Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Trabajo Fin de Grado

EASELBOOK

PROTOTIPO DE APLICACIÓN WEB PARA UNA RED SOCIAL DEDICADA AL MUNDO DE LAS ARTES PLÁSTICAS

Alumno: David Clajer Sánchez

Tutor: José Ramón Balsas Almagro
Dpto: Departamento de informática

Septiembre, 2021

Índice

Índice de Tablas

Índice de Figuras

1.- Introducción

Este proyecto consiste en el desarrollo de un prototipo de aplicación web destinado a personas interesadas en el mundo del arte, en concreto, en el dibujo y la pintura. Con este proyecto se pretende que los artistas puedan divulgar sus creaciones artísticas a través de internet como plataforma. También se pretende crear oportunidades de trabajo para futuros artistas, facilitar la búsqueda de profesionales o material artístico para su uso en proyectos, corporaciones, educación, etc.

Los usuarios dispondrán de un perfil propio en el que podrían subir sus creaciones artísticas a la aplicación, compartiéndolas, de este modo, con el resto de usuarios. Cada obra artística podría ser clasificada en distintos apartados (cómo por ejemplo, en el tipo de técnica empleada, temática, paleta de color, licencia de uso...), con el fin de que se pueda realizar una búsqueda selectiva de pinturas y dibujos dentro de la aplicación. Además, la aplicación contará con mecanismos para la interacción entre los diferentes participantes de la comunidad.

1.1.- Objetivos

Este proyecto tiene como objetivos:

- Realizar un estudio de necesidades y soluciones existentes para el sistema propuesto.
- Diseñar un sistema informático especialmente orientado a la utilización de arquitecturas y metodologías de desarrollo web.
- Implementar un prototipo de aplicación web usando tecnologías JEE en el lado del servidor y tecnologías híbridas para el desarrollo de una aplicación móvil multiplataforma.

1.2.- Metodología

La metodología a emplear en este proyecto consistirá en una metodología ágil, basada en elementos adaptados de Scrum, el cual es “un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto” [1].

Al tratarse de una metodología que se basa en el trabajo colaborativo, no es posible aplicarla directamente a este proyecto, ya que hay partes que no se pueden adaptar, como son los roles y las reuniones del equipo de trabajo. No obstante, sí se pueden utilizar algunos elementos de esta metodología, como pueden ser:

- Product Backlog: para definir los requisitos en forma de historias de usuario y para estimar y/o re-estimar la puntuación de las historias de usuario. Como se dice en Proyectos Agiles [2]: “La lista priorizada de objetivos/requisitos representa la visión y expectativas del cliente respecto a los objetivos y entregas del producto o proyecto”.
- Sprint Backlog. Se hará una breve declaración de cuál será el foco del trabajo durante el sprint, se escogerán las tareas, se realizará la estimación restante y se actualizará el trabajo restante conforme se avance en el proyecto.

Asimismo, se empleará la metodología MoSCoW para priorizar y determinar la importancia de las características del proyecto. Esta metodología sirve para determinar la priorización dentro de proyectos con limitaciones de tiempo [3].

1.3.- Estructura del documento.

El presente trabajo se ha organizado de la siguiente manera:

- En el apartado 2 se realizará un análisis preliminar tanto de los requisitos como de las aplicaciones similares que podemos encontrar en el mercado. Además, se detallará la selección de tecnologías empleadas, objetivos, planificación inicial, análisis de la interfaz y el estudio de viabilidad.
- En el apartado 3 se estudiarán los distintos aspectos del diseño de la aplicación, así como los diagramas que han ayudado a la realización del mismo y el diseño de la interfaz de la aplicación.
- En el apartado 4 se explicarán los aspectos relacionados con la implementación y el tipo de arquitecturas empleadas.
- En el apartado 5 se expondrán las pruebas realizadas para garantizar el correcto funcionamiento de la aplicación.

- Por último, se encuentran las conclusiones, la bibliografía, y los apéndices de instalación, funcionamiento y esquema de la entrega.

2.- Análisis

2.1.- Análisis preliminar

La idea de desarrollar esta aplicación web surge de la necesidad de crear una red social destinada específicamente a personas interesadas en el mundo del arte, pues hasta ahora no existe ninguna plataforma de este tipo en el mercado. Actualmente, si una persona quisiera exponer sus obras artísticas y ver las de otros, tendría que recurrir a otras redes sociales generales, como son Twitter e Instagram. Por el contrario, con esta red social específica que se plantea, se podría aprender viendo las obras de otros artistas de la plataforma y darse a conocer interaccionando con ellos a través de dicha aplicación. Esto puede ayudar a crear oportunidades de trabajo para futuros artistas, búsqueda de profesionales o material artístico para uso en proyectos, corporaciones, educación, etc.

2.1.1.- Estudio de soluciones similares

En este apartado se estudiarán distintas aplicaciones similares que pueden encontrarse en el mercado, con el objetivo de extraer ideas o justificar mejoras que tendrá este prototipo respecto a ellas.

Instagram (https://www.instagram.com/)**.** Se trata de una red social que permite a los usuarios subir imágenes, fotos o vídeos, comentarlas entre ellos y comunicarse entre sí mediante chat privado. De esta red social se extrajo la idea de la estructura para la aplicación, pues la forma en la que tiene organizada la galería de imágenes que el usuario sube a su perfil resulta muy intuitiva y fácil de utilizar (ver *Figura 1*). Una de las desventajas que tiene esta aplicación es la dificultad que tiene un usuario para darse a conocer a otros dentro de la plataforma, pues no permite compartir publicaciones de otras personas. Instagram dispone de aplicación móvil y de interfaz web (aunque esta versión es más limitada que la móvil, pues no permite subir post desde la web).

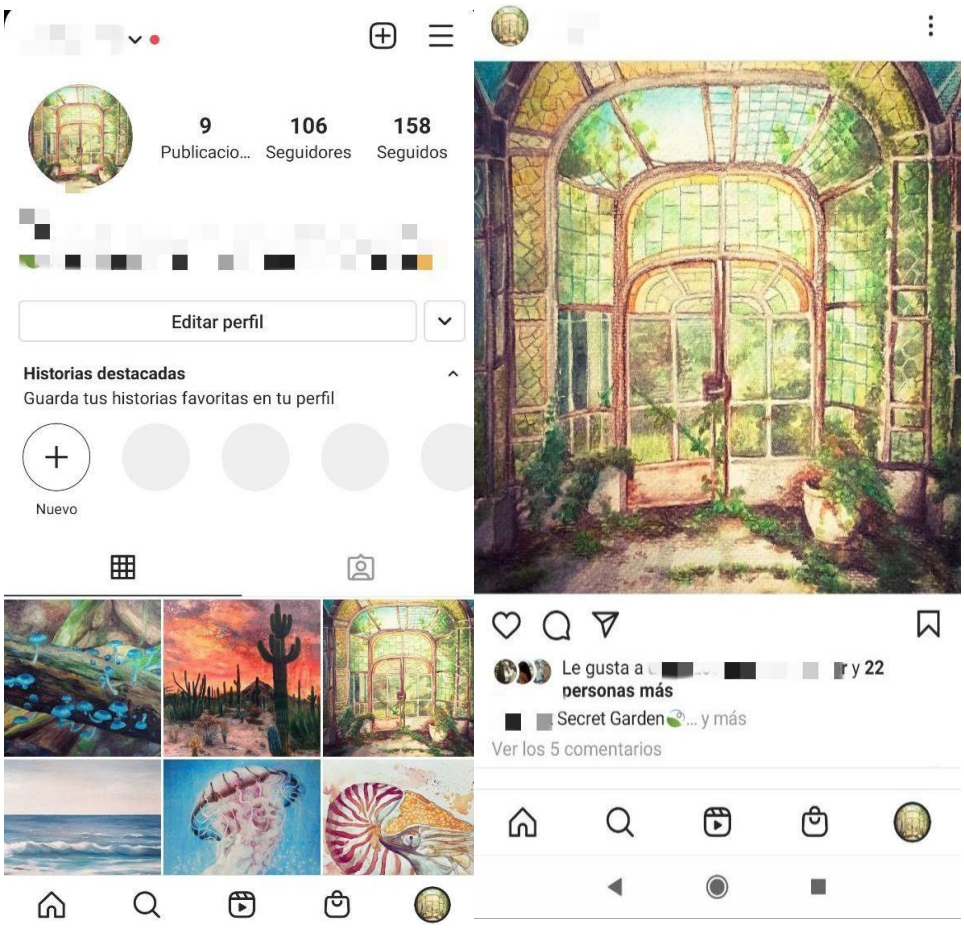

Figura 1. Publicaciones en la red social Instagram.

Twitter (https://twitter.com/). Al igual que Instagram, se trata de una red social que permite subir publicaciones con imágenes o vídeos, además de enviar comentarios y mensajes a otros usuarios (ver *Figura 2*). De esta aplicación se extrajo la idea de que un usuario pueda comentar sus propias creaciones. Uno de los inconvenientes que presenta Twitter es que en el apartado de inicio no muestra las publicaciones al resto de usuarios en orden cronológico, sino que aparecen antes aquellas que recomienda la propia aplicación. Además, no permite tener las publicaciones organizadas en forma de galería en el perfil, lo que dificultaría la búsqueda de las creaciones de un artista para el resto de usuarios. Esta aplicación dispone tanto de interfaz web como de aplicación móvil.

Tanto Twitter como Instagram gestionan los amigos del usuario como seguidores o seguidos. Además, permiten al usuario poner una biografía en el perfil, que consiste en una pequeña descripción de la actividad que realiza o del

tema que trata. Estas ideas también fueron extraídas para la elaboración de la aplicación.

Figura 2. Publicaciones en la red social Twitter.

Artfol (https://artfol.co/). Se trata de una red social para artistas (ver *Figura 3*), pero presenta algunos fallos ya que se encuentra todavía en desarrollo. Además, no es del todo intuitiva, no está bien traducida y solo está disponible para dispositivos móviles (su interfaz web solo funciona a nivel informativo y para buscar usuarios de la red social). Por el contrario, el prototipo de aplicación que se pretende realizar en este proyecto sí se podría visitar tanto en aplicación web como móvil. No obstante, Artfol tiene funciones muy interesantes, entre ellas, la de exponer en la pantalla de registro una obra aleatoria de uno de sus usuarios de fondo y un banner interactivo con los distintos tipos de obras en la aplicación.

Esta última funcionalidad se ha utilizado en este proyecto para realizar la clasificación de las obras en categorías.

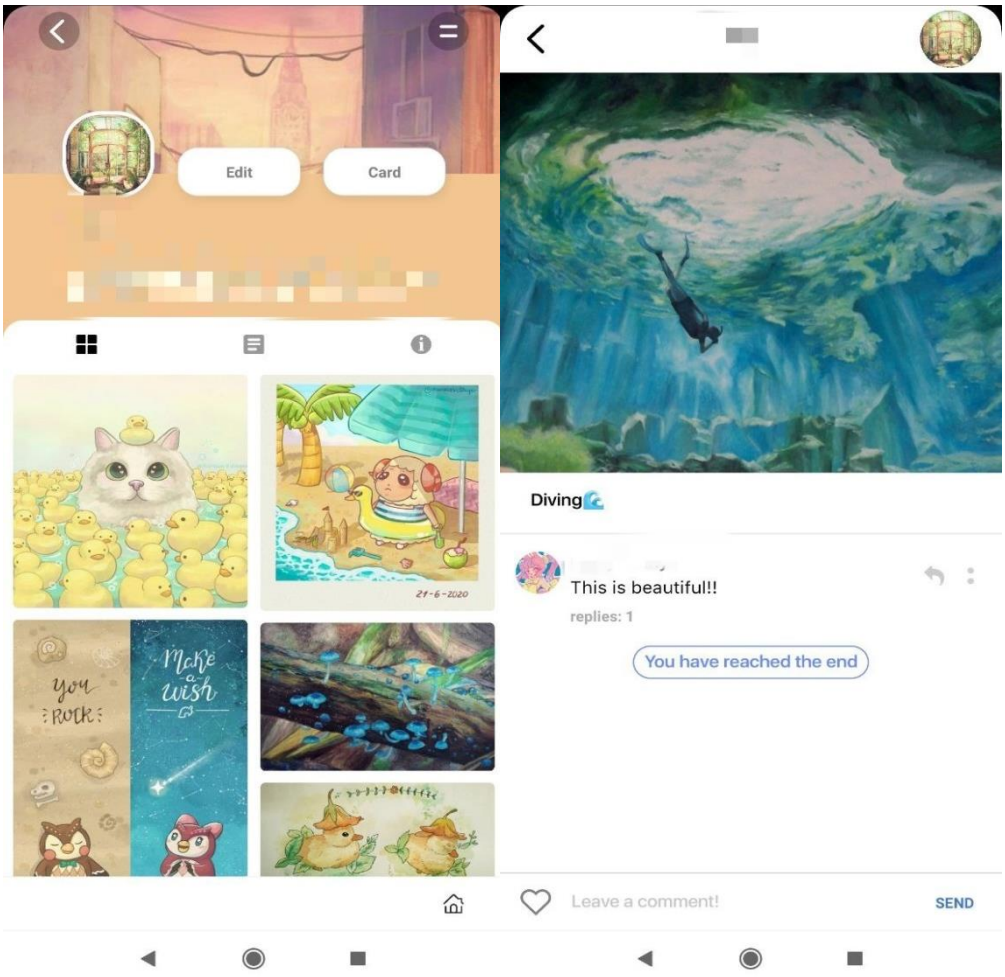

Figura 3. Publicaciones en la red social Artfol.

2.1.2.- Selección de tecnologías

Para este proyecto se podría hacer uso de varias soluciones o enfoques tecnológicos, como pueden ser:

- **Aplicaciones nativas.** Son específicas para un sistema operativo móvil en concreto y, si se quisiese desarrollar una app multiplataforma, implicaría aumentar el trabajo y el coste [4]. Para el desarrollo de estas aplicaciones en Android se puede emplear Java o Kotlin. Aunque Java es uno de los lenguajes más utilizados para programación, Kotlin mejora algunos de los problemas que presenta, siendo además compatible con

el código Java y con un excelente soporte en Android Studio [5]. Por otro lado, en iOS se puede utilizar Swift, que se trata de un lenguaje de programación creado por Apple, de código abierto y compatible con los Frameworks Cocoa y Cocoa Touch [6].

- **Aplicaciones web.** Se desarrollan normalmente con lenguaje JavaScript, junto con CSS y HTML. Se trata de páginas web con apariencia de aplicación nativa a las que se accede desde cualquier navegador, por lo que su ejecución es posible en diferentes sistemas operativos. Para el desarrollo de una aplicación web se podrían utilizar frameworks como Spring o Java Server Faces (JSF). Para la implementación en el lado del cliente existen también distintos frameworks como, por ejemplo, Angular, React o Vue, entre otros [4].
- **Aplicaciones híbridas.** Se trata de aplicaciones web que, en lugar de visualizarse en un navegador desde un servidor remoto, se insertan en un contenedor creado para la plataforma a la que pertenece el dispositivo móvil. De esta forma, se ejecutan como una aplicación nativa, evitando así tener que desarrollar una aplicación para cada sistema operativo [4]. Algunos de los Kit de Desarrollo de Software (SDK) que permiten desarrollar aplicaciones en Android, iOS y Web son Ionic [7] o Flutter [8], que se tratan de frameworks de código abierto.

Finalmente, se decidió realizar una aplicación híbrida que pudiera utilizarse en dispositivos móviles con un back-end REST para intercambio de información. De este modo, en un futuro sería posible crear un sistema basado en un API REST a la que se conectarían distintos tipos de clientes, como aplicaciones móviles nativas o conexión desde otros portales especializados. Para ello, se podrían utilizar diferentes lenguajes, como, por ejemplo, PHP (Laravel...), Python (Django, Flask) o java (SpringMVC, JAX-RS).

2.2.- Propuesta de solución

Este proyecto pretende que la aplicación permita a los usuarios darse de alta y de baja en la red social, modificar su perfil, subir sus creaciones a la

aplicación y realizar distintas operaciones sobre ellas, así como interactuar con otros usuarios a través de comentarios en las publicaciones.

Para implementar el servidor se hará uso del framework Spring (api/rest), a través del cual se irán diseñando los distintos recursos REST (urls) y métodos soportados (GET, POST, PUT...). Además, se requerirá también de la Base de Datos relacional Apache Derby.

Asimismo, se utilizará el framework Angular junto con la tecnología Ionic para implementar la aplicación híbrida por parte del cliente. De este modo, se podrán obtener fácilmente versiones de la aplicación tanto para Android como para IOS.

2.3.- Historias de Usuario

En este apartado se van a definir las distintas historias de usuarios que tendrá el sistema para su posterior planificación e implementación. Dichas historias son las siguientes:

1. **Registro de un usuario**
    - **Descripción:** Como usuario quiero registrarme en el sistema para poder utilizar la aplicación.
    - **Criterios de aceptación:** Que en la interfaz haya un botón para registrarse y que al pulsarlo y rellenar los datos, pueda utilizar la aplicación.
2. **Baja de un usuario**
    - **Descripción:** Como usuario quiero darme de baja en el sistema para borrar mis datos.
    - **Criterios de aceptación:** Que en el perfil haya un botón para darse de baja y que al pulsarlo y verificarlo, se borren todos mis datos de la aplicación.
3. **Modificación de un usuario**
    - **Descripción.** Como usuario quiero modificar mis datos para tenerlos actualizados.
    - **Criterios de aceptación:** Que en el perfil haya una opción para poder ver/editar los datos de usuario.

4. **Tipo de perfil de un usuario**
    - **Descripción.** Como usuario quiero seleccionar un perfil público o privado para permitir o no que todos los usuarios puedan ver mis creaciones.
    - **Criterios de aceptación:** Que en la aplicación haya una opción de seleccionar perfil público o privado.
5. **Subir creación**
    - **Descripción.** Como usuario quiero subir cualquier creación para tenerla en mi perfil.
    - **Criterios de aceptación:** Que en el perfil haya una opción de subir creación.
6. **Eliminar creación**
    - **Descripción.** Como usuario quiero eliminar cualquier creación para no tenerla en mi perfil.
    - **Criterios de aceptación:** Que en el perfil haya una opción de eliminar creación.
7. **Modificar creación**
    - **Descripción.** Como usuario quiero modificar cualquier creación para asignarle otro título y/o categoría.
    - **Criterios de aceptación:** Que en una creación subida haya una opción para modificar dicha creación.
8. **Buscar creaciones**
    - **Descripción.** Como usuario quiero buscar cualquier creación para poder visualizarla.
    - **Criterios de aceptación:** Que en la aplicación haya un buscador para buscar cualquier creación por su nombre, categoría o autor.
9. **Clasificar creaciones**
    - **Descripción.** Como usuario quiero clasificar cualquier creación que suba para que se corresponda con su categoría.
    - **Criterios de aceptación:** Que en la aplicación haya una opción para seleccionar la categoría a la que pertenece la creación.

10. Tipos de licencia en creaciones

- **Descripción.** Como usuario quiero clasificar cualquier creación que suba en un tipo de licencia creative commons para decidir la manera en la que mi creación va a circular en la red.
- **Criterios de aceptación:** Que en la aplicación haya una opción para que el autor de su creación decida qué tipo de licencia quiere asignarle.

11. Buscar usuarios

- **Descripción.** Como usuario quiero buscar cualquier usuario para añadirlo como amigo, visualizar sus creaciones o mandarle un mensaje.
- **Criterios de aceptación:** Que en la aplicación haya un buscador para buscar cualquier usuario por su nombre.

12. Añadir amigos

- **Descripción.** Como usuario quiero añadir a otro usuario de la aplicación como amigo para poder visualizar sus creaciones o mandarle un mensaje.
- **Criterios de aceptación:** Que en la aplicación haya una opción para añadir a usuarios como seguidos.

13. Eliminar amigos

- **Descripción.** Como usuario quiero eliminar a un usuario de la aplicación como amigo para quitarlo de mi círculo.
- **Criterios de aceptación:** Que en la aplicación haya una opción para eliminar a usuarios seguidos.

14. Enviar Mensajes

- **Descripción.** Como usuario quiero enviar a un usuario un mensaje para poder comunicarme con él.
- **Criterios de aceptación:** Que en la aplicación haya una opción para enviar mensajes a un usuario.

15. Comentar creaciones

- **Descripción.** Como usuario quiero comentar una creación de otro usuario.

- **Criterios de aceptación:** Que en la aplicación haya una opción para comentar las creaciones de los usuarios.

16. Valorar creaciones

- **Descripción.** Como usuario quiero valorar con un “me gusta” una creación de un usuario.
- **Criterios de aceptación:** Que en la aplicación haya una opción de dar “me gusta” a la obra de un autor.

17. Ver creaciones públicas mejor valoradas

- **Descripción.** Como usuario quiero ver las creaciones públicas mejor valoradas de los diferentes autores de la plataforma.
- **Criterios de aceptación:** Que en la aplicación haya una sección donde se pueda ver las creaciones públicas con un mayor número de “me gustas”.

2.4.- Planificación temporal

En primer lugar, se llevó a cabo la definición del proyecto, dividiéndolo en objetivos expresados como **historias de usuario** (descritas en el apartado anterior). Asimismo, se indicaron cuáles iban a ser los **criterios de aceptación** de cada historia de usuario, es decir, las pruebas que se realizan para comprobar si el sistema funciona de la manera esperada y si se satisfacen los objetivos propuestos [9].

En segundo lugar, las historias de usuario fueron divididas en tareas, indicando los **puntos de historia** que le correspondían a cada una. Estos puntos representan el esfuerzo que requiere realizar una historia de usuario.

Por último, se realizó una priorización de tareas por medio de la metodología MoSCoW. Este método, desarrollado por Dai Clegg, sirve para determinar la priorización de tareas en proyectos con limitaciones de tiempo. Las siglas de MoSCoW significan: Must Have (Debe Tener), Should Have (Debería incluir), Could Have (Podría Incluir) y Won’t Have (No se van a hacer). Estos requisitos se agrupan en cuatro categorías:

Must have (Debe tener). Estas características serán absolutamente críticas para tu nuevo proyecto, y sin ellas el proyecto es un fracaso.

Should have (Debería incluir). Estos aspectos de tu proyecto son críticos también, pero no imprescindibles.

Could have (Podría incluir). Estas iniciativas son las que estaría bien tenerlas, ya que añadirían valor al proyecto, pero no son críticas.

Won't have (No se van a hacer). Estas características no se merecen la inversión y no aportan ningún beneficio en este momento y se podrían considerar más tarde [3].

Las tareas en las que se descompusieron las historias de usuario son las siguientes:

1. **Registro de un usuario**
    - Descripción: Como usuario quiero registrarme en el sistema para poder utilizar la aplicación.
    - Criterios de aceptación: Que en la interfaz haya un botón para registrarse y que al pulsarlo y rellenar los datos, pueda utilizar la aplicación.
        - **Descomposición en tareas**:
            - Generar un botón con confirmación en la vista ajustes del usuario.
            - Enviar una petición DELETE al servidor.
            - Eliminar de la base de datos, los datos del usuario.
2. **Baja de un usuario**
    - Descripción: Como usuario quiero darme de baja en el sistema para borrar mis datos.
    - Criterios de aceptación: Que en el perfil haya un botón para darse de baja y que al pulsarlo y verificarlo, se borren todos mis datos de la aplicación.
        - **Descomposición en tareas**:
            - Generar un botón con confirmación en la vista ajustes del usuario.
            - Enviar una petición DELETE al servidor.
            - Eliminar de la base de datos, los datos del usuario.

3. **Modificación de un usuario**
    - Descripción. Como usuario quiero modificar mis datos para tenerlos actualizados.
    - Criterios de aceptación: Que en el perfil haya una opción para poder ver/editar los datos del usuario.
        - **Descomposición en tareas**:
            - Generar una vista (ajustes del usuario), en la que pueda modificar sus datos.
            - Enviar los datos en una petición PUT al servidor.
            - Modificar de la base de datos los datos del usuario.
4. **Tipo de perfil de un usuario**
    - Descripción. Como usuario quiero poder seleccionar un perfil público o privado para que ciertos usuarios puedan ver mis creaciones.
    - Criterios de aceptación: Que en la aplicación haya una opción de seleccionar perfil público o privado.
        - **Descomposición en tareas**:
            - Generar un botón para modificar el tipo de perfil (público o privado) de un usuario.
            - Enviar los datos en una petición POST al servidor.
            - Modificar de la base de datos los datos del usuario.
5. **Subir creación**
    - Descripción. Como usuario quiero subir cualquier creación para tenerla en mi perfil.
    - Criterios de aceptación: Que en el perfil haya una opción de subir creación.
        - **Descomposición en tareas**:
            - Generar un botón para poder subir una creación.
            - Enviar los datos en una petición POST al servidor.
            - Guardar los datos de la creación en la base de datos.

6. **Eliminar creación**
    - Descripción. Como usuario quiero eliminar cualquier creación para no tenerla en mi perfil.
    - Criterios de aceptación: Que en una creación subida haya una opción para eliminar dicha creación.
        - **Descomposición en tareas**:
            - Generar un botón con confirmación en la vista de la creación seleccionada.
            - Enviar una petición DELETE al servidor.
            - Eliminar de la base de datos, los datos de la creación.

7. **Modificar creación**
    - Descripción. Como usuario quiero modificar cualquier creación para asignarle otro título y/o categoría.
    - Criterios de aceptación: Que en una creación subida haya una opción para modificar dicha creación.
        - **Descomposición en tareas**:
            - Generar un botón para seleccionar que el usuario quiere modificar la creación.
            - Generar una vista/formulario/desplegable para modificar los datos de la creación.
            - Enviar los datos modificados en una petición PUT al servidor.

8. **Buscar creaciones**
    - Descripción. Como usuario quiero buscar cualquier creación para poder visualizarla.
    - Criterios de aceptación: Que en la aplicación haya un buscador para buscar cualquier creación por su nombre, categoría o autor.
        - **Descomposición en tareas**:
            - Generar un buscador para que el usuario pueda buscar creaciones por nombre.
            - Mostrar mediante una petición GET del usuario al servidor, las creaciones que coincidan con el nombre a buscar.

9. Clasificar creaciones

- Descripción. Como usuario quiero clasificar cualquier creación que suba para que se corresponda con su categoría.
- Criterios de aceptación: Que en la aplicación haya una opción para seleccionar la categoría a la que pertenece la creación.
    - **Descomposición en tareas**:
        - Generar un campo de tipo enum (con las clasificaciones artísticas que tiene la aplicación) en los datos de las creaciones.
        - Generar un desplegable para que el usuario pueda seleccionar la clasificación a la que pertenece su creación.

10. Tipos de licencia en creaciones

- Descripción. Como usuario quiero clasificar cualquier creación que suba con un tipo de licencia creative commons para decidir la manera en la que mi creación va a circular en la red.
- Criterios de aceptación: Que en la aplicación haya una opción para que el autor de su creación decida qué tipo de licencia quiere asignarle.
    - **Descomposición en tareas**:
        - Generar un campo de tipo enum (con los tipos de licencia creative commons que tiene la aplicación) en los datos de las creaciones.
        - Generar un desplegable para que el usuario pueda seleccionar la licencia que va a tener su creación.

11. Buscar usuarios

- Descripción. Como usuario quiero buscar cualquier usuario para añadirlo como amigo, visualizar sus creaciones o mandarle un mensaje.
- Criterios de aceptación: Que en la aplicación haya un buscador para buscar cualquier usuario por su nombre.
    - **Descomposición en tareas**:

- Generar un buscador para que el usuario pueda buscar usuarios por nombre de usuario.
- Mostrar los usuarios que coincidan con el nombre a buscar mediante una petición GET del usuario al servidor.

12. Añadir amigos

- Descripción. Como usuario quiero añadir a otro usuario de la aplicación como amigo para poder visualizar sus creaciones o mandarle un mensaje.
- Criterios de aceptación: Que en la aplicación haya una opción para añadir a usuarios como amigos.
    - **Descomposición en tareas**:
        - Generar un botón para añadir amigos en la aplicación.
        - Enviar los datos de la petición y el usuario mediante una petición POST al servidor.
        - Incrementar los amigos del usuario.

