ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA DISTRIBUIDO DE GESTIÓN DE FOTOGRAFÍAS

SUBSISTEMA DE ADQUISICIÓN DE FOTOGRAFÍAS

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN TECNOLOGÍAS DE LA INFORMACIÓN

JENIFER CRISTINA JÁCOME ARAUZ

jenifer.jacome@epn.edu.ec

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE

david.mejia@epn.edu.ec

DMQ, marzo 2023

CERTIFICACIONES

Yo, JENIFER CRISTINA JÁCOME ARAUZ declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

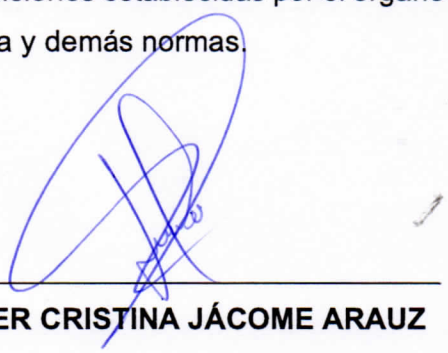
JENIFER CRISTINA JÁCOME ARAUZ

Certifico que el presente trabajo de integración curricular fue desarrollado por JENIFER CRISTINA JÁCOME ARAUZ, bajo mi supervisión.

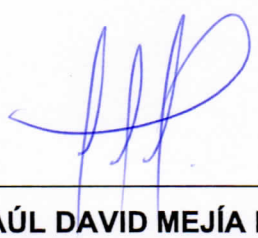
RAÚL DAVID MEJÍA NAVARRETE
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.



JENIFER CRISTINA JÁCOME ARAUZ



RAÚL DAVID MEJÍA NAVARRETE
DIRECTOR

DEDICATORIA

Este Trabajo de Integración Curricular está dedicado a mi familia, a mi amado esposo y en especial a mi abuelito, quien desde el cielo siempre me ayudo a salir adelante y me ayudó a convertirme en la mujer que soy.

AGRADECIMIENTO

Agradezco infinitamente a Dios, quién puso en mi camino a un hombre ejemplar, este hombre confía en mí y me apoya para salir adelante juntos. Es el mejor padre, hermano, hijo, esposo y amigo que puedo tener, ya que con su apoyo logré concluir este trabajo y esperó realizar mucho más.

ÍNDICE DE CONTENIDO

RESUMEN

En el presente Trabajo de Integración Curricular se presenta el desarrollo de un componente que es parte de un sistema distribuido y tiene como objetivo permitir al usuario el almacenar fotografías y clasificarlas mediante etiquetas. El subsistema fue desarrollado para el sistema operativo Android y consta de cuatro módulos: inicio de sesión, registro de usuario, consulta de fotografías, y subida de imágenes, así también dispone de un *stub* que permite emular el comportamiento del servidor.

El presente documento está conformado por tres capítulos.

En el primer capítulo se presentan los objetivos y el alcance; posteriormente se describen las principales características y componentes de las aplicaciones Android y se analiza las opciones para el manejo de imágenes en el sistema operativo Android, la definición de *stub* y la descripción de la metodología Kanban.

En el segundo capítulo se encuentra el diseño del subsistema para lo cual en primer lugar se realiza el análisis de requerimientos para la gestión y etiquetado de imágenes. Además, se indica el tablero Kanban empleado para el desarrollo del componente. Por último, se resume la implementación realizada del subsistema de acuerdo con el diseño establecido.

En el tercer capítulo se detallan las pruebas ejecutadas a cada componente del subsistema y las pruebas de validación. Finalmente se presentan las conclusiones y recomendaciones obtenidas en el presente Trabajo de Integración Curricular.

En los anexos se incluyen: los resultados de las encuestas de los requerimientos de los usuarios, el proyecto del cliente Android y el tablero Kanban utilizado para el desarrollo del subsistema.

PALABRAS CLAVE: aplicación Android, *stub*, gestión de fotografías, etiquetas.

ABSTRACT

This Curricular Integration Work presents the development of a component that is part of a distributed system and aims to allow the user to store photographs and classify them through tags. The subsystem was developed for the Android operating system and consists of four modules: login, user registration, photo query, and image upload, and also has a stub that emulates the behavior of the server.

This document is made up of three chapters.

In the first chapter, the objectives and scope are presented, followed by a description of the main features and components of Android applications and an analysis of the options for image management in the Android operating system, the definition of stub and the description of the Kanban methodology.

In the second chapter find the subsystem design for which, first of all, the requirements analysis for image management and labeling is performed. In addition, the Kanban board used for the development of the component is indicated. Finally, the implementation of the subsystem according to the established design is summarized.

The third chapter details the tests executed for each component of the subsystem and the validation tests. Finally, the conclusions and recommendations obtained in this Curricular Integration Work are presented.

The annexes include: the results of the user requirements surveys, the Android client project and the Kanban board used for the development of the subsystem.

KEYWORDS: Android app, stub, photo management, tags.

1 INTRODUCCIÓN

Como parte del presente Trabajo de Integración Curricular se desarrolló un componente que es parte de un sistema distribuido y tiene como objetivo permitir al usuario el almacenar fotografías y clasificarlas mediante etiquetas.

Actualmente la gestión de imágenes en las aplicaciones móviles es sumamente importante, debido a que un gran porcentaje de las aplicaciones necesitan manipular imágenes y etiquetarlas para diferentes fines, algunos ejemplos son las aplicaciones para acceder a redes sociales, tiendas en línea, alquiler de artículos, entre otros.

El componente será una aplicación para el sistema operativo Android[1], a través de la cual, el usuario escogerá las fotografías que serán enviadas para su posterior almacenamiento en el sistema distribuido, así como las etiquetas que se les asignará a las mismas; así también, con base en las etiquetas, el usuario podrá realizar consultas al sistema distribuido que permitan recuperar las fotografías que tengan asociadas dichas etiquetas; finalmente, el componente permitirá al usuario ingresar con sus credenciales al sistema distribuido, para lo cual se diseñará e implementará una vista de acceso. También permitirá el registro de usuarios nuevos de ser necesario.

En la selección y etiquetado de fotografías se permitirá escoger uno o varios archivos que se encuentren en el dispositivo móvil, para después de esto proceder a su respectivo etiquetado y envío al servidor. Finalmente, los usuarios podrán realizar la búsqueda de las imágenes de acuerdo con las etiquetas asignadas previamente a cada fotografía.

Para realizar las pruebas de funcionamiento se generará un *stub*[2] para emular el funcionamiento del resto de componentes y permitir que usuarios puedan probar el correcto funcionamiento del subsistema. El *stub* es un objeto configurado para que, al llamar a un método se tenga como respuesta un valor predeterminado, con lo cual se puedan realizar las pruebas unitarias del subsistema sin la necesidad de integrar los otros componentes.

El componente se desarrolló utilizando el IDE (*Integrated Development Environment*)[3] Android Studio, el cual incluye las librerías que permiten escribir el código de las aplicaciones móviles mediante el lenguaje de programación Java[4]. Para la elaboración de

[1] Android: Sistema operativo móvil, de código abierto, basado en el núcleo de Linux y otras aplicaciones.

[2] *Stub*: Fragmento de código utilizado como reemplazo de alguna otra funcionalidad.

[3] IDE (*Integrated Development Environment*): Herramienta para el diseño de aplicaciones que combina aplicaciones comunes para el desarrollador en una sola interfaz gráfica.

[4] Java: Lenguaje de programación orientado a objetos.

las interfaces de la aplicación se utilizó el formato universal de etiquetas para la representación y transferencia de datos estructurados XML[5] (*Extensible Markup Languaje*). Para el desarrollo del componente se empleó la metodología de desarrollo ágil Kanban[6], con lo cual se generará un tablero Kanban para gestionar las diferentes actividades requeridas para implementar el componente y para poder visualizar el progreso del trabajo durante el desarrollo de la aplicación.

1.1 OBJETIVO GENERAL

Desarrollar un subsistema de adquisición de fotografías, que permita cargar fotografías y asignarles etiquetas, así como realizar consultas de fotografías con base en etiquetas.

1.2 OBJETIVOS ESPECÍFICOS

1. Analizar los fundamentos teóricos y las herramientas necesarias para el manejo de fotografías en un subsistema de adquisición de fotografías a ser desarrollado en el sistema operativo Android.
2. Diseñar un subsistema de adquisición de fotografías que permita cargar fotografías con etiquetas y realizar consultas.
3. Implementar el subsistema de adquisición de fotografías de acuerdo con el diseño realizado.

1.3 ALCANCE

En este Trabajo de Integración Curricular se generará un subsistema para la adquisición de fotografías, el cual estará conformado por una aplicación para el sistema operativo Android, a través de la cual se seleccionarán fotografías del teléfono celular y permitirá al usuario asignar las etiquetas necesarias, las que serán enviadas al servidor para su almacenamiento en conjunto con la fotografía. Además, mediante esta aplicación se podrá

[5] XML (*Extensible Markup Languaje*): Lenguaje de marcado universal que utiliza etiquetas y determina un conjunto de reglas para la codificación de documentos.

[6] Kanban: sistema de información que abarca una metodología para desarrollar productos de software que permite administrar tareas dentro de un ciclo de trabajo.

buscar fotografías mediante etiquetas y como resultado el sistema reflejará todas las fotografías asociadas a dichas etiquetas.

El subsistema de adquisición de fotografías contará con 3 vistas:

- Vista de acceso, que permitirá al usuario ingresar mediante clave y nombre de usuario, también permite al usuario registrarse en caso de no tener cuenta en el subsistema.
- Vista de selección y etiquetado de imágenes, permitirá al usuario seleccionar una o varias fotografías que se encuentren en su dispositivo móvil, en conjunto con la colocación de etiquetas para cada una de las fotografías que ha seleccionado.
- Vista de búsqueda, que presentará las fotografías que contengan las etiquetas, que el usuario ha indicado para la búsqueda.

El presente trabajo de titulación constará de cuatro fases:

Fase Teórica: Para la implementación de este subsistema se revisará la teoría relacionada con la programación de aplicaciones para el sistema operativo Android, en conjunto con el uso de las herramientas necesarias para cargar y descargar imágenes a través de la red y mediante una aplicación para Android. Así también se presentarán detalles de la metodología Kanban.

Fase de Diseño: Se establecerán los requerimientos necesarios para el subsistema, se utilizará el tablero Kanban para definir las tareas requeridas para implementar el subsistema. Se establecerá la arquitectura del subsistema y se generarán diagramas de clase. Posteriormente se crearán *sketches* para las vistas que permitan al usuario utilizar la aplicación. Finalmente se diseñará el *stub* que permita emular la parte del servidor para verificar el almacenamiento de las imágenes con sus respectivas etiquetas.

Fase de implementación: Se instalará el entorno de desarrollo Android Studio, en conjunto con las librerías necesarias para su funcionamiento. Después de esto se programará cada una de las clases y métodos de acuerdo con el diagrama establecido. Finalmente se implementarán las vistas y por último se codificará el *stub*.

Fase de pruebas: En primer lugar, se probará el código conforme se va desarrollando el subsistema. Una vez finalizada la aplicación, se solicitará a 4 personas que empleen el subsistema, para posteriormente aplicar una encuesta para determinar si el subsistema es adecuado. En caso de ser necesario, se corregirán los errores encontrados.

1.4 MARCO TEÓRICO

En esta sección del documento se presentan los conceptos teóricos necesarios para la realización de este trabajo.

1.4.1 APLICACIONES MÓVILES

Se desarrollan aplicaciones móviles para ser ejecutadas en dispositivos móviles como *smartphones* o *tablets*. La alta penetración que se tiene en la actualidad, en el mercado de las aplicaciones móviles, se debe a que la telefonía celular ofreció servicios alternos a los canales de voz tradicionales, y posteriormente la integración de diversas tecnologías en los dispositivos móviles permitió la aceptación masiva de los usuarios [1]. El término móvil hace referencia a poder acceder a la aplicación desde cualquier lugar, sin embargo, para brindar estas características a un producto de software hay que considerar ciertas limitaciones que vienen dadas por el hardware de los dispositivos móviles: dimensiones pequeñas, procesadores no tan potentes, poca capacidad de almacenamiento, conexión a Internet limitada. Ejemplos de aplicaciones móviles que pese a las limitantes indicadas funcionan de manera adecuada son: navegadores web, mapas, juegos, mensajería instantánea, bancas móviles, redes sociales, edición de fotos y videos [2].

En la actualidad se utilizan dos plataformas para desarrollar aplicaciones móviles nativas son Android de Google e iOS de Apple. Cada sistema operativo acepta diferentes lenguajes de programación para la programación de las aplicaciones móviles. En el caso de las aplicaciones para el sistema operativo Android pueden ser desarrolladas en los lenguajes de programación Java o Kotlin[7] y las aplicaciones para iOS en los lenguajes de programación Objetive-C[8] o Swift[9] [3].

1.4.2 ARQUITECTURA DE ANDROID

Android posee una arquitectura constituida por cinco niveles, los cuales se relacionan entre sí, para permitir al desarrollador la creación de aplicaciones [4]. En la Figura 1.1 se presenta la arquitectura de Android con sus respectivos componentes.

[7] Kotlin: es un lenguaje de programación que se ejecuta en la máquina virtual de Java.

[8] Objective-C: lenguaje de programación orientado a objetos creado como un subconjunto del lenguaje de programación C.

[9] Swift: Lenguaje de programación de múltiples paradigmas producido por Apple orientado en el desarrollo de aplicaciones para iOS y macOS.

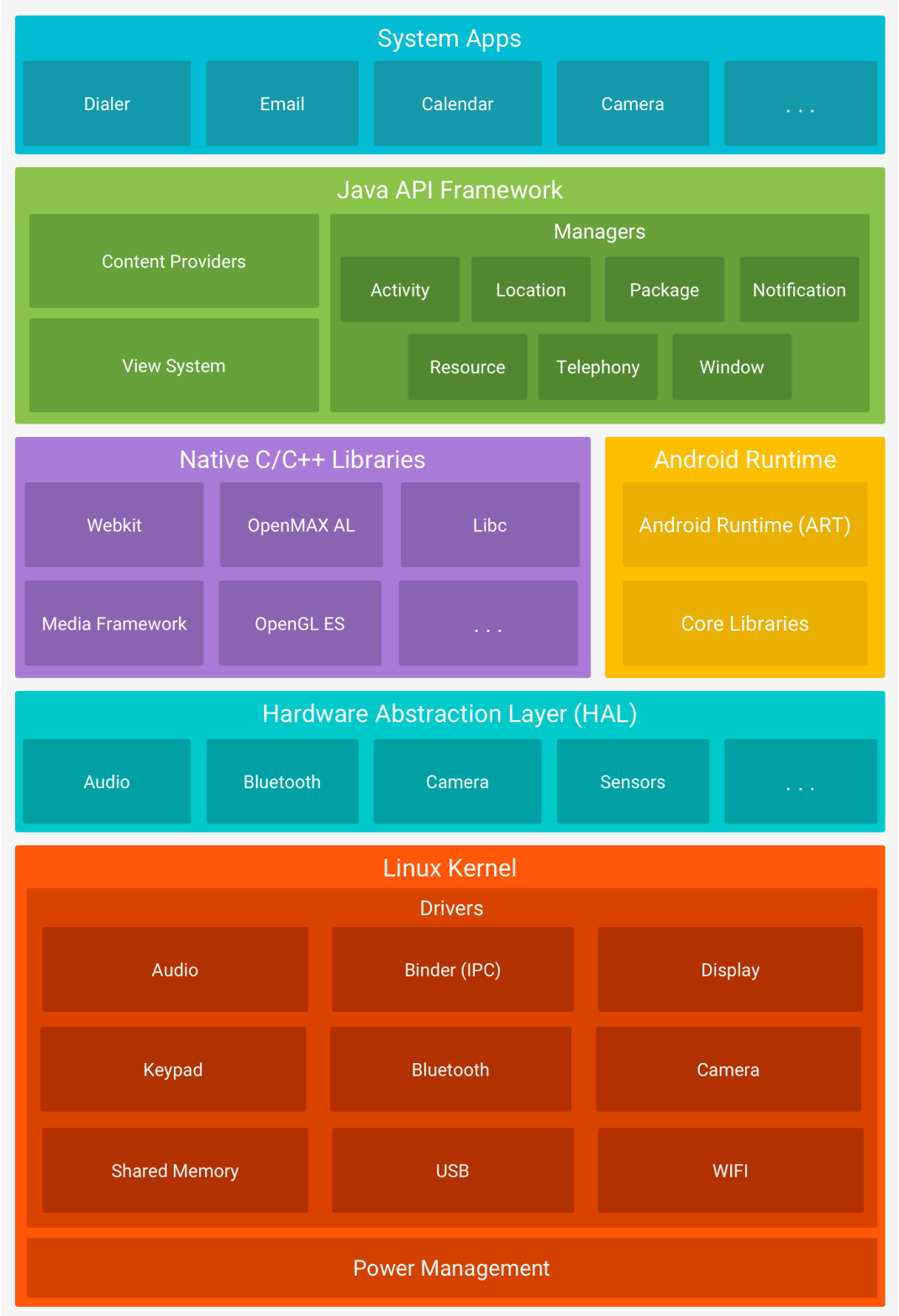


Figura 1.1. Arquitectura de Android [5]

Núcleo Linux: Constituye la base de la arquitectura de Android, y maneja las funciones básicas del sistema.

