repositorios separados.
 - Tener un repositorio de publicaciones, y diferenciar como se guardan en persistencia en los Adaptadores del DAO. Esta versión puede ser algo mas costosa, pero creemos que es como mejor se respeta el diagrama de clases.
- **Clase Utils:** En esta clase definimos funciones estáticas que usamos en varios puntos de nuestro modelo de la aplicación para que podamos acceder a ellas de manera sencilla y sin repetir código. Ejemplos de estas funciones pueden ser funciones **auxiliares** para tratamiento de cadenas, etc.
- **Clase EncryptDecrypt:** Siguiendo la misma idea que con la clase **Utils**, definimos una clase encargada de la **encriptación** de las contraseñas para introducirlas a la base de datos. Aunque no se pida como requisito en la práctica, creemos que almacenar contraseñas en claro es una pésima idea, y nos vimos obligados a implementar esta funcionalidad.
Cuando un usuario se da de alta, se encripta su contraseña usando una variante del algoritmo **AES 256**, teniendo como **clave compartida** la cadena: **my-super-secret-key**, la cual es muy segura.
- **Login Obligatorio:** Para poder usar cualquier funcionalidad del controlador previamente ha tenido que ser logueado el usuario.

Queremos hacer notar, que aunque la clase **Controller** tenga muchas líneas de código, no se trata de una clase **Dios**. En otras palabras, está programada de forma que no se

viole ninguno de los patrones **GRASP** y se puede comprobar debido a que no únicamente ofrece lógica correspondiente a su abstracción y el resto es delegado a las demás clases. El hecho de que posea un **javadoc**, hace que aumente considerablemente en tamaño.

Aunque no sea una decisión de diseño propiamente dicha, queremos recalcar el uso de las expresiones **lambda** y **streams** dentro del proyecto. Si indagamos un poco por el código, nos daremos cuenta de que hemos intentado explotarlas al máximo para conocer su **potencial**. Y hemos de decir, que hemos quedado gratamente sorprendidos. Veamos un ejemplo en el que se implementa una de las funcionalidades para un usuario premium en el controlador: obtener las 10 fotos con mas likes del usuario.

```
1  public List<Publication> getMoreLikedFotos(String user) {  
2      ...  
3      return u.getPublications().stream()  
4          .sorted(Comparator.comparing(Publication::getLikes).  
5             reversed())  
6          .limit(User.NUM_LIKES_PREMIUM)  
7          .collect(Collectors.toList());  
8  }
```

8. Componentes JavaBean.

- **Cargador de Fotos:** Es un componente **Java Bean** que permite seleccionar un archivo **XML** donde se indican varias fotos que se desean subir a la aplicación. Hace uso del **Modelo de Delegación de Eventos** de Java para programar una propiedad ligada. En concreto, nosotros usamos la clase **PropertyChangeSupport** dado que facilita la implementación de componentes Java Bean.

Para que el cargador interaccione con nuestra aplicación, es necesario que el controlador se **suscriba** a los eventos que emite este cargador (cuando se selecciona un archivo XML y lo ha procesado). Una vez procesado, el cargador guarda todos los atributos en forma de objetos consultables y **mapeables** a nuestro dominio. Veamos un ejemplo:

```
1  // Controlador  
2  public void propertyChange(PropertyChangeEvent evt) {  
3  
4      umu.tds.fotos.fotos = umu.tds.fotos.  
5          MapperFotosXMLtoJava.cargarFotos(evt.getNewValue().  
6              toString());  
7  
8      ...  
9      for (umu.tds.fotos.Foto f : fotos.getFoto()) {  
10          Foto foto = u.createPhoto(...);  
11  
12          this.pubRepo.createPublication(foto);  
13          this.userRepo.updateUser(u);  
14      }  
15  }
```