13. Eliminar amigos

- Descripción. Como usuario quiero eliminar a un usuario de la aplicación como amigo para quitarlo de mi círculo.
- Criterios de aceptación: Que en la aplicación haya una opción para eliminar usuarios para dejar de ser amigos.
    - **Descomposición en tareas**:
        - Generar un botón para eliminar amigos en la aplicación.
        - Enviar los datos de la petición y el usuario mediante una petición DELETE al servidor.
        - Decrementar los amigos del usuario.

14. Enviar Mensajes

- Descripción. Como usuario quiero enviar a un usuario un mensaje para poder comunicarme con él.

- Criterios de aceptación: Que en la aplicación haya una opción para enviar mensajes a un usuario.
    - **Descomposición en tareas**:
        - Generar un botón para poder generar un mensaje a un usuario.
        - Enviar los datos del mensaje y el usuario mediante una petición POST al servidor.

15. Comentar creaciones

- Descripción. Como usuario quiero comentar una creación de otro usuario para poder comentarle lo que opino respecto a su creación.
- Criterios de aceptación: Que en la aplicación haya una opción para poder comentar las creaciones de los usuarios.
    - **Descomposición en tareas**:
        - Generar un botón en las creaciones para poder comentarlas.
        - Enviar los datos del comentario y el usuario mediante una petición POST al servidor.

16. Valorar creaciones

- Descripción. Como usuario quiero valorar con un “me gusta”, una creación de un usuario para que pueda ver la repercusión que tiene su obra.
- Criterios de aceptación: Que en la aplicación haya una opción de dar “me gusta” a la obra de un autor.
    - **Descomposición en tareas**:
        - Generar un botón que permita la valoración de la creación por parte de un usuario.
        - Enviar los datos de la valoración mediante una petición POST al servidor.

17. Ver creaciones públicas mejor valoradas

- Descripción. Como usuario quiero ver las creaciones públicas mejor valoradas de los diferentes autores de la plataforma para observar lo que más valora la comunidad.

- Criterios de aceptación: Que en la aplicación haya una sección donde se puedan ver las creaciones públicas mejor valoradas.
    - **Descomposición en tareas**:
        - Generar una vista.
        - Mostrar mediante una petición GET del servidor al cliente, las creaciones públicas mejor valoradas.

Para estimar los **Puntos de Historia** de cada historia del usuario se empleó una escala creciente con valores comprendidos entre el 1 y el 10, donde el valor más pequeño significa un menor esfuerzo o dificultad y el valor más grande, mayor esfuerzo. Para calcular el valor de la estimación, se empleó la técnica de estimación por analogía [10], la cual permite comparar el elemento a estimar con otros de proyectos anteriores o entre sí. De esta forma, se hizo una comparativa entre el nivel de dificultad de las distintas historias de usuario iniciales. Además, se calculó la media entre la dificultad/esfuerzo en la implementación del servidor y del cliente. Por último, se tuvo en cuenta que la primera implementación (por ejemplo, la del registro del usuario) supusiera un esfuerzo mayor que su modificación o eliminación (ver *Tabla 1*).

Historias de usuario	Estimación Servidor	Estimación Cliente	Puntos de Historia	Horas Estimadas
1. Registro de un usuario	5	7	6	23
2. Baja de un usuario	3	4	4	8
3. Modificación de un usuario	3	6	5	15
4. Tipo de perfil de un usuario	2	4	3	9
5. Subir creación	10	10	10	48
6. Modificar creación	3	7	5	15
7. Eliminar creación	3	5	4	12
8. Buscar creaciones	8	9	9	
9. Clasificar creaciones	5	7	6	25
10. Tipos de licencia en creaciones	3	3	3	9
11. Buscar usuarios	8	9	9	37
12. Añadir amigos	10	10	10	48

13. Eliminar amigos	7	9	8	24
14. Enviar mensajes	10	10	10	
15. Comentar creaciones	8	10	9	27
16. Valorar creaciones	8	10	9	
17. Ver creaciones mejor valoradas	6	7	7	

Tabla 1. Puntos de Historia de las historias de usuario

Por último, mediante el método **MoSCoW** se priorizaron las tareas que se iban a llevar a cabo en el proyecto:

- Must (“Tiene que estar”):
  - Registro de un usuario.
  - Baja de un usuario.
  - Modificación de un usuario.
  - Subir creación.
  - Modificar creación.
  - Eliminar creación.
  - Añadir amigos.
  - Eliminar amigos.
  - Buscar usuarios.

Estas tareas son esenciales para crear un prototipo de una red social funcional, pues permiten las opciones básicas de registrar usuarios, compartir creaciones y agregar amigos.

- Should (“Debería estar”):
  - Tipos de licencia en creaciones.
  - Clasificar creaciones.
  - Buscar creaciones.
  - Comentar creaciones.

Es muy importante que estas funcionalidades estén presentes en la aplicación, ya que es muy práctico que los artistas puedan clasificar y buscar las obras por categorías, así como comentar las creaciones de otros usuarios. También es importante que puedan seleccionar la licencia que va a tener cada una de sus creaciones para saber el uso que se le pueden dar.

- Could (“Podría estar”):
    - Valorar creaciones.
    - Ver creaciones públicas mejor valoradas.
    - Tipo de perfil de un usuario.

Estos requisitos no son obligatorios pero sería interesante que estuviesen.

- Won´t (“No estarán”):
    - Enviar mensajes.

Este requisito es descartado a priori pero podría ser implementado en el futuro. Ha sido excluido debido a que los usuarios podrían poner en la descripción de su perfil sus datos de contacto (otras redes sociales o correo electrónico).

2.4.1.- Planificación inicial

La planificación inicial de las tareas fue la siguiente:

- **Mes 1:** Inicio del proyecto, planificación, organización inicial de la memoria e implementación inicial de los entornos de desarrollo.
- **Mes 2, 3 y 4:** Elaboración progresiva de la memoria según se iban tomando decisiones y obteniendo resultados durante la implementación del proyecto (historias de usuario).
- **Mes 5:** Últimos retoques del proyecto y la memoria, y corrección de posibles errores.

En la *Figura 4* se presenta la planificación temporal del proyecto mediante un diagrama de Gantt, en el que se estimó una duración aproximada de dos semanas por iteración.

	Iteraciones	Iteración 1	Iteración 2	Iteración 3	Iteración 4	Iteración 5	Iteración 6	Iteración 7	Iteración 8	Iteración 9	Iteración 10
Códigos Historias Usuario	Nombre de la tarea	01.02.2021	15.02.2021	01.03.2021	15.03.2021	01.04.2021	15.04.2021	01.05.2021	15.05.2021	01.06.2021	30.06.2021
	Estudio de tecnlogías										
	Historias de Usuario										
	Memoria										
	Imp. Servidor										
1	Registro										
	Login										
	Imp. GBD										
	Perfil										
3 y 4	Modificar Perfil										
2	Eliminar Cuenta										
	Listas Amigos										
	Logout										
5	SubirCreaciones										
6	Modificar Creaciones										
7	Eliminar Creaciones										
9	Categorias										
10	Licencias										
12	Añadir Amigos										
13	Eliminar Amigos										
15	Comentarios										
	Swagger										
11	Usuarios Buscados										

Figura 4. Diagrama de Gantt para la planificación temporal del proyecto.

Con el objetivo de familiarizarse con las tecnologías Ionic[1], Angular[2] y Spring[3], las dos primeras semanas del proyecto se dedicaran al estudio de estas herramientas. Al mismo tiempo, durante la primera semana, se definirán las historias de usuario y se implementara la base de datos en la parte del servidor. Además, en la segunda semana, se realizará la implementación del registro de usuarios.

Una vez llevados a cabo el registro y las bases de datos, se comenzará con la implementación del login en el servidor y con el registro de usuarios en el lado del cliente. Las tareas restantes serán implementadas por orden (subir creación, visualizar creación, modificar creación…), primero en el servidor, y, posteriormente, en el cliente.

Por último, hay que indicar que la memoria se realizará de forma progresiva durante todo el proyecto, para ir documentando los tiempos de inicio de tareas y finalización, referencias bibliográficas, etc.; y, de este modo, llevar un correcto control del proyecto. Además al finalizar cada iteración, se realizará la correspondiente comprobación mediante el backlog del proyecto.

[1] https://ionicframework.com/
[2] https://angular.io/docs
[3] https://www.baeldung.com/

2.4.2.- Evolución de la planificación inicial

En este apartado se analizará la evolución de la planificación respecto al planteamiento inicial mostrado en el diagrama de Gantt.

En primer lugar, el estudio de las tecnologías Angular e Ionic requirió más tiempo del estimado, pues hubo que investigar cómo se debía realizar la implementación del cliente de forma que conectara correctamente con el servidor. Por tanto, no se pudo comenzar al mismo tiempo con la realización de las historias de usuario.

Más adelante, surgieron problemas y errores en algunos procesos, que se describen a continuación:

En cuanto al registro de usuarios en la aplicación, se tuvo que cambiar el mecanismo que se estaba utilizando para gestionar la autenticación del login (que en un principio se implementó con un token) por el uso de cookie de sesión. También surgieron errores con el interceptor en la parte del cliente (pues el usuario no llegaba a registrarse correctamente y el navegador pedía de nuevo las credenciales), lo que fue solucionado después de realizar una documentación sobre varios ejemplos de http interceptor y sobre su funcionamiento.

Por otro lado, al no tener en cuenta aspectos sobre tercera forma normal (3FN) en el diseño inicial del modelo de datos, hubo algunos problemas relacionados con la persistencia de clases en la base de datos entre las distintas entidades. Esto se solventó eliminando atributos de dichas clases y apoyándose en una clase auxiliar creada solamente para obtener ciertos valores de la base de datos en unas peticiones concretas.

Asimismo, se tuvieron que resolver algunos errores relacionados con los procesos de subida y visualización de creaciones. Por ejemplo, las creaciones del usuario en la aplicación móvil no se subían correctamente al servidor (debido a que había que asignarle un tiempo concreto de subida) y no se podían visualizar y editar las imágenes en la parte del cliente (debido a que no se conocía como pasar objetos entre vistas en Angular).

Por último, también se encontraron dificultades en la exportación de la aplicación a Android. Uno de los fallos se corrigió configurando correctamente un parámetro del CORS en el servidor. CORS (Cross-Origin Resource Sharing) es “un mecanismo o política de seguridad que permite controlar las peticiones

HTTP asíncronas que se pueden realizar desde un navegador a un servidor con un dominio diferente de la página cargada originalmente. Este tipo de peticiones se llaman peticiones de origen cruzado (cross-origin)" [11].

Estos problemas obligaron a reorganizar las tareas en algunas iteraciones y a reconsiderar la priorización de ciertas historias de usuario o su inclusión/descarte del backlog para poder ajustarse a las restricciones temporales del proyecto. Aquellas historias de usuario que fueron modificadas son las siguientes:

12. Añadir amigos

- Descripción. Como usuario quiero añadir a otro usuario de la aplicación como amigo para poder visualizar sus creaciones o mandarle un mensaje.
- Criterios de aceptación: Que en la aplicación haya una opción para añadir a usuarios como amigos.
- Descomposición en tareas:
  - Generar un botón para añadir amigos en la aplicación.
  - Enviar mediante una petición POST al servidor los datos de la petición y el usuario.
  - Incrementar los amigos del usuario.

13. Eliminar amigos

- Descripción. Como usuario quiero eliminar a un usuario de la aplicación como amigo para quitarlo de mi círculo.
- Criterios de aceptación: Que en la aplicación haya una opción para eliminar usuarios para dejar de ser amigos.
- Descomposición en tareas:
  - Generar un botón para eliminar amigos en la aplicación.
  - Enviar los datos de la petición y el usuario mediante una petición DELETE al servidor.
  - Decrementar los amigos del usuario.

Las historias de usuario 12 (añadir amigos) y 13 (eliminar amigos) se modificaron y cambiaron por las historias 18 (seguir usuarios) y 19 (dejar de

seguir usuarios), ya que estas opciones son más utilizadas actualmente en las redes sociales, como, por ejemplo, Instagram o Twitter.

18. Seguir usuarios

- Descripción. Como usuario quiero seguir a otro usuario de la aplicación para poder visualizar e interactuar con las creaciones de su perfil.
- Criterios de aceptación: Que en el perfil del usuario haya un botón para seguir usuarios.
- Descomposición en tareas:
    - Generar un botón para seguir usuarios en la aplicación.
    - Enviar los datos de la petición y el usuario mediante una petición POST al servidor.
    - Incrementar el número de seguidos del usuario y el de seguidores del usuario al que se le acaba de seguir.

19. Dejar de Seguir Usuarios

- Descripción. Como usuario quiero dejar de seguir a un usuario de la aplicación para que no aparezca en mi lista de seguidos.
- Criterios de aceptación: Que en el perfil del usuario haya un botón para dejar de seguir usuarios en caso de que se les siga previamente.
- Descomposición en tareas:
    - Generar un botón para dejar de seguir usuarios en la aplicación.
    - Enviar los datos de la petición y el usuario mediante una petición DELETE al servidor.
    - Decrementar el número de seguidos del usuario y el de seguidores del usuario al que se le acaba de dejar de seguir.

Además, se añadió una nueva historia de usuario para permitir la visualización de las creaciones dentro de las distintas categorías en la aplicación:

20. Visualizar categorías

- Descripción. Como usuario quiero visualizar las creaciones que pertenecen a las distintas categorías artísticas.

- Criterios de aceptación: Que en la aplicación haya una sección para seleccionar cualquier categoría con el fin de visualizar las creaciones que pertenezcan a dicha categoría.
- Descomposición en tareas:
    - Generar un banner/carrusel con distintas categorías de obras artísticas.
    - Obtener las creaciones pertenecientes a una categoría dada mediante una petición GET al servidor.

En la *Figura 5*, se puede observar el diagrama *Burndown*, en el que se muestra el desempeño final respecto a la planificación inicial, de acuerdo a los puntos de historia asignados a las historias de usuario. A pesar de que dichas historias de usuario se completaron en las 10 iteraciones que se muestran (de dos semanas de duración cada una), se tuvo que continuar con el proyecto después de ese plazo para resolver algunos fallos de la aplicación y terminar la documentación.

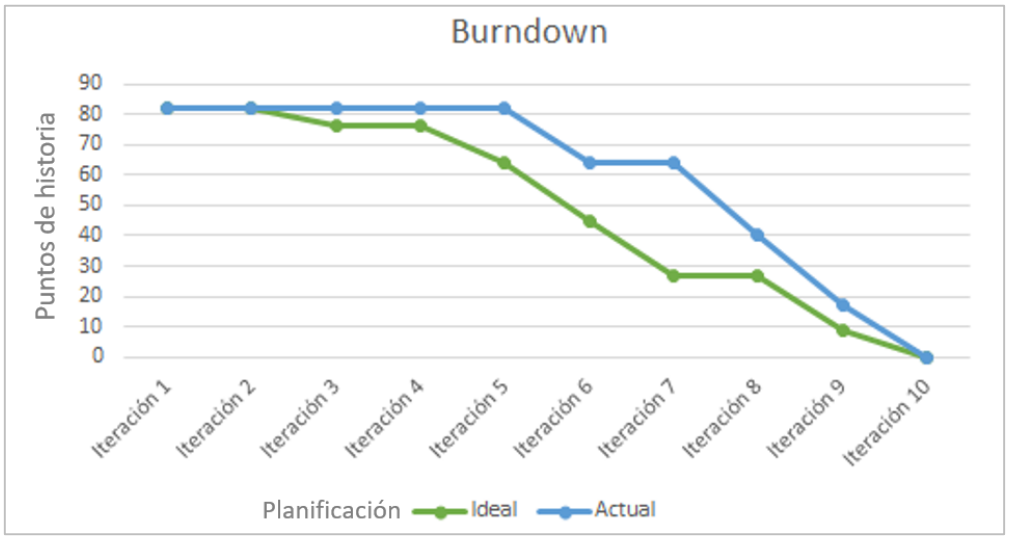

Figura 5. Gráfica Burndown.

2.5.- Estudio de viabilidad

Para explicar la viabilidad del proyecto, se van a detallar los costes que supone su realización a nivel de hardware, software y mano de obra:

- En cuanto a la parte **hardware**:

- El equipo que se ha utilizado es un Asus x570zd y tiene las siguientes especificaciones técnicas que permiten obtener un buen rendimiento para el proyecto:
  - Procesador AMD Quad Core Ryzen 5 2500U hasta 3.6 GHz.
  - 8GB de memoria RAM.
  - SSD de 256 GB.
  - Tarjeta Gráfica Nvidia GTX 1050 de 2 GB.
  - Sistema Operativo Windows 10 Home 64 bits.

  El precio de este equipo ha sido de 540 €. Haciendo una estimación de un tiempo de vida útil de unos 6 años, el coste al mes sería de 7,5€. Por lo tanto, el coste total para los 6 meses en los que se va a realizar el proyecto sería de 37,5€.
- La infraestructura de red es de fibra simétrica de la compañía MásMóvil con un precio mensual de 49,90€ al mes, lo que tiene un coste total para el proyecto de 249,5 €.

- Respecto al **software**, en la *Tabla 2* se muestran los distintos programas que se han utilizado, todos ellos gratuitos, por lo que el coste en esta parte ha sido de 0€.

Programa	Uso	Licencia	Coste
Visual paradigm	Realización de los diagramas del proyecto	Proporcionada por la Universidad	0
Netbeans	Implementación del servidor	Gratuita	0
Ionic	Implementación del cliente	Gratuita	0
Visual studio code	Implementación del cliente	Gratuita	0
Android Studio	Exportación de la apk a android	Gratuita	0
Postman	Realización de pruebas con el servidor	Gratuita	0
Microsoft Office	Realización del documento de la memoria	Gratuita con la compra del equipo	0
Google Drive	Guardado de una copia de seguridad	Gratuita	0

	de la memoria y desarrollo en Google Docs		
GitLab	Guardado de copias de seguridad	Gratuita	0

Tabla 2. Costes de software.

- Por último, respecto a los costes de **mano de obra** previstos, el precio por hora de un analista programador, según los datos que se reflejan en el portal de empleo Indeed [12], es de 14,36 €. Por tanto, suponiendo que el proyecto tiene una duración mínima de 300 horas, el coste de mano de obra sería de 4308 €, que puede aumentar en el caso de que fueran necesarias más horas de trabajo.

En la *tabla 3*, se muestran, a modo resumen, los costes totales del desarrollo del proyecto, tanto en hardware, software como mano de obra:

	Hardware	Software	Mano de obra	Total
Coste	287 €	0€	4308 €	4595 €

Tabla 3. Costes Totales

2.6.- Modelo de dominio

Con el objetivo de detallar las entidades principales que componen el sistema, se realizó un modelo de dominio (ver *Figura 6*), es decir, “un modelo conceptual de todos los temas relacionados con un problema específico. En él se describen las distintas entidades, sus atributos, papeles y relaciones, además de las restricciones que rigen el dominio del problema” [13].

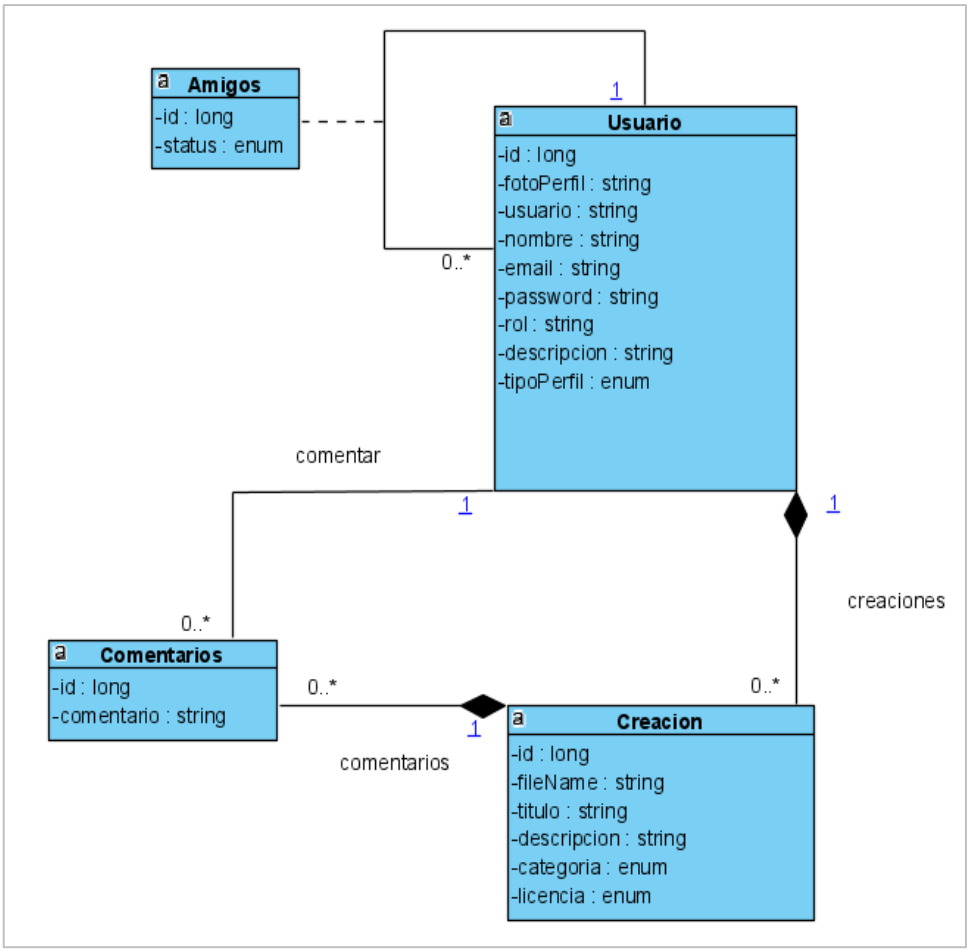

Figura 6. Modelo de dominio.

Como se puede observar en el diagrama, las clases que componen el sistema son las siguientes:

- **Clase *Usuario*.** Clase encargada de almacenar todos los datos del usuario.
    - Atributos:
        - ***id*.** Atributo único para identificar al usuario y gestionado por el propio sistema.
        - ***fotoPerfil*.** Atributo para guardar el nombre y extensión de la foto de perfil subida por el usuario.
        - ***password*.** Atributo para almacenar la contraseña del usuario.
        - ***email*.** Atributo para almacenar el email del usuario.
        - ***nombre*.** Atributo para almacenar el nombre del usuario.
        - ***descripción*.** Atributo para almacenar la descripción del usuario.

- ***rol*.** Atributo para gestionar el rol de un usuario.
- ***tipoPerfil*.** Atributo para gestionar el perfil del usuario (Público o Privado).

- **Clase *Creacion*.** Clase empleada para almacenar los datos de las creaciones de los usuarios.
  - Atributos:
    - ***id*.** Atributo único para identificar la creación y gestionado por el propio sistema.
    - ***fileName*.** Atributo para guardar el nombre y extensión de la creación subida por el usuario.
    - ***titulo*.** Atributo para almacenar el título de la creación.
    - ***descripcion*.** Atributo para almacenar la descripción de la creación.
    - ***categoria*.** Atributo para gestionar la categoría a la que pertenece la creación del usuario. Las categorías son: Acrilico, Acuarela, Oleo, Rotulador, Carboncillo, Gouache, LapicesColores, LapizGrafito y Mixta.
    - ***licencia*.** Atributo para gestionar la categoría a la que pertenece la creación del usuario. Las licencias son: Reconocimiento, ReconocimientoNoComercial, ReconocimientoNoComercialCompartirIgual, ReconocimientoNoComercialSinObraDerivada, ReconocimientoCompartirIgual, y ReconocimientoSinObraDerivada.

- **Clase *Amigos*.** Clase encargada de gestionar la relación “reflexiva” entre usuarios (seguidores y seguidos).
  - Atributos:
    - ***id*.** Atributo único para identificar la relación entre usuarios y amigos, y gestionado por el propio sistema.
    - ***status***. Atributo para gestionar el estado de la petición de amistad en un momento dado. Dependerá del tipo de perfil del usuario. Los estados son: ENVIADA, PENDIENTE, ACEPTADA y RECHAZADA.

- **Clase *Comentarios*.** Clase encargada de gestionar los comentarios en las distintas creaciones.
    - Atributos:
        - ***id*.** Atributo único para identificar el comentario de una creación y gestionado por el propio sistema.
        - ***comentario*.** Atributo para almacenar el comentario de la creación.

En cuanto a las relaciones existentes entre las clases anteriores, la clase *Usuario* y la de *Amigos* están conectadas mediante una relación de asociación, ya que, en el caso de que un usuario fuera eliminado de la aplicación, los demás usuarios seguirían existiendo.

Por otro lado, la relación entre *Usuario* y *Creación* es una asociación de composición, ya que, si el usuario fuera borrado, todas sus creaciones serían igualmente eliminadas. Además, como se puede observar en el diagrama, un *Usuario* puede estar a su vez compuesto por cero o varios objetos de la clase *Creación*. Esta cardinalidad se debe a que un usuario registrado puede utilizar la aplicación sin necesidad de subir alguna creación, ya que dispone de otras funcionalidades. Esto mismo ocurre con la relación amigos respecto a los usuarios seguidos y seguidores.

Por último, la relación entre *Creación* y *Comentarios* es también una composición, pues una creación puede tener cero, uno o varios comentarios, y, en el caso de que fuera eliminada, todos los comentarios también desaparecerían.

2.7.- Análisis de la interfaz

En este apartado se va a explicar la identificación de los distintos tipos de usuarios, así como los aspectos generales que se han tenido en cuenta al diseñar la interfaz.

Respecto a la **identificación de los usuarios**, aunque este proyecto actualmente no lo contempla, se ha planteado que en un futuro la aplicación pueda tener varios tipos de usuarios. De este modo, el atributo rol permitiría clasificar a los usuarios en administradores y en usuarios sin privilegios.

Por otro lado, para **diseñar una interfaz** sencilla que facilitase la usabilidad de la aplicación, se tuvieron en cuenta los siguientes aspectos:

- Utilizar iconos que representasen correctamente objetos y funcionalidades, haciendo uso de buenas metáforas. Para ello, se aplicaron los iconos que ofrece Ionic por defecto. De este modo, se dispone de un elemento visual para la realización de cada acción y el aprendizaje de la aplicación resulta más fácil.
- Emplear las reglas de Marcus para el uso de colores [14], utilizando un tono azul para el fondo, manteniendo un número pequeño de colores presentes en la aplicación, y empleando un rojo brillante para alertar al usuario de la ocurrencia de un error, por ejemplo, en el caso de que dejase un campo vacío o incorrecto en los formularios.
- También se han seguido las reglas de Murch [15], evitando el despliegue simultáneo de los colores altamente saturados que sean espectralmente extremos y descartando el color azul puro para el texto, figuras pequeñas y líneas delgadas.

3.- Diseño

3.1.- Diagrama Entidad-Relación

Para realizar el diagrama entidad-relación de la base de datos (ver *Figura 7*), que muestra cómo se relacionan entre sí las entidades dentro del sistema [16], se hizo uso de un ORM (Object Relational Mapping) o Mapeador de Objetos Relacionales. Un ORM se refiere a “un modelo de programación que permite mapear las estructuras de una base de datos relacional (SQL Server, Oracle, MySQL, etc.), en adelante RDBMS (Relational Database Management System), sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones” [17].

Para implementar el ORM, se empleó la especificación de persistencia JPA (Java Persistence API), cuya principal ventaja es que requiere una información mínima (unas anotaciones en el código de Java) para generar el mapeado.