Capa de abstracción de *hardware*: Presenta varios módulos de biblioteca, los cuales implementan una interfaz para un determinado componente de *hardware*, como por ejemplo el módulo de Bluetooth[10] o el sensor biométrico del dispositivo móvil [5].

Librerías nativas y *runtime*: Contiene bibliotecas de bajo nivel escritas en lenguajes de programación C[11] y C++[12] que realizan diferentes funciones como: SQLite[13], para permanencia de datos; OpenGL[14], para administración de gráficos; y, Webkit[15], como navegador web embebido. También en este nivel se encuentra el *runtime*[16] de Android y las librerías del núcleo del sistema operativo [6].

Entorno de Aplicación: Se encuentra fragmentado en subsistemas que permiten a desarrolladores acceso completo a las distintas API[17] (*Application Programming Interfaces*) del *framework*[18], simplificando la reutilización de componentes. Incluye sistemas de vistas para manipular la interfaz gráfica de usuario [6].

Aplicaciones: Son aplicaciones base que contiene el sistema Android las cuales a su vez pueden ser utilizadas por otras aplicaciones [7].

1.4.3 COMPONENTES DE UNA APLICACIÓN ANDROID

Las aplicaciones de Android constan de tres componentes para proporcionar la reutilización del código y acelerar el desarrollo. Los tres componentes son actividades, servicios y proveedores de contenido [8]:

Actividades: Son las interfaces gráficas de usuario adecuadas para su propósito, que permiten que el usuario puede realizar ciertas acciones, tales como: escribir texto, ver un

[10] Bluetooth: Tecnología industrial desarrollada para redes inalámbricas de área personal.

[11] Lenguaje C: Lenguaje de programación de bajo nivel.

[12] Lenguaje C++: Lenguaje de programación que amplia al lenguaje de programación C e incluye mecanismos que posibilita la orientación de objetos.

[13] SQLite: Librería del lenguaje de programación que gestiona de bases de datos relacionales.

[14] OpenGL: Es una plataforma que contiene una API de múltiples lenguajes y proporciona herramientas para que las aplicaciones que representen gráficos vectoriales en 2D y 3D.

[15] WebKit: plataforma de diseño que permite a los navegadores web representar páginas web para aplicaciones. Tienen como base para su funcionamiento el navegador web Safari y Epiphany, entre otros.

[16] *Runtine*: Conforma el entorno de ejecución de las aplicaciones Android.

[17] API (*Application Programming Interfaces*): Contrato de servicios entre dos aplicaciones.

[18] *Framework*: Es un entorno real o conceptual, cuyo objetivo es servir como soporte y guía para la construcción de una estructura.

mapa, hacer una llamada, etc. Cada aplicación tiene una o más actividades que pueden llamarse entre sí. Cada actividad tiene un ciclo de vida y una actividad está asociada a cada uno de sus procesos, como se especifica en la Tabla 1.1**.** En Android no existe un método `main`, método que en otros lenguajes de programación corresponde al punto de arranque de la aplicación; pero si existen varios métodos para gestionar los eventos que se produzcan en la aplicación [9]. Por otro lado, los eventos del ciclo de vida de una actividad se presentan en la Figura 1.2. Ciclo de vida de una Actividad en Android .

Tabla 1.1. Eventos del ciclo de vida de una actividad de Android [10]

Evento	Descripción
`onCreate`	Se ejecuta al crearse una actividad y permite inicializar todos los objetos que la actividad defina.
`onStart`	Se ejecuta cuando la actividad pasa a estar en pantalla, aunque no precisamente de forma visible.
`onRestart`	Se ejecuta luego de a `onStart` cuando proviene de una llamada a `onStop`
`onResume`	Se ejecutan cuando la aplicación inicia una actividad en pantalla, pero no necesariamente visible, este es el estado en el cual el usuario interactúa con la aplicación
`onPause`	Se ejecuta cuando la actividad suspende la interacción con el usuario.
`onStop`	Se lleva a cabo cuando la actividad se encuentra ejecutándose totalmente en segundo plano
`onDestroy`	Se ejecuta cuando la actividad va a ser destruida y sus recursos liberados.

Servicios: Los servicios son los responsables de ejecutar tareas en segundo plano, por lo que no tienen una interfaz gráfica. Entre los servicios se encuentran: recibir notificaciones, actualizar datos sin necesidad de que el usuario interactúe directamente en ningún momento, entre otros.

Proveedores de Contenidos: A través de este componente una aplicación puede poner a disposición su información para que otras aplicaciones puedan hacer uso de ella. Las aplicaciones pueden consultar, modificar o eliminar esta información, la cual puede ser almacenada en una base de datos SQLite, en la nube o en otros lugares desde los cuales las otras aplicaciones puedan acceder a los mismos.

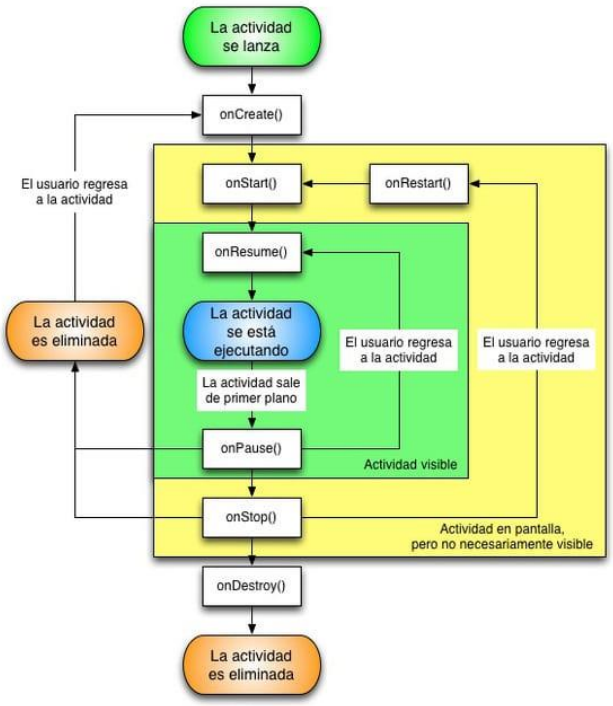


Figura 1.2. Ciclo de vida de una Actividad en Android [10]

1.4.4 ANDROID STUDIO

Android Studio es un IDE para el desarrollo de aplicaciones Android. Las características principales de Android Studio son [11]:

- Permite crear diseños dinámicos con *Jetpack Compose*[19], que permite obtener vistas previas de los diseños realizados en cualquier tamaño de pantalla.
- Posee un editor de código inteligente con el cual se puede escribir un mejor código, trabajar más rápido y ser más productivo puesto que está en capacidad de completar el código escrito en los lenguajes de programación Kotlin, Java y C/C++. Adicionalmente, cuando se edita con *Jetpack Compose* se puede observar los cambios realizados en el código de manera inmediata con *LiveEdit*[20].
- Es un sistema de construcción flexible. Posee *Gradle*[21], el cual permite personalizar la configuración para generar diferentes variantes para diferentes dispositivos Android a partir de un solo proyecto. Otra herramienta poderosa de construcción es

[19] *Jetpack Compose*: kit de herramientas moderno de Android para compilar interfaces de usuario nativas.

[20] *LiveEdit*: es un poderoso controlador destinado a producir eventos en vivo.

[21] *Gradle*: Sistema de automatización de construcción de código de *software*.

Build Analyzer[22], que analiza el rendimiento de las compilaciones para detectar posibles problemas.

- Para realizar las pruebas de *software*, Android Studio permite emular fácilmente cualquier dispositivo. Por lo que permite usar diseños receptivos, es decir que se adapten a teléfonos inteligentes, *tablets*, dispositivos plegables y Chrome OS.

Android Studio permite usar Java, Kotlin o C# para el desarrollo de aplicaciones. La versión móvil de Java se basa en la versión *Micro Edition*[23] de la plataforma Java. Es un lenguaje rápido, sencillo y permite generar una amplia gama de aplicaciones, razón por la cual sigue siendo uno de los más utilizados para el desarrollo de aplicaciones Android [12]. Este lenguaje es recomendado para principiantes debido a que existe una amplia comunidad de programadores Java, lo que significa que se encontrará respuestas más rápidas a dudas o problemas encontrados [13].

Kotlin es un lenguaje de programación relativamente nuevo, ofrece un código intuitivo y eficiente. Está inspirado en Scala[24], pero con una mayor velocidad de compilación. Desde mayo del 2018 es el lenguaje oficial para Android [13].

C# es un lenguaje multiplataforma y es una gran alternativa para la construcción de aplicaciones híbridas.

1.4.5 OPCIONES PARA EL MANEJO DE IMÁGENES EN APLICACIONES ANDROID

Para presentar una imagen en una aplicación Android, en primer lugar, se le debe asignar un espacio en la interfaz de usuario en el que se mostrará la imagen, para lo cual se utiliza un elemento denominado `Image`, el mismo que permite controlar como se presenta la imagen, pudiendo, por ejemplo, ajustarlo al tamaño que ocupa la pantalla. Por otro lado, se utiliza la clase `ImageView` para mostrar cualquier tipo de imagen en una aplicación Android, y permite presentar cualquier recurso de imagen, por ejemplo, mapas de bits o recursos dibujables. `ImageView` también se usa comúnmente para aplicar tintes a una y

[22] *Build Analyzer*: Archivos de compilación de Android Studio.

[23] *Micro Edition*: Es una plataforma informática que se utiliza para el desarrollo de código portátil para dispositivos móviles e integrados y es un subconjunto de la plataforma Java.

[24] Scala: Es un lenguaje de programación multiparadigma ayuda a prevenir errores en el desarrollo de aplicaciones complejas y a producir sistemas de alto rendimiento.

manejar la escala de la imagen. Se presenta en la Figura 1.3 un fragmento de código escrito en XML el cual indica el uso de `ImageView` para mostrar un recurso de imagen [14].

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/my_image"
        android:contentDescription="@string/my_image_description"
        />
</LinearLayout>
```

Figura 1.3. Uso de la etiqueta `ImageView` para mostrar un recurso de imagen [14]

En la Figura 1.3 se muestra la etiqueta `LinearLayout` y su propiedad `xmlns:Android` la cual define el `namespace` de Android. A continuación, se utiliza las propiedades `layout_with` y `layout_height` con el valor `match_parent` lo que significa que este contenedor utilizará todo el espacio disponible considernado el ancho y alto de la pantalla. Entre las etiquetas `LinearLayout` se agrega la etiqueta `ImageView` que corresponde a un contenedor de imagen y sus propiedades `layout_with y layout_height` con el valor `wrap_content` indica que el contenedor tendrá el ancho y alto del contenido, posteriormente la propiedad `src` indica la ubicación de la imagen que se mostrará en el componente, y finalmente la propiedad `contentDescription` en la cual se indica descripción en texto del contenido de la etiqueta.

Otra librería eficiente para cargar imágenes en una aplicación Android es *Glide*, la misma proporciona una API fácil de usar, un mecanismo de decodificación extensible de recursos y alto rendimiento en la agrupación automática de recursos [15].

Glide permite la adquisición, decodificación y visualización de imágenes fijas de video, imágenes y GIF (*Graphics Interchange Format*)[25] animados, utiliza un *stack* personalizado a través de un elemento `HttpUrlConnection`. De la misma manera contiene bibliotecas de utilidades que se enlazan al proyecto *Volley* de Google o a la biblioteca `OkHttp` de *Square*.

[25] GIF (*Graphics Interchange Format*): es un archivo de imagen animado a través de la agrupación de varias imágenes o un solo vídeo.

La finalidad de *Glide* es hacer que mover cualquier tipo de lista de imágenes sea lo más suave y rápido posible; sin embargo, también es efectivo para buscar, modificar la medida y ver una imagen remota.

Glide dispone de una API sencilla y fluida que permite a los usuarios realizar la mayoría de las solicitudes en una sola línea, como se observa en la Figura 1.4. se invoca al método `with` y se pasa como argumento un `fragment` el cual representa a la clase en la cual se lodeclara. A continuación, se llama al método `load` el cual recibe como argumento la URL (*Uniform Resource Locator*)[26] de la imagen que se espera que se cargue. Finalmente se llama al método `into` el cual recibe como argumento el componente `ImageView` en donde se quiere mostrar la imagen.

```
Glide.with(fragment)
     .load(url)
     .into(imageView);
```

Figura 1.4. Uso de la API *Glide* [15]

1.4.6 VOLLEY CLIENT

Es un cliente para un servicio REST (*Representational State Transfer*)[27], que dispone de una biblioteca HTTP (*Hypertext Transfer Protocol*)[28] que habilita y acelera el uso de redes en las aplicaciones de Android, y que está disponible en *GitHub*[29].

Para incorporar *Volley* a un proyecto Android se debe agregar la dependencia al archivo `build.gradle` [16] indicada en la Figura 1.5 para agregar la librería de *Volley* se modifica el archivo `build.gradle` dentro de la sección `dependencies` se añade la línea `implementation` con la ubicación, el nombre y la versión de la librería que se utilizará en el proyecto de Android.

Una de las ventajas que ofrece *Volley* es la programación automática de requerimientos de red, la posibilidad de gestionar varias conexiones paralelas, así como el almacenamiento de respuestas en caché.

---

[26] URL (*Uniform Resource Locator*): dirección de un recurso para su identificación de forma única en la web.

[27] REST (*Representational State Transfer*): arquitectura para conectar sistemas basada en el protocolo HTTP-

[28] HTTP (*Hypertext Transfer Protocol*): protocolo que permite gestionar las solicitudes del cliente y las respuestas del servidor en la web.

[29] GitHub: Lugar para albergar proyectos informáticos utilizando el sistema de control de versiones Git.

```
dependencies {
    ...
    implementation 'com.android.volley:volley:1.1.1'
}
```

Figura 1.5. Incorporación de *Volley* a un proyecto de Android [16]

Se integra fácilmente con cualquier protocolo e incluye compatibilidad con cadenas de texto, imágenes y objetos JSON (*JavaScript Object Notation*)[30]. No se recomienda el uso de *Volley* para operaciones de transmisión que realicen descargas grandes de archivos. *Volley* admite dos tipos de solicitudes comunes. La primera es de tipo `StringRequest`, en la cual especifica una URL y recibe como respuesta un objeto `string` sin procesar. El segundo tipo de solicitud requiere el uso de las clases: `JsonObjectRequest` y `JsonArrayRequest`, a través de estas clases se especifica una URL para realizar el pedido y se obtiene como respuesta un objeto JSON (*JavaScript Object Notation*) [17].

En la Figura 1.6 se detalla un ejemplo de un fragmento de código escrito en el lenguaje de programación Java que realiza la búsqueda de un *feed*[31] usando JSON y lo muestra como texto en la interfaz de usuario.

Para crear una petición con *Volley* primero se debe establecer la URL del recurso del cual se quiere obtener la información, se crea un objeto `JsonObjectRequest` al cual se le pasa como argumento el método de la petición por ejemplo `get`, la URL que corresponde a la ubicación del recurso, un elemento `null` que se envía en el elemento `body` de la petición y un `Response.Listener` que indica que se espera por una respuesta que será de tipo `JSONObject` y un `Response.ErrorListener` que indica que se espera por un posible error.

Se sobrescribe el método `onResponse` que se ejecutará si llega una respuesta y como resultado al `Response.Listener`; en el ejemplo se aprecia que la respuesta obtenida por el *endpoint*[32] será colocada en un `textView`. Posteriormente se sobrescribe el método `onErrorResponse` que se ejecutará si llega una respuesta con un error y como resultado al `ResponseErrorListener`.

[30] JSON (*JavaScript Object Notation*): Es un formato de texto simple para el envío y recepción de datos.

[31] *Feed*: se refiere a la provisión y actualización de datos electrónicos, se utiliza para designar a los documentos con formato RSS (*Really Simple Syndication*).

[32] *Endpoint*: dispositivo informático remoto que se comunica con una red a la que está conectado.