- **Luz:** Componente Java Beans que proporcionado en los recursos de la asignatura que nos dibuja un **botón** circular en la ventana. Se puede usar como cualquier componente de **swing**, y se integra con WindowsBuilder. Para poder usar este componente, tenemos que implementar la interfaz **IEncendidoListener**, por tanto implementamos el método heredado:

```

1  public void enteradoCambioEncendido(EventObject arg0) {
2      JFileChooser chooser = new JFileChooser();
3      chooser.showSaveDialog(null);
4
5      if (chooser.getSelectedFile() != null) {
6          String fichero = chooser.getSelectedFile().
7              getAbsolutePath();
8          Controller.getInstancia().uploadPhotos(user, fichero);
9      }
}

```

Dentro del método del controlador, disparamos el evento, estableciendo el archivo de las Fotos.

```

1  public void uploadPhotos(String u, String f) {
2      ...
3      umu.tds.fotos.CargadorFotos.getUnicaInstancia().
4          setArchivoFotos(f);
}

```

- **JCalendar:** Componente que usamos dentro de la **Ventana Login** en donde tenemos la opción de escoger la fecha de nacimiento haciendo uso de un calendario visual interactivo. Para usarlo simplemente basta con añadir una entrada en el archivo **pom.xml**, buscando la referencia al mismo en maven repository.

9. Test unitarios

Hemos usado el Framework **JUnit** para realizar nuestras pruebas unitarias. En concreto, hemos probado tres métodos del controlador. Para poder discutir con algo mas de profundidad los test unitarios implementados, debemos situarnos dentro de la carpeta **src/main/java**, en el paquete **umu.tds.app.PhotoTDS**, concretamente en la clase **AppTest** donde podemos encontrar los siguientes métodos.

- **testFollower():** Dentro de este método comprobamos que cuando un usuario sigue a otro, efectivamente se añade a su lista de seguidores.
- **testChangeDescription():** En esta ocasión probamos que cuando un usuario quiere cambiar su descripción, esta realmente es actualizada con el valor correcto.
- **testCheckFinder():** Probamos la funcionalidad de búsqueda de usuarios. La variable **userToFind** pretende modelar el input del usuario referente a quien quiere buscar dentro de PhotoTDS. Por otro lado, **userLogged** pretende modelar el usuario actualmente logueado en la aplicación.

10. Manual de usuario

• **Ventana Login:** La ventana de login es la primera ventana que nos encontraremos cuando accedamos a la aplicación. Esta ventana nos servirá para acceder a la aplicación si estamos registrados. Nos encontramos con un campo para escribir el nombre de usuario, otro para escribir la contraseña y dos botones, el botón de *login* que una vez se completa los campos de *Username* y *Password* correctamente podremos acceder a la aplicación. El botón de *register* nos permitirá ir a otra ventana para poder registrar a un nuevo usuario.

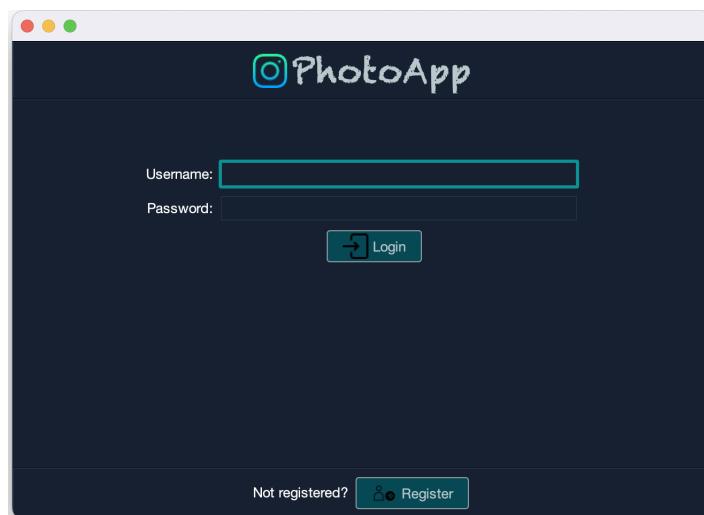


Figura 6: Ventana Login

• **Ventana Register:** La ventana de registro se accede a través de la ventana de login y sirve para realizar el registro de un nuevo usuario a la aplicación. Podemos ver sus campos en la siguiente Figura.