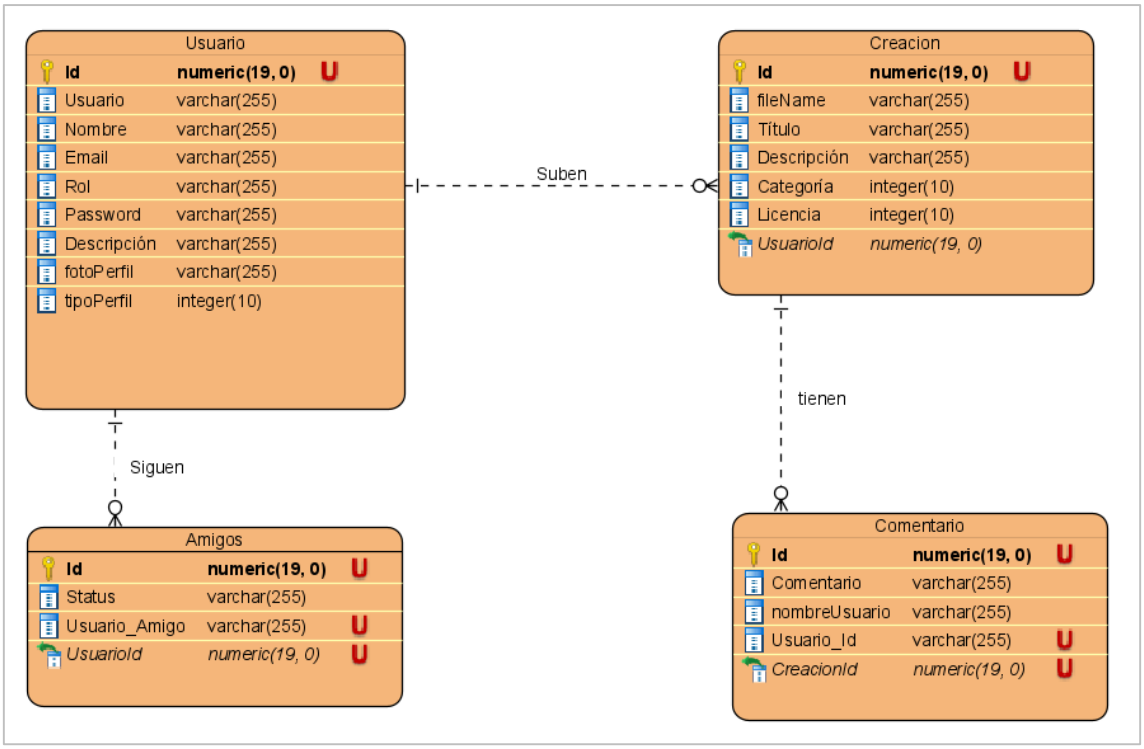

Figura 7. Diagrama Entidad Relación.

Como se puede observar en la *Figura 7*, la base de datos está en 3ª forma normal, por lo que no existen redundancias en la información de las tablas.

Respecto a las creaciones, se estableció un máximo de 10MB para cada una de las imágenes, las cuales eran almacenadas en un directorio externo del servidor (por ejemplo, en la raíz del sistema de archivos o en una unidad específica) referenciadas desde la base de datos. Hay que indicar que, para llevar a cabo pruebas con la subida de imágenes, se utilizó la base de datos de tal forma que se eliminase todo cada vez que se cerrase el servidor.

3.2.- Diagrama de Clases

En este apartado se mostrarán los distintos diagramas de clases, tanto del servidor, como del cliente, que se han realizado para la correcta implementación de la aplicación. Estos diagramas describen la estructura del sistema mostrando sus clases, atributos, operaciones y relaciones entre los objetos que lo componen [18].

3.2.1.- Diagrama de clases del servidor

En este apartado se mostrará la arquitectura de clases empleada en el servidor a través del diagrama de **servicios del sistema**; el de **Entidades y Objetos de Acceso a Datos**; y el de **Objetos de Transferencia de Datos, Excepciones y Formato**.

3.2.1.1- Servicios del sistema

Como se puede observar en la *Figura 8*, el sistema está compuesto por múltiples paquetes:

- ***REST*.** En este paquete se encuentra el controlador del sistema. En la clase del controlador del sistema se gestionan los métodos relacionados con las peticiones *http CRUD*[2]. Además dichos métodos hacen uso de los métodos necesarios para su correcto funcionamiento a través de la clase sistema.
- ***CORS*.** En este paquete se encuentra la clase que gestiona una política de seguridad al acceso de las distintas urls del sistema.
- ***SEGURIDAD*.** En este paquete se encuentran las clases que gestionan la autenticación del sistema y los mecanismos de seguridad de la aplicación.
- ***SWAGGER*.** En este paquete se encuentra la clase que permite hacer de forma "automática" la documentación del servidor y además permite que se pueda realizar distintas pruebas del funcionamiento, todo ello a través de una url accesible desde el navegador.
- ***SERVICIOS.*** En este paquete se encuentran las clases necesarias para gestionar correctamente algunos atributos de las entidades. Además la clase Datos tiene como utilidad recuperar datos del usuario cuando se realicen ciertas *peticiones http CRUD* al servidor.

Todos estos aspectos se detallarán con más profundidad en el apartado de implementación.

[2] Se trata del acrónimo de "Crear, Leer, Actualizar y Borrar" (del original en inglés: Create, Read, Update and Delete), se utiliza para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software [19].

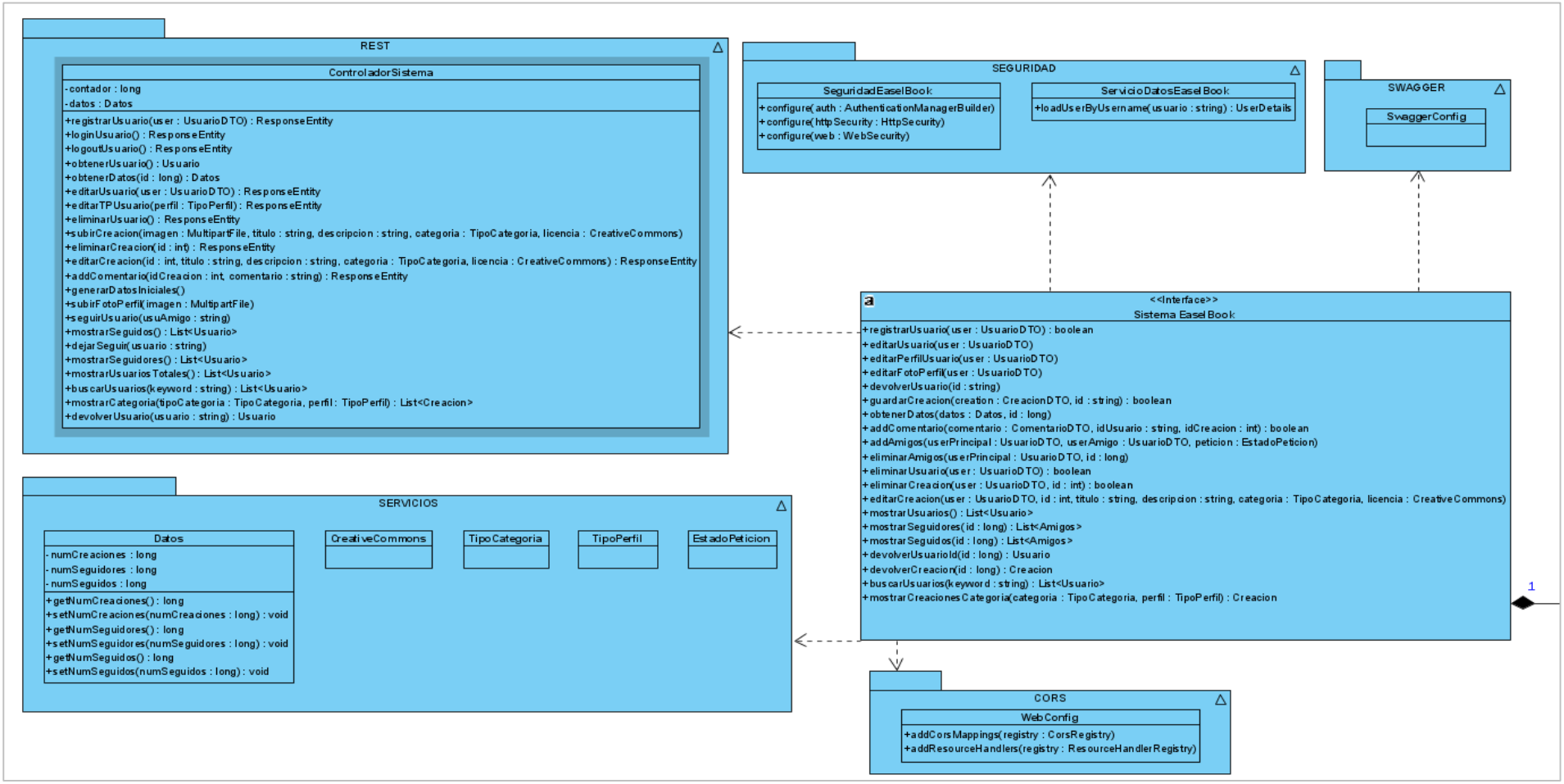

Figura 8. Diagrama de servicios del sistema.

3.2.1.2- Entidades y Objetos de Acceso a Datos

Como se puede observar en la *Figura 9*, se muestran las distintas entidades y sus DAOs dentro del sistema. DAO (Data Access Object), se trata del patrón arquitectónico “el cual permite separar la lógica de acceso a datos de los Business Objects u Objetos de negocio, de tal forma que el DAO encapsula toda la lógica de acceso de datos al resto de la aplicación” [20].

Los DAOs interactuarán con el ORM basado en JPA mediante elementos EntityManager.

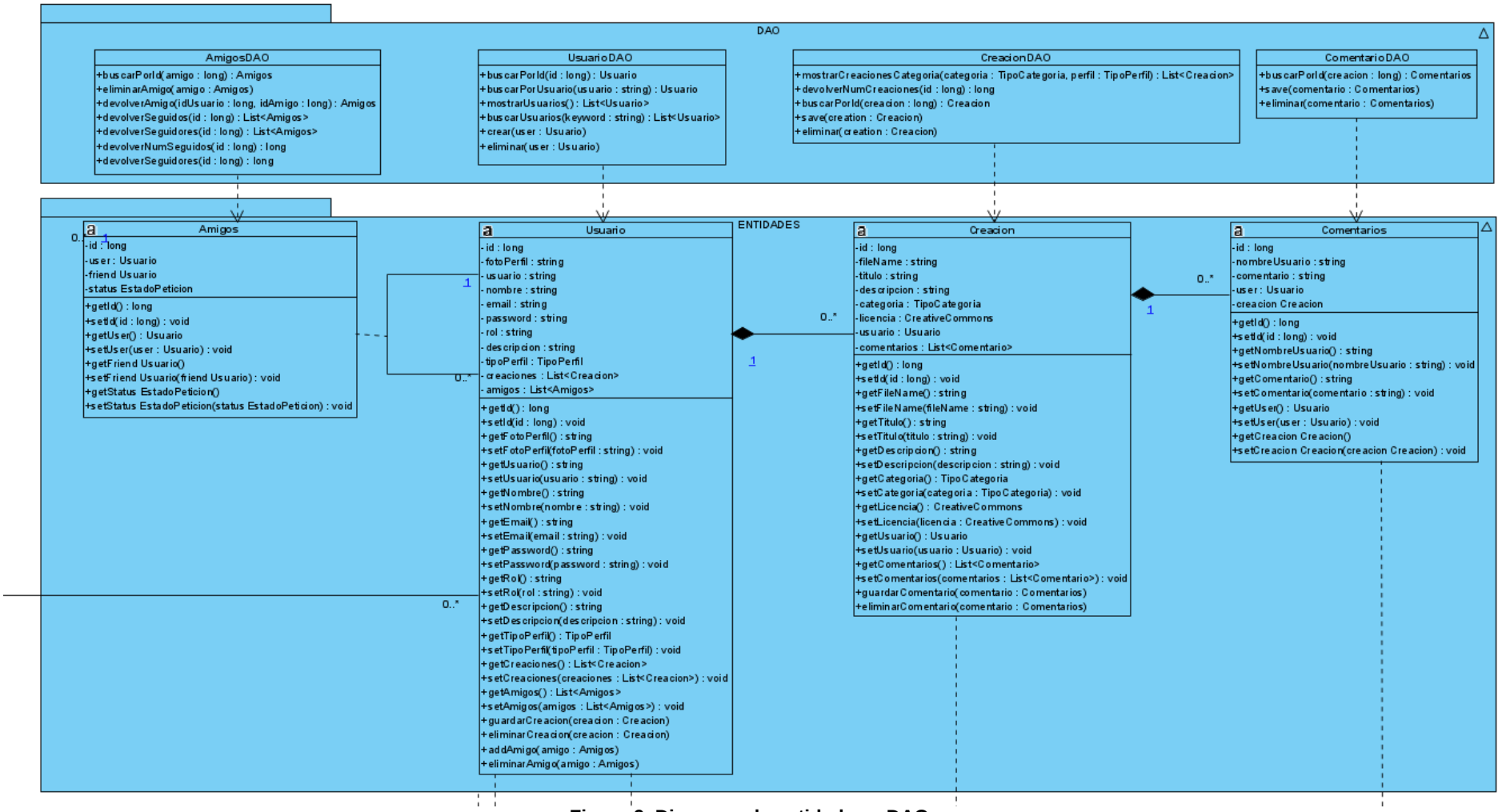

Figura 9. Diagrama de entidades y DAOs.

3.2.1.3- Objetos de Transferencia de Datos, Excepciones y Formato

Como se puede observar en la *Figura 10*, se muestran los distintos DTOs de las entidades anteriormente vistas, así como las clases formato y excepciones de dentro del sistema. Un DTO (Data Transfer Object) se trata del patrón el cual "tiene como finalidad la creación de objetos planos (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarlas en una única clase simple" [21].

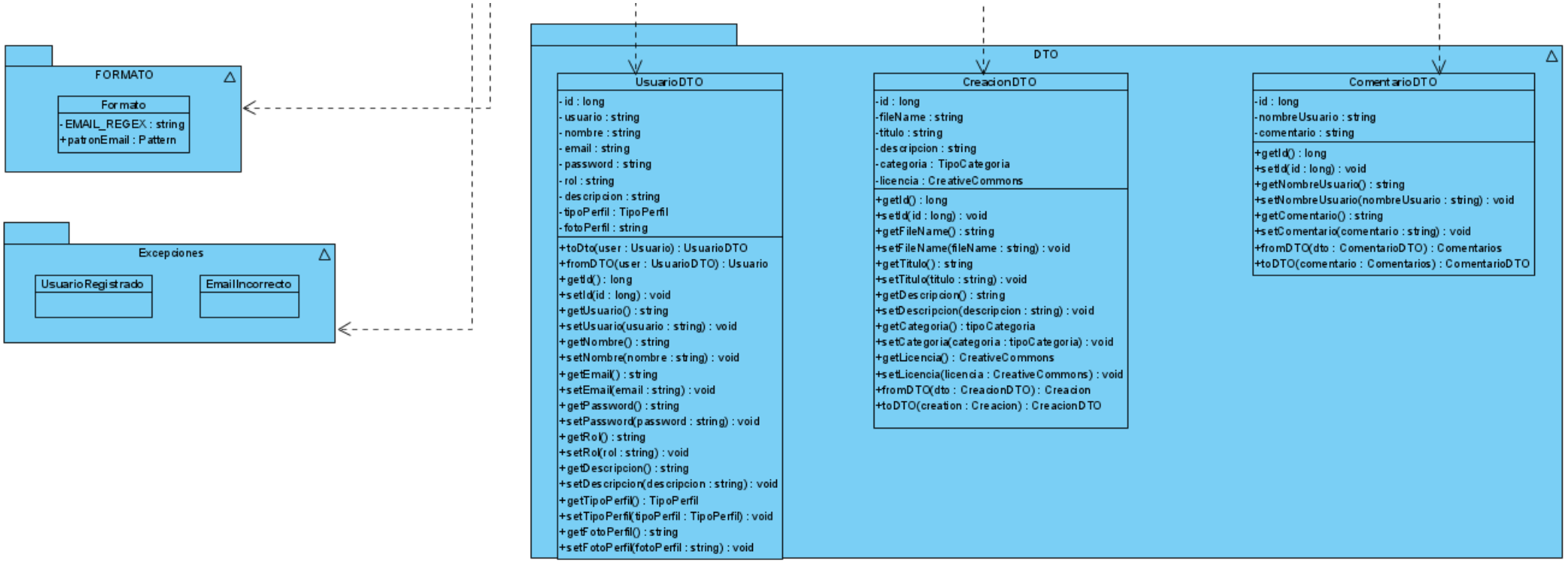

Figura 10. Diagrama de DTOs, excepciones y formato.

3.2.2.- Diagrama de clases del cliente

En este apartado se mostrará la arquitectura de clases empleada en el cliente a través del diagrama de la *Figura 11*.

También se van a comentar algunos aspectos más relevantes y/o consideraciones oportunas sobre decisiones de diseño o cambios que se han ido introduciendo a lo largo del tiempo. Además dicho desarrollo ha sido orientado a componentes que propone el propio framework Angular:

Respecto al diseño de la *Figura 11*. Diagrama de clases del cliente, se ha de aclarar que las distintas componentes están compuestas por varios archivos como pueden ser las más relevantes:

- Complemento.html: Es empleado para gestionar la vista html con la que el usuario interactúa.
- Complemento.scss: Es empleado para definir los estilos de las páginas html.
- Complemento.ts: Es empleado para definir la propia clase, aquí es dónde se gestionan las distintas funcionalidades y atributos.

Un aspecto relevante relacionado con el diseño es que se han agrupado todas las clases y atributos de las distintas entidades mencionadas anteriormente en el servidor, en una misma clase llamada *user*. Esto se debe a que solamente se utilizan para definir los atributos concretos para poder especificarlos en ciertos atributos de las distintas vistas o en las funciones, ya que en la parte cliente no va a ver relaciones entre ellas (de esta parte se encarga el servidor).

Otro aspecto relevante relacionado con el diseño es que se ha utilizado un paquete de Servicios para la implementación del servicio de autenticación y para gestionar mensajes de alerta para el caso en él que no haya creaciones en una categoría dada aún. Además para para el servicio de autenticación es necesario utilizar una clase que implementa un http interceptor, ya que permite gestionar correctamente las credenciales del usuario conectado en todo momento, así como los inicios de sesión o cierres de sesión.

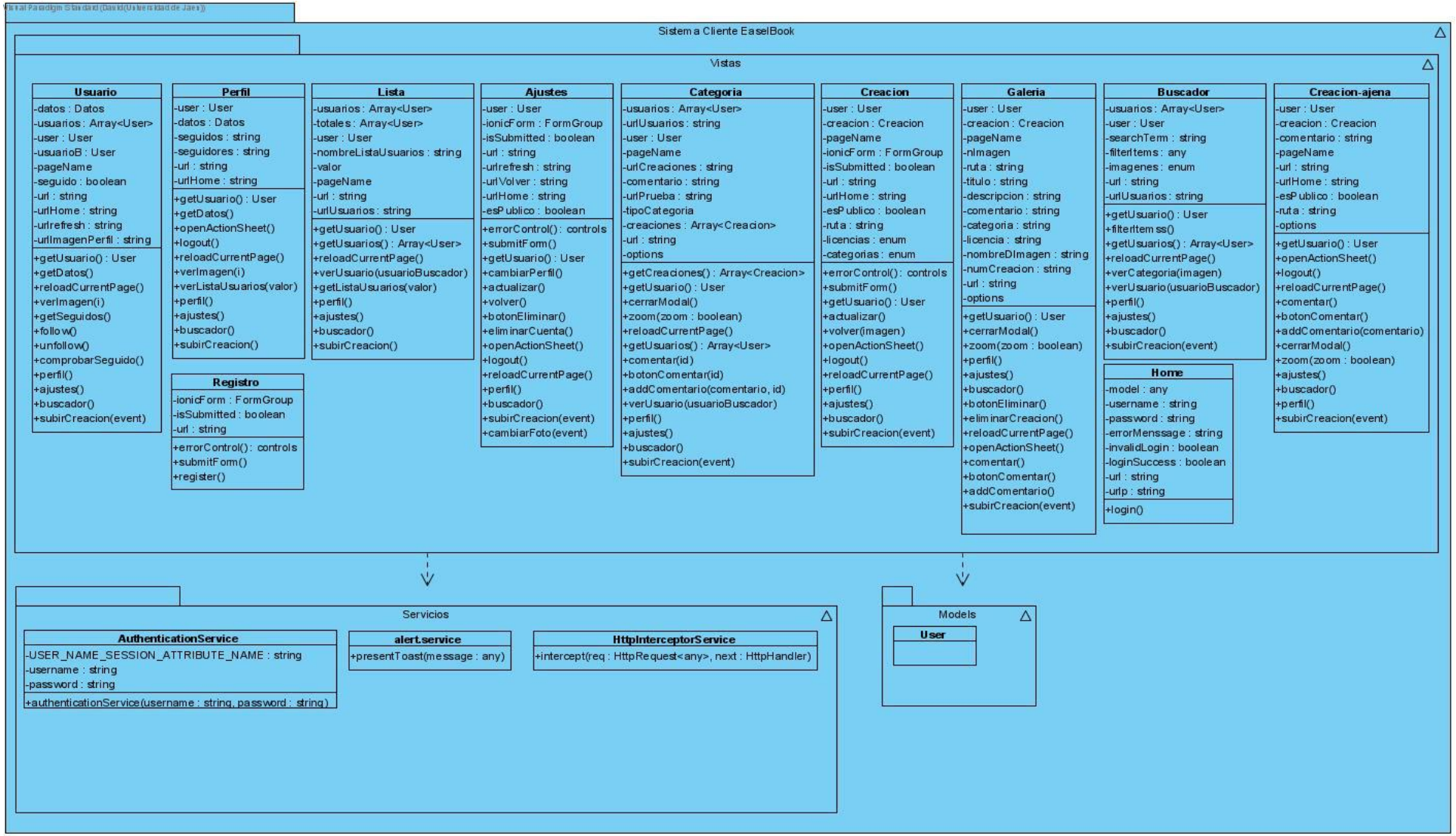

Figura 11. Diagrama de clases del cliente.

3.3.- Diagramas de secuencia

En este apartado se va a mostrar el funcionamiento interno de la aplicación estudiando cómo interaccionan los diferentes elementos del servidor y el cliente, representados en los diagramas de clases presentados en el apartado anterior.

Para ello se hará uso de unos diagramas de secuencia y a continuación se mostrarán los más relevantes para comprender el funcionamiento de los diagramas de clases propuestos, ya que el resto de funcionalidades tienen un funcionamiento similar.

Dichos diagramas son un tipo de diagrama que se emplean para modelar interacción entre objetos en un sistema según UML [22].

Para diferenciar los distintos elementos de la aplicación cliente y servidor en los diagramas de secuencia se han utilizado distintos colores: rojo para el servidor, amarillo para el cliente y azul para el usuario.

3.3.1 Búsqueda de usuarios

En este apartado se va a mostrar la interacción entre los objetos que intervienen en la búsqueda de usuarios a través de la *Figura 12*.

Dicha Figura representa cuando un usuario quiere realizar una búsqueda por nombre de usuario en la aplicación. Para ello el usuario introducirá el nombre del usuario a buscar.

El componente buscador es el encargado tanto de representar el buscador con el que el usuario interactúa, como de obtener previamente un listado de usuarios a través de una petición http GET al servidor.

Una vez recibida dicha petición del cliente en el servidor mediante el rest, este llamará al beans del sistema para su ejecución. En el beans se llamará al DAO correspondiente, el cual obtiene el listado de todos los usuarios registrados en la aplicación. Este listado se le devuelve al componente buscador de la aplicación cliente y dicho componente comprobará si el nombre que va introduciendo el usuario coincide con algún usuario registrado en la aplicación. En caso de que sí coincida, se le mostrará el usuario un listado con los usuarios que coinciden con el nombre introducido por el usuario.

Posteriormente el usuario que realiza la búsqueda podrá seleccionar un usuario del listado para consultar el perfil del usuario buscado. Además el

componente buscador se encarga de hacer una comprobación previa antes de visualizar el perfil del usuario buscado. Esta comprobación consiste en comprobar si el propio usuario ha pulsado en su mismo usuario a través del listado o si se trata de un usuario diferente. En caso de que sea el propio usuario se le redirigirá a su propio perfil, en caso contrario se le redirigirá al perfil del usuario buscado.

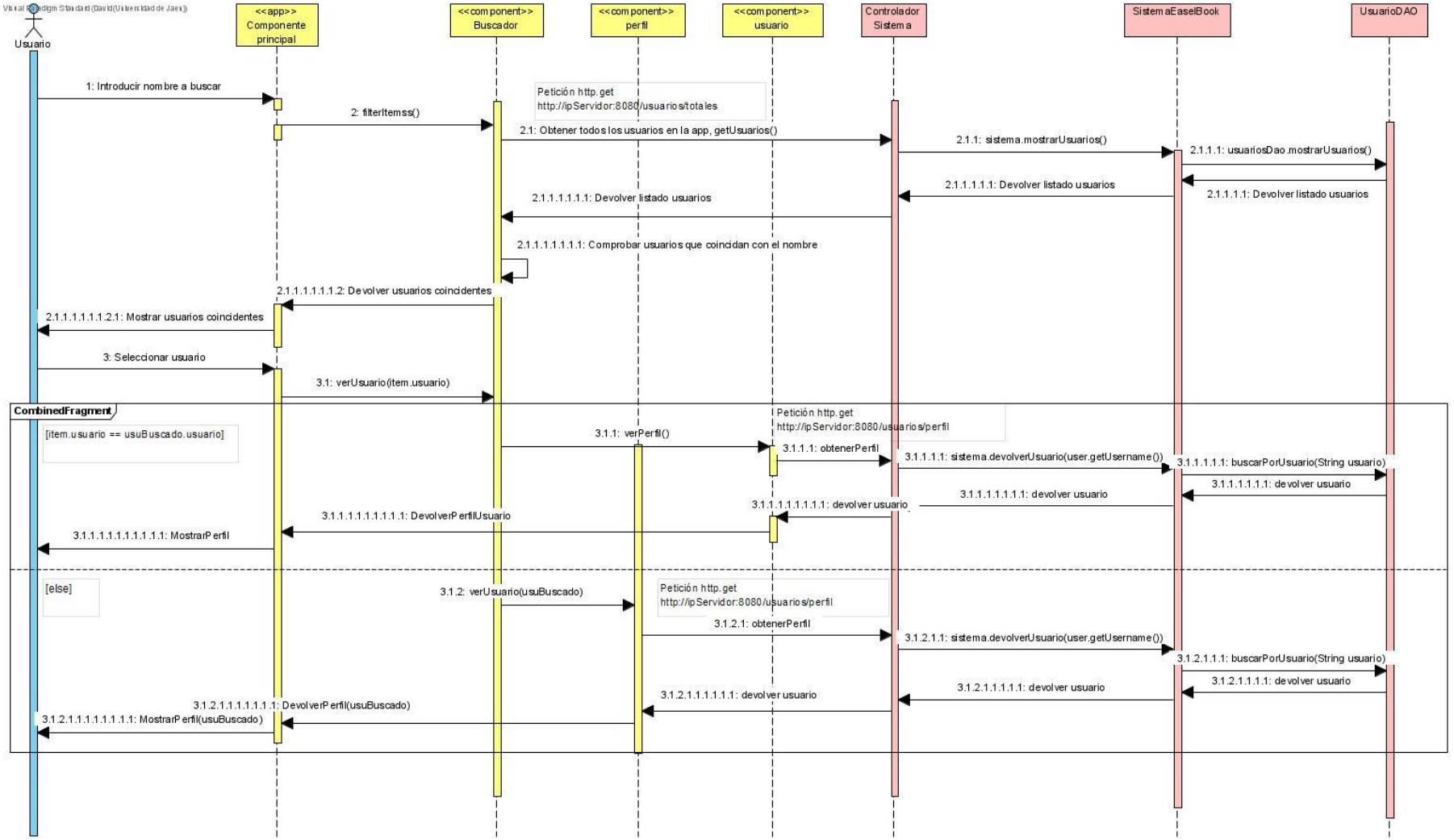

Figura 12. Diagrama de secuencia de búsqueda de usuarios.

3.3.2 Seguir y dejar de seguir usuarios

En este apartado se va a mostrar la interacción entre los objetos que intervienen cuando un usuario sigue y/o deja de seguir a un usuario a través de las *Figuras 13 y 14*.