```java
String url = "http://my-json-feed";

JsonObjectRequest jsonObjectRequest = new JsonObjectRequest
        (Request.Method.GET, url, null, new Response.Listener<JSONObject>()

    @Override
    public void onResponse(JSONObject response) {
        textView.setText("Response: " + response.toString());
    }
}, new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
        // TODO: Handle error

    }
});

// Access the RequestQueue through your singleton class.
MySingleton.getInstance(this).addToRequestQueue(jsonObjectRequest);
```

Figura 1.6. Extracto de código de Java que permite solicitar y recuperar información de un *feed* en formato JSON [17]

1.4.7 STUB

Un *stub* es un componente de software que simula la actividad de otro componente para poder probar un sistema sin necesidad de contar todas las partes que conforman dicho sistema. Es un código que contiene los mismos parámetros de entrada y salida que los componentes faltantes; pero con un comportamiento altamente simplificado. El costo de realización de un *stub* es inferior que el de las partes faltantes [18]. Un ejemplo simple de *stub* se presenta la Figura 1.7, la imagen muestra un seudocódigo el cual contiene una variable `temperatura` que será asignada al ejecutar el método `Termómetro_Leer`, cuyo resultado será el valor “¡hace calor!” si el parámetro recibido es mayor a 40 grados, así también el código terminará si el método `ThermoterRead` devuelve un valor numérico fuera del rango.

Idealmente este método debería entregar un resultado a partir de un componente de hardware, pero como se aprecia, el código es simplificado y no contiene lo necesario para extraer la información desde un hardware.

```
temperatura = Termómetro Leer (exterior) si la temperatura> 40 entonces imprimir "¡Hace calor!"terminara si