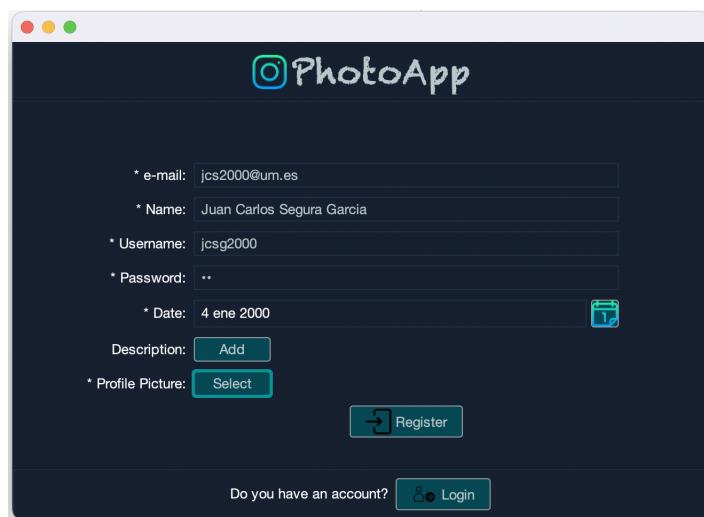


Figura 7: Ventana Registro

Todos los campos que precedidos con un * se consideran campos **obligatorios** y deben de llenarse para que se pueda registrar un usuario.

Cuando un campo obligatorio se ha llenado de una manera **incorrecta**, saltará una **ventana** notificando del **error** con un comentario enfatizándolo y como solucionarlo.

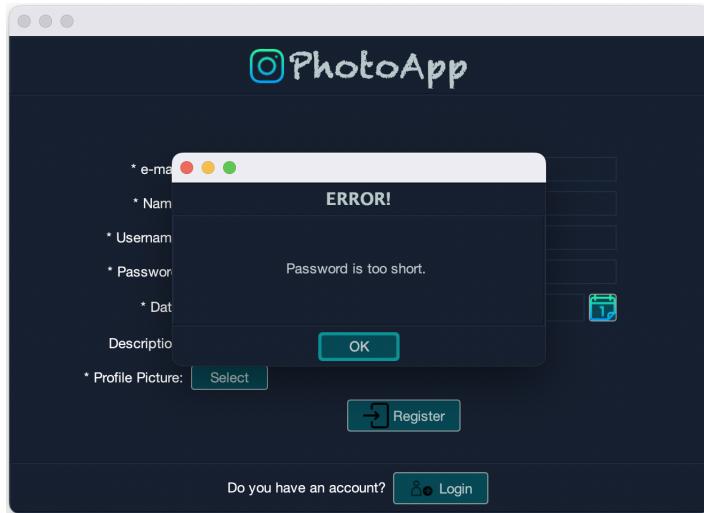


Figura 8: Error en contraseña

Si todos los campos obligatorios han sido rellenados correctamente el registro de usuario se efectuará con éxito y se creará un nuevo usuario en la aplicación con lo que ahora podríamos irnos a la ventana de inicio y hacer un **login** con nuestro nombre de usuario y contraseña.

- **Ventana Inicio:** La ventana de inicio corresponde con la pagina principal de la aplicación, la cual se accede haciendo login con un usuario. Es esta ventana aparecerán todas las publicaciones de todos los usuarios que sigue en el transcurso de la **última** vez que el que se desconectó hasta la **conexión actual**, es decir, todas las publicaciones que se subieron cuando el usuario que realiza el *login* se encontraba desconectado.