Cuando un usuario accede al perfil de otro usuario registrado en la aplicación le aparecerá un botón de seguir o de dejar de seguir a un usuario (en caso de que lo siga previamente), esto es posible mediante una comprobación previa que hace el sistema para comprobar si el usuario lo sigue o no.

A continuación se van a explicar las dos posibles situaciones del funcionamiento de esta interacción:

- En el caso de que el usuario no siga al usuario al que ha accedido a su perfil, este pulsará un botón de seguir a dicho usuario en la aplicación, a través de ese botón, el componente usuario de la aplicación enviará una petición http POST al sistema, el cual a través del DAO correspondiente realizará la creación de una relación entre los usuarios. De esta forma para el usuario seguido aumentan sus usuarios seguidores y para el otro usuario, aumentan sus usuarios seguidos.
- En el caso de que el usuario siga previamente al usuario al que ha accedido a su perfil, este pulsará un botón de dejar de seguir a dicho usuario en la aplicación, a través de ese botón, el componente usuario de la aplicación enviará una petición http DELETE al sistema, el cual a través del DAO correspondiente realizará la eliminación de la relación entre los usuarios. De esta forma para el usuario dejado de seguir, disminuyen sus usuarios seguidores y para el otro usuario, disminuyen sus usuarios seguidos.

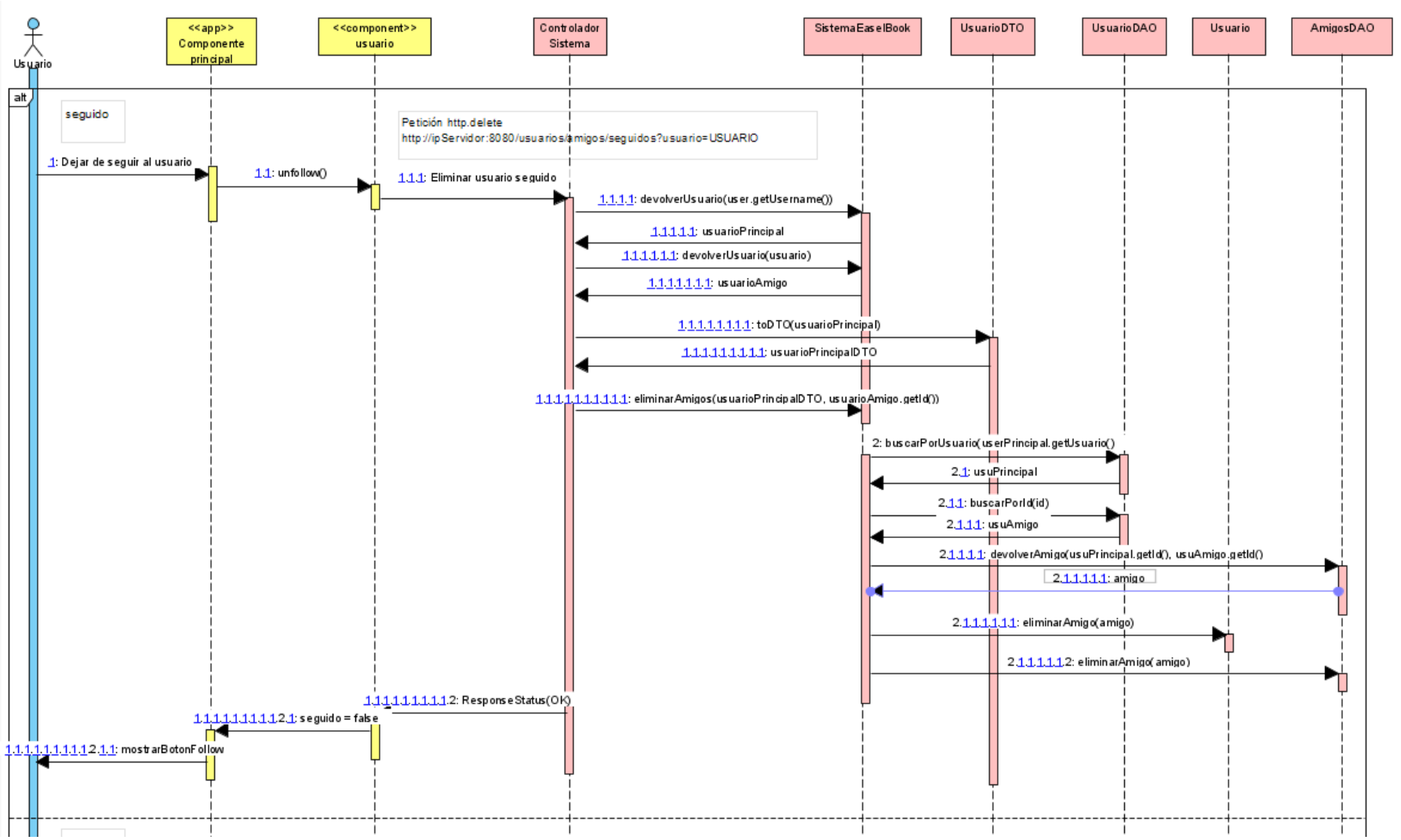

Figura 13. Diagrama de secuencia de seguir y dejar de seguir usuarios.

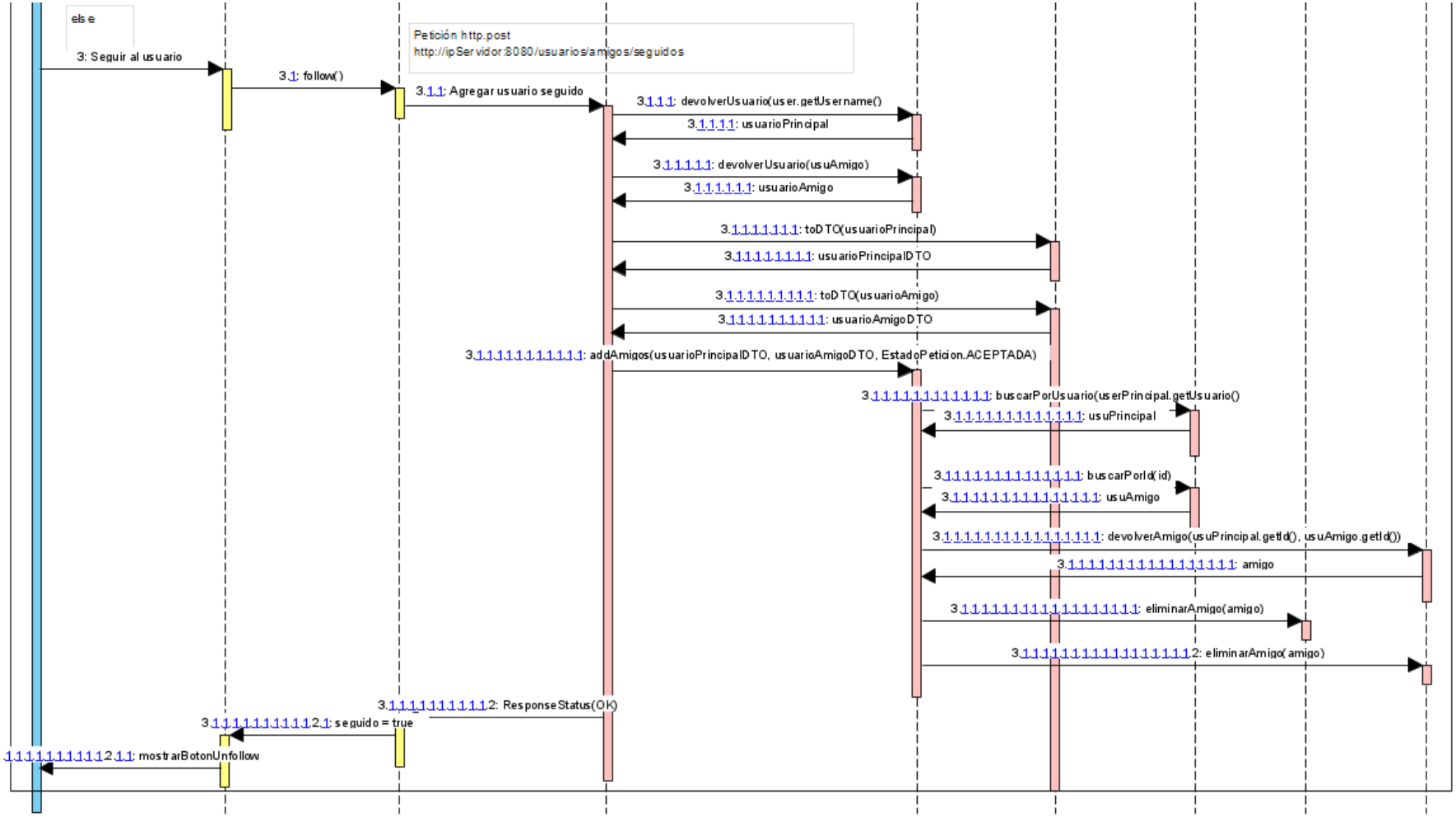

Figura 14. Diagrama de secuencia de seguir y dejar de seguir usuarios.

3.3.3 Editar creación

En este apartado se va a mostrar la interacción entre los objetos que intervienen cuando un usuario procede a editar una creación a través de las *Figuras 15 y 16*.

Para ello el usuario deberá de estar ubicado en su perfil. A continuación el usuario selecciona una imagen y procede a visualizarla a través del componente galería. Además mediante este componente el usuario puede realizar zoom en la propia creación, también puede comentarla, eliminarla o editarla.

Para editar la creación el usuario pulsara en la opción editar creación a través de un botón elipsis. Al pulsar en dicha opción se redirigirá al usuario al componente creación. Dicho componente permite el usuario modificar los atributos de la creación mediante un formulario.

Si el usuario rellena el formulario y guarda los cambios, realiza una petición http PUT al sistema para modificar los datos de dicha creación a través del DAO correspondiente. Si por el contrario el usuario borra algún dato del formulario, lo deja vacío y pulsa en guardar cambios, la aplicación le muestra un mensaje de error al usuario y no se guardarán los cambios.

Por último destacar que el usuario puede volver en cualquier momento al componente anterior mediante el botón volver.

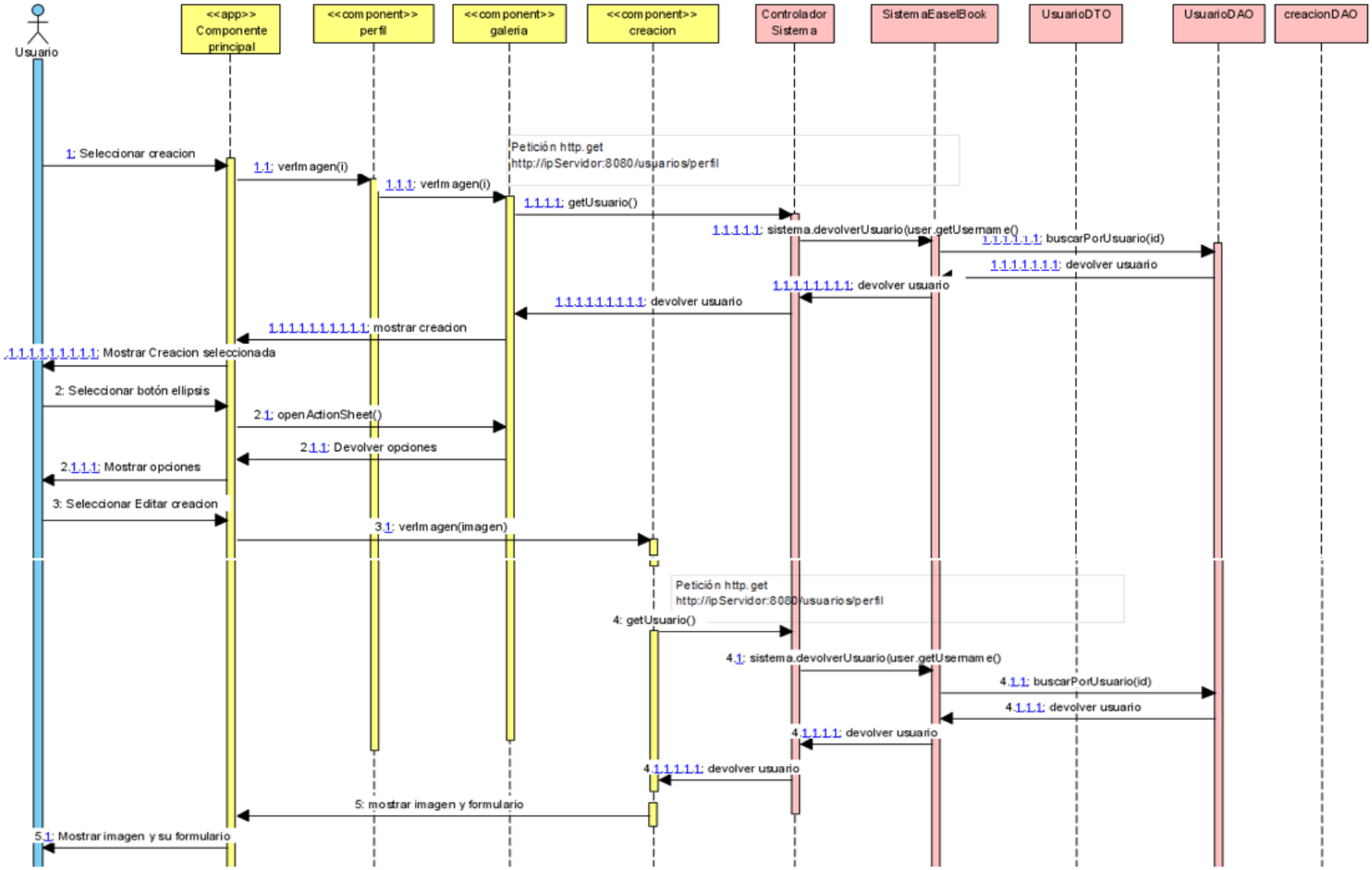

Figura 15. Diagrama de secuencia editar creación.

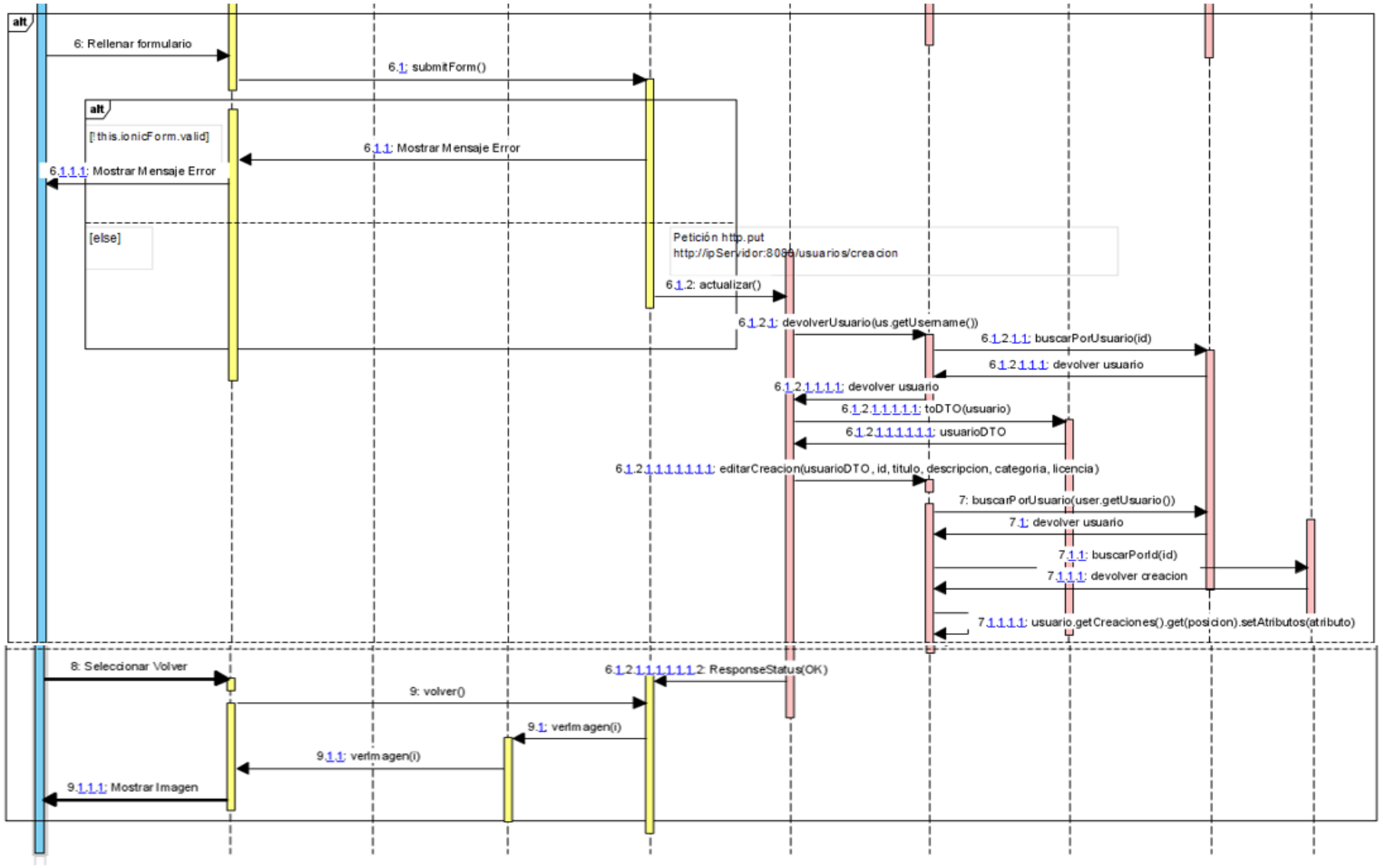

Figura 16. Diagrama de secuencia editar creación.

3.4.- Diseño de la Interfaz

En este apartado se van a determinar los detalles de las interfaces de comunicación con cada una de las aplicaciones que componen el sistema propuesto: Api/Rest e interfaz gráfica de la aplicación móvil.

3.4.1 Diseño Api/Rest

En este apartado se mostrará el diseño del Api/Rest. Una API REST define un conjunto de funciones que los desarrolladores pueden utilizar para realizar solicitudes y recibir respuestas a través del protocolo HTTP, como GET y POST.

Además debido a que la API REST usa HTTP, esto implica que puedan ser utilizados por prácticamente cualquier lenguaje de programación y son fáciles de probar un requisito de una API REST es que el cliente y el servidor sean independientes entre sí, lo cual permite codificarlo en cualquier idioma y mejorar al soportar la longevidad y evolución) [23].

Por lo tanto, para el servidor de la aplicación se detallan a continuación los diferentes recursos y sus características para poder interactuar con él. Mostrando así las urls de los distintos métodos de acceso: GET, POST, PUT, DELETE.

Para dichos métodos ha sido fácil poder mostrar su funcionalidad gracias a un plugin de spring conocido como swagger.io[3] que permite documentar automáticamente el Api/Rest.

3.4.1.1- Peticiones GET

En este apartado se muestran las distintas peticiones HTTP GET a través de la documentación generada por el plugin de swagger.io.

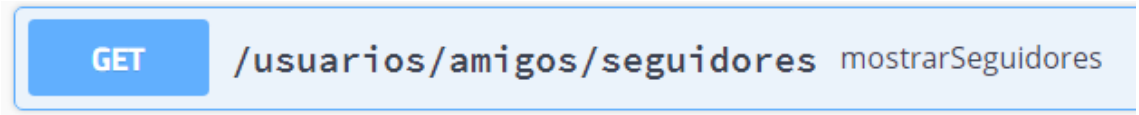

Figura 17. Petición HTTP GET mostrar usuarios seguidores.

Descripción: La petición de la *Figura 17* permite obtener los usuarios seguidores de un usuario.

Parámetros de entrada: Sin parámetros de entrada.

[3] https://swagger.io/

Datos de salida:

- Código 200, OK y muestra los usuarios seguidores del usuario.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

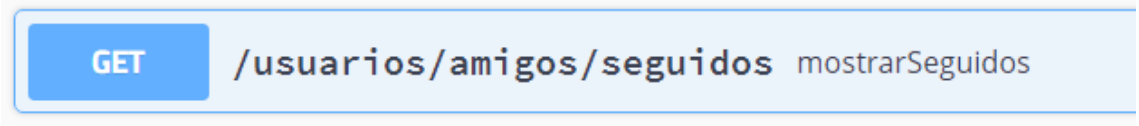

Figura 18. Petición HTTP GET mostrar usuarios seguidos.

Descripción: La petición de la *Figura 18* permite obtener los usuarios seguidos de un usuario.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK y muestra los usuarios seguidores del usuario.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

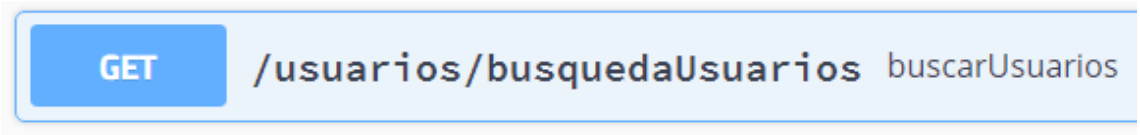

Figura 19. Petición HTTP GET buscar usuarios.

Descripción: La petición de la *Figura 19* permite obtener los usuarios cuyo nombre de usuario coincide con el parámetro de entrada keyword. (Esta funcionalidad está presente en la aplicación pero al final el buscador se implementó de manera distinta, como se ha comentado previamente en el apartado 3.3.1 Búsqueda de usuarios).

Parámetros de entrada: keyword de tipo string.

Datos de salida:

- Código 200, OK y muestra los usuarios cuyo nombre de usuario coincide con el parámetro de entrada keyword.
- Código 401, Unauthorized.
- Código 403, Forbidden.

- Código 404, Not Found.

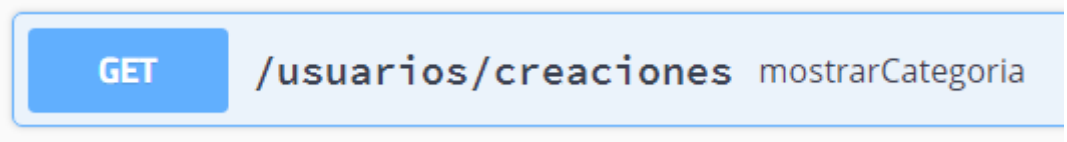

Figura 20. Petición HTTP GET mostrar creaciones por categoría.

Descripción: La petición de la *Figura 20* permite obtener las creaciones de los usuarios cuyas creaciones coinciden con el tipo de categoría introducido en el parámetro de entrada.

Parámetros de entrada:

- tipoCategoria. Valores: Acrilico, Acuarela, Carboncillo, Gouache, LapicesColores, LapizGrafito, Mixta, Oleo, Rotulador, de tipo string.
- tipoPerfil. Valores: PRIVADO, PUBLICO, de tipo string.

Datos de salida:

- Código 200, OK y muestra las creaciones de los usuarios cuyas creaciones coinciden con el tipo de categoría introducido en el parámetro de entrada. Destacar que se ha dejado el tipoPerfil como parámetro de entrada aunque no se tenga en cuenta por si en un futuro se continuase el proyecto poder limitar las creaciones a mostrar dependiendo del tipo de perfil de un usuario.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

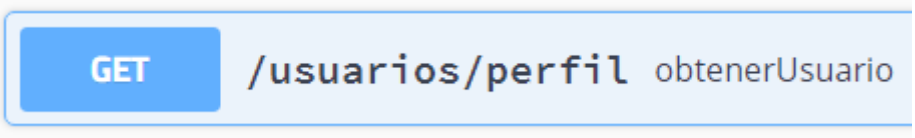

Figura 21. Petición HTTP GET obtener el perfil del usuario.

Descripción: La petición de la *Figura 21* permite obtener el perfil de un usuario.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK y muestra el perfil del usuario.
- Código 401, Unauthorized.

- Código 403, Forbidden.
- Código 404, Not Found.

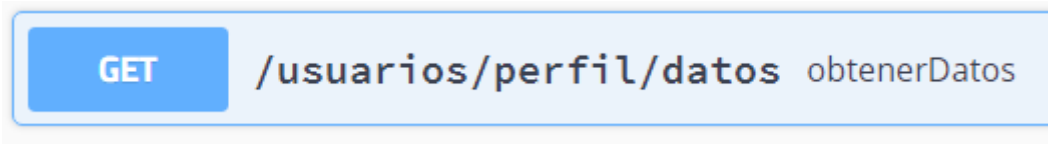

Figura 22. Petición HTTP GET obtener datos.

Descripción: La petición de la *Figura 22* permite obtener los datos de un usuario, en concreto el número de creaciones, número de usuarios seguidos y número de usuarios seguidores.

Parámetros de entrada: id del tipo integer.

Datos de salida:

- Código 200, OK y muestra el número de creaciones, número de usuarios seguidos y número de usuarios seguidores.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

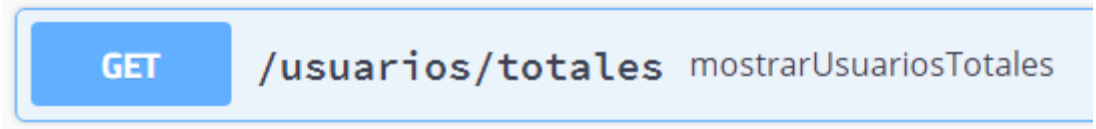

Figura 23. Petición HTTP GET mostrar usuarios totales.

Descripción: La petición de la *Figura 23* permite obtener los usuarios totales que están registrados en el sistema.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK y muestra los usuarios totales del sistema.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

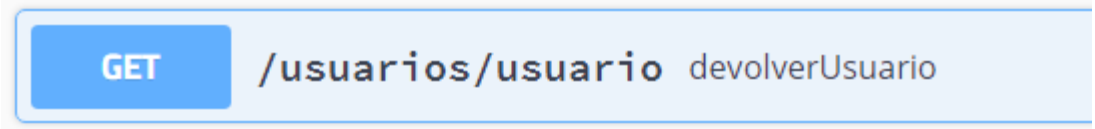

Figura 24. Petición HTTP GET devolver usuario.

Descripción: La petición de la *Figura 24* permite obtener un usuario al pasarle por parámetro de entrada un nombre de usuario.

Parámetros de entrada: usuario del tipo string.

Datos de salida:

- Código 200, OK y devuelve el usuario que coincide con el nombre de usuario.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

3.4.1.2- Peticiones POST

En este apartado se muestran las distintas peticiones HTTP POST a través de la documentación generada por el plugin de swagger.io.

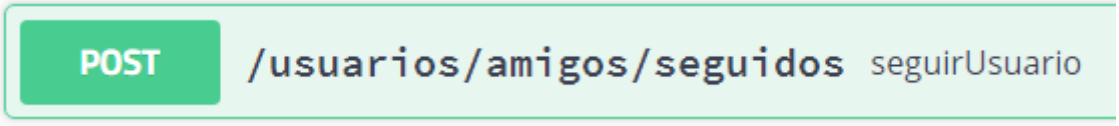

Figura 25. Petición HTTP POST seguir a un usuario.

Descripción: La petición de la *Figura 25* permite seguir a un usuario específico por el parámetro de entrada.

Parámetros de entrada: usuario del tipo string.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

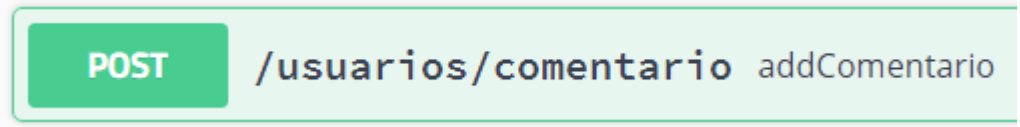

Figura 26. Petición HTTP POST añadir comentario.

Descripción: La petición de la *Figura 26* permite añadir un comentario a una creación.

Parámetros de entrada:

- comentario de tipo string.
- idCreacion de tipo integer.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

Figura 27. Petición HTTP POST subir creación.

Descripción: La petición de la *Figura 27* permite guardar una creación de un usuario en concreto.