función TermómetroRead (Fuente dentro o fuera) retorno 28 función final
```

Figura 1.7. Ejemplo sencillo de *stub* [19]

1.4.8 METODOLOGÍA KANBAN

La metodología Kanban tiene como fin lograr un proceso productivo, organizado y eficiente. Esta metodología fue creada en Toyota para controlar el progreso del trabajo realizado a lo largo de una cadena de suministros. Kanban busca garantizar una producción sostenible para evitar el uso excesivo de recursos, los cuellos de botella y demoras en la entrega, adicional muestra transparencia durante los procesos del trabajo [20]; por lo que es utilizado como una metodología ágil en el desarrollo de software. Kanban incluye una herramienta visual denominada tablero Kanban que proporciona a los equipos de desarrollo una visión general del estado actual de un proyecto.

Un ejemplo de tablero Kanban se presenta en la Figura 1.8.

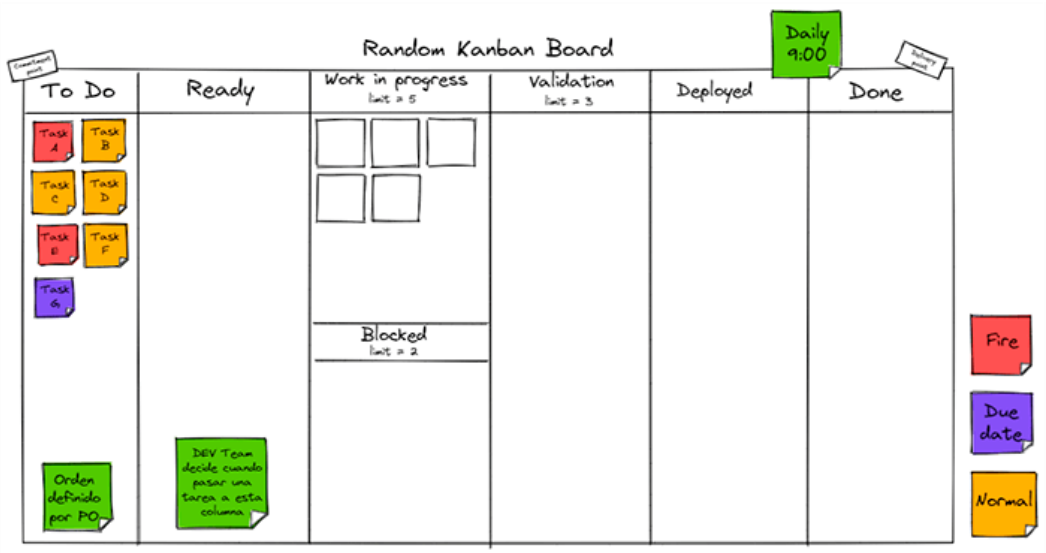


Figura 1.8. Ejemplo tablero Kanban [21]

El tablero permite enlistar las tareas propias de cada proyecto y seguir su evolución a lo largo de las diferentes etapas que se gestionan en el proceso.

El tablero representa los estados y las transacciones de las asignaciones los cuales se pueden personalizar de acuerdo con el proyecto que se esté realizando; por defecto en el sistema las tareas deben moverse en un flujo de izquierda a derecha [21].

El uso de Kanban en el desarrollo de programas requiere una división del trabajo; pero Kanban no define la distribución o dimensión de los trabajos o tareas.

El tablero Kanban se apoya en una serie de columnas que simbolizan los diferentes estados en los que se pueden encontrar las tareas a lo largo del proceso de desarrollo de *software*.

Con el avance del trabajo las tareas se van moviendo de un estado a otro hasta llegar a la última columna. Los componentes de un tablero Kanban son los expuestos en la Tabla 1.2.

Tabla 1.2. Componentes tablero Kanban [21].

Componente	**Descripción**
Indicadores visuales	En un tablero Kanban sobresalen las tarjetas visuales (*post-it*, tarjetas u otros). Al colocar las tarjetas sobre el tablero, estas señales visuales ayudan a percibir con mayor rapidez qué trabajo se está realizando.
Columnas	El conjunto de columnas conforma un flujo de trabajo y cada columna representa una tarea específica. Las tarjetas circulan por medio del flujo de trabajo desde el inicio hasta su terminación. Los flujos de trabajo son sencillos como "Por realizar", "En progreso", "Finalizadas" o presentar mayor complejidad.
Columna de entrada	Por lo general se dispone de una lista de tareas iniciales en el tablero, para comenzar con el trabajo. La columna de entrada es donde se colocan dichas tareas.
Columna de salida	Constituye el final del flujo de trabajo. El objetivo de la metodología Kanban es mover tarjetas a través del flujo de trabajo en el menor tiempo posible.

2 METODOLOGÍA

En este capítulo proporciona un resumen del proceso realizado como parte del diseño y desarrollo del subsistema distribuido de gestión de fotografías mediante una aplicación móvil. El propósito de la aplicación móvil es permitirle al cliente el almacenar fotos y etiquetas y permitirle la búsqueda de las fotografías asociadas a las etiquetas.

Para la creación del subsistema se manejará la metodología ágil Kanban, y se dispondrá de un tablero Kanban para realizar un seguimiento de las tareas y poder visualizarlas en cada una de las etapas del desarrollo.

Posteriormente, para la parte del diseño del sistema se definirán los requerimientos funcionales y no funcionales; además se realizará el diagrama de clases para presentar las clases que contendrá el sistema y su relación entre ellas. Luego utilizando el lenguaje UML (*Unified Modeling Language*)[33] se realizarán los diagramas de actividades para las funcionalidades de acceso a usuarios, registro de nuevos usuarios, carga de imágenes y etiquetas, y búsqueda de imágenes a partir de etiquetas.

También para el diseño de las vistas de la aplicación Android se realizarán *mockups*[34]. Después se explicará de forma resumida la implementación del prototipo con base a lo señalado en el diseño. Posteriormente se implementará un *stub* en NodeJS[35], el cual permitirá emular la parte del servidor para verificar el almacenamiento de las imágenes con sus respectivas etiquetas. Finalmente se realizarán las pruebas unitarias de cada módulo del subsistema y las pruebas de validación para verificar el correcto funcionamiento del subsistema.

2.1 DISEÑO

Como parte del diseño, en esta sección se presenta, en primer lugar, el levantamiento de requerimientos funcionales y no funcionales del subsistema; después se indica la arquitectura a utilizarse, el diseño del tablero Kanban y los diagramas UML correspondientes para finalmente elaborar los *mockups* de la aplicación.

[33] UML (*Unified Modeling Language*): Lenguaje de modelado de desarrollo de propósito general en la ingeniería de *software*.

[34] *Mockups*: Diseño y maquetado a escala que representa la idea de las vistas una aplicación o un desarrollo de *software*.

[35] NodeJS: Es un entorno de ejecución JavaScript multiplataforma y de código abierto.

2.1.1 LEVANTAMIENTO DE REQUERIMIENTOS

Para el levantamiento de requerimientos se evaluó los requerimientos definidos en el proyecto del cual es parte este componente, así como las necesidades de disponer de una aplicación Android que admita la gestión de imágenes y etiquetas. Los requerimientos funcionales del prototipo se establecieron para cada módulo y se detallan en la Tabla 2.1.

Tabla 2.1. Requerimientos funcionales

Id	Módulos	Requerimiento funcional
RF1	*Login*	• El componente permitirá ingresar usuario y contraseña. • Si el inicio de sesión es satisfactorio se recibirá los datos del usuario y un *token* de autenticación, caso contrario se recibirá un error HTTP.
RF2	Registro	• El componente permitirá registrar nuevos usuarios. • El componente permitirá ingresar los datos del usuario (nombre, correo, nombre de usuario y la contraseña). • El componente exigirá confirmar la contraseña antes de enviar los datos. • Si el registro es exitoso se ingresará al sistema recibiendo los datos del usuario y el *token* de autenticación del nuevo usuario.
RF3	Ver imágenes	• El componente permitirá ingresar varias etiquetas como criterio de búsqueda de las imágenes. • El componente permitirá restablecer los criterios de búsqueda. • El componente mostrará todas las imágenes recibidas por el *stub* presentando versiones en miniatura de las imágenes en una cuadrícula. • El componente permitirá seleccionar cada imagen para verla en mayor tamaño.
RF4	Subida de imágenes	• El componente permitirá al usuario escoger las fotografías desde el dispositivo Android. • El componente permitirá quitar imágenes seleccionadas por error. • El componente permitirá ingresar una o varias etiquetas antes de enviar los datos al *stub*. • El componente mostrará el mensaje de éxito o error recibido por el *stub*.
RF5	*Stub*	• El *stub* simulará una API REST con cuatro rutas (`/login`, `/registro`, `/upload`, `/images`). • La ruta `/login` recibirá como parámetros `user` y `password` y generará la respuesta de éxito enviando un *token* y los datos de un usuario. Caso contrario enviará la respuesta de error de autenticación enviando un estado `HTTP 401` con el mensaje "Usuario o Clave incorrecto". • La ruta `/registro` mostrará los datos recibidos y devolverá como respuesta un *token* y los datos de un usuario. • La ruta `/upload` e `/images` deben verificar que se reciba el *token* de autenticación, caso contrario enviar un error `HTTP 401`. • La ruta `/upload` recibirá en la petición un arreglo de imágenes que serán almacenadas en una carpeta `public/images` y mostrará el arreglo de etiquetas recibidas. • La ruta `/images` enviará como respuesta un arreglo de información de imágenes almacenadas (nombre, URL y etiquetas).

En la Tabla 2.2 se enlistan los requerimientos no funcionales del prototipo que establecen las características generales del subsistema.

Tabla 2.2. Requerimientos no funcionales

Id	Requerimiento no funcional
RNF1	La aplicación será desarrollada para el sistema operativo Android.
RNF2	Se utilizará Java como lenguaje de programación para escribir el código de la aplicación Android.
RNF3	Se utilizará NodeJS para la implementación del *stub.*
RFN4	Para poder consumir los recursos de la aplicación los usuarios deberán tener conexión a Internet.
RFN5	La aplicación Android consumirá el servicio REST.

2.1.2 ARQUITECTURA DEL SUBSISTEMA

La arquitectura en capas es un patrón de diseño de *software* que se utiliza para estructurar aplicaciones en varias capas, donde cada capa tiene una responsabilidad específica en la aplicación y se comunica con las capas adyacentes.

Para el presente trabajo, la aplicación tendrá una arquitectura en tres capas: la capa de presentación (o interfaz de usuario), la capa de negocio (o lógica de negocio) y la capa de datos (o acceso a datos), como se indica en la Figura 2.1.

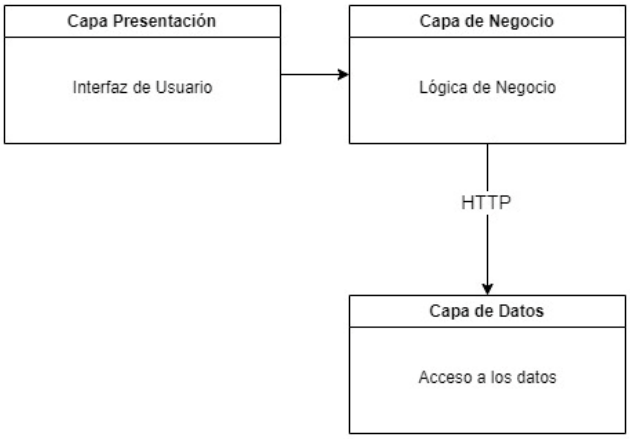


Figura 2.1. Arquitectura del subsistema

Capa de presentación: Es la capa que se encarga de la interacción con el usuario, es decir, muestra información y recibe entradas del usuario.

Capa de negocio: Es la capa que contiene la lógica de negocio de la aplicación, es decir, las reglas y procesos que se aplican para procesar la información. Esta capa es encargada

de validar la información recibida de la capa de presentación, estructurar la información, tomar decisiones y enviar los resultados a la capa de presentación.

Capa de datos: Es la capa que se encarga del acceso a los datos y su almacenamiento, ya sea en una base de datos, un archivo o cualquier otro tipo de almacenamiento.

2.1.3 TABLERO KANBAN

El tablero Kanban cuenta con las siguientes columnas: entrada, diseño, desarrollo, pruebas y salida, correspondientes a fases del desarrollo de software. Para la gestión del Tablero Kanban se empleó la aplicación gratuita Trello[36], la cual permite administrar proyectos que cuenten con una interfaz web y con clientes móviles para iOS y Android. Para este trabajo se implementó el tablero Kanban el enlace para visualizar el tablero se encuentra en el Anexo I.

Para el desarrollo del subsistema se dividió el trabajo en dos grupos de tareas: tareas del *stub* y tareas para el desarrollo de la aplicación Android. En las actividades del *stub* se encuentran las tareas que posibilitarán establecer las operaciones del servidor (ver Tabla 2.3). Mientras que en las tareas de desarrollo de la aplicación Android están las actividades que permitirán obtener los entregables funcionales del subsistema correspondiente a cada módulo como lo indica la Tabla 2.4.

Tabla 2.3. Tablero Kanban para las tareas del *stub*

Tareas del *stub*				
Entrada	Diseño	Desarrollo	Pruebas	Salida
RF5	Diagrama de actividades	Utilizando el diagrama de actividades se creará el servidor web con las rutas `/login`, `/registro`, `/upload`, `/images`	Peticiones a las diferentes rutas del *stub* utilizando un cliente REST	*Stub* desarrollado en NodeJS

[36] Trello: software de administración de proyectos con interfaz web y con cliente para iOS y Android para organizar proyectos.

Tabla 2.4. Tablero Kanban de las tareas de la aplicación Android

Tareas de la aplicación Android				
Entrada	**Diseño**	**Desarrollo**	**Pruebas**	**Salida**
RF1	Diagrama de actividades y *mockups* de la interfaz de usuario	Enviar la petición HTTP al *stub* para iniciar la sesión	Iniciar sesión exitosa y con error en los datos ingresados	Módulo *Login*
RF2		Validación de datos y envío de petición HTTP al *stub*	Crear un usuario nuevo	Módulo Registro de usuario
RF3		Envío de etiquetas ingresadas en una petición HTTP al *stub*	Ver las imágenes enviadas por el *stub*	Módulo Ver imágenes
RF4		Envío de una petición HTTP `multipart/form-data` con las imágenes seleccionadas	Guardar imágenes en el *stub*	Módulo Subir imágenes

2.1.4 DIAGRAMA DE CLASES

En el diagrama de clases se trabaja con 5 paquetes principales: `Backend.java`, `DataPart.java`, `Imagen.java`, `ImagesAdapter.java`, `VolleyMultipartRequest.java`. En la Figura 2.2 se indica la correspondencia entre cada una de ellas. Mientras que en la Tabla 2.5 se encuentra detallada las características de cada elemento del diagrama de clases.

Tabla 2.5. Descripción de los elementos del diagrama de clases

Clase	Descripción
`Backend.java`	Tiene dos métodos: `servername`, que retorna un `string`[37] que contiene el nombre del servidor y `port`, que retorna el puerto de escucha del servidor.
`DataPart.java`	Almacena arreglos de bytes, que corresponden a imágenes
`Imagen.java`	Gestiona a una imagen almacenada en el servidor
`ImagesAdapter.java`	Representa la lógica de la vista que se va a integrar en cada elemento del `gridview`[38]
`VolleyMultipartRequest.java`	Crea una petición HTTP que envía imágenes como `multipart/form-data`[39] en el `body`[40] de la petición.

[37] `String`: Objeto que se utiliza para representar y manipular una secuencia de caracteres.

[38] `Gridview`: Control que se utiliza para mostrar los valores del origen de datos de una tabla, cada una de las columna representa un campo y cada una de las filas representa un registro.

[39] `multipart/form-data`: uno de los valores del atributo `enctype`, que se usa en el elemento de formulario el que tiene la carga de archivo.

[40] `Body`: es el que contiene todos los aspectos representables de un documento.

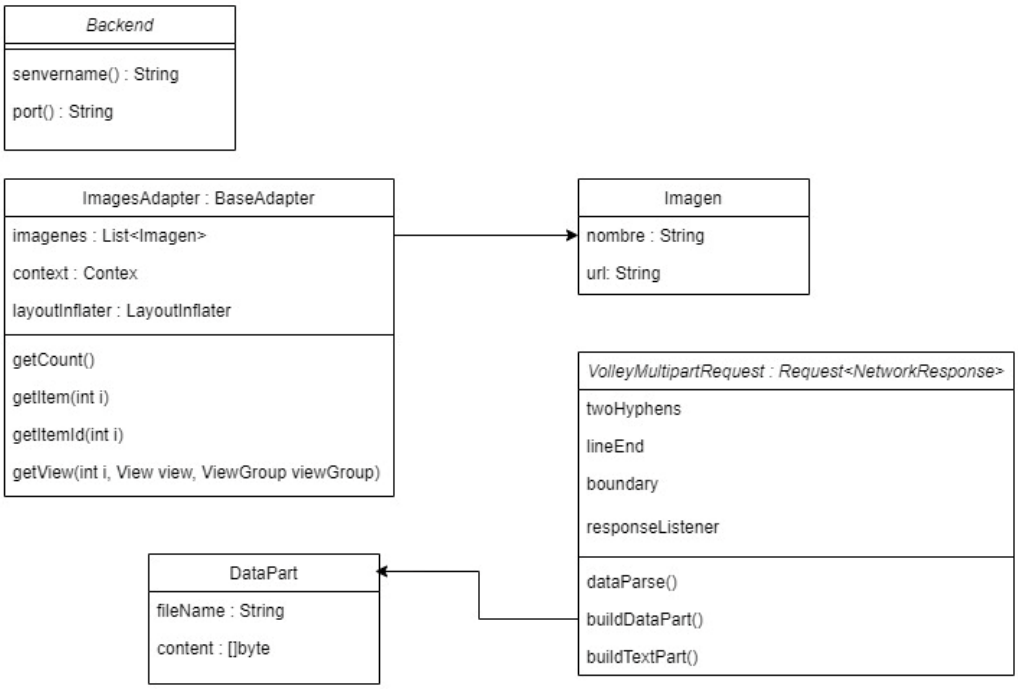


Figura 2.2. Diagrama de clases del subsistema

2.1.5 DIAGRAMA DE ACTIVIDADES

El diagrama de actividades muestra las actividades que el subsistema ejecutará. Para el subsistema se han definido tres actores: usuario, aplicación móvil y *stub.*

La primera actividad del subsistema corresponde al acceso del usuario, el cual se detalla en la Figura 2.3. Como se aprecia, cuando el usuario quiere usar el subsistema debe ingresar su nombre de usuario y contraseña, datos que serán enviados por la aplicación Android en formato JSON, mediante una petición POST[41] de HTTP; luego el *stub* verifica si la información ingresada es correcta y se remitirá la respuesta respectiva y la aplicación Android presentará al usuario la vista del menú de la aplicación; el *stub* debe remitir en la respuesta un *token*[42] y la información del usuario para que la aplicación Android pueda iniciar la actividad denominada `ActivityMenu`.

En caso de que los datos ingresados fuesen incorrectos se envía código de `error 401`, para que la aplicación móvil muestre el mensaje de usuario o contraseña incorrecta.

[41] POST: método que se utiliza para enviar una entidad a un recurso en específico, provocando constantemente una alteración en el estado o efectos secundarios en el servidor.

[42] *Token*: Utilizado en la seguridad de datos, se refiere al proceso de reemplazar un dato sensible por un equivalente no sensible; que no tiene un valor extrínseco o exportable.

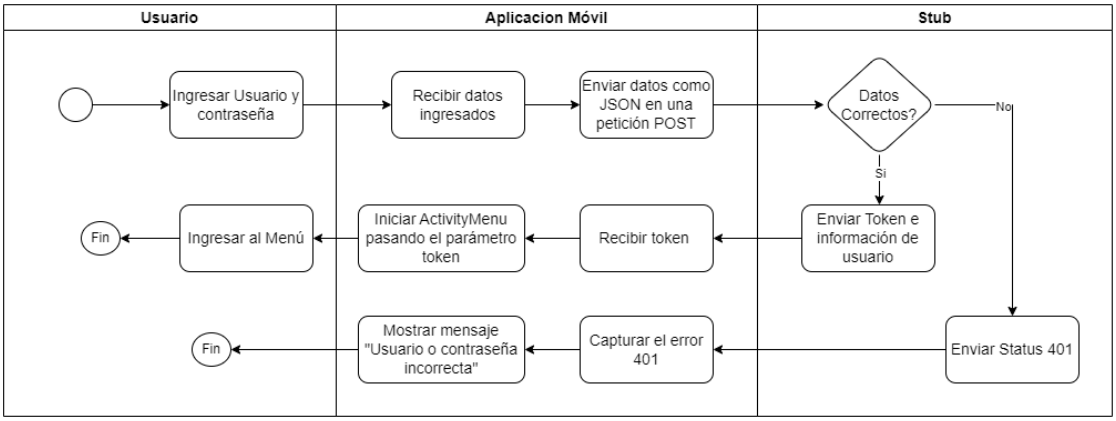


Figura 2.3. Diagrama de actividades para el acceso de usuarios

La siguiente actividad del subsistema es el registro de nuevos usuarios para lo cual el usuario debe ingresar los datos a registrar, a continuación, la aplicación móvil valida los datos ingresados, y una vez registrado el usuario, este pueda acceder al subsistema, como se detalla en la Figura 2.4**.**

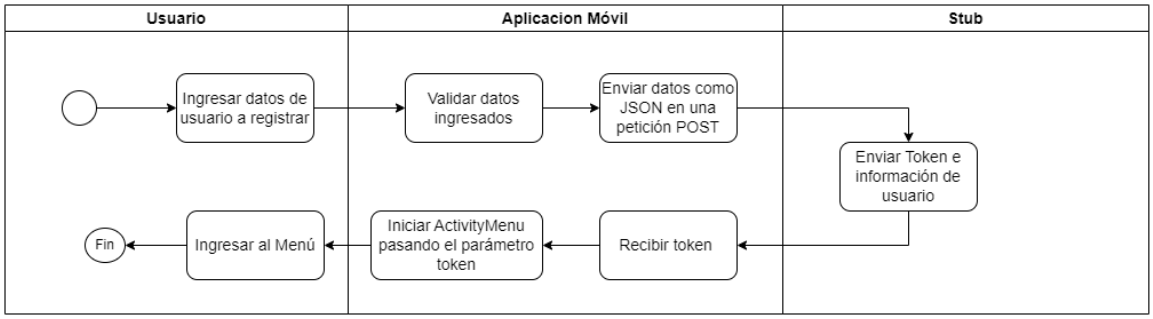


Figura 2.4. Diagrama de actividades para el registro de nuevos usuarios

En la Figura 2.5, se presenta el diagrama de actividades de la actividad para ver imágenes. En primer lugar el usuario ingresa las etiquetas que desea que el sistema busque, posteriormente la aplicación móvil genera un arreglo de etiquetas para enviar como parámetros JSON mediante una petición POST de HTTP al *stub*, el cual verifica si la petición contiene el *token*, en caso de que sea afirmativa la respuesta se envía como respuesta las imágenes y etiquetas que concuerdan con la petición hacia la aplicación móvil; en la aplicación móvil, con base en la respuesta, se presenta en un control `gridview` las imágenes recibidas, el usuario puede presionar sobre una imagen y la aplicación Android se encargará de ampliarla.

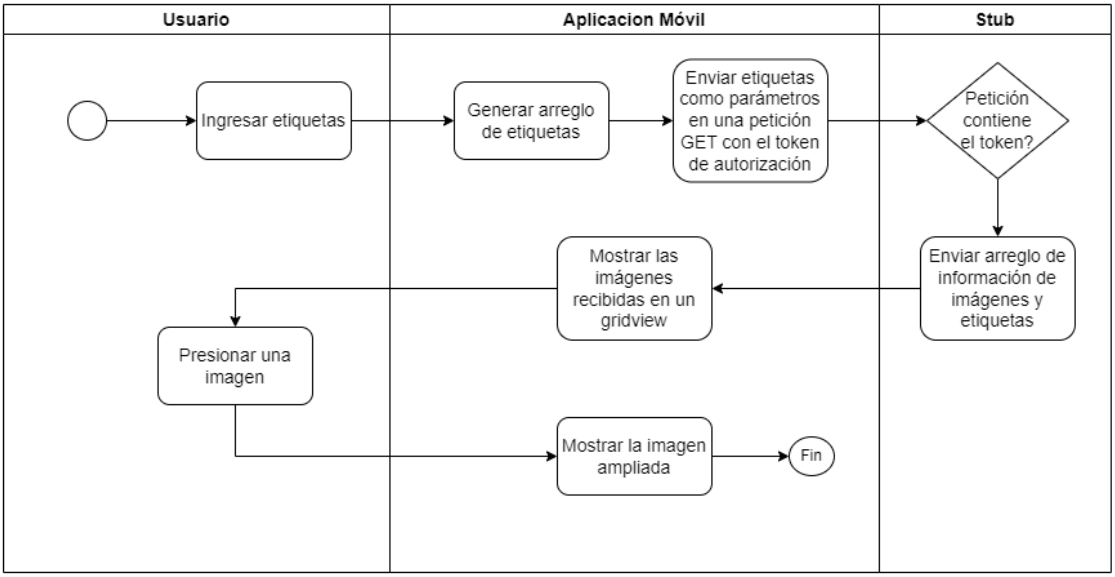


Figura 2.5. Diagrama de actividades para ver imágenes

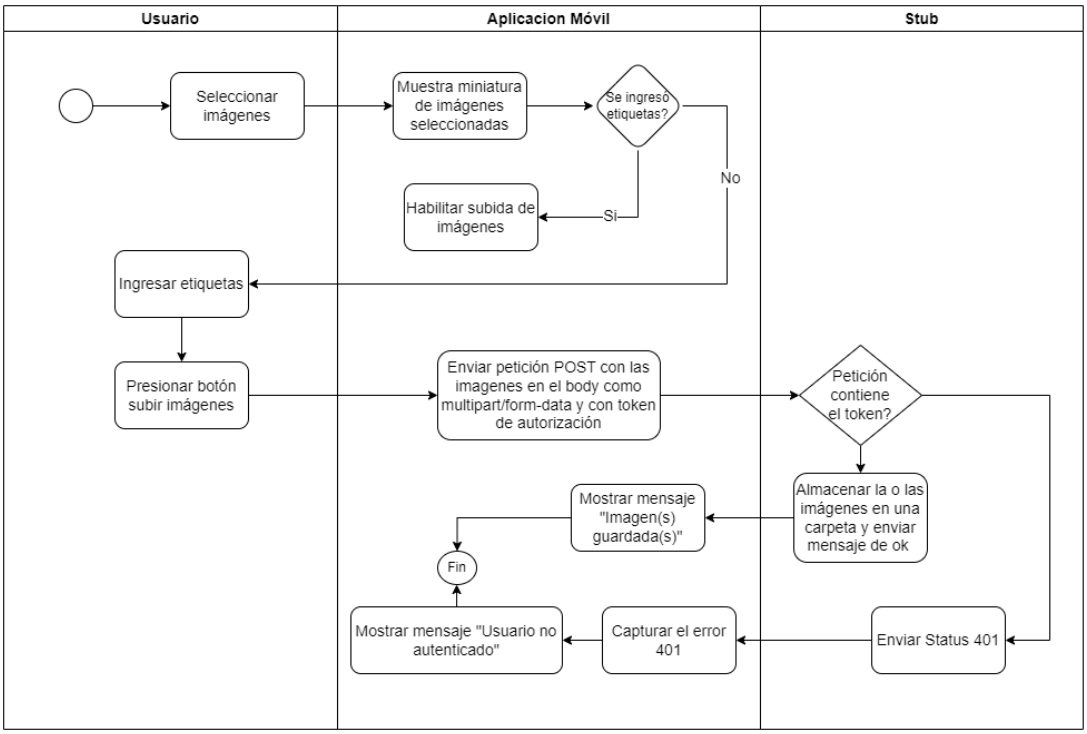


Figura 2.6. Diagrama de actividades para subir imágenes

La última actividad es subir imágenes (ver Figura 2.6). Para iniciar el usuario selecciona las imágenes que desea enviar al servidor, la aplicación móvil muestra en miniatura las imágenes seleccionadas. Posteriormente la aplicación revisa si se ingresó o no las

etiquetas de las imágenes, de ser el caso habilita la subida de imágenes; caso contrario, el usuario debe ingresar las etiquetas para que se pueda enviar las imágenes en el `body` de la petición POST, posteriormente el *stub* verifica si la petición contiene el *token*, si se envió el *token* se almacena las imágenes en una carpeta y se responde con un mensaje para que la aplicación a su vez muestra el mensaje “Imagen guardada”; caso contrario el *stub* envía el `error 401` con el cual, la aplicación muestra el mensaje “Usuario no autenticado”.

2.1.6 INTERFACES DE LA APLICACIÓN MÓVIL

Para las interfaces de la aplicación móvil se desarrollaron mockups tomando en consideración los requerimientos funcionales descritos. En primer lugar, se diseñaron los *mockups* correspondientes al acceso de usuario, así como del menú que se presenta al usuario una vez validadas sus credenciales, como se observa en la Figura 2.7.

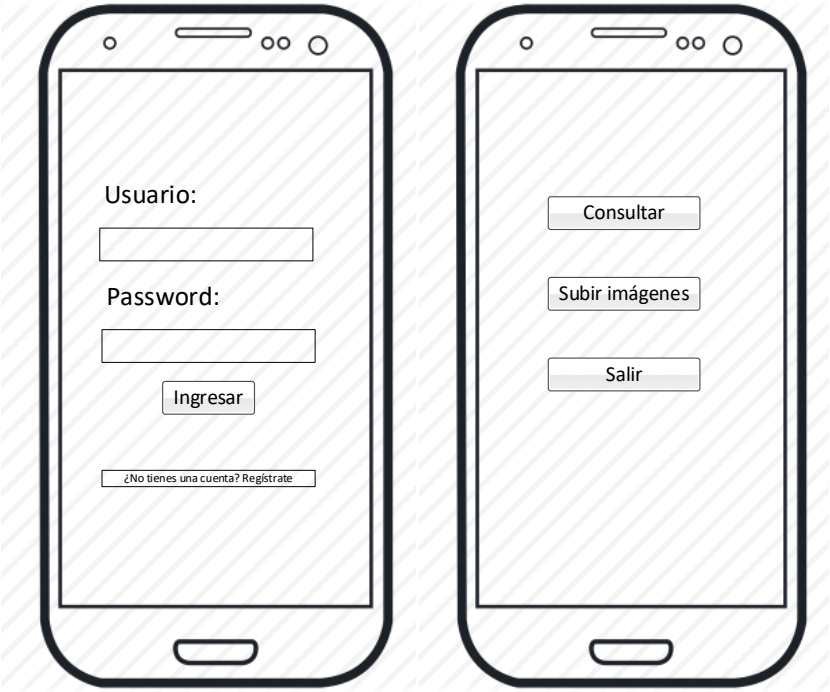


Figura 2.7. Mockups de *Login* y Menú

El primero permite el ingreso del nombre de usuario y contraseña de usuarios registrados y en caso de que no se encuentren registrados, posee un enlace que lo llevará a otra interfaz para registrarse. El segundo correspondiente al menú que permite al usuario escoger que acción desea realizar: consultar, subir imágenes o salir de la aplicación; para lo cual se agregaron los tres botones correspondientes. Posteriormente se diseñó la

interfaz para la creación de usuarios nuevos (ver Figura 2.8) a través de la cual se solicitan los datos necesarios para su respectivo registro, los datos son: nombre, nombre de usuario, correo electrónico, contraseña y confirmación de la contraseña ingresada. Adicionalmente también contiene un botón para completar el registro una vez que los datos hayan sido ingresados.

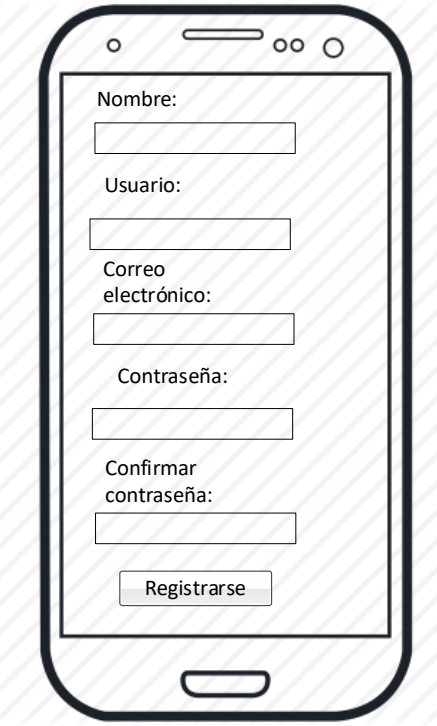


Figura 2.8. *Mockup* de registro de usuario

Finalmente se realizó el diseño de las vistas de búsqueda y subida de imágenes. En la Figura 2.9 correspondiente a la búsqueda de imágenes existe un campo de texto para añadir etiquetas y dos botones; el botón actualizar recupera todas las imágenes correspondientes a las etiquetas ingresadas y el botón limpiar borra las etiquetas ingresadas.

La segunda interfaz corresponde a la subida de imágenes y contiene un botón que permite la selección de imágenes del dispositivo móvil, también contiene un campo de texto para añadir las etiquetas que se desee guardar.

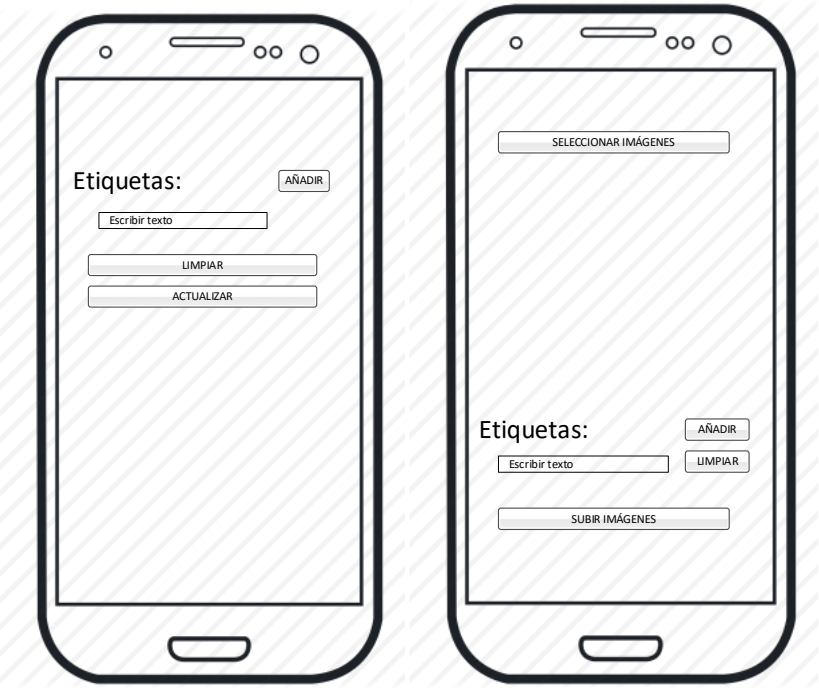


Figura 2.9. *Mockups* de búsqueda y subida de imágenes

2.2 IMPLEMENTACIÓN

En esta sección se presenta un el proceso de implementación del subsistema de acuerdo con lo establecido en la sección de diseño.

2.2.1 CONFIGURACIÓN DE ENTORNO DE DESARROLLO

Para el desarrollo del subsistema se va a usar el entorno de desarrollo Android Studio. Los pasos realizados para la configuración del entorno se describen a continuación:

1. Descargar e instalar Android Studio desde la página oficial: https://developer.android.com.
2. Una vez descargado, hacer clic derecho sobre el instalador y ejecutar, con permisos de administrador, y seguir los pasos de instalación.
3. Después de realizar la instalación se procede a crear un nuevo proyecto. En el proyecto se agrega una `Empty Activity`, como se indica en la Figura 2.10.

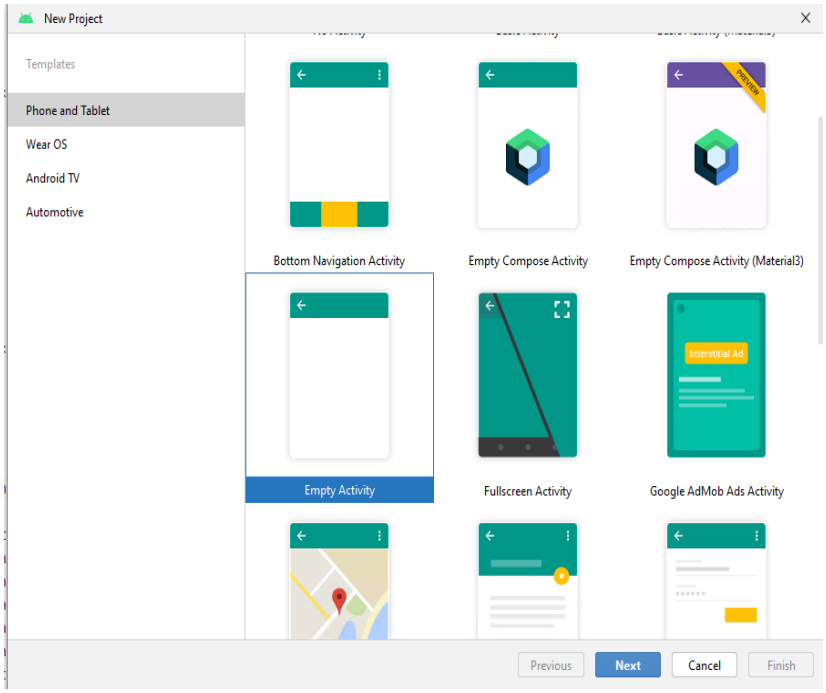


Figura 2.10. Creación de un nuevo proyecto en Android Studio

4. A continuación, se colocan las características del proyecto como el nombre, el lenguaje de programación, y la versión de la API de Android. Para este trabajo se usó como nombre `GestiónFotografías`, como lenguaje de programación se escogió Java y en versión de API se escogió `API 24: Android 7.0 (Nougat)`, como se detalla en la Figura 2.11.

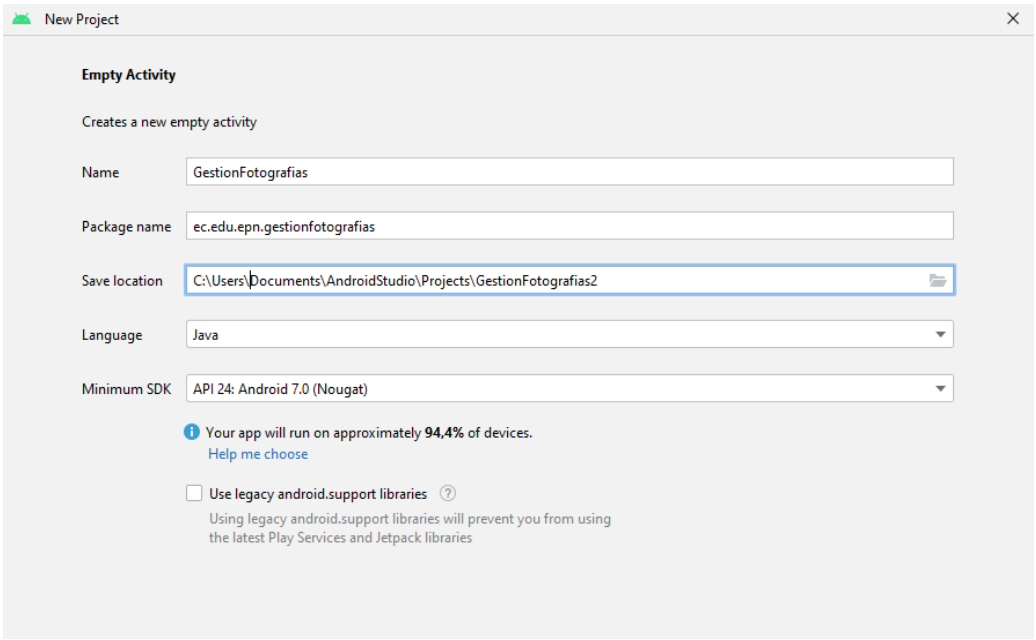


Figura 2.11. Ingreso de las características del nuevo proyecto

2.2.2 IMPLEMENTACIÓN DE LIBRERÍAS EXTERNAS

Las librerías externas que se utilizaron para el desarrollo del subsistema son *Volley* y *Glide*; para poder usar las mismas, en primer lugar en el archivo `build.graddle(module)` (ver Figura 2.12), se ingresan las dependencias (bibliotecas externas utilizadas por la aplicación), así también se configuran las opciones de compilación, como la versión mínima y máxima de Android que soporta la aplicación.

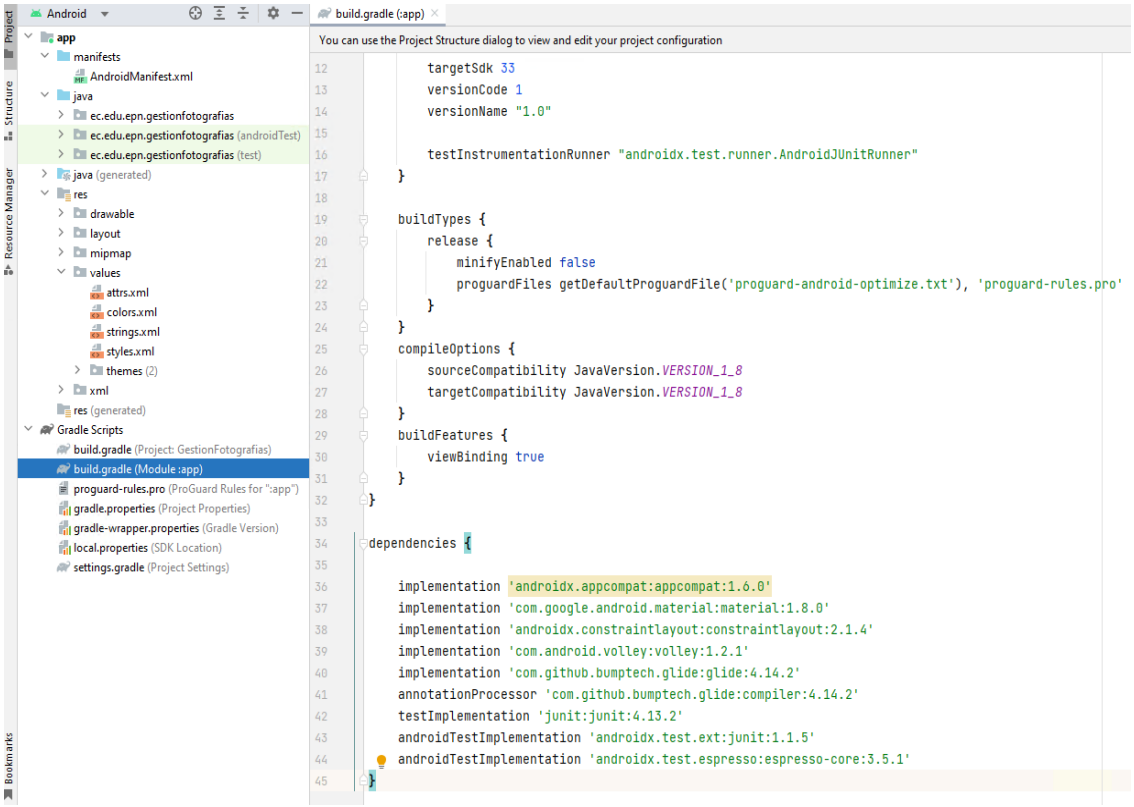


Figura 2.12. Archivo `build.graddle(module)`

En la Figura 2.13 se aprecia que se añadió la línea 39, la cual indica que se va a utilizar la libraría *Volley* en su versión 1.2.1, así como la línea 40, la cual indica que se utilizará la librería *Glide* en su versión 4.14.2, y la línea 42, en la cual se añade `annotationProcessor` de *Glide*, que es necesaria cuando se utilizan algunas de las características de *Glide* que requieren el procesamiento de anotaciones. El procesamiento de anotaciones se realiza en tiempo de compilación para generar código adicional para una clase, interfaz o método, y se utiliza para automatizar ciertas tareas repetitivas en el código. Por otro lado, en la sección de dependencia se agregan las líneas de código señaladas con color celeste en la Figura 2.13, luego de lo cual se debe sincronizar el

proyecto presionando en el botón `Syn Now` ubicado en la parte superior derecha, el cual aparece solo cuando se edita el archivo.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.6.0'
    implementation 'com.google.android.material:material:1.8.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    implementation 'com.android.volley:volley:1.2.1'
    implementation 'com.github.bumptech.glide:glide:4.14.2'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.14.2'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'
}
```

