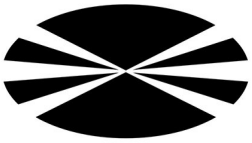


UNIVERSIDADE DA CORUÑA

Marcos de Desarrollo

Segunda iteración

Grupo 1.2

2021/2022

Jose Manuel Turnes Pazos	**jose.turnes**
Miguel Quiroga Rodriguez	**miguel.quiroga**
Rita Cernadas Tubío	**rita.cernadas**

Índice

1. Arquitectura global.

La estructura global de la aplicación está formada por 3 paquetes principales: Model, ModelTest y Web. El paquete Model, el primero de todos los paquetes, está formado tanto por la creación de la base de datos como la creación de las tablas, el EDMX con el contenido de las entidades y su relación entre ellas, los Dao's de la aplicación (CommentDao, ImageUploadDao, PublicationDao, UserProfileDao, TagDao y CategoryDao) y por último pero no por ello menos importante los Servicios necesarios para la elaboración de la aplicación (CommentService, ImageUploadService, PublicationService, UserProfileService y TagService).

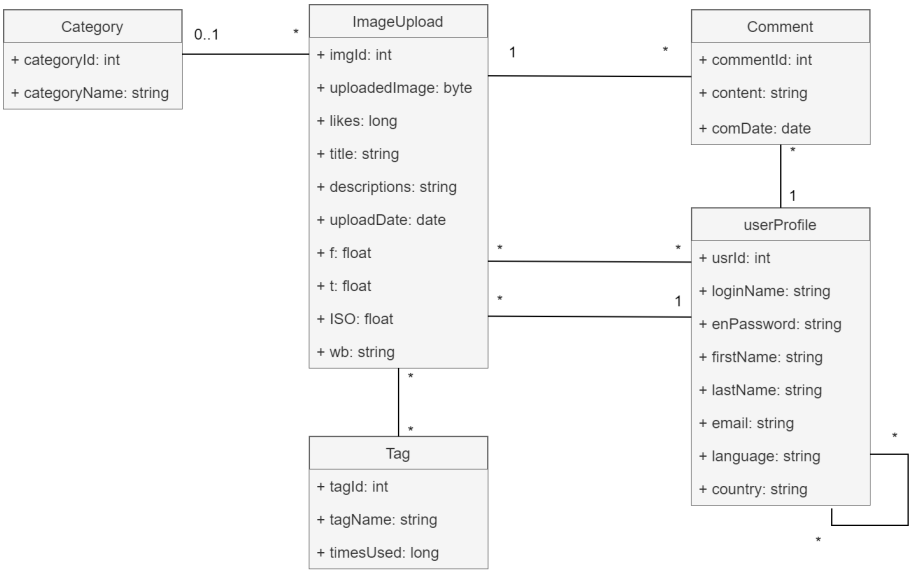
El paquete ModelTest, se encarga de comprobar si todas las funcionalidades implementadas en los Servicios de la capa Model están funcionando correctamente. Para ello, se elabora para los Servicios más relevantes los test necesarios (CommentService, ImageUploadService, PublicationService, UserProfileService). También contamos con las pruebas de NinjectXML.

Y por último contamos con el paquete Web, que es el que implementa la interfaz del usuario y está formado por las carpetas Content que contiene los contenidos utilizados en la aplicación, la carpeta Css que contiene el CSS de la aplicación, la carpeta fonts que contiene las fuentes que utilizamos, la carpeta HTTP, que incluye subcarpetas como Session, Util y View, luego tenemos la carpeta lib donde se encuentran todas las librerías que utilizamos, la carpeta Pages que es donde se encuentran todas las páginas de la aplicación, también tenemos los ficheros de internacionalización globales y locales y tenemos archivos como Global.asax, PracticaMaD.Master, .. entre otros.

2. Modelo.

2.1. Clases persistentes.

En este apartado se muestran todas las clases persistentes y la relación entre ellas.