Parámetros de entrada:

- categoria. Valores: Acrilico, Acuarela, Carboncillo, Gouache, LapicesColores, LapizGrafito, Mixta, Oleo, Rotulador, de tipo string.
- descripcion del tipo string.
- file del tipo multipart.
- licencia. Valores: Reconocimiento, ReconocimientoCompartirIgual, ReconocimientoNoComercial, ReconocimientoNoComercialCompartirIgual, ReconocimientoNoComercialSinObraDerivada, ReconocimientoSinObraDerivada.
- titulo del tipo string.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

Figura 28. Petición HTTP POST generar datos de prueba.

Descripción: La petición de la *Figura 28* permite crear un usuario de prueba.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

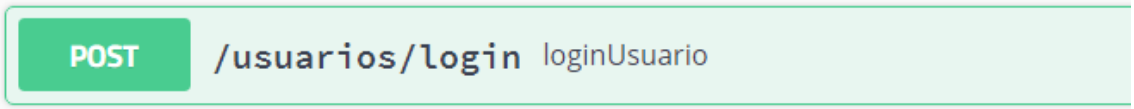

Figura 29. Petición HTTP POST hacer login.

Descripción: La petición de la *Figura 29* permite al usuario iniciar sesión.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

Figura 30. Petición HTTP POST hacer logout.

Descripción: La petición de la *Figura 30* permite al usuario cerrar la sesión.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

Figura 31. Petición HTTP POST registrar un usuario.

Descripción: La petición de la *Figura 31* permite el registro de un usuario.

Parámetros de entrada: user del tipo UsuarioDto.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

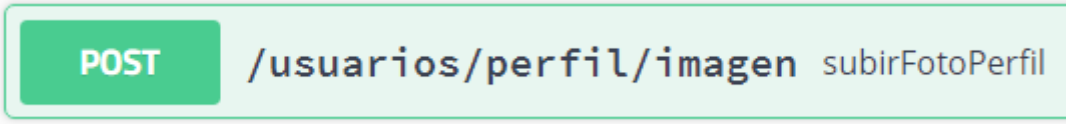

Figura 32. Petición HTTP POST subir una foto de perfil.

Descripción: La petición de la *Figura 32* permite al usuario cambiar su imagen de perfil.

Parámetros de entrada: file del tipo multipart.

Datos de salida:

- Código 200, OK.

- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

3.4.1.3- Peticiones PUT

En este apartado se muestran las distintas peticiones HTTP PUT a través de la documentación generada por el plugin de swagger.io.

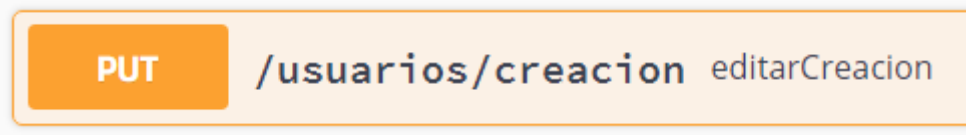

Figura 33. Petición HTTP PUT editar creación.

Descripción: La petición de la *Figura 33* permite editar los datos de una creación subida previamente por el usuario.

Parámetros de entrada:

- categoria. Valores: Acrilico, Acuarela, Carboncillo, Gouache, LapicesColores, LapizGrafito, Mixta, Oleo, Rotulador, de tipo string.
- descripcion del tipo string.
- id del tipo integer.
- licencia. Valores: Reconocimiento, ReconocimientoCompartirIgual, ReconocimientoNoComercial, ReconocimientoNoComercialCompartirIgual, ReconocimientoNoComercialSinObraDerivada, ReconocimientoSinObraDerivada.
- titulo del tipo string.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

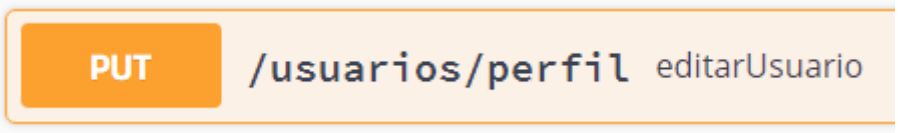

Figura 34. Petición HTTP PUT editar usuario.

Descripción: La petición de la *Figura 34* permite obtener los usuarios seguidores de un usuario.

Parámetros de entrada: user del tipo UsuarioDto.

Datos de salida:

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

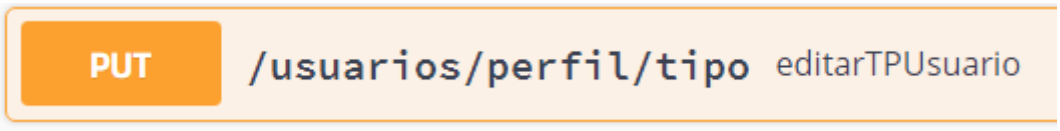

Figura 35. Petición HTTP PUT editar tipo de perfil del usuario.

Descripción: La petición de la *Figura 35* permite cambiar el tipo de perfil del usuario.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida: tipoPerfil. Valores: PRIVADO, PUBLICO, de tipo string.

- Código 200, OK.
- Código 201, Created.
- Código 401, Unauthorized.
- Código 403, Forbidden.
- Código 404, Not Found.

3.4.1.4- Peticiones DELETE

En este apartado se muestran las distintas peticiones HTTP DELETE a través de la documentación generada por el plugin de swagger.io.

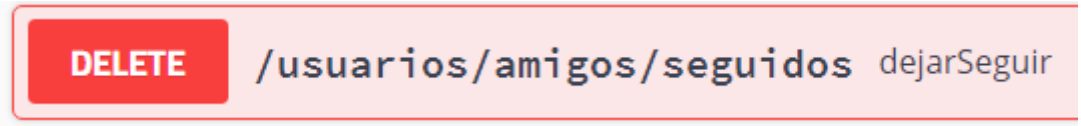

Figura 36. Petición HTTP DELETE dejar de seguir a un usuario.

Descripción: La petición de la *Figura 36* permite a un usuario dejar de seguir a un usuario.

Parámetros de entrada: usuario del tipo string.

Datos de salida:

- Código 200, OK.
- Código 204, No Content.
- Código 401, Unauthorized.
- Código 403, Forbidden.

Figura 37. Petición HTTP DELETE eliminar una creación.

Descripción: La petición de la *Figura 37* permite eliminar una creación de un usuario.

Parámetros de entrada: id del tipo integer.

Datos de salida:

- Código 200, OK.
- Código 204, No Content.
- Código 401, Unauthorized.
- Código 403, Forbidden.

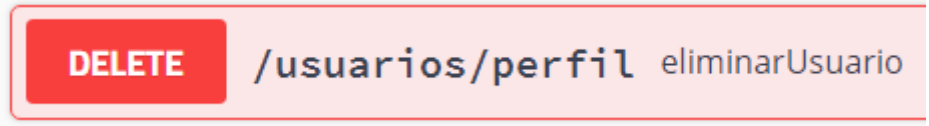

Figura 38. Petición HTTP DELETE eliminar un usuario.

Descripción: La petición de la *Figura 38* permite eliminar la cuenta del usuario.

Parámetros de entrada: Sin parámetros de entrada.

Datos de salida:

- Código 200, OK y muestra los usuarios seguidores del usuario.
- Código 204, No Content.
- Código 401, Unauthorized.
- Código 403, Forbidden.

3.4.2 Diseño de la interfaz de la aplicación

En este apartado se va a mostrar cómo se ha realizado la interfaz de la aplicación móvil.

Para la interfaz de la aplicación se ha empleado la herramienta Creator[4] para crear un prototipo inicial de las vistas más básicas como se puede observar en la *Figura 39.*

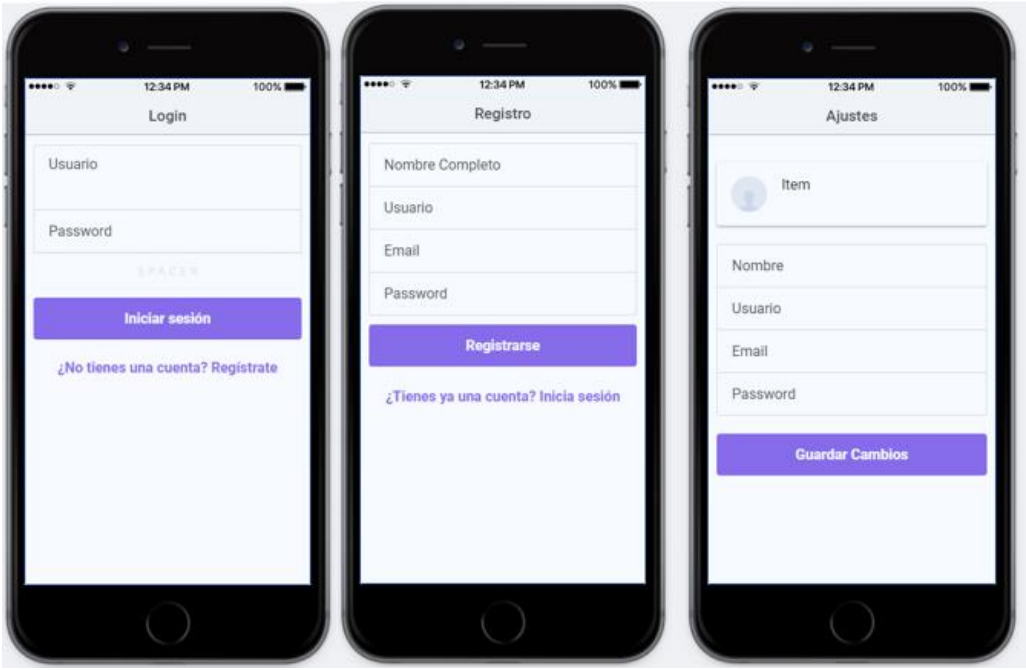

Figura 39. Prototipo inicial de las vistas: login, registro y ajustes.

Más tarde se decidió trasladar este prototipo a Ionic. Como resultó ser bastante simple, se decidió buscar algún tutorial para aportarle una vista más atractiva y poder implementarlo en toda la aplicación. Para la conversión se siguieron las indicaciones de un tutorial [23] para modificar la apariencia de la aplicación.

[4] https://ionicframework.com/creator

Además se buscaron plantillas gratuitas para Ionic a través de la página: https://market.ionicframework.com/themes, la cual contiene plantillas gratis y de pago para Ionic.

Revisando dichas plantillas, se encontró una plantilla[5] gratuita, similar a la apariencia de la red social Instagram y que parecía encajar bastante bien con este prototipo de aplicación, ya que se buscaba una interfaz simple, intuitiva y fácil de utilizar para el usuario. Dicha plantilla se puede observar en la *Figura 40*.

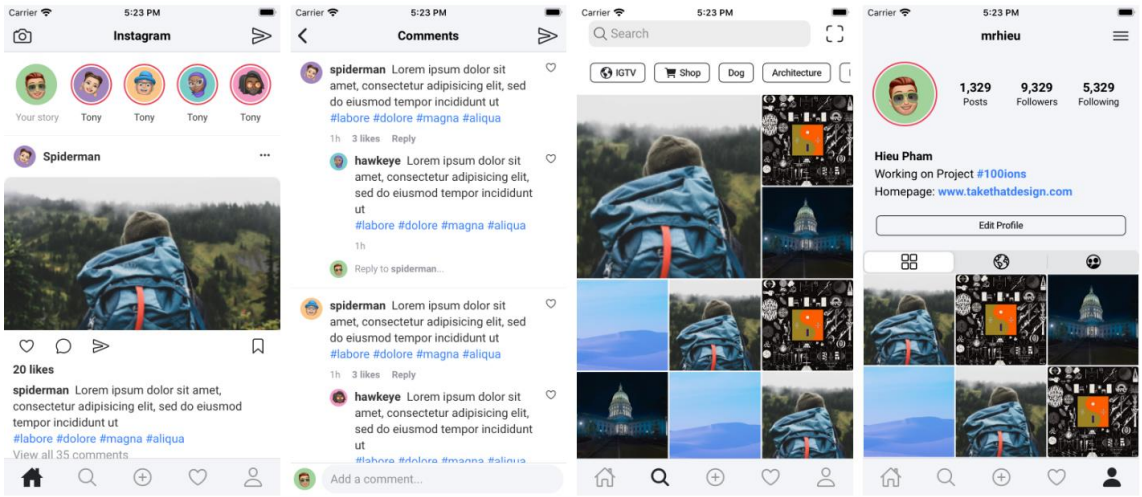

Figura 40. Plantilla gratuita de Ionic, ionic-instagram de Hieu Pham.

Para el estilo de las vistas además de la plantilla comentada anteriormente, se ha empleado css.

También en el diseño de la interfaz se utilizaron formularios de angular con sus respectivas validaciones, por ejemplo, *AlertController* para mostrar mensajes de alerta y *ActionSheetController* para tener menús desplegables llamativos e intuitivos.

Para los distintos iconos empleados en la aplicación se han utilizado los propios iconos de angular y para ello se han consultado en la siguiente página: https://ionic.io/ionicons.

Respecto a las distintas imágenes utilizadas en la interfaz o como ejemplos, decir que son de uso libre de la página: https://pixabay.com/es/.

En el **Apéndice II. Manual de Usuario**, se pueden consultar con detalle todas las vistas de la aplicación cliente.

[5] https://market.ionicframework.com/themes/ionic-instagram

3.5.- Plan de pruebas

En este apartado se va a determinar cómo se va a plantear lo que se va a probar y cómo, detallando más adelante los detalles más específicos.

Para comprobar si las cosas se están haciendo correctamente se realizarán dos tipos de pruebas: del Api/Rest con Postman para comprobar el funcionamiento de la aplicación y, de forma complementaria, pruebas manuales sobre la interfaz de usuario web usando el navegador para comprobar el cumplimiento de los criterios de aceptación. Para ello para cada historia de usuario implementada se comprobarán los criterios de aceptación establecidos en el análisis para verificar que el comportamiento es satisfactorio.

Dichas pruebas se realizarán sobre todo después de cada iteración para verificar que se completan las historias de usuario satisfactoriamente, todo ello usando la herramienta Postman[6]. La herramienta Postman permite de una forma rápida e intuitiva comprobar las distintas funcionalidades del sistema probando si funcionan correctamente las peticiones http CRUD.

Además como apoyo se va a emplear en determinados momentos la herramienta proporcionada por Swagger. Esto se debe a que además de generar la documentación de forma automática como se comentó anteriormente, también se podrán consultar las distintas peticiones http CRUD accediendo fácilmente mediante el navegador a la url http://localhost:8080/swagger-ui.html para comprobar su funcionamiento.

Por último destacar que estas pruebas las realizaré yo mismo. Esto se debe a que no existe una persona específica o cliente para este caso en particular que pueda realizar dicha tarea.

4.- Implementación

4.1.- Arquitectura

En este apartado se va a mostrar el diagrama arquitectónico del sistema desarrollado como se puede observar en la *Figura 41*, donde se observan los elementos constituyentes, ya sean desarrollados o externos, y la relación entre ellos, explicando brevemente la función de cada uno de ellos.

[6] https://www.postman.com/

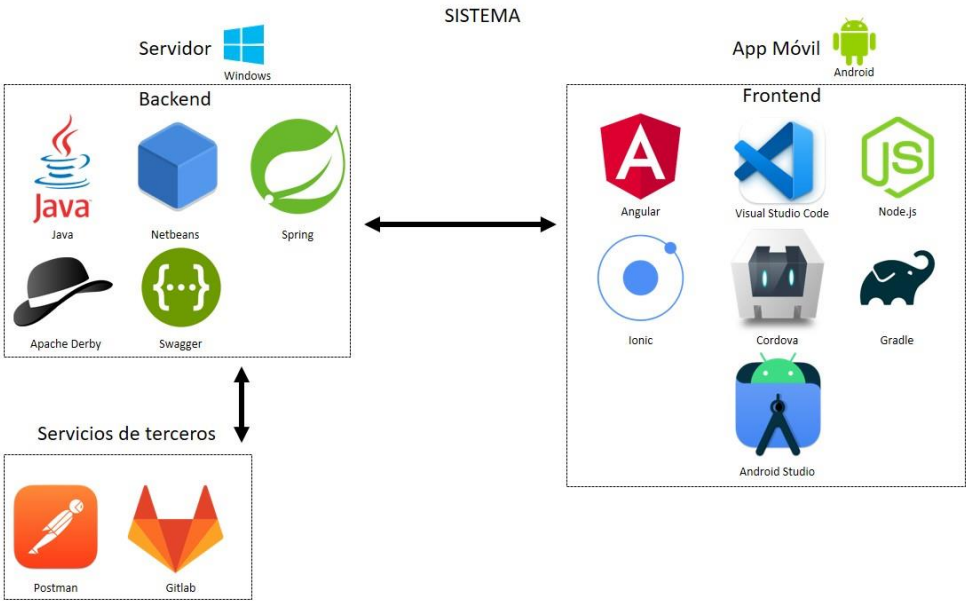

Figura 41. Diagrama arquitectónico del sistema desarrollado.

El servidor ha sido implementado en Windows 10, para el backend se ha utilizado:

- **Apache Netbeans.** Se trata de un entorno de desarrollo gratuito y de código abierto empleado para el desarrollo de aplicaciones. Es el software que se ha utilizado para la implementación del backend.
- **Java 8.** Se trata de un lenguaje de programación que se ha utilizado para la implementación del Backend. Se ha decidido utilizar este lenguaje de programación debido a que Spring Framework emplea este lenguaje de programación para su desarrollo y permite facilitar la implementación del código debido al uso de anotaciones del propio framework.
- **Spring[7].** Se trata de un framework utilizado en el backend, el cual proporciona un modelo integral de programación para aplicaciones empresariales basadas en Java. Las características principales de este framework son la inyección de dependencias, validación, soporte DAO, etc.
- **Apache Derby[8].** Se ha utilizado esta base de datos para su implementación en el servidor.

[7] https://spring.io/projects/spring-framework
[8] https://db.apache.org/derby/

- **Swagger[9].** Se trata de una herramienta que permite documentar de forma automática en el Api/Rest, urls o endpoints de las diferentes funcionalidades, métodos de acceso: GET, POST, PUT, etc. descripción de cada operación, parámetros de entrada, datos de salida y códigos de estado/error devueltos. Se ha utilizado para la documentación y pruebas del servidor.

También se ha hecho el uso de herramientas externas como pueden ser:

- **Postman[10].** Se trata de una aplicación que permite hacer pruebas de las distintas peticiones http CRUD en el servidor.
- **Gitlab[11].** Se trata de un servicio web de control de versiones.

Para la implementación de la aplicación móvil se ha utilizado:

- **Ionic[12].** Se trata de un framework utilizado para el desarrollo de aplicaciones hibridas. Se ha empleado para la creación de la aplicación móvil.
- **Angular[13] 11.2.4.** Se trata de un framework utilizado para crear aplicaciones web, además está desarrollado en Typescript. Se ha empleado para el desarrollo de la aplicación en Ionic.
- **Typescript.** Se trata de una especie de superset de JavaScript, cuyo resultado final es un código de JavaScript. Además lo que hace es encapsular varios elementos: una serie de funcionalidades de JavaScript 5, contiene ECMAScript 6 (ES6), añade el uso un tipado muy estricto y también añade la funcionalidad de genérica, que permite poder definir funciones que sean reutilizables, independientemente del tipo de datos que vayamos a tratar [24].
- **Node.js[14].** Se trata de un entorno empleado para poder hacer usos de algunos plugins de Cordova y Angular, necesarios para la creación de la aplicación en Android.

[9] https://swagger.io/
[10] https://www.postman.com/
[11] https://gitlab.com/
[12] https://ionicframework.com/
[13] https://angular.io/guide/what-is-angular
[14] https://nodejs.org/es/

- **Gradle**[15] **7.0.2**. Se trata de una herramienta que permite la automatización de compilación de código abierto. Además es el sistema de compilación oficial para Android y cuenta con soporte para diversas tecnologías y lenguajes [25]. Se ha empleado esta herramienta en Ionic y en Android Studio para la exportación de la aplicación al formato .apk de Android.
- **Cordova 10.** Se trata de un entorno de desarrollo de aplicaciones móviles y se ha empleado para compilar y crear el archivo necesario de la aplicación para su posterior uso en Android Studio.
- **Android Studio 16.** Se trata de un software que ofrece las herramientas necesarias para poder compilar aplicaciones de Android. Mediante este software se han podido realizar diversas pruebas a través del emulador de Android que contiene el software y además ha sido empleado para poder exportar la aplicación correctamente a Android.

4.2.- Detalles sobre implementación

En este apartado se van a comentar los aspectos metodológicos y técnicos más relevantes sobre la implementación.

Para la implementación del servidor se ha seguido la documentación del sitio web Baeldung[17] y para la implementación de la aplicación cliente se ha seguido la documentación de los sitios web de Ionicframework[18], Javaguides[19] y Angular[20].

4.2.1 Detalles del backend

La tecnología que se ha empleado en el servidor ha sido el **API Rest de Spring MVC.** El entorno que se ha empleado para su implementación es **Netbeans** y para la gestión de la base de datos, se ha hecho uso de **Apache Derby**.

[15] https://gradle.org/
[16] https://developer.android.com/
[17] https://www.baeldung.com/
[18] https://ionicframework.com/
[19] https://www.javaguides.net/
[20] https://angular.io/docs

La estructura del proyecto del servidor se ha realizado separando por paquetes las distintas funcionalidades del sistema. Dichos paquetes son:

Swagger. En este paquete se ha definido la clase *SwaggerConfig* necesaria para que la herramienta generase la documentación del Api/Rest. Además es necesario declarar las dependencias en el archivo pom.xml e indicarle al sistema con la etiqueta *@SpringBootApplication* en que paquete se encuentra el archivo de configuración. Para la configuración y uso de la herramienta Swagger se siguieron las recomendaciones indicadas en [26,27].

Entidades. En este paquete se han definido las distintas entidades que se utilizan en la aplicación. Estas clases deben tener una etiqueta *@Entity* para indicarle al sistema que son entidades. Todas las entidades generan automáticamente un identificador único mediante la etiqueta *@Id* y *@GeneratedValue*. Las distintas entidades son:

- **Usuario.** Entidad utilizada para gestionar correctamente los usuarios de la aplicación.
- **Creacion.** Entidad utilizada para gestionar correctamente las creaciones subidas por los usuarios.
- **Comentario.** Entidad utilizada para gestionar correctamente los comentarios de las creaciones creadas por los usuarios.
- **Amigos.** Entidad utilizada para gestionar correctamente la relación entre los usuarios seguidos y los usuarios seguidores de un usuario.

Para explicar la implementación y funcionamiento de las entidades se van a mostrar unos ejemplos:

Como se puede observar en la *Figura 42* en la clase de la entidad usuario, además de los atributos y métodos esenciales, se tienen dos atributos mapeados que representan la relación de las creaciones y amigos mediante una lista. Para estos atributos es necesario indicar la cardinalidad con una etiqueta *@OneToMany*. Estos se debe a que un usuario puede tener muchas creaciones y amigos. Además el tipo de carga será del tipo EAGER [28], ya que cuando se carga el usuario es muy importante que se carguen todas sus creaciones. También se hace uso de la etiqueta *@JsonManagedReference*, la cual administra la parte hacia adelante de la referencia y los campos marcados por

esta anotación son los que se serializan [29]. Además dicha relación es de tipo cascada, de esta forma si se elimina el usuario, se eliminan todas sus creaciones y las relaciones que tenga el usuario con los distintos usuarios de la aplicación.

```
@Entity
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "Usuario_ID")
    private long id;
    private String usuario;
    private String nombre;
    private String email;
    private String password;
    private String rol;
    private String descripcion;
    private TipoPerfil tipoPerfil;
    private String fotoPerfil;
    @OneToMany(mappedBy = "usuario", cascade = CascadeType.ALL,
            fetch = FetchType.EAGER)
    @JsonManagedReference
    private List<Creacion> creaciones;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    @JsonManagedReference
    private List<Amigos> amigos;
```

Figura 42. Entidad Usuario.

Como se puede observar en la *Figura 43* en la clase de la entidad usuario, además de los atributos y métodos esenciales, se tienen dos atributos mapeados.

Un atributo representa la relación de las creaciones del usuario, para ello es necesario usar la etiqueta *@ManyToOne* porque un usuario puede tener muchas creaciones, además el tipo de carga será del tipo LAZY [30]. De esta forma se recuperan los objetos sólo cuando se acceden a los objetos de la asociación.

El otro atributo representa la relación de los comentarios de las creaciones mediante una lista, para ello es necesario usar la etiqueta *@OneToMany* porque una creación puede tener muchos comentarios, además el tipo de carga será del tipo EAGER. Esto es muy importante ya que de esta forma se cargan los comentarios de la creación cuando esta se carga, además dicha relación será de tipo cascada. De esta forma si se elimina la creación, se eliminan todos sus comentarios. También es necesario usar la etiqueta @JsonBackReference para indicar la referencia de retorno, evitando de esta forma ciclos infinitos. La etiqueta

@JsonBackReference, administra la parte inversa de la referencia y los campos/colecciones marcados con esta anotación no se serializan [29].

```
@Entity
public class Creacion {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "Creacion_ID")
    private long id;
    private String fileName;
    private String titulo;
    private String descripcion;
    private TipoCategoria categoria;
    private CreativeCommons licencia;
    @ManyToOne(targetEntity = Usuario.class, fetch = FetchType.LAZY)
    @JoinColumn(name = "Usuario_ID", referencedColumnName = "Usuario_ID")
    @JsonBackReference
    private Usuario usuario;

    @OneToMany(mappedBy = "creacion", cascade = CascadeType.ALL,
            fetch = FetchType.EAGER)
    @JsonManagedReference
    private List<Comentario> comentarios;
```

Figura 43. Entidad Creación.

El resto de entidades tienen un funcionamiento similar acorde a las relaciones entre las demás entidades.

DTO. En este paquete se han implementado los distintos DTO (ComentarioDto, CreacionDto y UsuarioDto), los cuales son las entidades puras, es decir, sin atributos que tengan que ver con otras entidades distintas o métodos distintos a getters/setters a excepción de los métodos fromDTO y toDTO que permiten pasar una entidad a DTO y viceversa. Para la entidad amigos se ha decidido no crear un DTO ya que lo más importante son sus atributos relacionados con la entidad usuario.

DAO. En este paquete se han definido los distintos DAO necesarios para encapsular toda la lógica del acceso de datos respecto al resto de la aplicación, de esta forma en cada DAO se encuentran aquellas operaciones que se realizan en la base de datos. Además, para la dependencia entre los DAOs y el *EntityManager* para el acceso a JPA es necesario declarar en los DAOs la etiqueta *@Repository*. Con esta etiqueta más la comentada anteriormente (*@Entity* en las entidades), el framework puede gestionarlo correctamente. También es necesario incluir la dependencia *spring-boot-starter-data-jpa* en el archivo *pom.xml*. A continuación se va a mostrar la clase UsuarioDao para explicar el funcionamiento de los DAO:

- **UsuarioDao.** En este DAO se pueden encontrar las operaciones o consultas a la base de datos relacionadas con los usuarios. A continuación se van a mostrar algunos aspectos más relevantes:

Como se puede observar en la *Figura 44* es necesario declarar la etiqueta @Repository al principio de la clase y un atributo de tipo EntityManager con la etiqueta @PersistenceContext. Dicho atributo es con el que se trabaja para hacer las distintas operaciones sobre la base de datos respecto a la entidad Usuario.

```
@Repository
public class UsuarioDao {

    @PersistenceContext
    EntityManager em;
```

Figura 44.Clase UsuarioDao.

Seguidamente se van a mostrar unos métodos de ejemplo de este DAO:

Como se puede observar en la *Figura 45* se muestra la operación de eliminar usuarios. Para ello se le pasa por la cabecera el usuario que se va a eliminar y se invoca al método *remove* seguido del *merge* que se utiliza para devolver un objeto que existe dentro del *PeristenceContext*.

```
@Transactional(propagation = Propagation.REQUIRED, readOnly = true)
public void eliminar(Usuario user) {
    em.remove(em.merge(user));
}
```

Figura 45. Operación eliminar usuarios.