Figura 2.13. Definición de dependencias requeridas (*Volley* y *Glide*)

2.2.3 IMPLEMENTACIÓN DE VISTAS

Para la implementación de la vista de *Login* se edita el archivo `activity_main.xml` dentro de la carpeta `res/layout` y se ingresan los componentes gráficos usando etiquetas escritas en XML. En particular se colocan dos componentes `textview` para el ingreso del nombre de usuario y contraseña, un `checkedtextview` esto permitirá la creación de una cuenta en el caso de nuevos usuarios, y un `button` con el texto `ingresar`, como se puede observar en la Figura 2.14.

Para implementar la vista de *login* en el archivo `activity_main.xlm` dentro de la etiqueta `layout` se agregan dos etiquetas `EditText`. La primera para el ingreso de usuario y la siguiente con las propiedades `layout_width` con el valor `match_parent`, para que utilice todo el ancho disponible y `layout_height` con el valor `wrap_content`, para que tome el alto del texto ingresado. La propiedad `inputType` del `EditText` para el ingreso del *password* tiene el valor `textPassword` para que no se presente el valor ingresado en el componente. En la etiqueta `Button`se utilizan las propiedades `layout_height` y `width` de la misma manera que en los `EditText`. Luego se añade la etiqueta `CheckedTextView` que permite presentar texto en el cual el usuario puede hacer clic. Y finalmente se agrega un `TextView` con `id txvMensaje`, el cual se usará parar publicar mensajes de error de autenticación en caso de existir.

Para la implementación de la vista que permitirá a los usuarios nuevos registrarse en el sistema se creó el archivo `activity_registro.xml`, en el cual se tienen varios

`editText` para el ingreso de los datos del nuevo usuario y un `button` con el texto `Registrarse`, como se indica en la Figura 2.15.

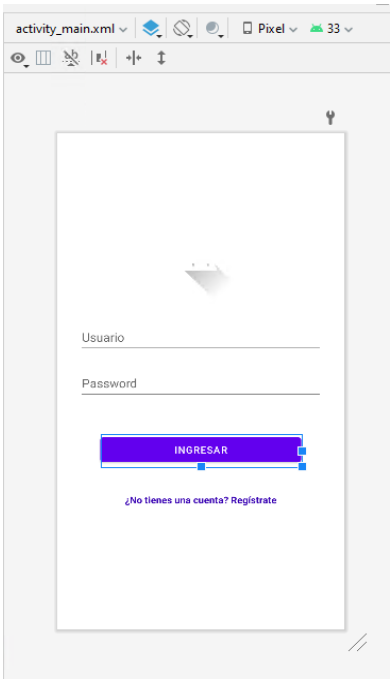


Figura 2.14. Implementación de la vista *Login*

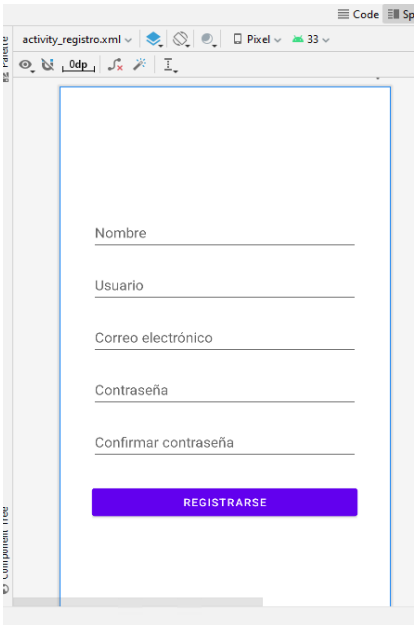


Figura 2.15. Creación de la vista de creación de nuevos usuarios.