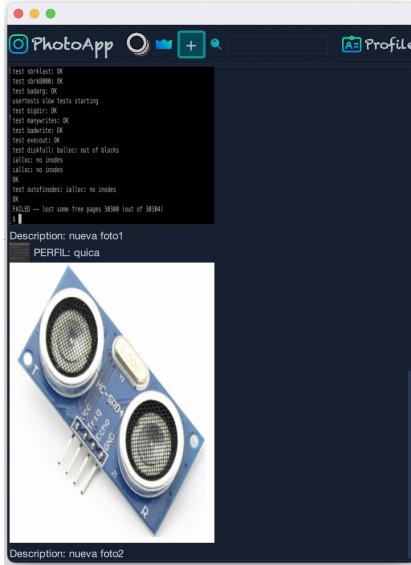


Figura 9: Ventana Inicio

En esta ventana y en las posteriores nos encontraremos con un panel **fijo** en la parte superior que contiene en orden:

- **PhotoApp:** Logotipo de la aplicación que al pulsar sobre él nos dirigirá a esta ventana de inicio estemos donde estemos.

- **Cargador de fotos:** Cargador de fotos que se encarga de cargar un archivo xml para cargar toda una aplicación de acciones para la aplicación. Subir publicaciones, álbumes, usuarios, etc.
- **Sección premium:** Sección donde nos encontraremos toda la funcionalidad para convertir en premium en el caso de no ser premium o con la funcionalidad especial para usuarios premium.
- **Botón para subir fotos:** Botón utilizado para subir fotos a la aplicación.
- **Buscador:** Buscador de usuarios y de publicaciones a través de un HashTag.
- **Perfil:** Botón que nos dirige al perfil del usuario.

•**Subir foto:** Para subir una foto bastará con pulsar el botón del panel superior con un **+**. Cuando es pulsado aparecerá una nueva **ventana** donde nos pide que insertemos un título y una descripción para la foto a subir. En descripción podremos darle una descripción a la foto y además poner HashTag **incrustados** en la **descripción**. Los HashTag serán útiles para una vez subida la foto se pueda buscar por dicho HashTag. Es importante no poner más de cuatro HashTag a una publicación. De lo contrario, los hashtags sobrantes **no se añadirán** a la misma. La longitud **máxima** de un HashTag es de **16** caracteres, por lo que si ponemos un hashtag con una longitud mayor, **tampoco** se añadirá a la lista de HashTag

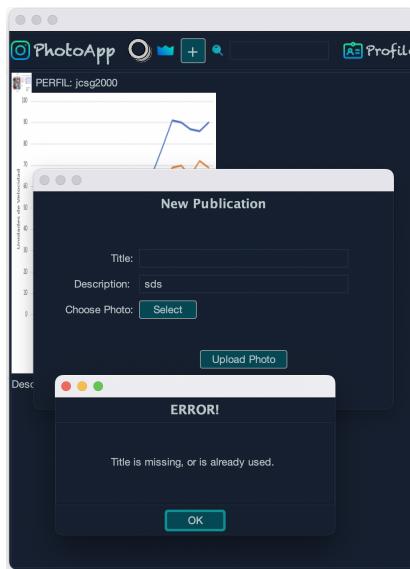


Figura 10: Ventana para subir fotos con error en el título

El título deberá de rellenarse **obligatoriamente** ya que si no es así saltará un error indicando que es necesario darle un título a la foto. Una vez subida la foto, aparecerá en la ventana de inicio de todos los usuarios que son **seguidores** nuestros dicha foto si esta fue subida cuando ellos estaban desconectados. A su vez la foto también aparecerá en nuestro **perfil**.

•**Foto:** Cuando pulsamos una imagen tanto si es **buscada** mediante HashTag, como desde la ventana de **inicio**, como desde el **perfil** se abrirá dicha foto donde nos encontraremos el usuario propietario de la foto, el título de la foto, la foto en cuestión, el número de *me*

gustas, un botón para darle *me gusta* a la foto, un recuadro para publicar un comentario a la foto y todos los comentarios publicados. También debajo de los comentarios nos encontramos con un botón que nos permitirá borrar la foto.