2.2. Interfaces de los servicios ofrecidos por el modelo.

Para agrupar los distintos casos de uso en los cinco Servicios ofrecidos decidimos seguir un criterio basado en la relación entre ellos, es decir, agrupamos los casos de uso según con lo que estuviesen relacionados. Un ejemplo sería, todos los casos de uso relacionados con los usuarios, estarán en el servicio de UserProfile, mientras que los casos de uso relacionados con los comentarios estarán en el servicio de Comment.

El servicio de UserProfile, engloba a las casos de uso de “Registro de usuarios”, “Autenticación y salidas”, “Seguimiento de usuarios”, “Lista de seguidores” y “Lista de seguidos”. Concretamente se desarrollan las funcionalidades que vemos a continuación.

IUserService
+ ChangePassword(long userProfileId, string oldClearPassword, string newClearPassword): void
+ FindUserProfileDetails(long userProfileId): UserProfileDetails
+ Login(string loginName, string password, bool passwordIsEncrypted): LoginResult
+ RegisterUser(string loginName, string clearPassword, UserProfileDetails userProfileDetails): Long
+ UpdateUserProfileDetails(long userProfileId, UserProfileDetails userProfileDetails): void
+ UserExists(string loginName): bool
+ Follow(string followedLogin, string followerLogin): void
+ FollowerList(long userId, int startIndex, int count): List<UserProfileDto>
+ ListOfFollows(long userId, int startIndex, int count):List<UserProfileDto>
+ GetNumberOfFollows(long userId) : int
+ GetNumberOfFollowers(long userId): int
+ IsFollowed(long userId1, long userId2): bool
+ FindUserNameById(long userId): string

El servicio de Comment, engloba a la funcionalidad “Añadir comentario”, en la que podemos losl caso de uso a continuación.

ICommentService
+ ShowComments(pubId: long, startIndex: int, count: int): List<Comment>
+ AddComment(pubId: long, comment: string, usrId: long): void
+ UpdateComment(commentId: long, content: string): void
+ RemoveComment(commentId: long): void
+ CountComents(long imgId): long

El servicio de ImageUpload, este podemos decir que es uno de los servicios más importante, ya que es donde se gestionan la mayoría de las funcionalidades. En él se implementan los casos de uso relacionados con “Búsqueda de imágenes” , “Indicar “Me gusta” y “Subida de imágenes”. Concretamente, se desarrollan los casos de uso siguientes.

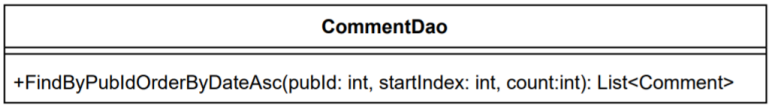
IImageUploadService
+ UploadImage(ImageUploadDetails img, List<string> tags, string category): long
+ UpdateImage(long pubId, ImageUploadDetails imageDetails): void
+ RemoveImage(long imgId): void
+ LikedImage(long imgId, long usrId): void
+ UnlikeImage(long imgId, long usrId): void
+ RecentUploads(long userId, int startIndex, int count):List<ImageUploadDto>
+ FindImage(long imgId): ImageUpload
+ SearchComments(long imgId, int startIndex, int count): List<CommentDto>
+ CountComments(long imgId): int
+ GetNumberOfImages(long userId): int
+ FindByKeywordAndCategory(string keywords,long categoryId, int startIndex,int count): List<ImageUpload>
+ IsLiked(long imgId,long userId):bool
+ CountSearchKeywords(string keywords, long categoryId): int
+ FindRecentUploads(): List<ImageUpload>
+ CountRecentUploads(): int
+ FindImageTags(long imgId, int startIndex, int count): List<Tag>
+ CountImageTags(long imgId): int
+ AddTag(Tag tag, long imgId): void

El servicio de Tag, engloba al caso de uso adicional de “Etiquetado de imágenes”. En él creamos las etiquetas y las recuperamos. Podemos ver los casos de uso a continuación.