En el archivo `activity_menu.xml` se tienen tres botones para presentar otras vistas del subsistema, como se detalla en la Figura 2.16.

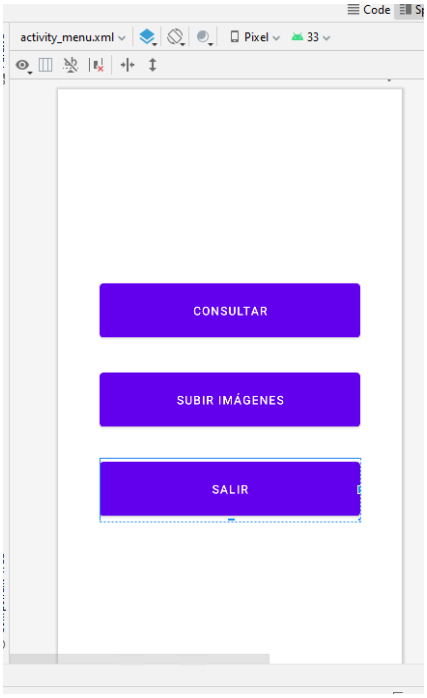


Figura 2.16. Creación de la vista menú

También se creó el archivo `activity_ver_imagenes.xml`, el cual contiene botones para añadir y borrar etiquetas. Adicionalmente posee un `GridView` que permitirá la visualización las imágenes en miniatura, como se indica en la Figura 2.17.

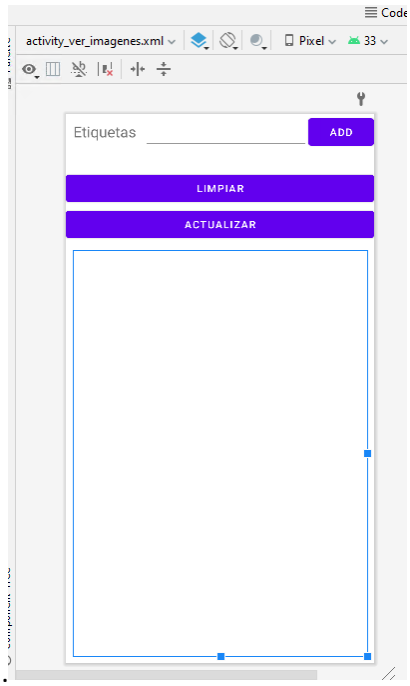


Figura 2.17. Creación de la vista para presentar imágenes

Adicionalmente se creó el archivo `activity_upload.xml` para la subida de imágenes que se seleccionarán desde el dispositivo móvil.

La vista contiene dos botones uno que permite la selección de imágenes y otro que permite borrar la información del `GridView`, como se observa en la Figura 2.18.

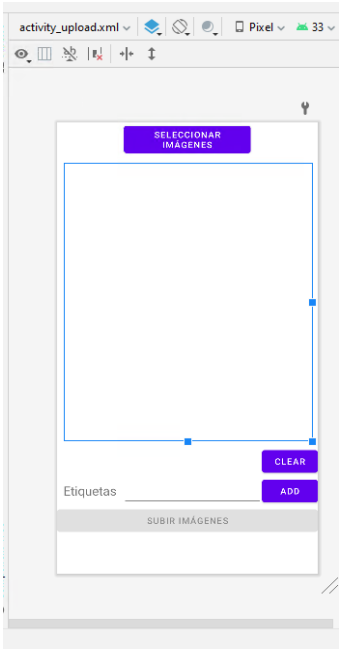


Figura 2.18. Creación de la vista para cargar imágenes

Aparte se creó el componente gráfico que se presentará en el `GridView`, el cual se indica en la Figura 2.19, como se aprecia se tiene `ImageView` y `TextView`, los cuales permitirán presentar la imagen y un texto, respectivamente.

2.2.4 PERMISOS DE LA APLICACIÓN MÓVIL

En Android se necesita otorgar permisos para el funcionamiento de la aplicación, lo cual se realiza en el archivo `AndriodManifest.xml`. Como se observa la Figura 2.20, con la línea 5 se permite el uso de Internet, mientras que con la línea 6 se otorga el permiso para escribir en el sistema de almacenamiento del teléfono y con la línea 7 se permite la lectura del sistema de almacenamiento del dispositivo móvil. Después en con la línea 15 se permite el uso alto de memoria RAM, y finalmente con la línea 18 se posibilita el tráfico HTTP en texto plano.

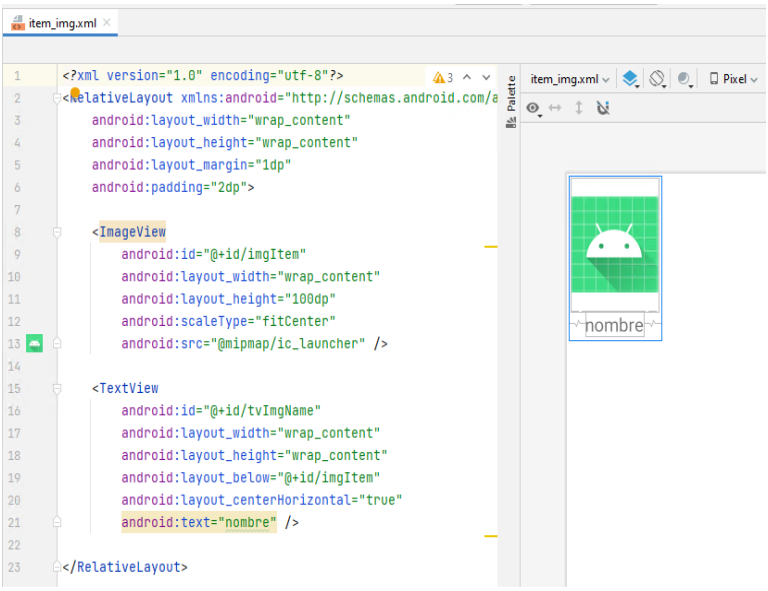


Figura 2.19. Componente gráfico de los elementos del `GridView`

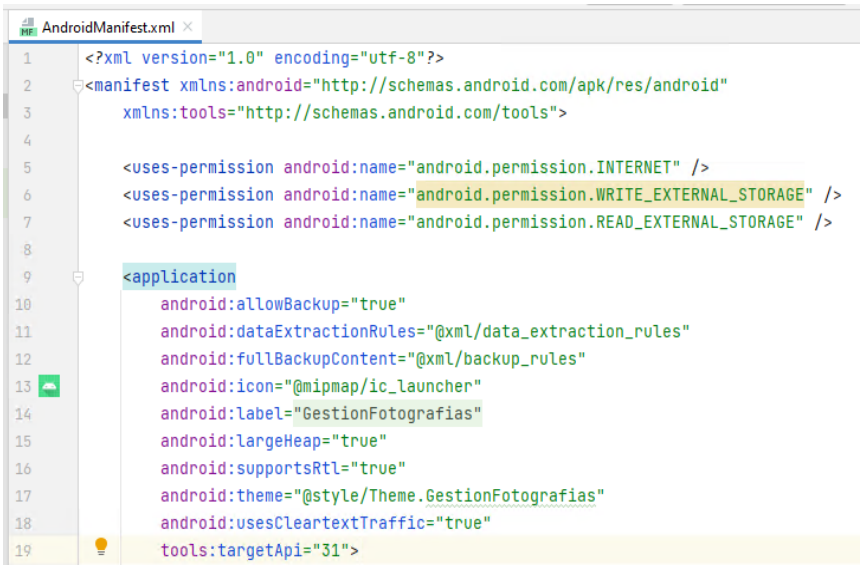


Figura 2.20. Permisos necesarios para el uso de la aplicación móvil

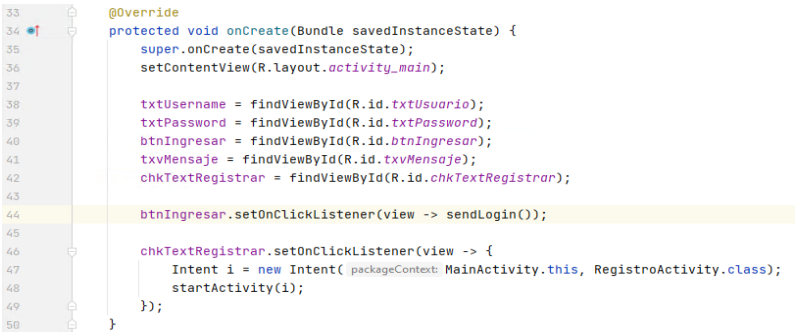
2.2.5 IMPLEMENTACIÓN DE LAS CLASES

Implementación de la clase MainActivity.java

Como parte de la implementación de la clase `MainActivity.java` se realizó lo siguiente:

En el método `onCreate` (ver Figura 2.21) se asocian las vistas con los objetos respectivos de la clase `MainActivity.java` (línea 38 a la 42), mientras que en las líneas 44 y 46 se

asocia el escuchador respectivo al evento `onclick` del botón `Ingresar` y del texto `Registrarse`, respectivamente.



```
33      @Override
34      protected void onCreate(Bundle savedInstanceState) {
35          super.onCreate(savedInstanceState);
36          setContentView(R.layout.activity_main);
37
38          txtUsername = findViewById(R.id.txtUsuario);
39          txtPassword = findViewById(R.id.txtPassword);
40          btnIngresar = findViewById(R.id.btnIngresar);
41          txvMensaje = findViewById(R.id.txvMensaje);
42          chkTextRegistrar = findViewById(R.id.chkTextRegistrar);
43
44          btnIngresar.setOnClickListener(view -> sendLogin());
45
46          chkTextRegistrar.setOnClickListener(view -> {
47              Intent i = new Intent( packageContext: MainActivity.this, RegistroActivity.class);
48              startActivity(i);
49          });
50      }
```

Figura 2.21. Implementación de la clase `MainActivity.java` (parte 1)

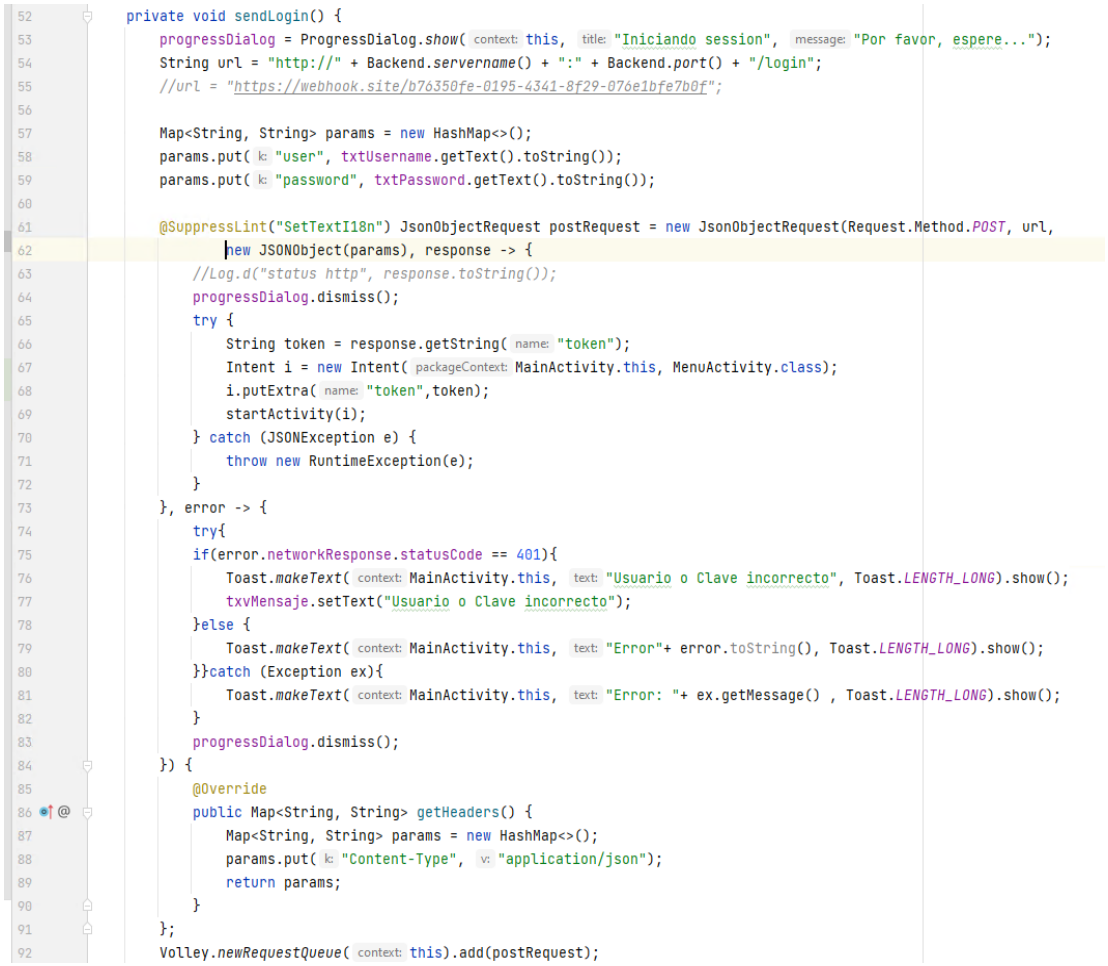
El código del método `sendLogin`, el cual realiza una petición POST hacia el *stub*, y se ejecuta al presionar el botón `Ingresar`, se presenta en la Figura 2.22, y permite mostrar un `ProgressDialog` que se visualiza mientras se realiza la petición y se espera por la respuesta. Se genera la información necesaria para la petición y se crea la petición POST hacia la URL enviando un JSON con el usuario y *password* (ver línea 61).

Se sobrescribe el método `ResponseListener` y `ErrorListener`. En el método `ReponseListener`, en la variable `response`, se obtiene el *token* el cual se envía al `menuActivity`, mientras que en `ErrorListener` se verifica si existe un `error 401` para que se muestre el mensaje “Usuario o clave incorrecta”, caso contrario se captura una excepción (líneas 64 a 72). Se sobrescribe las cabeceras de la petición agregando el campo `Content-Type`, `application/json` (línea 86 a 90). Finalmente se ejecuta la petición en la línea 92.

Implementación de la clase RegistroActivity.java

La clase `RegistroActivity.java`, que se presenta en la Figura 2.23 y contiene la lógica de la pantalla de registro, captura la información ingresada por el usuario y los envía en una petición POST hacia el *stub*.