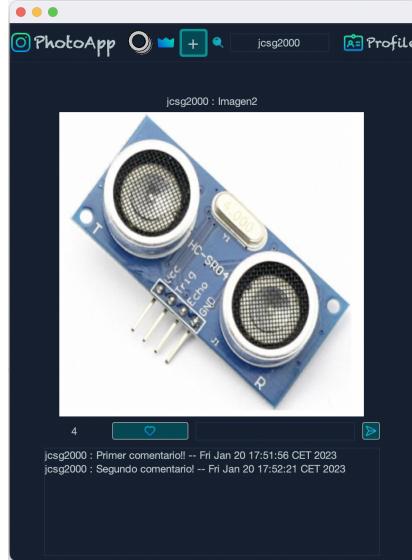


Figura 11: Foto

- **Perfil:** Cuando se pulsa en el botón del **perfil**, nos encontramos la foto de perfil de usuario, el nombre de usuario, un botón que nos permite cambiar cierta información del perfil, el número de seguidos y de seguidores, un botón para hacer *logout*, el nombre completo del usuario y la descripción del mismo. También se encuentran todas las **imágenes** que el usuario ha **subido** a la aplicación. **Debajo** de todas estas imágenes subidas nos encontramos con un botón que al pulsarlo nos llevará a los **álbumes** subidos.

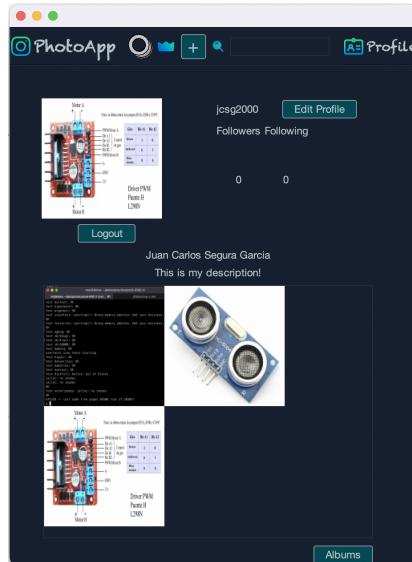


Figura 12: Perfil

Un usuario al pulsar el botón de **logout** se desconectará de la aplicación y volverá a la ventana de login.

•**Cambiar descripción:** Estando en el perfil, si pulsamos el botón *Edit Profile* nos llevará al panel para editar el perfil. Aquí tendremos la posibilidad de **editar** la descripción, la contraseña y la foto de perfil. El usuario podrá cambiar todos a la vez o los que quiera. Una vez todos los datos que se querían cambiar cambiados al pulsar el botón de **Save Changes** se guardarán todos los cambios.

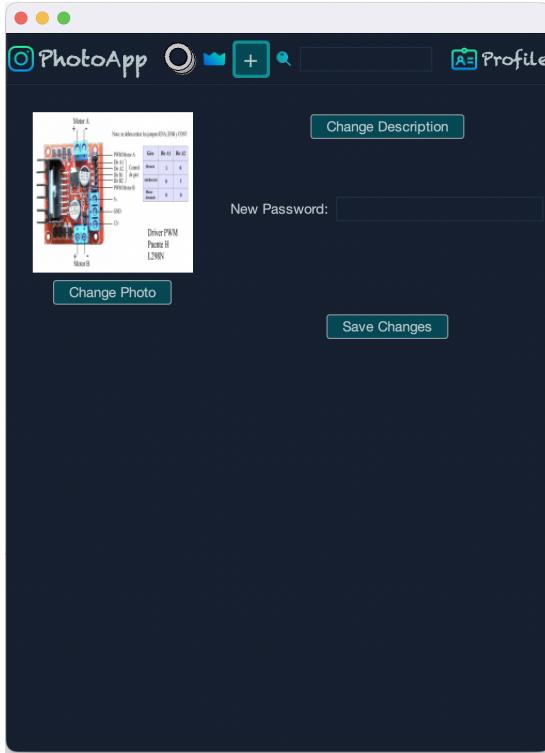


Figura 13: Pestaña cambiar perfil

•**Álbumes:** En el perfil al pulsar el botón de *Albums* nos dirigimos a la ventana donde se encuentran todos los álbumes del usuario.