ITagService
+ GetAllTags(): List<Tag>
+ FindMostUsedTags(int startIndex, int count): List<Tag>
+ CountTags(): int
+ FingImagesByTagId(long tagId, int startIndex, int count):List<ImageUpload>
+ CountImagesWithTag(long tagId): int
+ UpdateTags(long imgId, List<String> tags): void

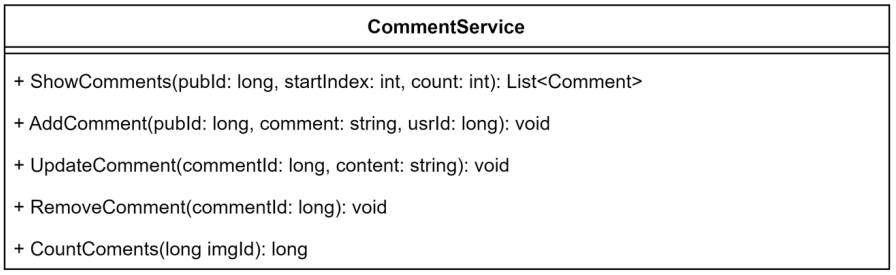
2.3. Diseño de un DAO.

En este apartado se incluye el diagrama de clases del CommentDao de la aplicación.

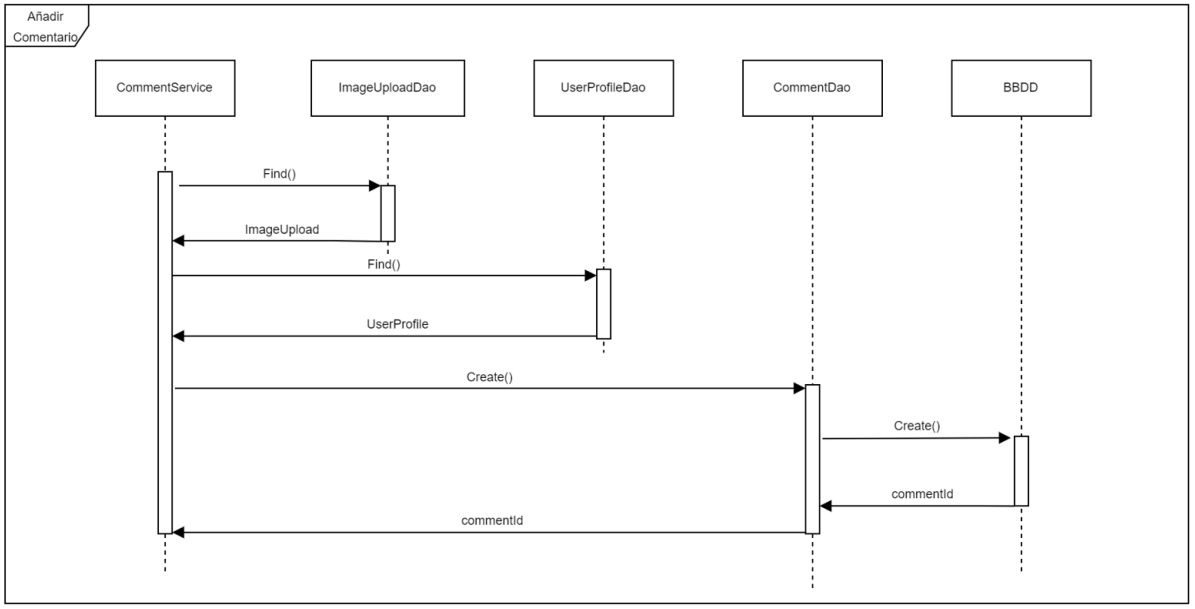


2.4. Diseño de un servicio del modelo.

En este apartado se incluye el diagrama de clases del CommentService de la aplicación.