Para esto, Se asigna el método `setOnClickListener` del botón registrar, se captura los datos ingresados por el usuario (líneas 41 y 44), se valida que los datos ingresados no se encuentren vacíos (línea 46), se compara que la contraseña y su confirmación sean iguales (línea 47), si las validaciones son correctas se ejecuta el método `registrarse` caso contrario se muestra sus respectivos mensajes de error.



```
52    private void sendLogin() {
53        progressDialog = ProgressDialog.show( context: this,  title: "Iniciando session",  message: "Por favor, espere...");
54        String url = "http://" + Backend.servername() + ":" + Backend.port() + "/login";
55        //url = "https://webhook.site/b76350fe-0195-4341-8f29-076e1bfe7b0f";
56
57        Map<String, String> params = new HashMap<>();
58        params.put( k: "user", txtUsername.getText().toString());
59        params.put( k: "password", txtPassword.getText().toString());
60
61        @SuppressLint("SetTextI18n") JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url,
62                new JSONObject(params), response -> {
63            //Log.d("status http", response.toString());
64            progressDialog.dismiss();
65            try {
66                String token = response.getString( name: "token");
67                Intent i = new Intent( packageContext: MainActivity.this, MenuActivity.class);
68                i.putExtra( name: "token",token);
69                startActivity(i);
70            } catch (JSONException e) {
71                throw new RuntimeException(e);
72            }
73        }, error -> {
74            try{
75            if(error.networkResponse.statusCode == 401){
76                Toast.makeText( context: MainActivity.this,  text: "Usuario o Clave incorrecto", Toast.LENGTH_LONG).show();
77                txvMensaje.setText("Usuario o Clave incorrecto");
78            }else {
79                Toast.makeText( context: MainActivity.this,  text: "Error"+ error.toString(), Toast.LENGTH_LONG).show();
80            }}catch (Exception ex){
81                Toast.makeText( context: MainActivity.this,  text: "Error: "+ ex.getMessage() , Toast.LENGTH_LONG).show();
82            }
83            progressDialog.dismiss();
84        }) {
85            @Override
86 @          public Map<String, String> getHeaders() {
87                Map<String, String> params = new HashMap<>();
88                params.put( k: "Content-Type",  v: "application/json");
89                return params;
90            }
91        };
92        Volley.newRequestQueue( context: this).add(postRequest);
```

Figura 2.22. Implementación de la clase MainActivity.java (parte 2)

```
29    @Override
30    protected void onCreate(Bundle savedInstanceState) {
31        super.onCreate(savedInstanceState);
32        setContentView(R.layout.activity_registro);
33        txtNewNombre = findViewById(R.id.txtNewNombre);
34        txtNewUser = findViewById(R.id.txtNewUser);
35        txtNewEmail = findViewById(R.id.txtNewEmail);
36        txtNewPass = findViewById(R.id.txtNewPass);
37        txtConfirmPass = findViewById(R.id.txtConfirmPass);
38        btnRegistrar = findViewById(R.id.btnRegistrar);
39
40        btnRegistrar.setOnClickListener(view -> {
41            nombre = txtNewNombre.getText().toString();
42            user = txtNewUser.getText().toString();
43            email = txtNewEmail.getText().toString();
44            pass = txtNewPass.getText().toString();
45
46            if (nombre.trim().length()!=0 && user.trim().length()!=0 && email.trim().length()!=0 && pass.trim().length()!=0) {
47                if (txtConfirmPass.getText().toString().equals(pass)) {
48                    registrarse();
49                } else {
50                    Toast.makeText( context: this,  text: "La contrseña no coincide", Toast.LENGTH_LONG).show();
51                }
52            } else{
53                Toast.makeText( context: this,  text: "Llene todos los campos", Toast.LENGTH_LONG).show();
54            }
55        });
56    }
```

Figura 2.23. Asociación de las vistas de la clase `RegistroActivity.java`

Posteriormente se crea el método `registrarse`, el cual genera una petición POST hacia el *stub* enviando un objeto JSON con las variables `nombre`, `userName`, `correo`, `password`, como se observa en la Figura 2.24, se procesan los datos ingresados por el usuario (línea 62 a 65), se crea la petición POST hacia la URL enviado un JSON con los datos del usuario, se sobrescribe el método `ResponseListener` y `ErrorListener`. En el método `ReponseListener` en la variable response se obtiene el *token* el cual se envía al `menuActivity`, mientras que en `ErrorListener` se verifica si existe un `error 401` para que se muestre el mensaje "`Error al registrarse`", caso contrario se captura una excepción (líneas 69 a 81). Finalmente se ejecuta la petición.

```
58      private void registrarse() {
59          String url = "http://" + Backend.servername() + ":" + Backend.port() + "/registro";
60          //url = "https://webhook.site/b76350fe-0195-4341-8f29-076e1bfe7b0f";
61
62          Map<String, String> params = new HashMap<>();
63          params.put( k: "uombre", txtNewNombre.getText().toString());
64          params.put( k: "userName", txtNewUser.getText().toString());
65          params.put( k: "correo", txtNewEmail.getText().toString());
66          params.put( k: "password", txtNewPass.getText().toString());
67
68          @SuppressLint("SetTextI18n") JsonObjectRequest postRequest = new JsonObjectRequest(Request.Method.POST, url, new JSONObject(params), response -> {
69              //Log.d("status http", response.toString());
70              try {
71                  String token = response.getString( name: "token");
72                  Intent i = new Intent( packageContext: RegistroActivity.this, MenuActivity.class);
73                  i.putExtra( name: "token", token);
74                  startActivity(i);
75              } catch (JSONException e) {
76                  throw new RuntimeException(e);
77              }
78          }, error -> {
79              if (error.networkResponse.statusCode == 401) {
80                  Toast.makeText( context: RegistroActivity.this, text: "Error al registrarse", Toast.LENGTH_LONG).show();
81              }
82          }) {
83              @Override
84              public Map<String, String> getHeaders() {
85                  Map<String, String> params = new HashMap<>();
86                  params.put( k: "Content-Type", v: "application/json");
87                  return params;
88              }
89          };
90          Volley.newRequestQueue( context: this).add(postRequest);
91      }
92  }
```

Figura 2.24. Método `registrarse` de la clase RegistroActivity.java

Implementación de la clase VerImagenes.java

La clase `VerImagenes.java` permite enviar un arreglo de etiquetas ingresadas por el usuario, obtener imágenes desde el *stub* y mostrarlas en la aplicación. Para la implementación se siguieron los siguientes pasos (ver Figura 2.25):

a. Se crea una petición GET enviando como parámetros las etiquetas ingresadas por el usuario hacia la URL del *stub* (línea 78). En el método `ResponseListener` utilizando la clase `ImageAdapter` se asocia la lista de imágenes recibidas desde

el *stub*, y se asigna el `adapter` al `gridview` para mostrar las imágenes en la aplicación.

b. Se agrega la cabecera `authorization` con el *token* recibido al ingresar a la aplicación móvil a través del método `getHeaders` como se muestra en la Figura 2.26.

```
67  @   private void getImages(String token, ArrayList<String> etiquetas) {
68          progressDialog = ProgressDialog.show( context: this,  title: "Obteniendo imágenes",  message: "Por favor, espere...");
69          String url = "http://" + Backend.servername() + ":" + Backend.port() + "/images";
70          StringBuilder etiq = new StringBuilder();
71
72          for (String etiqueta : etiquetas) {
73              etiq.append("etiquetas=").append(etiqueta).append('&');
74          }
75
76          ArrayList<Imagen> imagens = new ArrayList<>();
77
78          StringRequest getRequest = new StringRequest(Request.Method.GET,  url: url + "?" + etiq, response -> {
79              try {
80                  JSONArray jsonImages = new JSONArray(response);
81                  for (int i = 0; i < jsonImages.length(); i++) {
82                      JSONObject jsonImg = jsonImages.getJSONObject(i);
83                      String nombre = jsonImg.getString( name: "nombreImagen");
84                      String urlImg = jsonImg.getString( name: "img");
85                      imagens.add(new Imagen(nombre, urlImg));
86                  }
87                  ImagesAdapter imagesAdapter = new ImagesAdapter(imagens,  contex: this);
88                  gvImagenes.setAdapter(imagesAdapter);
89
90              } catch (JSONException e) {
91                  Log.e( tag: "Array Images", e.getMessage());
92              }
93              progressDialog.dismiss();
94          }, error -> {
95              try {
96                  Log.e( tag: "GET IMG", error.getMessage());
97                  Toast.makeText( context: this, error.getMessage(), Toast.LENGTH_LONG).show();
```

Figura 2.25. Clase `VerImagenes.java`

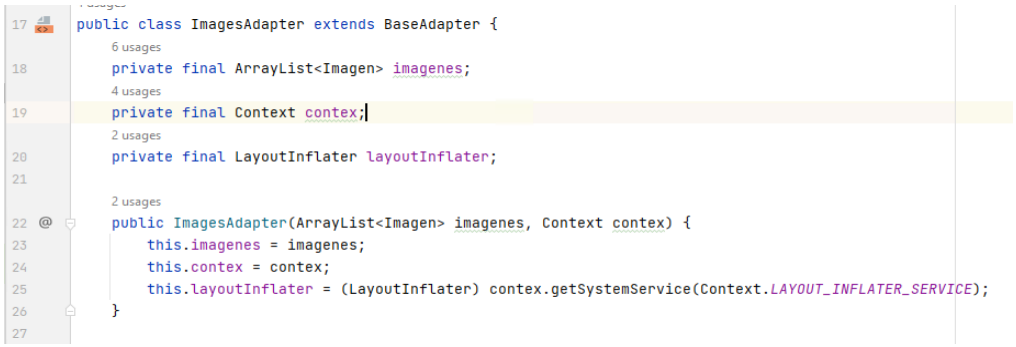
```
103             @Override
104 @           public Map<String, String> getHeaders() {
105                 HashMap<String, String> headers = new HashMap<>();
106                 headers.put("authorization", "Bearer " + token);
107                 return headers;
108             }
109
110         };
111         Volley.newRequestQueue( context: this).add(getRequest);
112     }
113 }
```

Figura 2.26. Envío de la petición GET

Implementación de la clase ImagesAdapter.java

En el archivo `ImagesAdapter.java` se tiene una clase que hereda de la clase `BaseAdapter`, en la cual se indican los atributos y métodos que se emplean para mostrar

elementos en un `gridView`. Para la implementación se siguieron los pasos a continuación (ver Figura 2.27):



```
17  public class ImagesAdapter extends BaseAdapter {
        6 usages
18      private final ArrayList<Imagen> imagenes;
        4 usages
19      private final Context contex;
        2 usages
20      private final LayoutInflater layoutInflater;
21
        2 usages
22  @   public ImagesAdapter(ArrayList<Imagen> imagenes, Context contex) {
23          this.imagenes = imagenes;
24          this.contex = contex;
25          this.layoutInflater = (LayoutInflater) contex.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
26      }
27
```

Figura 2.27. Clase `ImagesAdapter.java`

a. Se crea los atributos indicando el tipo de datos y su respectivo constructor (líneas 17 a 26).

b. Se sobrescribe el método `getView` en el cual se asocia la vista `item_img` y sus componentes.

   - Se asigna el nombre de cada imagen al `texview` de cada `item` y utilizando la librería *Glide* se asigna cada imagen recibida al `texView` de cada item. (líneas 52 y 53 de la Figura 2.28).

c. Se asigna el método `onClick` a cada elemento del `Adapter` el cual envía la URL de la imagen seleccionada hacia el `ImageActivity` para mostrarla en mayor tamaño.

```
44      public View getView(int i, View view, ViewGroup viewGroup) {
45
46          if (view == null) {
47              view = layoutInflater.inflate(R.layout.item_img, viewGroup,  attachToRoot: false);
48          }
49
50          ImageView imgItem = view.findViewById(R.id.imgItem);
51          TextView tvImgName = view.findViewById(R.id.tvImgName);
52          Glide.with(view).load(imagenes.get(i).getUrl()).into(imgItem);
53          tvImgName.setText(imagenes.get(i).getNombre());
54
55          view.setOnClickListener(view1 -> {
56              Toast.makeText(contex, imagenes.get(i).getNombre(), Toast.LENGTH_LONG).show();
57              Intent intent = new Intent(contex, ImageActivity.class);
58              intent.putExtra( name: "imgUrl", imagenes.get(i).getUrl());
59              contex.startActivity(intent);
60          });
61          return view;
62      }
63  }
```

Figura 2.28. Método `getView`

Implementación de la clase UploadActivity,Java

La clase `UploadActivity` permite al usuario seleccionar imágenes del dispositivo, ingresar etiquetas y enviar esta información al *stub*. En la Figura 2.29 se realiza lo siguiente:

a. Se crean y asocian las variables con las vistas.

b. Se obtiene el *token* recibido desde el `Activity` anterior.

c. Se revisa que la aplicación tenga permiso de lectura del almacenamiento externo. En caso de no tener se solicita el permiso (líneas 69 a la 71).

d. Se asigna el evento `onClick` del botón `seleccionar imágenes` quien permite la apertura del selector de archivos del dispositivo móvil; pero filtrado el tipo de archivo a través de la línea de 75 de la Figura 2.30.

e. Se asigna permiso de selección múltiple y la acción de obtener contenido a través de la línea de código: `setAction(Intent.ACTION_GET_CONTENT);`.

f. Finalmente, con las imágenes seleccionadas se procede a sobrescribir el método `onActivityResult` el cual está asociado al selector de imágenes y va a permitir ir colocando las imágenes seleccionadas en un `GridView`.

```
51  protected void onCreate(Bundle savedInstanceState) {
52      super.onCreate(savedInstanceState);
53      setContentView(R.layout.activity_upload);
54
55      Bundle extras = getIntent().getExtras();
56      token = extras.getString( key: "token");
57
58      btnSelecImgs = findViewById(R.id.btnSelecImgs);
59      btnAddEtiqueta = findViewById(R.id.btnAddEtiqueta);
60      btnClearEtiq = findViewById(R.id.btnClearEtiq);
61      btnSubirImgs = findViewById(R.id.btnSubirImgs);
62      gvSelecImgs = findViewById(R.id.gvSelecImgs);
63      txtCreateEtiqueta = findViewById(R.id.txtCreateEtiqueta);
64      tvCrearEtiq = findViewById(R.id.tvCrearEtiq);
65
66      imageSelecAdapter = new ImageSelecAdapter(uris,  context: this);
67      gvSelecImgs.setAdapter(imageSelecAdapter);
68
69      if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
70          ActivityCompat.requestPermissions( activity: UploadActivity.this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, Read_Permission)
71      }
```

Figura 2.29. Clase `UploadActivity.java`

```
73      btnSelecImgs.setOnClickListener(view -> {
74          Intent intent = new Intent();
75          intent.setType("image/*");
76          intent.putExtra(Intent.EXTRA_ALLOW_MULTIPLE,  value: true);
77          intent.setAction(Intent.ACTION_GET_CONTENT);
78          startActivityForResult(Intent.createChooser(intent,  title: "Seleccione imágenes"),  requestCode: 1);
79      });
```

Figura 2.30. Evento `OnClick` del botón `seleccionar imágenes`

El método `getFileDataFromDrawable` recibe una URI (*Uniform Resource Identifier*)[43] de la imagen, la convierte en un *bitmap* para posteriormente comprimirla como png y retornarla como arreglo de *bytes*, la comprensión establecida fue de un ochenta por ciento como se observa en la Figura 2.31.

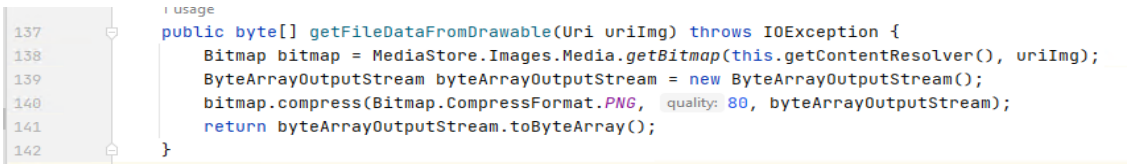


Figura 2.31. Método `getFileDataFromDrawable`

El método `uploadImage` (ver Figura 2.32) realiza una petición POST enviando un arreglo de imágenes hacia el *stub*. Se siguieron los siguientes pasos para su implementación:

a. Se muestra un `progressDialog` con el mensaje "`Subiendo imágenes`" (línea 145), hasta recibir una respuesta.

b. Utilizando la clase `VolleyMultipartRequest` la cual se sobrescribió para crear una petición de tipo `multipart/form-data` (línea 147), se crea una petición POST con cabecera de autorización con *token* (líneas 163-167) y se envía como parámetros un arreglo de *bytes* de las imágenes y una lista de etiquetas ingresadas (líneas 170-180).

   - Se sobrescribe el método `Response Listener` para crear un objeto `JSONObject` y mostrar el mensaje recibido en el atributo `message` del `response` (líneas 151 y 152).

Implementación de la clase VolleyMultipartRequest

La clase `VolleyMultipartRequest` permite realizar una petición POST la cual puede enviar en el archivos binarios y datos de formulario en el `body` de la petición. Para su implementación se siguieron los siguientes pasos, como se observa en la Figura 2.33:

a. Se sobrescribe el método `getBodyContentType` para retornar un `contentTye` "`multipart/form-data`".