En esta ventana nos encontramos todos los álbumes que el usuario ha subido. Al pulsar en cualquier álbum se abrirá y nos mostrará las imágenes que contiene el álbum así como los me gusta que tiene dicho álbum. Cabe recalcar que al darle **me gusta** al **álbum** también se le dará me gusta a **cada una** de las fotos contenidas en él.

En la parte **inferior** nos encontramos con dos botones. El botón de la derecha al pulsarlo nos devolverá a la **pestaña** donde se encuentran las **fotos** que ha subido el usuario y a la izquierda un botón para **subir un álbum**.

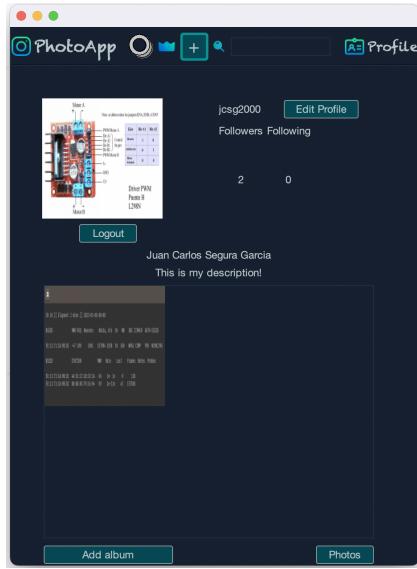


Figura 14: Pestaña álbumes

•**Crear álbumes:** Para crear un álbum nuevo, desde la pestaña de álbumes pulsamos el botón de **Add Album** y se nos abrirá una pestaña muy similar a la de **subir imágenes**. En esta pestaña tendremos que indicar el título del álbum y la descripción, elegir una foto que será la primera se que añada al álbum y ya al pulsar el botón de *Add Album* se subirá el álbum.

•**Subir fotos a álbumes:** Si pulsamos un álbum podemos ver el **creador** del álbum, el **título** del álbum y la primera imagen de este que consideramos que es la **portada** del álbum. También nos encontramos con un botón para añadir fotos al álbum y otro para eliminar el álbum.

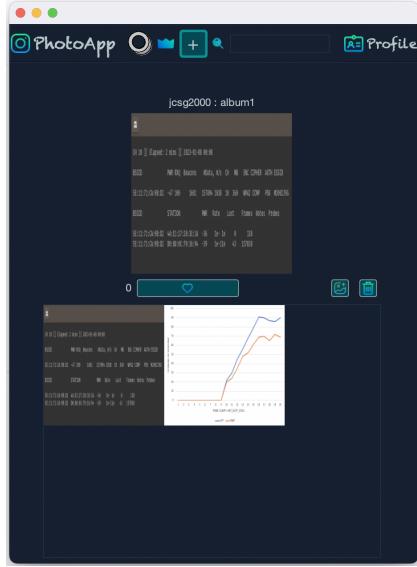


Figura 15: Álbum

Para añadir una nueva foto al álbum bastará con pulsar el botón de añadir una foto y nos saldrá una ventana similar a la de subir foto.

• **Premium:** Existe la posibilidad de que un usuario se haga premium de la aplicación, para ello basta con pulsar el botón de la corona del panel norte de la aplicación. Cuando es pulsado nos encontramos con la opción de hacerse premium y con un *free trial* de que nos podríamos encontrar si nos hiciésemos premium.