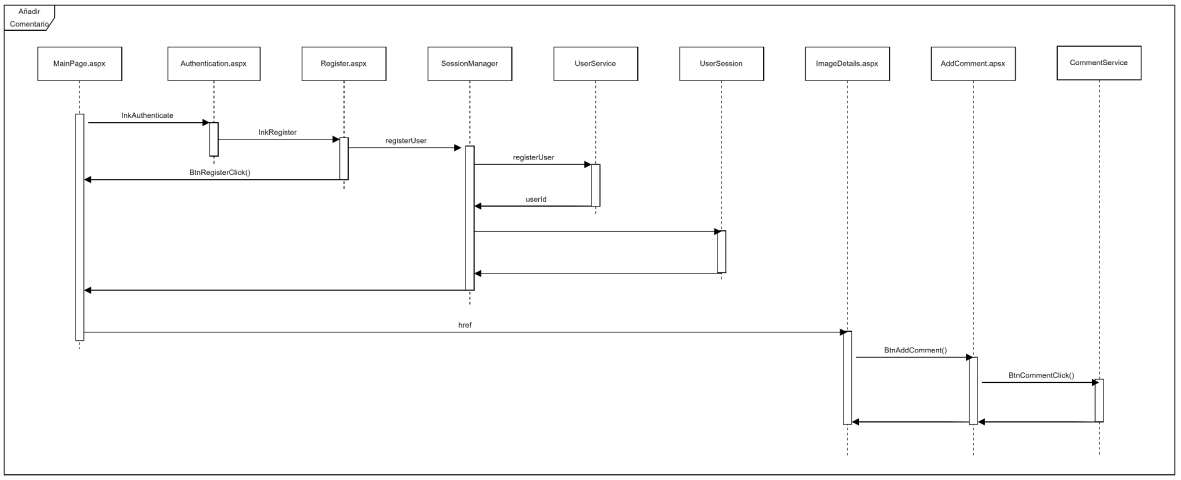


También se incluye el diagrama de secuencia que muestra la ejecución de AddComment en la capa modelo.



3. Interfaz gráfica.

En este apartado mostraremos el diagrama de secuencia que ilustra el caso de uso de añadir comentario, junto con el registro del usuario.



* Para ver el diagrama en mejor calidad mirar Anexo A.

4. Parte adicional.

4.1. Etiquetado de imágenes.

Para elaborar el apartado adicional “etiquetado de imágenes” creamos dos tablas en la base de datos a la que llamamos Tag y ImageTags. En la primera, Tag, se guardan los identificadores, nombre de la etiqueta y el número de veces usado cada etiqueta. En la segunda, ImageTags, se guardan las relaciones entre un identificador de etiqueta y una imagen a la que está asociada. La entidad Tag, como todas las otras, tiene su DAO y su servicio correspondiente, en el cual implementamos métodos como:

- **FindMostUsedTags**: Que busca los tags más usados para luego incluirlos en la nube de tags.
- **GetAllTags**: Que devuelve todos los tags que existen.
- **CountTags**: Que devuelve el número de etiquetas que tiene una imagen.
- **FindImagesByTagId**: Que devuelve todas las imágenes que usan una etiqueta con dicho id.
- **CountImagesWithTag**: Que devuelve el número de imágenes asociadas a una etiqueta.
- **UpdateTags**: Que actualiza las etiquetas de una imagen, pudiendo añadir o eliminar etiquetas.

En esta captura de pantalla podemos observar el formato que tenemos en todas las páginas para ver la nube de etiquetas. Se trata de una tabla, en el centro de cada página, donde indicamos el nombre de la etiqueta (Tag Name) y las veces que se usa cada una de ellas (Times Used).

Captura 1. Tabla/Nube de etiquetas.