Como se puede observar en la *Figura 46* se muestra la operación de buscar un usuario por su nombre de usuario. Para ello se le pasa por la cabecera el nombre de usuario y acto seguido se realiza una consulta para buscar dicho nombre y devolverlo. En algunas consultas se utiliza un pseudo nombre para aportar un poco más de seguridad a la consulta cuando se accede a los datos. En este caso se ha cambiado *u.usuario =:usuario* por *u.usuario =:nombre*.

```
public Usuario buscarPorUsuario(String usuario) {
    Usuario u = null;
    try {
        u = em.createQuery("select u from Usuario u "
                + "where u.usuario =:nombre", Usuario.class).setParameter
    ("nombre", usuario).getSingleResult();
    } catch (Exception e) {
    }
    return u;
}
```

Figura 46. Buscar por nombre de usuario.

También ha sido necesario declarar en la clase de tipo *beans SistemaEaselbook* un atributo por cada DAO. Para este caso (*usuarioDAO*), se puede observar en la Figura 47 que se ha declarado un atributo de tipo *UsuarioDao* para realizar las distintas operaciones en la base de datos relacionadas con los usuarios de la aplicación.

```
@Autowired
UsuarioDao usuariosDao;
```

Figura 47. Declaración del atributo usuariosDao.

Para entender mejor la explicación a continuación se va a mostrar como ejemplo el método de registrar un usuario en la aplicación de la clase *SistemaEaselBook* dónde interviene el atributo del *UsuarioDao*.

Como se puede observar en la *Figura 48*, a través del atributo usuariosDao se puede crear el registro del usuario en la base de datos.

Además cada vez que un usuario se registra en la aplicación su contraseña se codifica gracias a la funcionalidad de seguridad *BCryptPasswordEncoder* de Spring. De esta forma en la base de datos se muestran las contraseñas de los usuarios de la aplicación totalmente codificadas. Si el usuario quisiera cambiar su contraseña deberá de comunicárselo al administrador.

```
@Transactional
@Override
public boolean registrarUsuario(UsuarioDto user) {
    Usuario usuarioID = usuariosDao.buscarPorUsuario(user.getUsuario());
    if (usuarioID != null) {
        throw new UsuarioRegistrado();
    }
    usuarioID = user.fromDTO(user);
    BCryptPasswordEncoder pass = new BCryptPasswordEncoder();
    usuarioID.setPassword(pass.encode(user.getPassword()));
    usuariosDao.crear(usuarioID);
    return true;
}
```

Figura 48. Método registrar usuario.

Para el resto de DAOs (*AmigosDao, ComentarioDao y CreacionDao*), comentar que siguen el mismo patrón de implementación.

Servicios. En este paquete se han implementado las distintas clases de apoyo, así como de tipo *enum* para utilizarlas de atributo en las entidades. Por ejemplo como se puede observar en la *Figura 49* se muestra la clase *TipoCategoria*. Esta clase es de tipo *enum* y permite clasificar los distintos tipos de categorías que tienen las creaciones en la aplicación.

```
public enum TipoCategoria {
    Acrilico,
    Acuarela,
    Oleo,
    Rotulador,
    Carboncillo,
    Gouache,
    LapicesColores,
    LapizGrafito,
    Mixta
}
```

Figura 49. Clase TipoCategoria.

El resto de clases: *CreativeCommons*, *TipoPerfil* y *EstadoPeticion* (funcionalidad que no se tiene en cuenta actualmente), funcionan de manera similar. Sin embargo, la clase Datos tiene un funcionamiento distinto. Dicha clase se trata de una clase de apoyo empleada para obtener distintos datos del usuario (número de creaciones subidas a la aplicación, número de usuarios seguidores y número de usuarios seguidos), a través de las distintas peticiones http CRUD.

Formato. En este paquete se ha implementado la clase *Formato* para dar formato al tipo de email de los usuarios. En un futuro podría utilizarse para implementar más tipos de formato.

Excepciones. En este paquete se han implementado las clases de excepciones como pueden ser *EmailIncorrecto* y *UsuarioRegistrado*.

Beans. En este paquete se ha implementado la clase *SistemaEaselBook*. Dicha clase permite encapsular el contenido con el objetivo de conseguir una estructura mejor permitiendo la reutilización de código a través de una estructura sencilla. Lo más destacable aquí es que se da lugar a la inyección de dependencias, cuyo objetivo es el de suministrar objetos a una clase mediante el contenedor de Spring, sin necesidad de que la propia clase los tenga que crear. En esta clase se emplean funciones previamente declaradas en el paquete **Interfaces,** en el cual se tiene declarada una clase llamada **Servicios** que actúa de interfaz.

Además en esta clase se tienen declarados unos atributos de los DAO previamente indicados con la etiqueta *@Autowired,* ya que esta anotación permite inyectar unas dependencias con otras en Spring y algunas funciones tienen la etiqueta *@Transactional* para garantizar de esta forma que la transacción sea atómica.

A continuación se van a detallar las operaciones de esta clase:

- **Boolean registrarUsuario(UsuarioDto user).** Permite el registro de un usuario haciendo una comprobación previa comprobando que el usuario no esté registrado previamente en la aplicación. Además aquí se le codifica al usuario la contraseña.
- **Void editarUsuario(UsuarioDto user).** Permite modificar los parámetros del usuario, tales como el nombre, email y la descripción.
- **Void editarPerfilUsuario(UsuarioDto user).** Permite modificar el tipo de perfil de un usuario.
- **Void editarFotoPerfil(UsuarioDto user).** Permite cambiar la imagen del perfil del usuario.
- **Usuario devolverUsuario(String id).** Devuelve un usuario mediante su nombre de usuario.
- **Boolean guardarCreacion(CreacionDto creation, String id).** Guarda la creación de un usuario en la aplicación.
- **Void obtenerDatos(Datos datos, long id).** Obtiene el número de creaciones, usuarios seguidores y usuarios seguidos de un usuario.

- **Boolean addComentario(ComentarioDto comentario, String idUsuario, int idCreacion).** Añade un comentario de un usuario a una creación.
- **Void addAmigos(UsuarioDto userPrincipal, UsuarioDto userAmigo, EstadoPeticion peticion).** Permite a un usuario seguir a otro usuario y establecer dicha relación de amigos.
- **Void eliminarAmigos(UsuarioDto userPrincipal, long id).** Permite a un usuario dejar de seguir a otro usuario y eliminar dicha relación de amigos.
- **Boolean eliminarUsuario(UsuarioDto user).** Permite eliminar un usuario de la aplicación.
- **Boolean eliminarCreacion(UsuarioDto user, int id).** Elimina de la aplicación la creación de un usuario.
- **Void editarCreacion(UsuarioDto user, int id, String titulo, String descripcion, TipoCategoria categoria, CreativeCommons licencia).** Permite modificar los parámetros de una creación de un usuario, tales como, título, descripción, categoría y licencia.
- **List<Usuario> mostrarUsuarios().** Muestra todos los usuarios de la aplicación.
- **List<Amigos> mostrarSeguidores(long id).** Muestra los usuarios seguidores de un usuario.
- **List<Amigos> mostrarSeguidos(long id).** Muestra los usuarios seguidos por un usuario.
- **Usuario devolverUsuarioId(long id).** Devuelve un usuario mediante su id.
- **Creacion devolverCreacion(long id).** Devuelve una creación mediante su id.
- **List<Usuario> buscarUsuarios(String keyword).** Muestra los usuarios cuyo nombre coincide con el nombre pasado por parámetro.
- **List<Creacion> mostrarCreacionesCategoria(TipoCategoria categoria, TipoPerfil perfil).** Muestra las creaciones correspondientes al tipo de categoría pasada por parámetro.

Para terminar de explicar esta clase se va a mostrar un ejemplo de un método de los comentados anteriormente.

Como se puede observar en la *Figura 50*, la comunicación con el API Rest se hace a través de objetos DTO y que, por lo tanto, es necesario transformarlos a Entidades para poder persistirlos.

```
@Transactional
@Override
public boolean guardarCreacion(CreacionDto creation, String id) {
    Usuario usu = usuariosDao.buscarPorUsuario(id);
    Creacion nuevaCreacion = creation.fromDTO(creation);
    usu.setCreaciones(Arrays.asList(nuevaCreacion));
    nuevaCreacion.setUsuario(usu);
    return true;
}
```

Figura 50. Método guardar creación.

Servidor. En este paquete se ha implementado clase *ControladorSistema*, se trata del controlador principal del API Rest. Para ello se ha utilizado la etiqueta *@RestController* para indicar que es el controlador. También se ha utilizado la etiqueta *@CrossOrigin(origins = "*")* para aceptar las conexiones entrantes y la etiqueta *@RequestMapping("/usuarios")* para el mapeo de la url inicial.

En esta clase se pueden encontrar 3 tipos de atributos principales:

- **Datos datos.** Se utiliza para obtener el número de creaciones, usuarios seguidores y usuarios seguidos de un usuario.
- **Long contador.** Se utiliza para añadirle un número a cada nombre de las creaciones y fotos de perfil subidas en la aplicación, de forma que se tiene un nombre de archivo único.
- **SistemaEaselBook sistema.** Se trata del beans comentado anteriormente que se utiliza para realizar las distintas operaciones respecto a las peticiones http CRUD que reciba el servidor.

Además en esta clase se encuentran los distintos métodos http CRUD que se han visto previamente en el apartado 3.4.1 Diseño Api/Rest.

En la *Figura 51* se puede observar el método http CRUD de subir una creación [31, 32]. Al método se le pasan una serie de parámetros de la imagen (título, descripción, categoría y licencia), así como la propia imagen que es del tipo *MultipartFile*.

Mediante el mecanismo de seguridad de Spring se obtiene el usuario que hay en línea en ese momento. Después el fichero enviado es colocado por Spring en una ubicación temporal y por ello posteriormente debe copiarse a la ubicación definitiva.

```
@ResponseStatus(OK)
@CrossOrigin
@PostMapping("/creacion")
public void subirCreacion(@RequestParam("file") MultipartFile imagen,
        @RequestParam("titulo") String titulo,
        @RequestParam("descripcion") String descripcion,
        @RequestParam("categoria") TipoCategoria categoria,
        @RequestParam("licencia") CreativeCommons licencia) {
    UserDetails user = (UserDetails) SecurityContextHolder.getContext()
            .getAuthentication().getPrincipal();
    if (!imagen.isEmpty()) {
        Path directorioImagen = Paths.get(ubicacionImagenes);
        String rutaAbsoluta = directorioImagen.toFile().getAbsolutePath();
        try {
            byte[] bytesImg = imagen.getBytes();
            String nombreImagen = Long.toString(contador)
                    + imagen.getOriginalFilename();
            Path rutaCompleta = Paths
                    .get(rutaAbsoluta + "//" + nombreImagen);
            Files.write(rutaCompleta, bytesImg);
            CreacionDto creacion = new CreacionDto(0, nombreImagen,
                    titulo, descripcion, categoria, licencia);
            sistema.guardarCreacion(creacion, user.getUsername());
            contador++;
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Figura 51. Método subir creación.

CORS. En este paquete se ha implementado el CORS. El CORS es una política de seguridad que prohíbe accesos no autorizados al servidor.

Para ello se ha utilizado la clase *WebConfig [33, 34]*. En esta clase se ha utilizado la etiqueta de *@Configuration* para indicarle al Framework de Spring que es una clase de configuración. Además se ha utilizado la etiqueta *@Bean* para que los métodos estén disponibles.

En la *Figura 52* se puede observar la clase WebConfig, la cual dispone de los métodos *addCorsMappings y addResourceHandlers*.

El primer método es necesario para indicar que cualquier usuario/dispositivo puede acceder desde cualquier destino a las distintas urls http.

El segundo método permite que se pueda acceder al directorio de las creaciones e imágenes de los usuarios a través de las distintas peticiones http.

```
@Configuration
public class WebConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            String myExternalFilePath = "file:C:/Users/David/images/";

            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**").allowedOrigins("*")
                        .allowCredentials(true);
            }

            @Override
            public void addResourceHandlers(ResourceHandlerRegistry registry) {
                registry.addResourceHandler("/images/**")
                        .addResourceLocations(myExternalFilePath);
            }
        };
    }
}
```

Figura 52. CORS.

Seguridad. En este paquete se ha implementado la configuración de la seguridad del servidor. En dicho paquete se pueden encontrar 2 clases:

- **ServicioDatosEaselBook.** Esta clase implementa métodos de la interfaz *UserDetailsService* de Spring para gestionar la autenticación de los usuarios conectados en ese momento. En la *Figura 53* se puede observar su funcionamiento. El método *loadUserByUsername* se encarga de comprobar si existe el usuario que se le ha pasado por parámetro. Si dicho usuario existe, devuelve el usuario y su contraseña, en caso contrario salta una excepción comunicando que el usuario no se ha encontrado. Para la realización de esta clase se ha consultado la documentación disponible en Baeldung [35].

```
@Component
public class ServicioDatosEaselBook implements UserDetailsService {

    @Autowired
    UsuarioDao usuarioDao;

    @Override
    public UserDetails loadUserByUsername(String usuario)
            throws UsernameNotFoundException {
        Usuario user = usuarioDao.buscarPorUsuario(usuario);
        if (user == null) {
            throw new UsernameNotFoundException("No se ha encontrado");
        }

        return User.withUsername(usuario).password(user.getPassword())
                .roles("USUARIO").build();
    }
}
```

Figura 53. UserDetails.

- **SeguridadEaselBook.** Esta clase hereda de la clase *WebSecurityConfigurerAdapter* de Spring y se encarga de gestionar la autenticación de los usuarios a través de la autenticación del servicio *userDetailsService*. Además también permite configurar la seguridad del tipo *httpBasic*. Su funcionamiento se puede observar en la *Figura 54*. En esta clase se tienen 3 métodos. El primer método es utilizado para la autenticación de los usuarios. el segundo método es utilizado para definir el CORS y la seguridad del tipo httpBasic. Por último el tercer método se utiliza para declarar urls a las que se pueden acceder sin autenticación, como pueden ser la url para generar datos de pruebas y la url para que un usuario se registre en la aplicación.

```
@Configuration
@EnableWebSecurity
public class SeguridadEaselBook extends WebSecurityConfigurerAdapter {

    @Autowired
    ServicioDatosEaselBook servicio;

    @Override
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(servicio)
                .passwordEncoder(new BCryptPasswordEncoder());
    }

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception {
        httpSecurity.csrf().disable();
        httpSecurity.httpBasic();
        httpSecurity.cors();
        httpSecurity.authorizeRequests().antMatchers("/usuarios/**")
                .hasRole("USUARIO");
        httpSecurity.logout()
                .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
                .logoutSuccessUrl("/login");
    }

    @Override
    public void configure(WebSecurity web){
    web.ignoring().antMatchers(HttpMethod.POST,"/usuarios/generarDatosPruebas");
    web.ignoring().antMatchers(HttpMethod.POST,"/usuarios/nuevoUsuario");
    }
}
```

Figura 54. Clase SeguridadEaselBook.

Servidor. En este paquete se ha implementado la clase *ServidorEaselBook*, la cual hace como del "main" de la aplicación. Además en dicha clase se declararán los paquetes de tipo beans, del tipo configuration y las entidades con la etiqueta *@EntityScan*.

Se ha utilizado también una etiqueta de tipo *@Bean* para indicar el tamaño máximo permitido de las imágenes en el servidor, actualmente es de 10MB [36]. También ha sido necesario configurarlo también en el correspondiente *archivo.yml*.

Además de los paquetes anteriormente mencionados, también es necesario configurar una serie de archivos:

Application.yml. Este archivo se ha utilizado para configurar la base de datos utilizada por Spring. En la *Figura 55* se puede observar dicho archivo, al cual se le han indicado una serie de parámetros relacionados con la gestión de la base de datos para su correcto funcionamiento. Los distintos parámetros son el driver de la base de datos, la url de la base de datos, el usuario de la base de datos, la contraseña de la base de datos, el máximo de conexiones activas, el

tipo de esquema utilizado, la opción de mostrar consultas sql por pantalla, las propiedades de Hibernate y el máximo de archivos permitido.

```
spring.datasource.driver-class-name: org.apache.derby.jdbc.ClientDriver
spring.datasource.url: jdbc:derby://localhost:1527/EaselBook
spring.datasource.username: clajer
spring.datasource.password: clajer
spring.datasource.tomcat.max-active: 50

spring.jpa.hibernate.ddl-auto: create
spring.jpa.show-sql: true
spring.jpa.properties.hibernate.dialect: org.hibernate.dialect.DerbyTenSevenDialect

spring.servlet.mutipart.max-file-size: 10MB
spring.servlet.multipart.max-request-size: 10MB
```

Figura 55. Archivo Application.yml.

Pom.xml. Este archivo se ha utilizado para especificar todas las dependencias utilizadas en el proyecto. En la *Figura 56* se puede observar cómo se han declarado las dependencias de la base de datos Apache Derby.

En este archivo además de la dependencia comentada anteriormente, también se tienen declaradas las dependencias del Framework de Spring, también las dependencias de Spring Security para las funcionalidades relacionadas con la seguridad, las dependencias para la subida de archivos[21] y las dependencias de Swagger IO para la documentación.

```
<dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derbyclient</artifactId>
    <version>10.14.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derby</artifactId>
    <version>10.14.2.0</version>
</dependency>
<dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derbynet</artifactId>
    <version>10.14.2.0</version>
</dependency>
```

Figura 56. Dependencias de la BD Apache Derby.

4.2.2 Detalles del frontend

La tecnología que se ha empleado en la parte del cliente para realizar la aplicación ha sido el **Framework de Ionic**. Se trata de un conjunto de

[21] https://commons.apache.org/proper/commons-fileupload/

herramientas de interfaz de usuario de código abierto para crear aplicaciones móviles de escritorio de alta calidad y rendimiento utilizando tecnologías web (HTML, CSS y JavaScript) con integraciones para marcos populares, como por ejemplo Angular [37].

El entorno que se ha empleado para su implementación es Visual Studio Code, ya que este entorno es comúnmente utilizado para este tipo de implementaciones.

La estructura del proyecto del cliente se ha realizado separando por páginas las distintas funcionalidades del sistema.

En el proyecto se tiene una carpeta llamada *models,* que contiene una clase llamada *user.ts*. En esta clase se han definido las distintas entidades utilizadas del sistema pero solamente sus atributos. Esto se debe a que de esta forma está todo unificado en un único archivo. Además también en esta clase hay un atributo con la dirección ip del servidor y su correspondiente puerto, de esta forma se hace más sencillo el cambio de la ruta del servidor en la aplicación.

También se tiene una carpeta llamada servicios donde se encuentran una serie de clases con distintas funcionalidades, se van a detallar a continuación:

- **alert.service.ts.** Esta clase se utiliza para la implementación de un mensaje de alerta de tipo info, para que se muestre en el caso de que no haya creaciones en una categoría dada. Para ello hace uso de la dependencia *ToastController* de *@ionic/angular*.
- **AuthenticationService.ts.** En esta clase se ha implementado el servicio encargado de la autenticación de la aplicación [38].
- **HttpInterceptorService.ts.** En esta clase se ha implementado un interceptor para la autenticación en toda la aplicación [38].

Además como se puede observar en la *Figura 57* la clase AuthenticationService se encarga de identificar al usuario utilizando una autenticación básica y de generar el encabezado de autenticación que se reenviará automáticamente, mediante una clase interceptora (Ver *Figura 58*), en cada llamada posterior al API Rest. Por último destacar que dicho servicio (AuthenticationService), es llamado expresamente cuando el usuario pulsa el botón para conectarse a la aplicación.

```
authenticationService(username: String, password: String) {
  var options = {
    headers: new HttpHeaders()
      .set('Authorization', this.createBasicAuthToken(username, password))
  }
  return this.http.post(this.host+'/usuarios/login', {}, options).pipe(map((res) => {
    this.username = username;
    this.password = password;
    this.registerSuccessfulLogin(username, password);
  }));
}
```

Figura 57. AuthenticationService.

```
@Injectable()
export class HttpInterceptorService implements HttpInterceptor {


  constructor() { }

  intercept(req: HttpRequest<any>, next: HttpHandler) {

    if (sessionStorage.getItem('username') && sessionStorage.getItem('basicauth')) {
      req = req.clone({
        setHeaders: {
          Authorization: sessionStorage.getItem('basicauth')
        }
      })
    }

    return next.handle(req);

  }
}
```

Figura 58. HttpInterceptorService.

Respecto a las distintas componentes que componen la aplicación se pueden observar en la *Figura 59*.

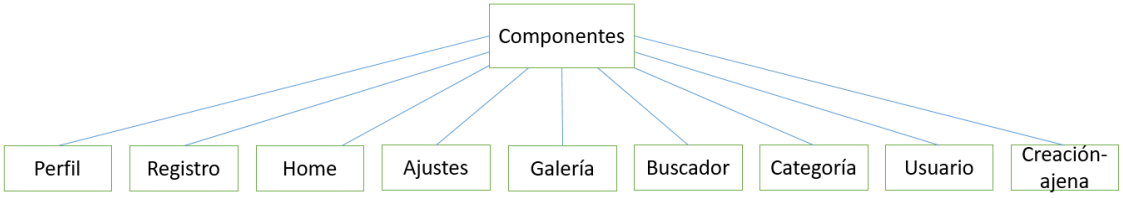

Figura 59. Componentes.

A continuación se van a detallar algunos aspectos técnicos más relevantes de las componentes.

4.2.2.1- Validación de formularios

En este apartado se va a mostrar un ejemplo de validación mediante el formulario de registro de los usuarios [39].

En la *Figura 60* se muestra como se han declarado los distintos campos del formulario con sus respectivas condiciones requeridas.

```
ngOnInit() {
  this.ionicForm = this.formBuilder.group({
    usuario: ['', [Validators.required, Validators.minLength(2)]],
    nombre: ['', [Validators.required, Validators.minLength(2)]],
    email: ['', [Validators.required, Validators.pattern('\\b[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}\\b')]],
    password: ['', [Validators.required, Validators.minLength(2)]]
  })
}
```

Figura 60. Campos del formulario de registro.

Como se puede observar en la *Figura 61*, ha sido necesario implementar un método para obtener los errores de control del formulario. Si los datos se validan correctamente la propiedad *valid* del formulario quedará establecida a true.

```
submitForm() {
  this.isSubmitted = true;
  if (!this.ionicForm.valid) {
    return false;
  } else {
    this.register();
  }
}
```

Figura 61. SubmitForm().

En la *Figura 62* se muestra el método que contiene la petición http POST que será enviada al servidor junto con los datos introducidos por el usuario, si dichos datos son correctos. Acto seguido se redirigirá al usuario a la página del inicio de sesión.

```
register() {

  const httpOptions = {
    headers: new HttpHeaders({ 'Content-Type': 'application/json' })
  }

  let body = {
    usuario: this.ionicForm.value['usuario'],
    nombre: this.ionicForm.value['nombre'],
    email: this.ionicForm.value['email'],
    password: this.ionicForm.value['password'],
    rol: "USUARIO"
  };

  this.httpClient.post(this.host + '/usuarios/nuevoUsuario', JSON.stringify(body), httpOptions)
    .subscribe();
  this.router.navigateByUrl(this.url);
}
```

Figura 62. Register().

En la *Figura 63* se muestra un fragmento del formulario de la entrada de datos.

```
<!-- NOMBRE -->
<span class="error ion-padding" *ngIf="isSubmitted && errorControl.nombre.errors?.required">
  El nombre es requerido
</span>
<span class="error ion-padding" *ngIf="isSubmitted && errorControl.nombre.errors?.minlength">
  El nombre debería de contener mínimo 3 carácteres
</span>
<ion-item class="wrap-input" lines="full">
  <ion-label position="floating">Nombre Completo</ion-label>
  <ion-input class="input" formControlName="nombre" type="text"></ion-input>
</ion-item>
<!-- NOMBRE -->
```

Figura 63. Entrada nombre de usuario.

4.2.2.2- Envío de imágenes

En este apartado se va a mostrar como ejemplo el envío de creaciones [40].

En la *Figura 64* se muestra el código html del botón de subir una creación. Además este código solo permite que se visualicen archivos del tipo imagen para subirlos a la aplicación.

```
<input #fileInput type="file" accept="image/*" hidden multiple (change)="subirCreacion($event)"/>
<ion-tab-button  tab="color-palette" (click)="fileInput.click()">
  <ion-icon name="color-palette"></ion-icon>
  <ion-label></ion-label>
</ion-tab-button>
```

Figura 64. Botón subir creación.

Como se puede observar en la *Figura 65* se muestra el método *subirCreación* que permite hacer una petición http POST al servidor con una creación seleccionada por el usuario. Además destacar que previamente se hace una comprobación para que el usuario no suba un tipo de archivo distinto al tipo imagen.

```
async subirCreacion(event) {
  if (!(event.target.files[0].type.match(/image.*/))) {
    alert('No puede subir archivos que no sean imagenes');
    return;
  }
  // Create a form data object using the FormData API
  let formData = new FormData();
  // Get a reference to the file that has just been added to the input
  console.log('ARCHIVO: ' + event.target.files[0]);
  formData.append('file', event.target.files[0]);
  formData.append('titulo', "tituloPrueba");
  formData.append('descripcion', "descripcionPrueba");
  formData.append('categoria', "Mixta");
  formData.append('licencia', "Reconocimiento");
  // POST formData to server using HttpClient
  this.http.post(this.host+'/usuarios/creacion', formData).subscribe((res: any) => {
    setTimeout(() => {
      this.reloadCurrentPage();
    }, 2000);
  },
  );
}
```

Figura 65. Subir Creación.

4.2.2.2- Enrutamiento de vistas

En este apartado se va a detallar cómo funciona el enrutamiento en angular [41,42], el cual ha sido utilizado para navegar entre las distintas creaciones de los usuarios, los distintos usuarios, categorías de creaciones, etc.

En la *Figura 66* se muestra el enrutamiento que se hace en el perfil del usuario para visualizar una creación suya. Para ello el usuario pulsa en una creación se llamará al método *verImagen(i).* Dicho método obtiene la imagen como parámetro para redirigir al usuario a la página donde se visualizará dicha imagen, la cual se le pasa a la componente de destino como *state*.

```
verImagen(i) {
  this.nav.navigateForward("/galeria", { state: i });
}
```

Figura 66. Ver Imagen.

Como se puede observar en la *Figura 67*, se muestra el código del html dónde se llama al método visto en la *Figura 66*.

```
<ion-row>
  <ion-col size="4"  *ngFor="let i of user.creaciones">
    <img class="post-item bg-image" height="60" width="110" src="{{host}}/images/{{i.fileName}}" (click)=verImagen(i)/>
  </ion-col>
</ion-row>
```

Figura 67. Html Ver Imagen.

Para obtener la imagen pasada como *state* al navegar a otra página es necesario declarar en el componente *app-routing.module.ts* el path de la ruta a la que se va a enviar dicha imagen. Como se puede observar en la *Figura 68* se muestran las rutas a las páginas dónde se envía el objeto para recogerlo y hacer las operaciones oportunas con él en la página de destino.

```
const routes: Routes = [
  {
    path: 'galeria/:nombreImagen',
    component: GaleriaPageModule

  },{
    path: 'categoria/:tipoCategoria',
    component: CategoriaPageModule

  },{
    path: 'usuario/:usuarioB',
    component: UsuarioPageModule

  },{
    path: 'lista/:valor',
    component: ListaPageModule

  },
```

Figura 68. Routes.

Por último en la *Figura 69* se muestra un fragmento del constructor de la página galería en la que se obtiene una imagen de otra página de origen mediante la propiedad *router.getCurrentNavigation().*

```
if (router.getCurrentNavigation().extras.state) {
  this.pageName = this.router.getCurrentNavigation().extras.state;
  console.log(this.pageName)
}
this.creacion = this.pageName;
this.nImagen = this._Activatedroute.snapshot.paramMap.get['nombreImagen'];
```

Figura 69. Obtener creación.

4.2.2.3- Creación de menús desplegables

En este apartado se va a mostrar cómo se han creado los distintos menús desplegables de la aplicación. En la *Figura 70* se muestra el método relacionado con la implementación del menú desplegable. Dicho método está constituido por un *actionSheet*, en el cuál se definen los nombres e iconos de las múltiples opciones del menú y también su funcionamiento.