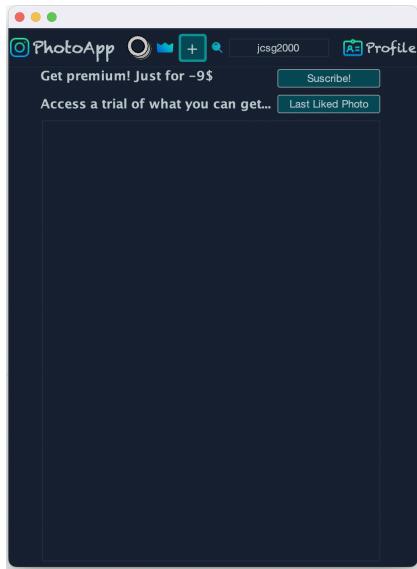


Figura 16: Premium

Si pulsamos el botón del *free trial* se nos mostrará la última foto de nuestro perfil a la que alguien le ha dado me gusta.

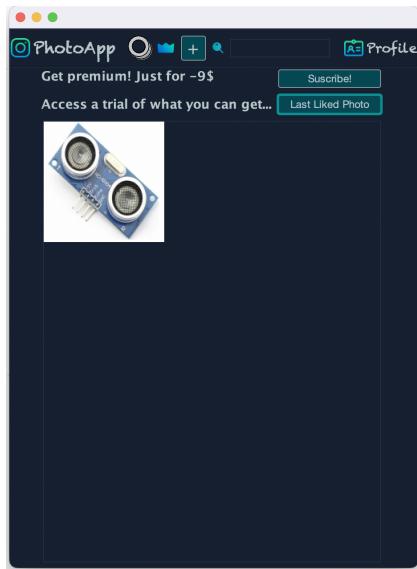


Figura 17: Free trial premium

Al hacerse premium contaremos con más funcionalidad. Automáticamente podremos obtener tanto un **PDF** como un **Excel** con nombre de usuario, correo y descripción de todos los usuarios que nos **siguen**. Además de esta funcionalidad también se podrán mostrar las 10 fotos con más me gusta que tenemos.

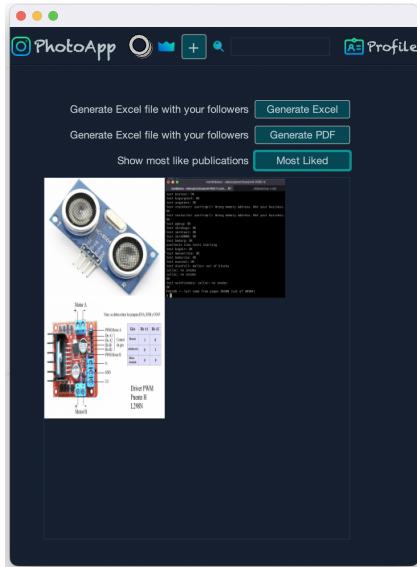


Figura 18: Funciones de usuario premium

- **Cargar Fotos:** Dentro de PhotoTDS podemos cargar también seleccionar un archivo XML indicando las fotos que queremos subir, con sus respectivo título, descripción y HashTags. Podemos ver un ejemplo de una subida efectiva en la siguiente Figura.

(a) Cuatro fotos subidas. (b) Archivo usado.

```

<?xml version="1.0"?>
<photos xmlns="http://www.tds.es/fotos" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.tds.es/fotos fotos.xsd">
<foto><Path><![CDATA[/Users/quico/Desktop/1.png]]</Path>
<Descripcion>Prueba de Imagen1</Descripcion>
<HashTags>
<HashTag>TDS</HashTag>
<HashTag>Cargador</HashTag>
</HashTags>
</foto>
<foto><Path><![CDATA[/Users/quico/Desktop/2.png]]</Path>
<Descripcion>Prueba de Imagen2</Descripcion>
<HashTags>
<HashTag>Pruebas</HashTag>
<HashTag>Cargador</HashTag>
</HashTags>
</foto>
<foto><Path><![CDATA[/Users/quico/Desktop/3.png]]</Path>
<Descripcion>Prueba de Imagen3</Descripcion>
<HashTags>
<HashTag>TDS</HashTag>
<HashTag>Pruebas</HashTag>
</HashTags>
</foto>
<foto><Path><![CDATA[/Users/quico/Desktop/4.png]]</Path>
<Descripcion>Prueba de Imagen4</Descripcion>
<HashTags>
<HashTag>Cargador</HashTag>
<HashTag>Pruebas</HashTag>
</HashTags>
</foto>
</photos>

```

Figura 19: Ejemplo subida varias fotos con XML.