Para saber qué etiquetas son las más usadas, a parte de que podemos ver el número de veces que esta es utilizada, también lo podremos observar en el tamaño de fuente del nombre de la etiqueta como podemos observar a en la captura 2. Para que esto fuese posible de implementar hicimos un bucle donde comprobamos que las etiquetas más usadas tuviesen un número mayor de tamaño de fuente, que difieren entre una y otra en un tamaño incremental. Al mismo tiempo que cambiamos de tamaño de fuente, también cambiamos el color de la fila de tabla. Todo esto está situado en PracticaMaD.Master.cs, para que de esta manera la tabla aparezca en todas las páginas de la aplicación.

Captura 2. Tabla/Nube de etiquetas.

```
var rows = gvTags.Rows;

for (int i = 0; i < rows.Count; i++)
{
    rows[i].Font.Size = 20 - 2 * i;
    rows[i].ControlStyle.ForeColor = System.Drawing.Color.White;
}
```

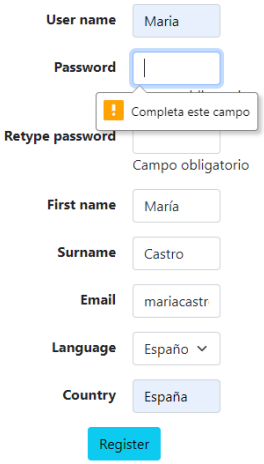
Captura 3. Código asociado al tamaño del nombre de la etiqueta.

4.2. Cacheado de búsquedas.

No implementado.

4.3. OAuth 2.0 y Bootstrap.

De esta funcionalidad tan solo implementamos el apartado relacionado con Bootstrap. Para poner en práctica esto, lo que hicimos fue añadir validaciones para cuando el usuario introduzca mal los datos, por ejemplo en la página de registro de usuarios:



Captura 4. Captura de registro de usuarios.

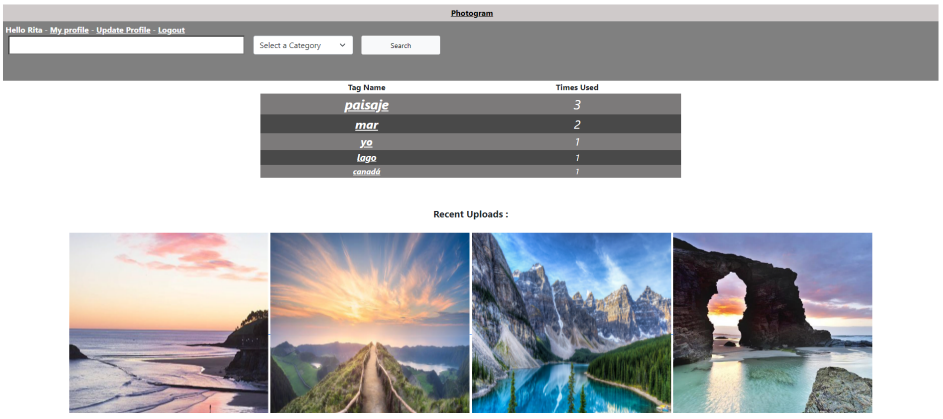
Como observamos en la captura 4, la aplicación indica al usuario que el campo de Password y Retype password son campos obligatorios, y por lo cual debe completarlos. Para poder incluir este tipo de validaciones en la aplicación tuvimos que implementarlos en las páginas .aspx, en concreto en el propio HTML añadiendo sentencias como:

```
<asp:RequiredFieldValidator ID="rfvUserName" runat="server" ControlToValidate="txtLogin"
    Display="Dynamic" Text="<%$ Resources:Common, mandatoryField %>"
    meta:resourcekey="rfvUserNameResource1"></asp:RequiredFieldValidator>
```

Captura 5. Código asociado a la implementación de la validación de los campos.

Además de esto, también añadimos la propiedad “required” en los controladores web que son obligatorios. Esto es el mensaje “Completa este campo”, como podemos observar en la captura 4, que será distinto en función del navegador usado.