```
openActionSheet() {
  this.actionSheet.create({
    header: 'Opciones',
    cssClass: 'my-custom-class',
    buttons: [{
      text: 'Eliminar creacion',
      icon: 'trash',
      handler: () => {
        this.botonEliminar();
      }
    }, {
      text: 'Editar creacion',
      icon: 'brush',
      handler: () => {
        this.verImagen(this.creacion);
      }
    }, {
      text: 'Cancelar',
      icon: 'close',
      role: 'cancel',
      handler: () => {
      }
    }]
  }).then(res => {
    res.present();
  });
}
```

Figura 70. OpenActionSheet.

4.2.2.4- Visualización opcional

En este apartado se va a mostrar cómo se ha gestionado la aparición del botón de seguir o dejar de seguir a un usuario.

En la *Figura 71* se muestra un fragmento de código html que contiene las opciones para mostrar un botón u otro.

```
<div *ngIf="seguido; then unfollowButton else followButton">
  button renders here
</div>
<ng-template #followButton>
  <ion-button mat-button fill="outline" size="small" (click)="follow()">Seguir</ion-button>
</ng-template>

<ng-template #unfollowButton>
  <ion-button mat-button fill="outline" size="small" (click)="unfollow()">Dejar de seguir</ion-button>
</ng-template>
</div>
```

Figura 71. Comprobar usuario seguido.

Para ello se utiliza el método *comprobarSeguido* que se muestra en la *Figura 72*. Dicho método comprueba si el usuario coincide con los usuarios seguidos por el usuario.

```
public comprobarSeguido() {
  if (this.usuarios.length > 0) {
    for (let usuario of this.usuarios) {
      if (this.usuarioB.usuario == usuario.usuario) {
        this.seguido = true;
      }
    }
  }
}
```

Figura 72. Comprobación de usuario seguido.

4.2.3 Generación de la aplicación en Android

En este apartado se van a mostrar los pasos necesarios para generar la aplicación en Android [43].

Para poder exportar la aplicación a el archivo .apk es necesario ejecutar el comando *ionic cordova build android --verbose* (verbose es una opción para depurar la exportación). Para esto es necesario el framework de cordova y node.js, ya que lo necesita cordova. Además también se necesita tener instalado gradle. Una vez hecho utilizado el comando anterior, ionic exportará el archivo .apk. Este archivo es necesario para poder compilarlo con Android Studio ya que ahora mismo no funcionará en Android.

En la *Figura 73* se muestra el archivo *network_security_config.xml* de la aplicación una vez importada a través del archivo .apk anterior en Android Studio. En dicho archivo es necesario indicar la dirección ip del servidor.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">192.168.1.131</domain>
    </domain-config>
</network-security-config>
```

Figura 73. Network_security_config.xml.

En la *Figura 74* se muestra el archivo *AndroidManifest.xml*. En este archivo es necesario declarar las dos líneas de código que se observan en la *Figura 74*, ya que se tratan de permisos que necesita la aplicación para funcionar correctamente.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 74. AndroidManifest.xml.

Por último para obtener el archivo .apk necesario para instalarlo en Android, hubo que compilar el proyecto y generar el archivo .apk en el apartado build.

5.- Pruebas

En este apartado se va a mostrar un resumen sobre las pruebas realizadas siguiendo el plan de pruebas diseñado anteriormente en el apartado 3.5.- Pruebas.

Para la realización de las pruebas en el servidor se ha hecho uso de la herramienta Postman y, de forma complementaria, pruebas manuales sobre la interfaz de usuario web usando el navegador para comprobar el cumplimiento de los criterios de aceptación. En dicha aplicación se realizó un registro creando cada una de las peticiones http CRUD que se le hicieron al servidor para comprobar si todo estaba correcto o tenía algún fallo.

Al hacer dicha petición se comprobaba que Postman devolviera un OK 200. Además también se comprobaba si la petición http CRUD había hecho su funcionamiento correcto respecto a las operaciones en la base de datos.

Todo esto se repetía por cada nueva petición http CRUD para comprobar su funcionamiento correcto.

En la *Figura 75* se muestra la colección de peticiones http CRUD que se tenían en Postman para hacer pruebas.

Figura 75. Registro peticiones Postman.

Como mínimo se han tenido que realizar en total 38 pruebas, ya que se han tenido 19 peticiones http CRUD con distintas funcionalidades, para las cuales había que hacer como mínimo 19 pruebas con resultados positivos y 19 con resultados negativos.

Gracias al registro de las peticiones http CRUD realizadas en Postman, el total de pruebas que se han hecho ha sido un total de 443 pruebas. Las cuales han permitido verificar todos los escenarios de las historias de usuario en relación a los criterios de aceptación de las mismas.

Un ejemplo de una prueba realizada en Postman fue la de comprobar la petición http POST de subir una creación por parte del usuario. Para ello previamente se tuvo que comprobar que se hacía correctamente la petición http POST del registro del usuario, acto seguido la petición http POST del inicio de sesión del usuario y finalmente dicha petición de subir la creación del usuario.

En la *Figura 76* se puede observar la petición http POST de subir una creación con un mensaje de Ok 200, indicando que dicha petición se ha realizado correctamente.

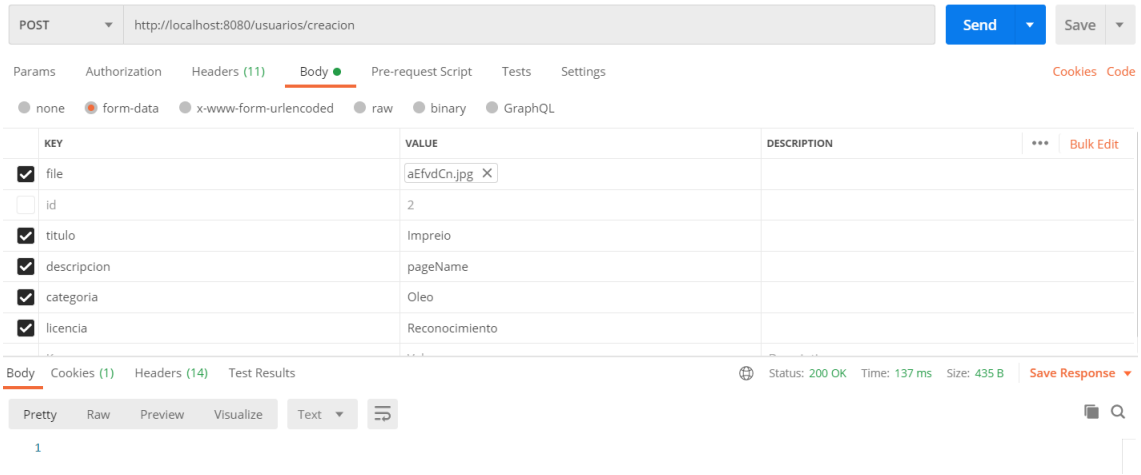

Figura 76. Petición http POST de subir creación.

Otro ejemplo de una prueba realizada en Postman fue el de comprobar la petición http GET de obtener el perfil del usuario. Para ello previamente se tuvo que comprobar que se hacía correctamente la petición http POST del registro del usuario, acto seguido la petición http POST del inicio de sesión del usuario y finalmente dicha petición de obtener su perfil.

En la *Figura 77* se puede observar la petición http GET de obtener el perfil del usuario con un mensaje de Ok 200, indicando que dicha petición se ha realizado correctamente. Además también se pueden observar sus atributos y creaciones subidas.

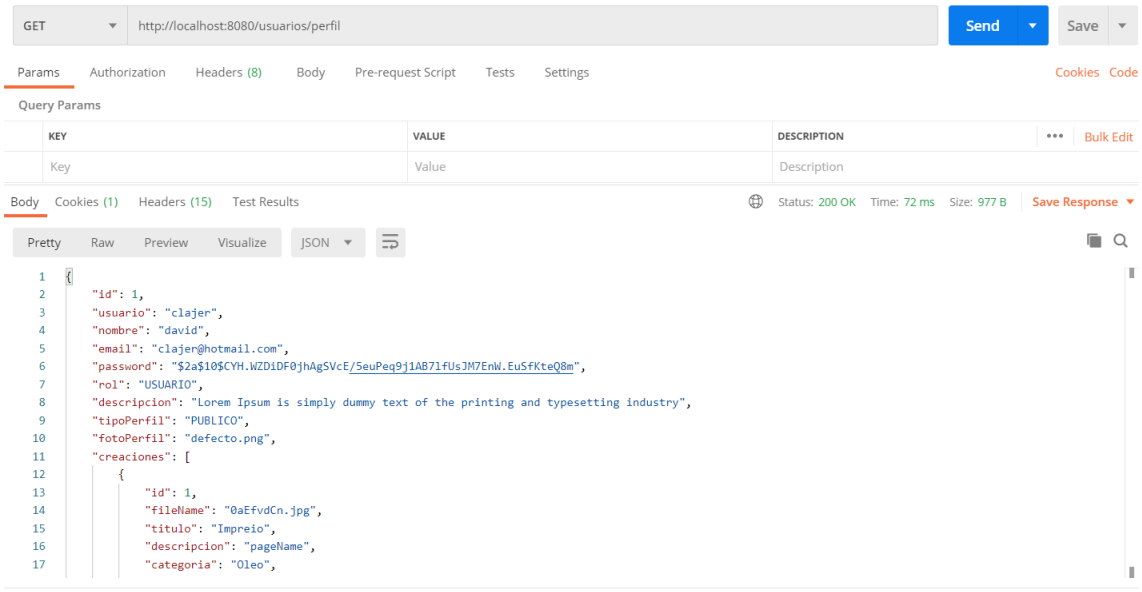

Figura 77. Petición http GET de obtener el perfil del usuario.

También hubo pruebas que ayudaron a detectar/corregir algunos errores. A continuación se mostrarán algunos de los resultados más relevantes obtenidos con estas pruebas:

- Al hacer la prueba para comprobar el correcto funcionamiento de la petición http POST de subir una creación, se comprobó que se duplicaba infinitamente el usuario y su creación. Esta prueba permitió dar con dicho error, el cual se trataba de un problema relacionado con las relaciones y atributos de la base de datos en las clases de las entidades.
- Al hacer la prueba para comprobar el correcto funcionamiento de la petición http POST de seguir a un usuario y la prueba de la petición http DELETE de dejar de seguir a unusuario, se comprobó que al hacerla un par de veces daba error. Esta prueba permitió dar con dicho error, el cual estaba relacionado con algunos atributos de las entidades y con las operaciones sobre las consultas de la base de datos.

Para la realización de las pruebas que se han ido realizando para el correcto funcionamiento de la aplicación móvil, se ha utilizado el comando de Ionic: *ionic serve,* el cual permite desplegar la aplicación en un navegador web, en este caso en Google Chrome.

Como mínimo se han realizado aproximadamente un total de 250 pruebas de forma manual sobre la interfaz de usuario web usando el navegador. A través de Google Chrome se ha seleccionado la opción de inspeccionar para comprobar en todo momento si surgía un error a través de la propia consola de Google Chrome y también revisando los distintos menús de la herramienta, como el apartado de Network. En este apartado se comprobaba si las peticiones se realizan correctamente o presentaban algún error.

Una vez exportada la aplicación y ejecutada con Android Studio, se han hecho pruebas para comprobar su correcto funcionamiento ejecutando el emulador que trae la aplicación de Android. De esta forma si ha podido comprobar si todo funcionaba correctamente o presentaba algún error. También usando la funcionalidad de Google Chrome: *inspect with Chrome Developer Tools*. Dicha funcionalidad se trata de una herramienta que permite depurar el funcionamiento de la aplicación ejecutándose en Android desde un navegador de Google Chrome en el equipo.

6.- Conclusiones

A modo de conclusión, comenzaré justificando el grado de cumplimiento de los objetivos inicialmente propuestos:

Se ha creado un prototipo de aplicación funcional que implementa las historias de usuario más importantes inicialmente planteadas. También, se ha conseguido diseñar un sistema informático especialmente orientado a la utilización de arquitecturas y metodologías de desarrollo web porque se ha hecho la implementación de un servidor mediante el Framework de Spring, el cual implementa un servicio REST con diferentes funcionalidades de tipo CRUD.

En el lado del cliente se ha utilizado el Framework Ionic para el desarrollo de una aplicación móvil multiplataforma ya que se podría utilizar tanto en dispositivos Smartphone como en ordenador (para este último no se ha generado ninguna aplicación cliente como tal aunque podría hacerse fácilmente si fuera necesario).

Para ello se ha realizado previamente un estudio de necesidades y soluciones existentes para el sistema propuesto.

De esta forma se ha obtenido un prototipo de aplicación totalmente operativa. No obstante, no se han podido implementar todas las historias de

usuario inicialmente propuestas como se justificado adecuadamente en el apartado 2.4.2.- Evolución de la planificación inicial. Sin embargo, gracias a la metodología de trabajo iterativo planteada y a la priorización inicial de las historias de usuario me resultó fácil adaptarme a los cambios que han sido necesarios introducir para conseguir al menos alcanzar las funcionalidades más relevantes.

Desde un punto de vista personal, me ha parecido muy interesante trabajar en la integración del servidor con la base de datos ya que me ha parecido muy útil de cara al mercado laboral, además también me ha parecido interesante contar con mecanismos de seguridad como el CORS y de sistemas de autenticación bastantes sólidos.

Este proyecto lo he realizado porque tenía bastante interés en aprender a crear una aplicación en Android desde 0 a través de los Frameworks y lenguajes empleados para aumentar mis conocimientos sobre ellos. Esto se debe a que dichos Frameworks son bastantes utilizados en el mercado laboral y me ha parecido una buena oportunidad para aprender a utilizarlos y ganar experiencia al trabajar con ellos.

En relación a los conocimientos adquiridos, el desarrollo del TFG me ha permitido profundizar en el desarrollo de aplicaciones web, en la gestión de la base de datos y en la planificación de proyectos, entre otras. Además de la oportunidad de aprender a utilizar nuevas tecnologías como Ionic y Angular.

7.- Mejoras y trabajos futuros

En este apartado se van a comentar algunas cuestiones sobre algunas de las funcionalidades no implementadas así como posibles mejoras de las existentes o desarrollos futuros que se podrían llevar a cabo

En relación a las historias de usuario iniciales, algunas fueron descartadas. Concretamente las historias de usuario de enviar mensajes y la de buscar creaciones. Respecto a la historia de usuario de enviar mensajes ya se comentó previamente en el apartado 2.4.- Planificación temporal, los motivos por los que fue descartada. Respecto a la historia de usuario de buscar creaciones se descartó porque al incorporar la visualización de creaciones por categorías y además con la visualización de los perfiles de los distintos usuarios en la aplicación, se consideró que no tendría mucha utilidad actualmente.

En la aplicación se tiene implementada la historia de usuario tipo de perfil de un usuario. Esta funcionalidad funciona correctamente de forma que el usuario puede cambiar su perfil de público a privado y viceversa en la aplicación, además los cambios se ven reflejados en la base de datos. Sin embargo, esta funcionalidad no se ha tenido en cuenta para restringir la visualización de los perfiles privados a los demás usuarios. Por lo tanto, podría ser interesante implementar esta funcionalidad y también la de restringir las peticiones de seguimiento dependiendo del perfil de un usuario, de cara al futuro.

En la aplicación se han dejado han dejado algunos botones o aspectos de la interfaz que no funcionan por si se decidiesen implementar en un futuro:

- Botón de home, botón de notificaciones, botón corazón que representa los “me gusta” de las creaciones, botón el botón de compartir creaciones, botón de guardar creaciones y un apartado para poder visualizarlas más fácilmente.

Otras funcionalidades interesantes que podrían implementarse en un futuro son:

- Incorporar a la aplicación otro idioma, como por ejemplo el inglés, de forma que el usuario pueda seleccionar el idioma que desee de los implementados. De esta forma la aplicación podría llegar a más usuarios.
- Valorar creaciones, ya que podría crear una funcionalidad para votar las mejores creaciones del mes respecto a la cantidad de “me gusta” que tengan. Además las creaciones públicas y mejor valoradas se podrían mostrar como fondo en la pantalla de inicio de sesión de la aplicación, siempre y cuando lo permitan respecto al tipo de licencia asignada por el usuario.

Por último destacar que los diseños propuestos tanto del Back-End como del Front-End pueden ayudar a su implementación en el futuro porque presentan una base bastante sólida de un proyecto prometedor, ya que actualmente hay muy pocas aplicaciones destinadas al uso de los artistas.

Además este proyecto podría abrir puertas a otros proyectos de redes sociales destinadas para un público en concreto, por ejemplo, resultaría fácil adaptar este proyecto a una red social destinada a la gente que le guste cocinar.

8.- Bibliografía

[1] *Qué es SCRUM.* Proyectosagiles.org. [Online] Disponible en: https://proyectosagiles.org/que-es-scrum/

[2] *Lista de objetivos / requisitos priorizada (Product Backlog)*. Proyectosagiles.org. [Online] Disponible en: https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog/

[3] C. Simões (julio, 2020). *MoSCoW. ¿Qué es y cómo priorizar en el desarrollo de tu aplicación?* [Online] Disponible en: https://www.itdo.com/blog/moscow-que-es-y-como-priorizar-en-el-desarrollo-de-tu-aplicacion/

[4] A. Martínez. Tipos de aplicaciones móviles: Nativas, web e híbridas. FutureSpace. [Online] Disponible en: https://www.futurespace.es/tipos-de-aplicaciones-moviles/

[5] J. Thornsby (diciembre, 2016). *Java vs. Kotlin: ¿Deberías Usar Kotlin en Desarrollo Android?* [Online] Disponible en: https://code.tutsplus.com/es/articles/java-vs-kotlin-should-you-be-using-kotlin-for-android-development--cms-27846

[6] *Swift (lenguaje de programación)*. (junio, 2021). Wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/Swift_(lenguaje_de_programaci%C3%B3n)

[7] J. Agüero y R. Maluenda (febrero, 2021). Qué es Ionic: ventajas y desventajas de usarlo para desarrollar apps móviles híbridas. [Online] Disponible en: https://profile.es/blog/que-es-ionic/

[8] *Flutter (software)*. (julio, 2021). Wikipedia. [Online] Disponible en: https://es.m.wikipedia.org/wiki/Flutter_(software)

[9] X. Quesada (junio, 2009). *Introducción a la estimación y planificación ágil*. Proyectosagiles.org. [Online] Disponible en: https://bit.ly/3fzSZUO. Acceso: Recuperado en Jun. 8, 2009.

[10] *10 Técnicas de estimación de software*. (agosto, 2018). PMOinformatica.com. [Online] Disponible en: http://www.pmoinformatica.com/2018/08/tecnicas-estimacion-software.html

[11] Manz. Política CORS. [Online] Disponible en: https://lenguajejs.com/javascript/peticiones-http/cors/

[12] *¿Cuánto se gana en España de Analista programador/a?* Indeed. [Online] Disponible en: https://es.indeed.com/career/analista-programador/salaries

[13] *Modelo de dominio*. (enero, 2020). Wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/Modelo_de_dominio

[14] *Principios de diseño de interfaces*. (abril. 2014). Augmented Reality [Online] Disponible en https://augrealityhci.wordpress.com/2014/04/09/principios-de-diseno-de-interfaces/

[15] J. F., Canté (2017). Psicología del color aplicada a los cursos virtuales para mejorar el nivel de aprendizaje en los estudiantes. *grafica, 5*(9), 51-56.

[16] *Qué es un diagrama entidad-relación*. Lucidchart. [Online] Disponible en: https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion

[17] J. A. Muro. *¿Qué es un ORM?* Deloitte. [Online] Disponible en: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html

[18] *Diagrama de clases*. (julio, 2021). Wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_clases

[19] *CRUD*. (junio, 2021). Wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/CRUD

[20] O. Blancarte. (diciembre, 2018). *Data Access Object (DAO) Pattern.* [Online] Disponible en: https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/

[21] O. Blancarte. *Data Transfer Object (DTO).* [Online] Disponible en: https://reactiveprogramming.io/blog/es/patrones-arquitectonicos/dto

[22] *Diagrama de secuencia*. (junio, 2021). Wikipedia. [Online] Disponible en: https://es.wikipedia.org/wiki/Diagrama_de_secuencia

[23] H. AlMawali. (agosto, 2020). Youtube. [Online] Disponible en: https://www.youtube.com/watch?v=MuJ47_t5H_E

[24] J. Cabrera. (diciembre, 2020). *Qué es TypeScript.* [Online] Disponible en: https://openwebinars.net/blog/que-es-typescript/

[25] Y. Muradas. (febrero, 2020). *Qué es Gradle: La herramienta para ser más productivo desarrollando.* [Online] Disponible en: https://openwebinars.net/blog/que-es-gradle/

[26] Baeldung. (enero, 2021). *Documenting a Spring REST API Using OpenAPI 3.0.* [Online] Disponible en: https://www.baeldung.com/spring-rest-openapi-documentation

[27] C. Álvarez. (junio, 2018). *Swagger documentando nuestro API REST.* [Online] Disponible en: https://www.arquitecturajava.com/swagger-documentando-nuestro-api-rest/

[28] Baeldung. (septiembre, 2020*). Eager/Lazy Loading In Hibernate.* [Online] Disponible en: https://www.baeldung.com/hibernate-lazy-eager-loading

[29] Ozgur. (abril, 2017). *JsonManagedReference vs JsonBackReference.* [Online] Disponible en: https://www.it-swarm-es.com/es/java/jsonmanagedreference-vs-jsonbackreference/1054975278/

[30] Baeldung. (julio, 2020). *Working with Lazy Element Collections in JPA.* [Online] Disponible en: https://www.baeldung.com/java-jpa-lazy-collections

[31] C. Jayatilake. (octubre, 2019). *File Upload & Download as Multipart File using Angular + Spring Boot.* [Online] Disponible en: https://medium.com/linkit-intecs/file-upload-download-as-multipart-file-using-angular-6-spring-boot-7ad06d841c21

[32] N. Ha Minh. (julio, 2019). *Upload Files to Database with Spring MVC and Hibernate.* [Online] Disponible en: https://www.codejava.net/coding/upload-files-to-database-with-spring-mvc-and-hibernate

[33] Baeldung. (mayo, 2021). *CORS with Spring.* [Online] Disponible en: https://www.baeldung.com/spring-cors

[34] R. Fadatare. *Spring Boot CORS @CrossOrigin Example.* [Online] Disponible en: https://www.javaguides.net/2019/09/spring-boot-cors-crossorigin-example.html

[35] Baeldung. (agosto, 2020). *Retrieve User Information in Spring Security.* [Online] Disponible en: https://www.baeldung.com/get-user-in-spring-security

[36] Baeldung. (diciembre, 2020). *MaxUploadSizeExceededException in Spring.* [Online] Disponible en: https://www.baeldung.com/spring-maxuploadsizeexceeded

[37] IonicFramework. *Ionic Framework.* [Online] Disponible en: https://ionicframework.com/docs

[38] R. Fadatare. *Angular 8 + Spring Boot Basic Authentication Example.* [Online] Disponible en: https://www.javaguides.net/2019/08/angular-8-spring-boot-basic-authentication-example.html

[39] J. Salazar. (abril, 2017). *Validaciones avanzadas y personalizadas de formularios en Ionic 3 y Angular 4.* [Online] Disponible en: https://gist.github.com/salazarr-js/8227c717e790e59b11c2728d2a4f4c9b

[40] J. Morony. (octubre, 2020). *Handling File Uploads in Ionic.* [Online] Disponible en: https://eliteionic.com/tutorials/handling-file-uploads-in-ionic-web/

[41] Angular. Common Routing Tasks. [Online] Disponible en: https://angular.io/guide/router#accessing-query-parameters-and-fragments

[42] TekTutorialsHub. Angular Route Parameters: Passing Parameters to Route. [Online] Disponible en: https://www.tektutorialshub.com/angular/angular-passing-parameters-to-route/#retrieve-the-parameter-in-the-component

[43] RobotSolar. (enero, 2019). Youtube. [Online] Disponible en: https://www.youtube.com/watch?v=81bsliZCj7E

Apéndice I. Manual de Instalación del sistema

En este apéndice se va a detallar las la instalación necesaria de los distintos elementos necesarios para la ejecución del sistema.

Para ello es necesario instalar Java JDK[22], en este caso ha sido la versión 1.8. Como se muestra en la *Figura 78*, además hay que declarar la ruta en el PATH de Windows.

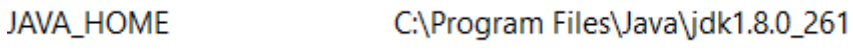

Figura 78. Java.

El proyecto del servidor, se puede lanzar mediante Spring Boot, a través de la línea de comandos desde Maven utilizando el comando: *$mvn spring-boot:run*[23].

Para utilizar la base de datos Apache Derby[24] es necesario descargarla previamente, concretamente la versión 10_14_2_0 que es la que se ha utilizado. También es necesario crear una conexión a dicha base de datos.

Para la aplicación cliente es necesario instalar Node.js[25] (necesario para angular), en este caso se ha utilizado la versión 14.16. Como se muestra en la *Figura 79*, es necesario añadir en la variable de entorno del PATH, la ruta de la carpeta npm.

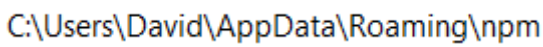

Figura 79. Npm.

Después hay que instalar los paquetes con el comando *npm install,* de tal forma que *npm* leerá el archivo *package.json* para comprobar las dependencias que tiene el proyecto e instalarlas.

Una vez hecho esto es necesario usar el comando *ionic serve* para transpilar el código TypeScript, de esta forma se ejecutará la aplicación en el navegador y generarán una serie de carpetas necesarias para el proyecto.

Acto seguido es necesario utilizar el comando *ionic cordova prepare* para restaurar las plataformas que están agregadas a la App y los plugins de Cordova

[22] https://www.oracle.com/
[23] https://www.baeldung.com/spring-boot-run-maven-vs-executable-jar
[24] https://db.apache.org/
[25] https://nodejs.org/es/

e Ionic. Además habrá que añadirlo al PATH tal y como se muestra en la *Figura 80*.

C:\Users\David\AppData\Roaming\npm\node_modules\cordova\bin

Figura 80. Cordova.

Para hacer la exportación de la aplicación a un archivo .apk es necesario descargar previamente Gradle[26], en este caso se ha utilizado la versión 7.0.2 y como se muestra en la *Figura 81*, hay que añadirlo al PATH.

GRADLE_HOME C:\Gradle\gradle-7.0.2

Figura 81. Gradle.

El siguiente paso consiste en utilizar el comando: *ionic cordova build android –verbose.* Dicho comando permite compilar y exportar la aplicación a un archivo .apk mostrando toda la información detallada por consola. De esta forma si ocurriese algún error, se puede comprobar fácilmente.

El archivo .apk generado anteriormente, no funcionará de momento en un dispositivo móvil, para ello es necesario hacer una serie de pasos previos:

- En primer lugar, hay que instalar Android Studio[27]. Para ello, se ha utilizado la versión 4.2.1. Una vez realizada la instalación, es necesario añadir al PATH las rutas que se muestran en la *Figura 82*.

ANDROID_HOME C:\Users\David\AppData\Local\Android\Sdk

C:\Users\David\AppData\Local\Android\Sdk\emulator

C:\Users\David\AppData\Local\Android\Sdk\tools\gradle\bin

Figura 82. Rutas.