11. Repositorio GitHub.

El **enlace** del repositorio de GitHub se puede encontrar aquí. Dentro del mismo, ambos integrantes del grupo hemos seguido la siguiente filosofía: crear una rama por funcionalidad, y cuando esté terminada fusionamos con la rama principal.

Siguiendo esta estrategia nos aseguramos que la rama principal siempre contenga una versión **sin errores**. En un mundo ideal, para poder integrar con esa rama debe pasar una serie de **test** para tener una mayor seguridad, sin embargo por falta de tiempo y

recursos nosotros no hemos pasado tests a la hora de **mergear** con main (en nuestro caso **dev**).

Pasamos a comentar las distintas ramas creadas:

- **dev:** es nuestra rama principal donde está la versión final de la aplicación. Cambiamos la rama principal de nuestro repositorio (2).
- **Controller:** es una rama que creamos al principio para implementar una pequeña versión del controlador.
- **Dominio:** creamos todas las clases del dominio y establecimos las relaciones entre ellas siguiendo el diagrama de clases.
- **login-window:** primera rama que creamos, en la que creamos las ventanas de login y registro para la aplicación.
- **front:** esta rama es donde se concentra casi todo el trabajo de front-end y arreglamos algunos pequeños bugs de ramas anteriores.
- **main:** rama obsoleta.
- **requisitos:** rama obsoleta.

12. Observaciones finales

Esta práctica nos ha resultado muy gratificante sobre todo por el resultado obtenido, gracias al esfuerzo de ambos. Ha sido la primera vez que hacíamos un desarrollo de software con cierto tamaño y nos llevamos varias lecciones para futuras ocasiones. Nos hemos dado cuenta de lo complicado que es manejar un código con cierto tamaño, y si nos ha costado a nosotros con PhotoTDS, no nos queremos imaginar en otras situaciones.

En líneas generales, pese a diversos problemas que nos han ido surgiendo sobre la marcha, creemos que las clases prácticas son de bastante ayuda pese a no avanzar con PhotoTDS en las mismas. Juntando esto con las clases teóricas, en líneas generales, hemos tenido una visión general bastante clara de PhotoTDS, lo cual ha agilizado bastante el desarrollo, sin contar con las incontables dudas relacionadas con aspectos técnicos que hemos tenido. De manera aproximada, ambos integrantes del grupo habremos dedicado a esta práctica al rededor de **40-50h** cada uno, siendo este un valor poco exacto.

Como aspecto final nos gustaría comentar que pese a que no hayamos incluido un apartado de **Refactoring**, hemos tenido que pasar por dos o tres procesos de refactorizar el código para poder avanzar de forma más eficiente en el proyecto. Nos ha ocurrido un par de veces el encontrarnos en una situación que nos estaba retrasando por una mala calidad del código y decidimos subsanarlo.

Para concluir, habiendo realizado este proyecto tomamos conciencia de la importancia de usar técnicas para facilitar el desarrollo, realizar pruebas, escribir código limpio y reusable, pero sobre todo intentar buscar la simplicidad.

Referencias

- [1] “Cambiar rama principal github.” <https://codely.com/blog/screencasts/arquitectura-hexagonal-ddd>.
- [2] “Cambiar rama principal github.” <https://docs.github.com/es/repositories/configuring-branches-and-merges-in-your-repository/managing-branches-in-your-repository/changing-the-default-branch>.