También añadir que contamos con un diseño responsive gracias a la colocación de las fotos en la pantalla principal (muro principal de la aplicación). Todo esto es gracias a la utilización de un ListView, ya que nos permite que las imágenes se adapten al tipo de espacio que tienen. Como podemos observar a continuación con una resolución de 1400x800 que es lo mismo que un ordenador portátil aproximadamente, veríamos la aplicación de esta manera:



Captura 6. Página principal de la aplicación.

Para una resolución de una tablet/teléfono móvil, la veríamos así:

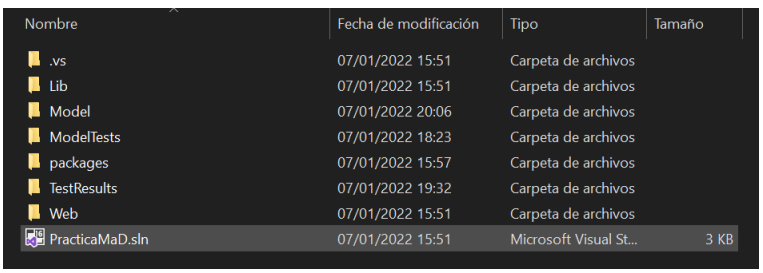
Captura 7. Página principal de la aplicación.

5. Compilación e instalación.

Para la compilación e instalación de la aplicación procederemos a seguir una serie de pasos:

1. Clonar el repositorio de GitHub haciendo:
   git clone https://github.com/joseturnes/MaD.21-22.git

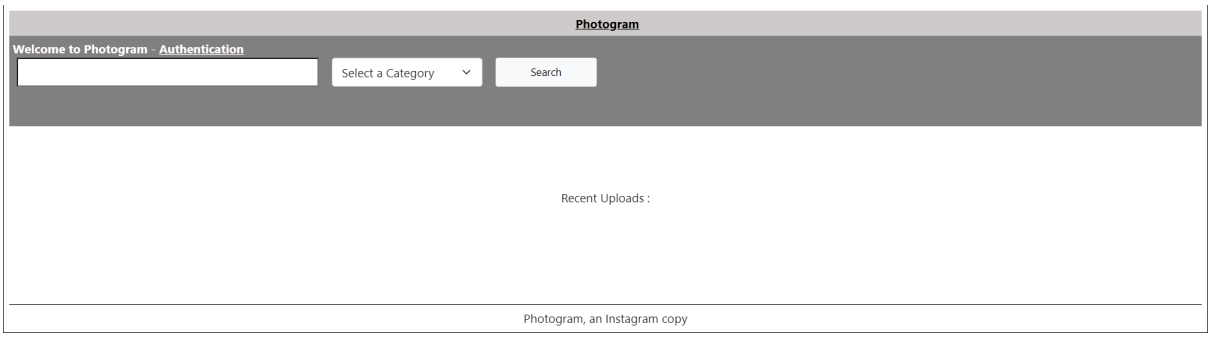
2. Abrir Visual Studio y seleccionar la opción de “Abrir un proyecto o una solución”, seleccionamos el directorio donde clonamos el repositorio, y una vez encontrado clicamos sobre la solución. En el caso de nuestra aplicación nos encontraríamos con algo como esto:



3. Una vez que ya tenemos el proyecto abierto en Visual Studio, compilaremos la solución.

4. Luego iremos a la carpeta Model>Sql dentro veremos que hay dos ficheros, uno que ejecutaremos primero que es SqlServerCreateDataBase.sql, y una vez que ejecutemos ese, pasaremos a ejecutar el SqlServerCreateTables.sql. Una vez hecho este paso la base de datos está lista.

5. Ahora llega el paso definitivo para poder ver la aplicación funcionando. Lo que tendremos que hacer es ir a la carpeta Web>Pages y ahí hacer click derecho sobre MainPage.aspx, donde le daremos a abrir con el explorador y ya podremos ver nuestra aplicación lista para funcionar!