- Como se muestra en la *Figura 83*, es necesario indicar la dirección ip del servidor en el archivo *network_security_config.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">192.168.1.131</domain>
    </domain-config>
</network-security-config>
```

Figura 83. Dirección IP.

[26] https://gradle.org/releases/
[27] https://developer.android.com/studio

- También se requiere incluir en el archivo *network_security_config.xml* los permisos de *internet* y *acces_network_state,* como se puede observar en la *Figura 84*.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Figura 84. Permisos.

Por último, hay que compilar el proyecto y, posteriormente, generar el archivo .apk en el apartado build de la aplicación. Una vez realizado este procedimiento, ya se podría instalar la aplicación en un dispositivo Android.

Apéndice II. Manual de Usuario

En este apéndice se van a detallar las funcionalidades implementadas de la aplicación.

Al iniciar la aplicación se muestra el inicio de sesión. En el caso de que no se tenga cuenta de usuario en la aplicación, habrá que pulsar en registrarse tal y como se puede observar en la *Figura 85*.

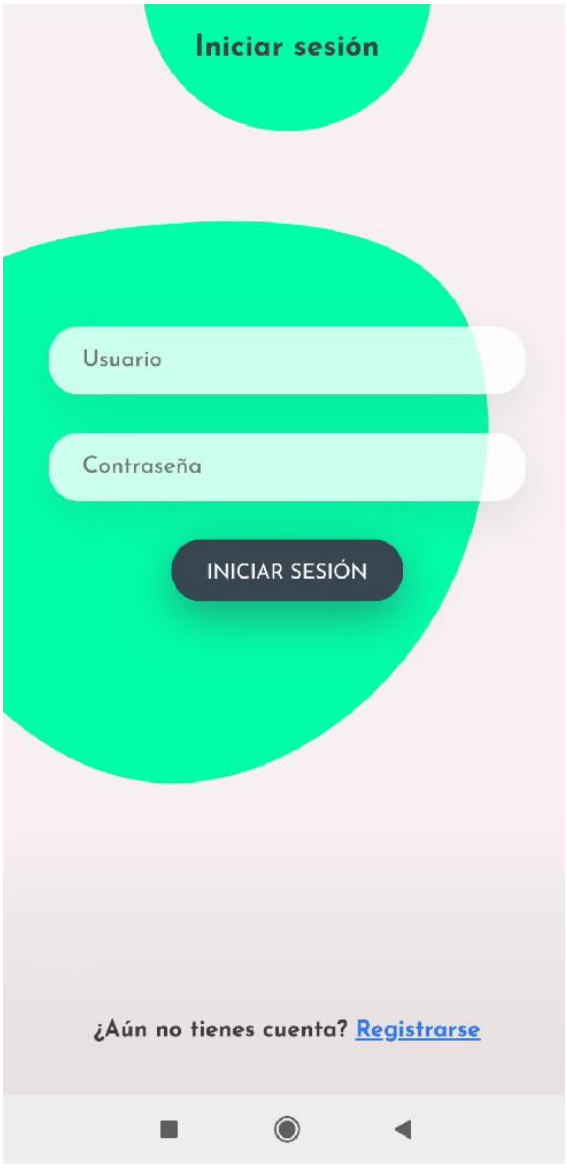

Figura 85. Login.

Para registrarse en la aplicación, se tendrá que rellenar el formulario que se muestra en la *Figura 86*. En dicho formulario se controla la longitud mínima de los campos, el formato correcto del email y que no se dejen campos vacíos. Cuando un usuario se registra en la aplicación se le redirigirá a la página que se

ha visto anteriormente de iniciar sesión. Si el usuario se equivoca al pulsar en registrarse, podrá volver a la página del inicio de sesión pulsando Inicia sesión.

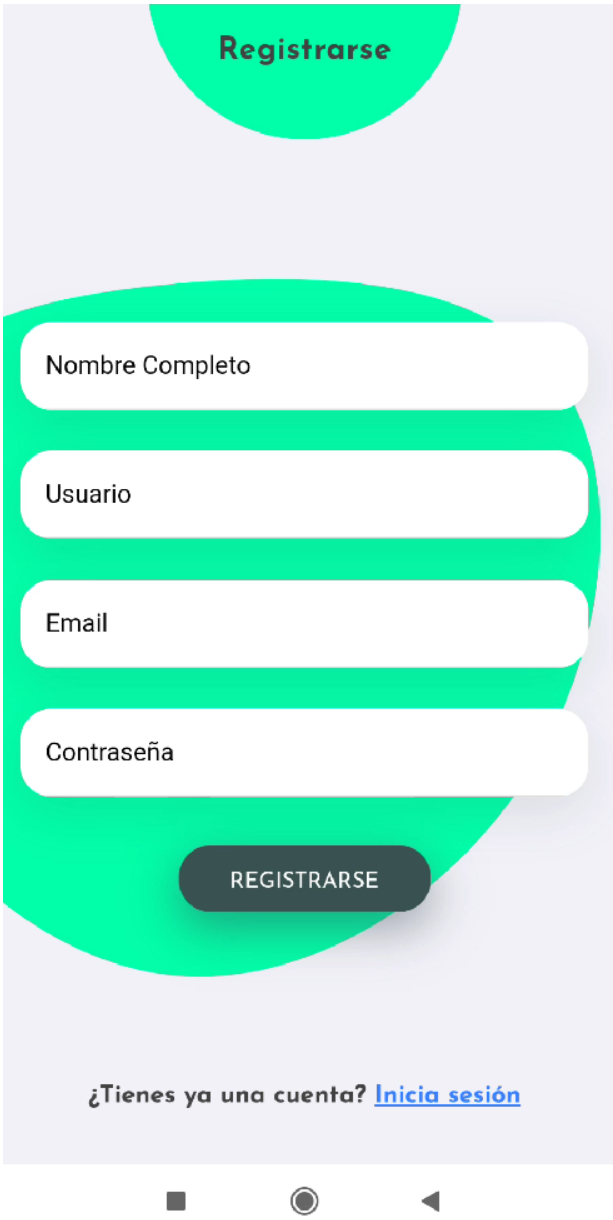

Figura 86. Registro.

Cuando un usuario inicia sesión, se controla que sus credenciales sean válidas. Si las credenciales del usuario son válidas se le redirigirá a su perfil, como se puede observar en la Figura 87. En el caso contrario, se muestra un mensaje indicando que dichas credenciales son inválidas.

En la aplicación los botones de la barra inferior que se muestran en la Figura 87, están disponibles en toda la aplicación. Dichos botones permiten ir a al buscador (lupa), subir una creación (paleta), ir al perfil del usuario (persona),

ir a ajustes (herramientas). El botón de ir a ajustes no está disponible en el perfil del usuario, ya que hay un botón específico para ir a esa página, llamado editar perfil.

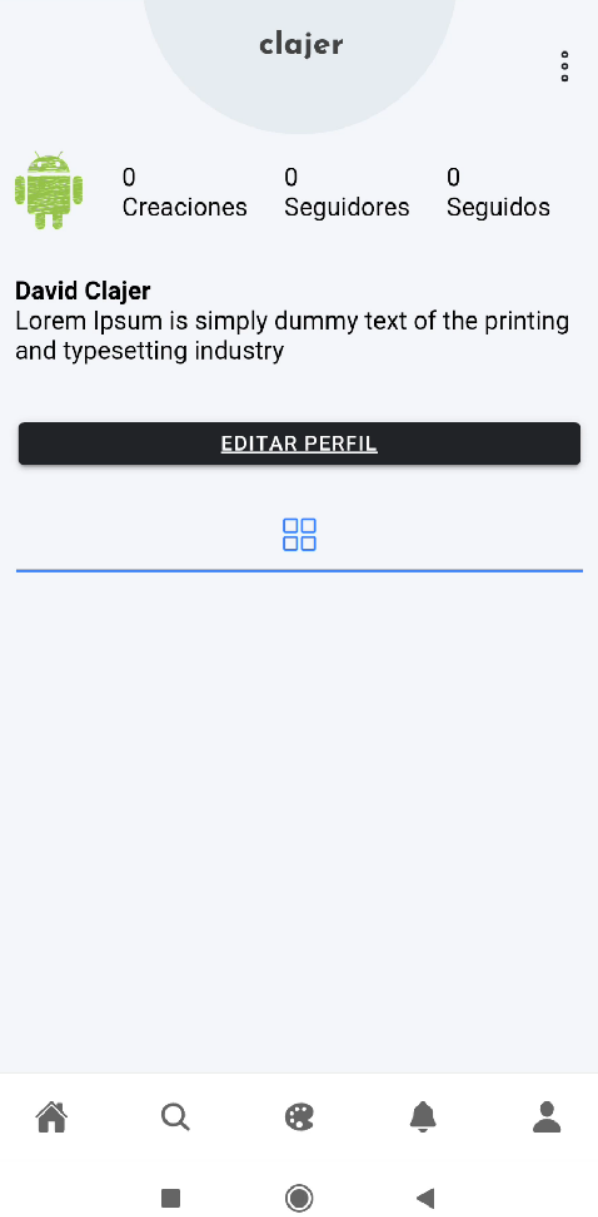

Figura 87. Perfil.

En la *Figura 88* se muestra el perfil del usuario con una creación subida. En el perfil del usuario se muestran sus creaciones subidas, su biografía, su nombre, su nombre de usuario, su imagen de perfil, el número de creaciones subidas, el número de usuarios seguidores y el número de usuarios seguidos. Además se tiene un botón de tres puntos en la parte superior derecha de la pantalla en el que al pulsarlo, el usuario podrá cerrar la sesión actual.

Figura 88. Perfil con creación.

En la *Figura 89* se muestra un formulario para editar los atributos de la creación subida. Para ello hemos tenido que pulsar previamente en la creación que se mostraba en la *Figura 88* y a continuación en un botón con tres puntos de la esquina superior derecha para que se muestre un menú desplegable, en el cual se habrá tenido que pulsar en editar creación.

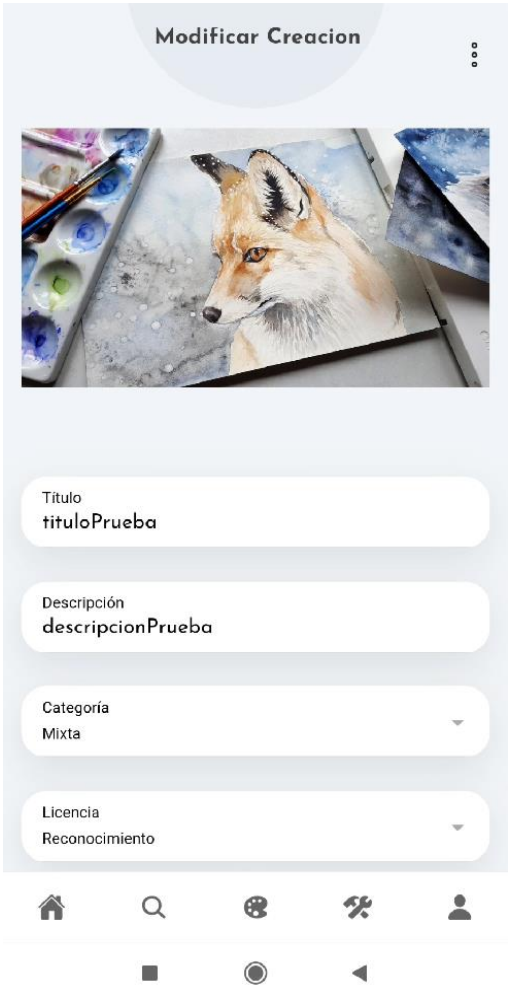

Figura 89. Editar creación.

En la Figura 90 se muestran los campos de la creación modificados. Si el usuario quiere guardar los cambios pulsará en guardar cambios, en caso contrario pulsará el botón volver para volver a la visualización de la creación. Además, destacar que en el formulario no se permiten dejar los campos vacíos, en caso de que sucediese, se le mostrará un mensaje de advertencia al usuario.

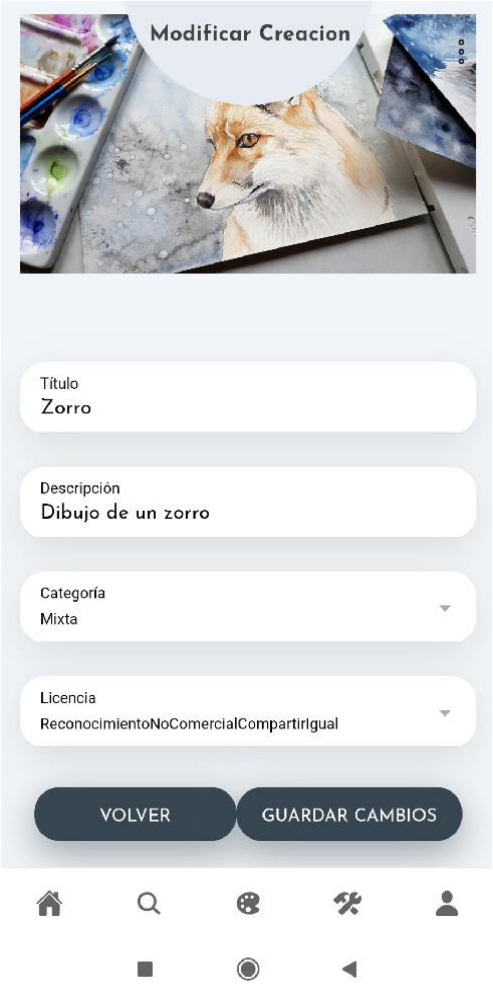

Figura 90. Creación editada.

En la *Figura 91* se muestra la visualización de una creación tras haber accedido a ella a través del perfil del usuario. En esta ventana el usuario puede comentar la creación, incluso si es su propia creación. El usuario también puede pulsar en la imagen dos veces para poder hacer zoom en ella y de estar forma visualizar los aspectos con más detalle. Para editar o eliminar la creación el usuario puede pulsar el botón de tres puntos que aparece en la esquina superior derecha, el cual, al pulsarlo, se muestra un menú desplegable con dichas opciones. Si el usuario quiere volver a su perfil puede hacerlo pulsando el la X que se muestra en la esquina superior izquierda.

Figura 91. Creación editada.

En la *Figura 92* se muestran los ajustes del usuario. Se puede acceder a dicha página desde el botón de las herramientas que se encuentra en la parte inferior de la pantalla o desde el perfil del usuario a través del botón editar perfil.

En esta página se puede cambiar la imagen de perfil del usuario, su nombre, su descripción, su email y su tipo de perfil (privado o público).

Figura 92. Ajustes.

En la *Figura 93* se muestran los datos del usuario modificados. Si el usuario quiere guardar los cambios pulsará en dicho botón. Además, destacar que en el formulario no se permiten dejar los campos vacíos, en caso de que sucediese, se le mostrará un mensaje de advertencia al usuario.

Figura 93. Ajustes editados.

En la *Figura 94* se muestra el buscador de usuarios y un carrusel de imágenes con las distintas categorías artísticas. Para acceder a esta ventana se ha tenido que pulsar previamente el botón de la lupa que aparece en la parte inferior de la pantalla.

Para buscar usuarios el usuario tendrá que introducir en el cuadro de texto el nombre del usuario a buscar y se le mostrará un listado con los nombres de usuario que coinciden con el que ha introducido.

El usuario puede visualizar las creaciones de una categoría artística, para ello el usuario navegará a través del carrusel de imágenes y seleccionará la categoría que le interese para visualizar las creaciones que se encuentren en dicha categoría.

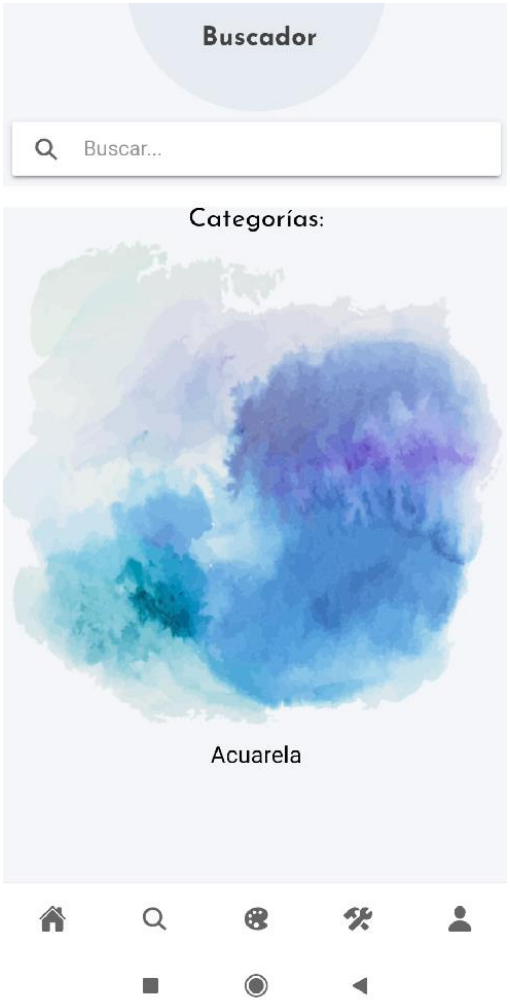

Figura 94. Buscador y categorías artísticas.

En la *Figura 95* se muestra un aviso informando al usuario de que hay creaciones en una categoría. Esto ocurre cuando en una categoría no hay creaciones con ese tipo.

Figura 95. Sin creaciones.

En la *Figura 96* se muestran las creaciones de una categoría. El usuario puede visualizar el perfil de los usuarios de las distintas creaciones pulsando en el nombre o la imagen de perfil de la creación de un usuario. Además, el usuario puede interactuar con otros usuarios comentando las creaciones de otros usuarios a través del icono del bocadillo de texto.

Figura 96. Categoría acuarela.

En la *Figura 97* se muestra cuando un usuario busca a otro usuario por el nombre de usuario. En el ejemplo se ve que escribiendo los caracteres vi, se le muestra al usuario el nombre de usuarios coincidentes, en este caso Vicent.

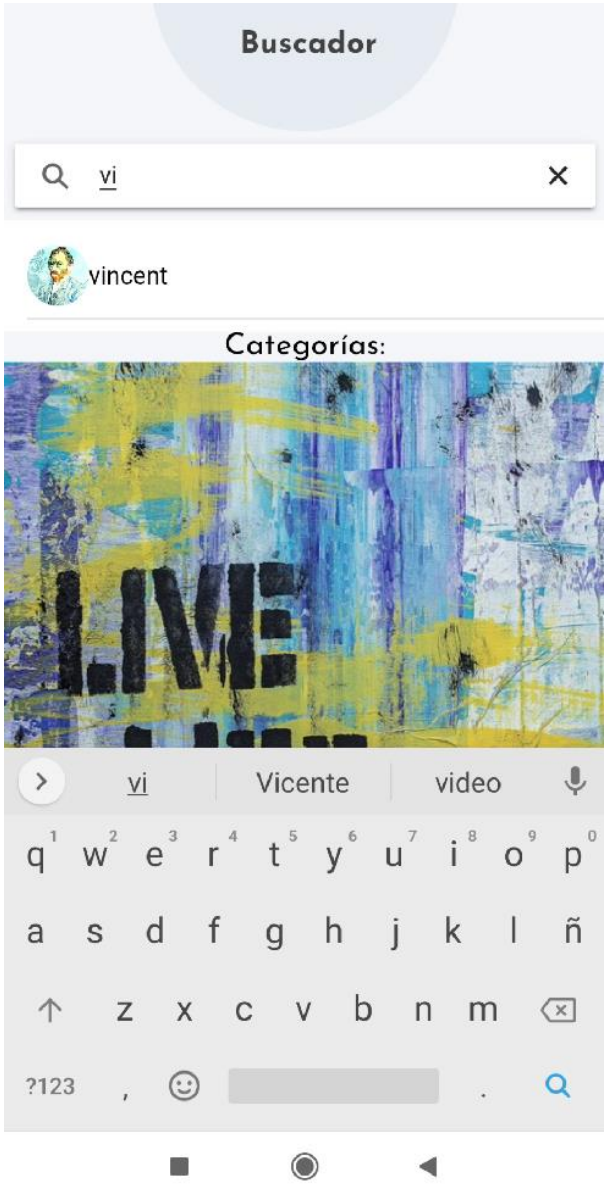

Figura 97. Búsqueda.

En la *Figura 98* se muestra el perfil del usuario Vincent al que el usuario busco y posteriormente pulso para acceder a su perfil. Cuando el usuario accede al perfil de un usuario buscado, este puede visualizar su perfil, en el cual puede visualizar las creaciones de dicho usuario. Además, el usuario puede seguir o dejar de seguir al usuario, dependiendo de si lo sigue actualmente o no.

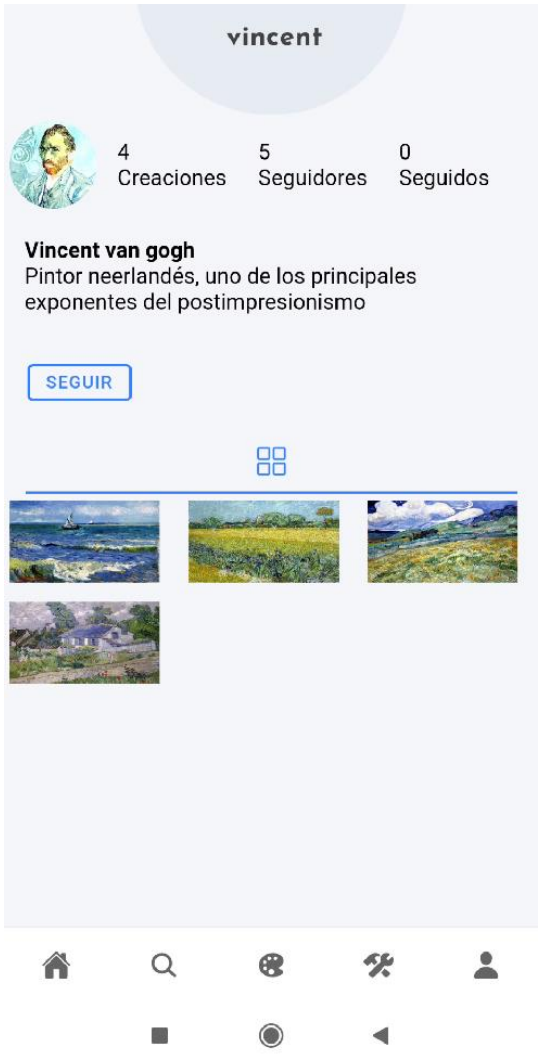

Figura 98. Perfil usuario Vincent.

En la *Figura 99* se muestra una creación seleccionada por el usuario del usuario Vincent, para poder visualizarla o comentarla.

Figura 99. Creación Vincent.

En la *Figura 100* se muestra el perfil del usuario cuando tiene al menos una creación subida, un usuario siguiéndole y siguiendo a un usuario. El usuario podrá visualizar los perfiles de los usuarios que le siguen o sigue él, pulsando sobre seguidores y seguidos. Además, se muestra el listado de los usuarios seguidores y seguidos del usuario.

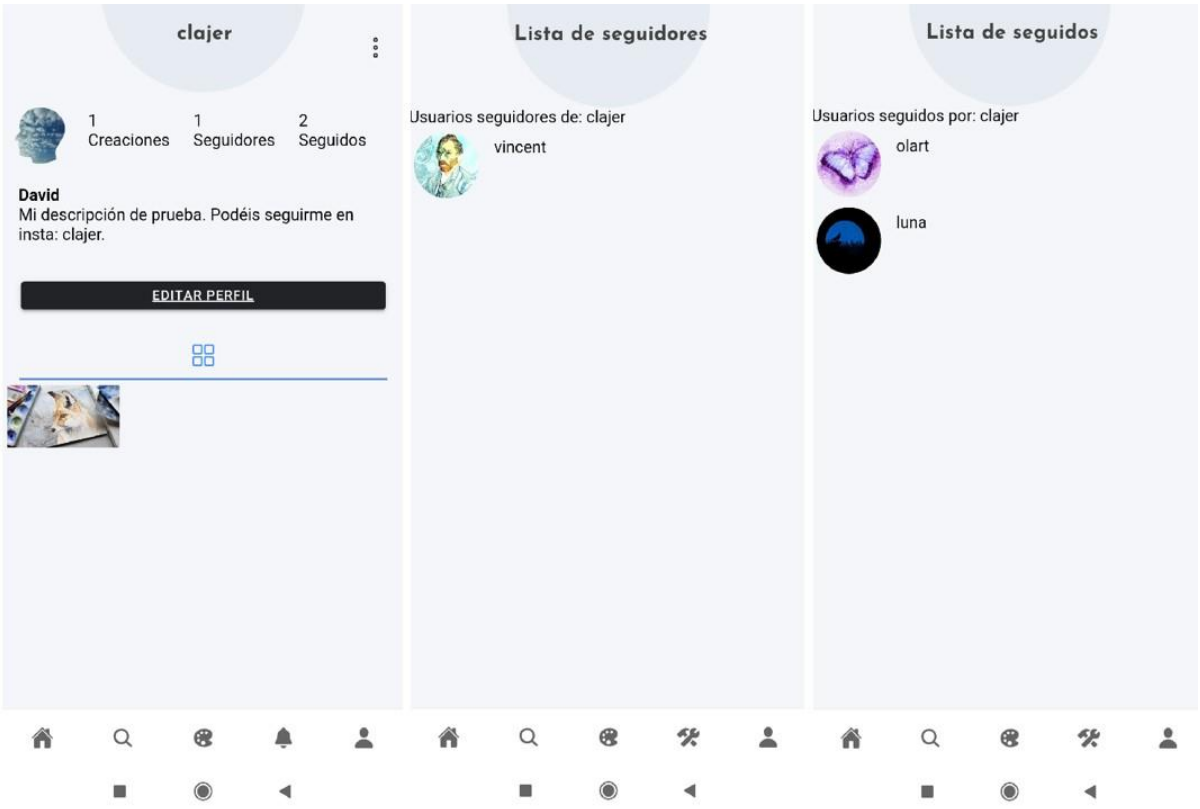

Figura 100. Perfil, seguidores y seguidos.

En la *Figura 101* se muestra la opción de eliminar la cuenta del usuario. Si el usuario quiere eliminar su cuenta, tendrá que acceder a ajustes y pulsar el botón de los tres puntos en la parte superior derecha de la pantalla. A continuación se le mostrará un desplegable con las opciones de cerrar la sesión actual, cancelar o eliminar su cuenta de usuario. En caso de que el usuario seleccione la opción del menú de eliminar cuenta, se le mostrará un mensaje de información comunicándole si está seguro de que quiere eliminarla, ya que se borrará toda la información asociada a su usuario. En caso de que la elimine, se redirigirá al usuario a la página del inicio de sesión y se habrán borrado todos sus datos.

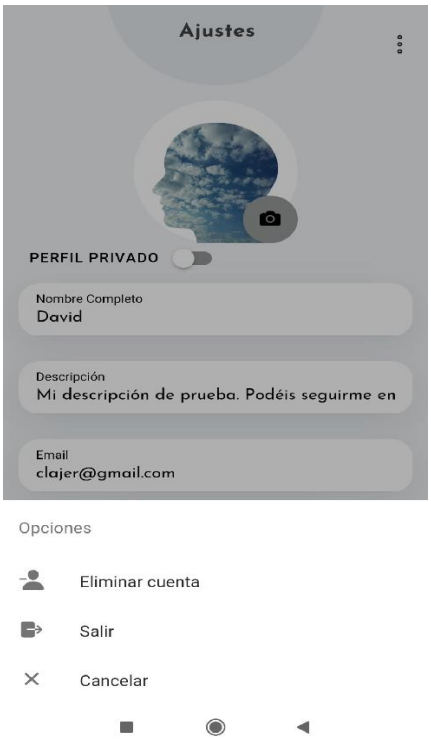

Figura 101. Eliminar Cuenta.

Apéndice III. Descripción de contenidos suministrados

En este apéndice se va a detallar la organización de la estructura de los contenidos complementarios suministrados junto al presente documento en un fichero comprimido. En los contenidos suministrados se pueden encontrar las siguientes carpetas:

- aplicacion_servidor_api_rest. Contiene el archivo .jar de la aplicación servidor.
- aplicacion_cliente_android. Contiene el archivo .apk de la aplicación cliente para dispositivos Android.
- fuentes_aplicacion_servidor. Contiene el proyecto de Netbeans de la aplicación servidor.
- fuentes_aplicacion_cliente. Contiene el proyecto VSCode de la aplicación cliente.
- diagramas. Contiene los distintos diagramas de diseño completos realizados con Visual Paradigm y los gráficos en Excel.
- documentación_swagger. Contiene la documentación del API REST generada mediante Swagger.