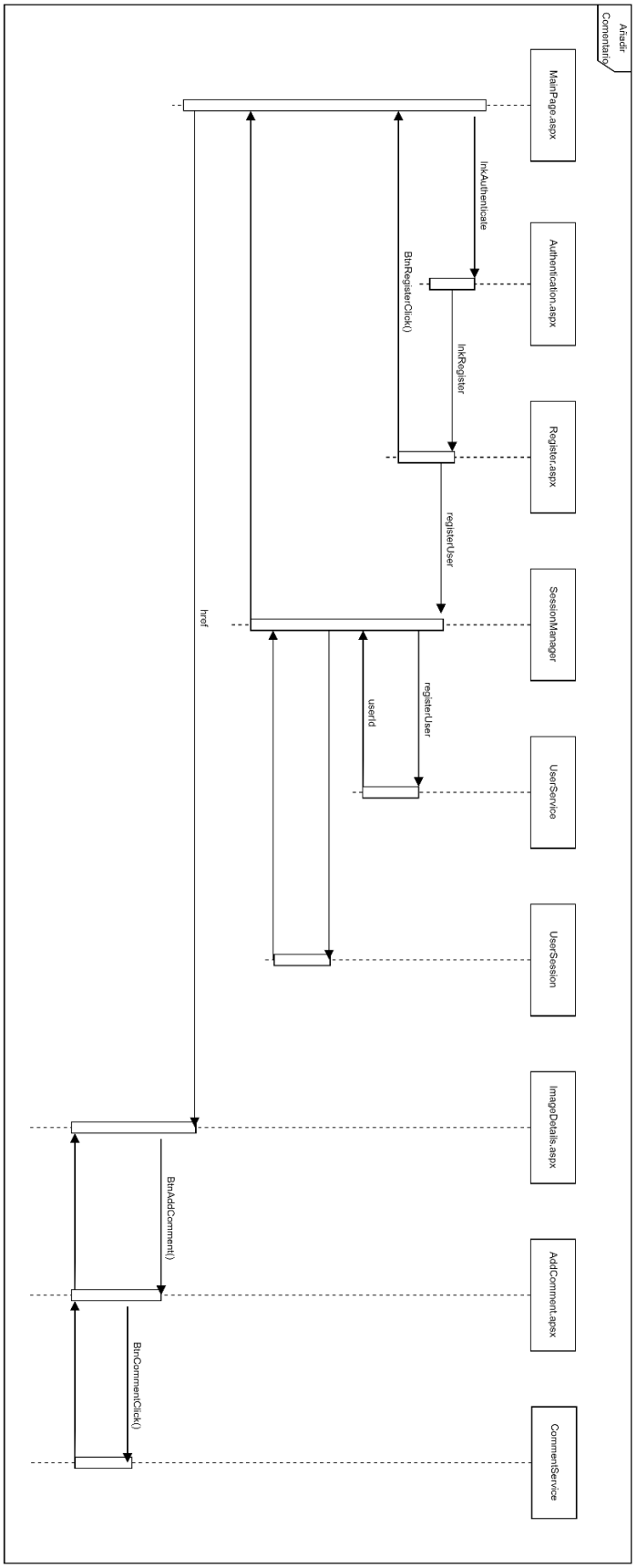
6. Problemas conocidos.

En este apartado trataremos los problemas que conocemos de la aplicación.

Uno de los problemas que tenemos es a la hora de realizar los test. Siempre que hacemos el recuento de etiquetas, imágenes, etc., el número que devuelve es erróneo, esto es debido a que la base de datos tiene basura y no somos capaces de eliminarla correctamente.

Otro de los problemas es que al ver los detalles de una imagen concreta, mostramos la imagen en el tamaño original, pudiendo causar esto que haya problemas de visualización.

ANEXO A.


Añadir Comentario
MainPage.aspx
Authentication.aspx
Register.aspx
SessionManager
UserService
UserSession
ImagoDetails.aspx
AddComment.apsx
CommentService
lnkAuthenticate
BtnRegisterClick()
lnkRegister
registerUser
href
userId
registerUser
BtnAddComment()
BtnCommentClick()