[43] URI (*Uniform Resource Identifier):* Es una secuencia de caracteres que identifica a un recurso lógico o físico.

b. Se crea el método `buildDataPart` que utiliza una lista de `Datapart (bytes de imágenes con sus filenames)` y un `inputname` para crear un `InputStream` y enviar en la petición mediante la función `dataOutputStream.writeBytes(lineEnd)`.

```
144    private void uploadImage() {
145        progressDialog = ProgressDialog.show(this, "Subiendo imagenes", "Por favor, espere...");
146        String url = "http://" + Backend.servername() + ":" + Backend.port() + "/upload";
147        VolleyMultipartRequest volleyMultipartRequest = new VolleyMultipartRequest(Request.Method.POST, url,
148                response -> {
149                    progressDialog.dismiss();
150                    try {
151                        JSONObject jsonObject = new JSONObject(new String(response.data));
152                        Toast.makeText(this, jsonObject.getString("message"), Toast.LENGTH_SHORT).show();
153                        limpiar();
154                    } catch (JSONException e) {
155                        Log.e("JsonError", e.getMessage());
156                    }
157                },
158                error -> {
159                    progressDialog.dismiss();
160                    Toast.makeText(this, "Error al subir imagen", Toast.LENGTH_SHORT).show();
161                }) {
162            @Override
163            public Map<String, String> getHeaders() {
164                HashMap<String, String> headers = new HashMap<>();
165                headers.put("authorization", "Bearer " + token);
166                return headers;
167            }
168
169            @Override
170            protected Map<String, ArrayList<DataPart>> getByteData() throws IOException {
171                Map<String, ArrayList<DataPart>> params = new HashMap<>();
172                ArrayList<DataPart> dataParts = new ArrayList<>();
173
174                for (int i = 0; i < uris.size(); i++) {
175                    long imagename = System.currentTimeMillis();
176                    dataParts.add(new DataPart(imagename + ".png", getFileDataFromDrawable(uris.get(i))));
177                }
178                params.put("Imgs", dataParts);
179                return params;
180            }
181
182            @Override
183            protected Map<String, String> getParams() {
184                Map<String, String> params = new HashMap<>();
185                params.put("Etiquetas", String.valueOf(new JSONArray(etiquetas)));
186                return params;
187            }
188        };
189        Volley.newRequestQueue(this).add(volleyMultipartRequest);
190    }
```

Figura 2.32. Método `uploadImage`

```
69 @    private void buildDataPart(DataOutputStream dataOutputStream, ArrayList<DataPart> dataFiles, String inputName)
70          for (DataPart dataFile : dataFiles) {
71              dataOutputStream.writeBytes( s: twoHyphens + boundary + lineEnd);
72              dataOutputStream.writeBytes( s: "Content-Disposition: form-data; name=\"" +
73                      inputName + "\"; filename=\"" + dataFile.getFileName() + "\"" + lineEnd);
74              if (dataFile.getType() != null && !dataFile.getType().trim().isEmpty()) {
75                  dataOutputStream.writeBytes( s: "Content-Type: " + dataFile.getType() + lineEnd);
76              }
77              dataOutputStream.writeBytes(lineEnd);
78
79              ByteArrayInputStream fileInputStream = new ByteArrayInputStream(dataFile.getContent());
80              int bytesAvailable = fileInputStream.available();
81
82              int maxBufferSize = 1024 * 1024;
83              int bufferSize = Math.min(bytesAvailable, maxBufferSize);
84              byte[] buffer = new byte[bufferSize];
85
86              int bytesRead = fileInputStream.read(buffer,  off: 0, bufferSize);
87
88              while (bytesRead > 0) {
89                  dataOutputStream.write(buffer,  off: 0, bufferSize);
90                  bytesAvailable = fileInputStream.available();
91                  bufferSize = Math.min(bytesAvailable, maxBufferSize);
92                  bytesRead = fileInputStream.read(buffer,  off: 0, bufferSize);
93              }
94
95              dataOutputStream.writeBytes(lineEnd);
96          }
97      }
```

Figura 2.33. Clase `VolleyMultipartRequest`

Implementación de la clase MenuActivity

En la clase `MenuActivity` se recibe el token del `MainActivity` y se asocia a los eventos `OnClick` a cada botón que redireccionará a las distintas actividades, enviando el *token* o permitirá finalizar la aplicación como se detalla en la Figura 2.34.

```
9   public class MenuActivity extends AppCompatActivity {
10
        2 usages
11      Button btnConsultar, btnCargarImagenes, btnSalir;
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_menu);
16
17          Bundle extras = getIntent().getExtras();
18          String token = extras.getString( key: "token");
19
20          btnConsultar = findViewById(R.id.btnConsultar);
21          btnCargarImagenes = findViewById(R.id.btnCargarImagenes);
22          btnSalir = findViewById(R.id.btnSalir);
23
24          btnConsultar.setOnClickListener(view -> {
25              Intent i = new Intent( packageContext: this, VerImagenes.class);
26              i.putExtra( name: "token",token);
27              startActivity(i);
28          });
29
30          btnCargarImagenes.setOnClickListener(view -> {
31              Intent i = new Intent( packageContext: this, UploadActivity.class);
32              i.putExtra( name: "token",token);
33              startActivity(i);
34          });
35
36          btnSalir.setOnClickListener(view -> this.finishAffinity());
37      }
38  }
```

Figura 2.34. Clase `MenuActivity`

2.2.6 CREACIÓN DEL STUB

Para la creación del *stub* se siguieron los siguientes pasos:

1. Descarga de NodeJS en la página oficial https://nodejs.org/en/.
2. Se crea una carpeta donde se va a crear el *stub*.
3. Se abre un terminal dentro de la carpeta y se ejecuta el comando `npm init` para inicializar un proyecto de NodeJS.
4. Se instalan las librerías necesarias para la creación de un servidor web con el comando `npm install express cors express-fileupload`.
5. Se crea el archivo `app.js` donde se implementarán todas las funciones y métodos.
   a. Importación de librerías (ver Figura 2.35).

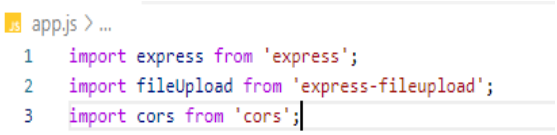


Figura 2.35. Importación de librerías del *stub*

   b. La creación de una aplicación de `express-js` se observa en la línea 5 de la Figura 2.36. En primer lugar, se inicia la aplicación, posteriormente en las líneas 6, 7 y 8 se agregan funciones para el manejo de peticiones, de archivos y la autorización, respectivamente y en la línea 10 se exponen los archivos que se localizan en la carpeta `public` en el servicio web.

```
5   var app = express();
6   app.use(cors())
7   app.use(express.json())
8   app.use(fileUpload());
9
10  app.use('/', express.static('public'));
```

Figura 2.36. Creación de una aplicación `express-js`

   c. Se implementa la ruta `/login`, la cual recibe mediante POST un usuario y *password*; si estos datos son correctos responde un objeto con el *token* y los

datos del usuario, caso contrario envía un mensaje de error, como se muestra en la Figura 2.37.

```
12  app.post('/login', (req, res) => {
13      const { user, password } = req.body
14      if (user == '' && password == '') {
15          res.json({
16              token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0
17              id: 3,
18              nombre: "Juan",
19              correo: "juan123@gmail.com",
20              userName: "Juan123",
21              message: "Login Correcto"
22          })
23      }
24      else {
25          res.status(401).json({ message: "Usuario o Clave incorrecto" });
26      }
27  })
```

Figura 2.37. Implementación de la ruta `/login`

d. Se implemente la ruta `/registro`, la misma que recibe una petición POST con los datos del usuario a registrar, imprime en consola los datos recibidos y envía el *token* y los datos de usuario, como se detalla en Figura 2.38.

e. Para la creación de la ruta `/Upload` se verifica que existan en el `body` del mensaje, la autorización a través del *token*, posteriormente recibe mediante POST un arreglo de etiquetas y un archivo o arreglo de archivos y los almacena dentro de la carpeta `/public/images/`. Después en consola imprime las etiquetas recibidas y envía el mensaje “imágenes guardadas” a la aplicación móvil, en caso de no recibir el t*oken* envía un mensaje de error; En la Figura 2.39 se presenta el código indicado.

```
29  app.post('/registro', (req, res) => {
30      console.log(req.body)
31      res.json({
32          token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODk
33          id: 3,
34          nombre: "Juan",
35          correo: "juan123@gmail.com",
36          userName: "Juan123",
37          message: "Login Correcto"
38      })
39  })
40
```

Figura 2.38. Implementación de la ruta `/registro`

```
41  app.post('/upload', (req, res) => {
42
43      const { authorization } = req.headers;
44      console.log(authorization);
45      if (authorization) {
46          const { Imgs } = req.files;
47          const { Etiquetas } = req.body;
48
49          if (!Imgs) return res.sendStatus(400);
50
51          if (Array.isArray(Imgs)) {
52              Imgs.forEach(img => {
53                  img.mv('./public/images/' + img.name);
54              });
55          } else {
56              Imgs.mv('./public/images/' + Imgs.name);
57          }
58          console.log(Etiquetas);
59          res.json({
60              message: "Imagen(s) guardada(s)"
61          });
62      } else {
63          res.status(401).json({ message: "Usuario no autenticado" });
64      }
65  });
```

Figura 2.39. Creación de la ruta `/Upload`

f. Se crea la ruta `/Images`, la cual permitirá consultar imágenes. Esta ruta recibe una petición GET con las etiquetas como parámetros y verificando la autorización con *token*. Después responde con un arreglo que contiene la URL de las imágenes con sus respectivas etiquetas y nombres, o un estado de error en caso de no recibir el t*oken,* como se indica en la Figura 2.40.

g. Se inicia el servidor web en el puerto 3000. Al iniciar, se mostrará el mensaje `Listening on http://localhost:3000` (ver Figura 2.41).

h. Finalmente se ejecuta el *stub* utilizando el comando `node app` con los comandos de la Figura 2.42.

```
app.get('/images', (req, res) => {
    console.log(req.query)
    const { authorization } = req.headers;
    console.log(authorization)
    if (authorization) {
        res.json([{
            id: 12,
            fechasubida: "23-01-14T10:25:45.9905282",
            nombreImagen: "imagen.jpg",
            img: "http://" + req.headers.host + "/images/galaxy-stars-space-24-4k.jpg",
            etiquetas: [
                "copa de la FIFA",
                "Messi",
                "Qatar"
            ]
        },
        {
            id: 15,
            fechasubida: "23-01-14T10:25:45.9905282",
            nombreImagen: "imagen2.jpg",
            img: "http://" + req.headers.host + "/images/K8S_DATA.png",
            etiquetas: [
                "copa de la FIFA",
                "Messi",
                "Qatar"
            ]
        },
        {
            id: 15,
            fechasubida: "23-01-14T10:25:45.9905282",
            nombreImagen: "imagen2.jpg",
            img: "http://" + req.headers.host + "/images/K8S_DATA.png",
            etiquetas: [
                "copa de la FIFA",
                "Messi",
                "Qatar"
            ]
        }])
    }
    else {
        res.sendStatus(401);
    }
})
```

Figura 2.40. Creación de la `ruta/Images`

```
app.listen(3000, () => {
    console.log('Listening on http://localhost:3000')
})
```

Figura 2.41. Código para iniciar el servidor web

```
PS D:\Documents\Tutos\stub> node app
Listening on http://localhost:3000
```

Figura 2.42 Ejecución del *stub*

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1 RESULTADOS

Con el fin de demostrar la funcionalidad del subsistema desarrollado se realizaron pruebas de funcionamiento y validación. Las primeras se hicieron para verificar el cumplimiento de los requerimientos establecidos para el presente trabajo. Para la realización de las pruebas de validación se aplicó una encuesta a cuatro personas a las cuales se les instaló la aplicación desarrollada para su respectivo uso.

Posteriormente a la realización de las pruebas se realizó la corrección de errores y mejoras solicitadas en el sistema.

3.1.1 PRUEBAS DE FUNCIONAMIENTO

Las pruebas de funcionamiento se realizaron con base en los requerimientos funcionales del sistema para cada componente.

Pruebas de funcionamiento del módulo de *Login*

Prueba: Registro de usuarios nuevos

Funcionamiento:

1. Presionar el texto: ¿No tienes una cuenta? Regístrate.
2. Llenar los datos solicitados en la nueva vista (ver Figura 3.1).
3. Presionar el botón `REGISTRAR`.

Figura 3.1.Registro de un usuario nuevo

Resultado: Se recibe los datos en el *stub* como se observa en la Figura 3.2, después se genera el *token* y en la aplicación inicia la sesión.

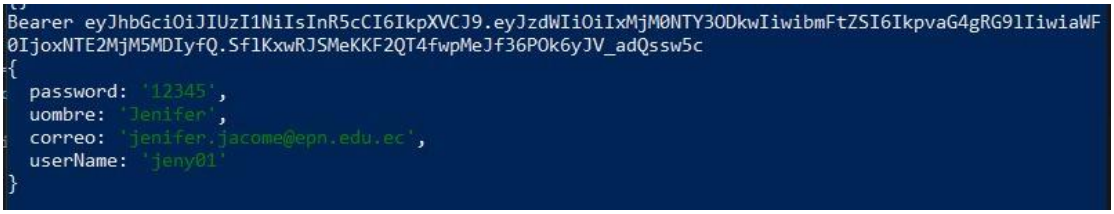


Figura 3.2. Respuesta del *stub* al registrar un nuevo usuario

Prueba: Inicio de sesión positiva.

Funcionamiento:

1. Ingresar correctamente los datos de usuario y contraseña como lo indica la Figura 3.3.
2. Presionar el botón `INGRESAR`.

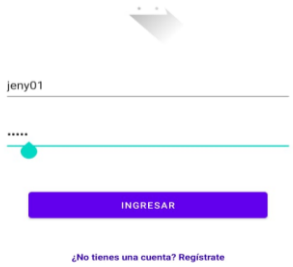


Figura 3.3. Ingreso de los datos en el módulo *Login*

Resultado: La aplicación redirige al usuario a el menú de la aplicación como se muestra en la Figura 3.4.

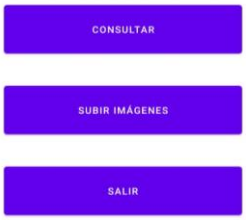


Figura 3.4. Menú de la aplicación al iniciar sesión

Prueba: Inicio de sesión de sesión negativa.

Procedimiento:

1. Ingresar usuario o contraseña incorrectos.
2. Presionar el botón `INGRESAR`.

Resultado:

Se muestra en el módulo de *Login* un mensaje de error: “Usuario o Clave Incorrecto” como se indica en la **Figura 3.5**.

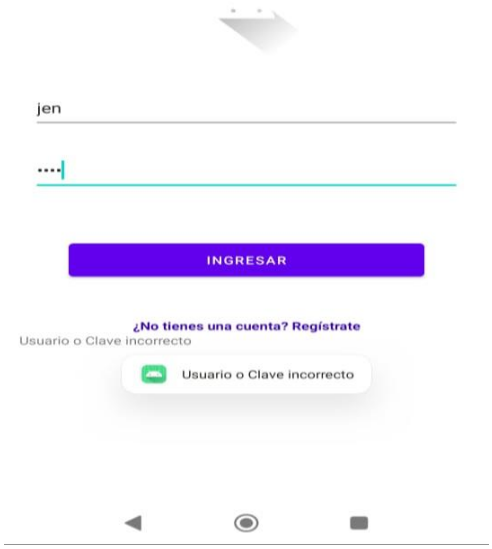


Figura 3.5. Resultado al ingresar un Usuario o Clave Incorrecta en el módulo de Inicio de Sesión

Módulo de consulta de imágenes

Prueba: Ingreso etiquetas.

Funcionamiento:

1. Ingreso de las etiquetas en el espacio correspondiente.
2. Presionar el botón `ADD`.

Resultado:

Se muestra en la pantalla el arreglo de etiquetas ingresadas separadas por comas como lo indica la Figura 3.6.

Figura 3.6. Resultado al ingresar diferentes etiquetas

Prueba: Despliegue de las imágenes con las etiquetas seleccionadas.

Funcionamiento:

1. Ingreso de etiquetas.
2. Presionar el botón `ACTUALIZAR`.
3. Visualización de las imágenes en la aplicación como indica la Figura 3.7.

Figura 3.7. Visualización de las imágenes en la aplicación

Resultado: En el *stub* se imprime las etiquetas recibidas (ver **Figura 3.8**) y el *token* enviado; mientras que en la aplicación se visualizan las imágenes enviadas por el *stub*.

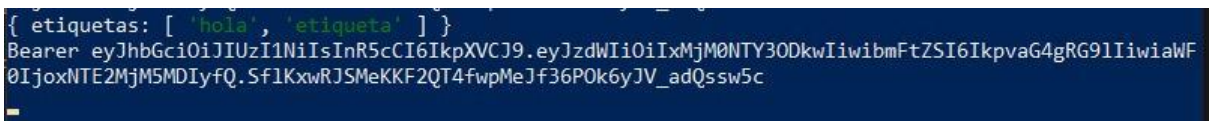


Figura 3.8. Respuesta del *stub* en la prueba Despliegue de las imágenes

Prueba: Limpieza de campos del módulo de consulta de imágenes.

Funcionamiento:

1. Después de traer las imágenes con las etiquetas ingresadas.
2. Presionar el botón `LIMPIAR`.

Resultado:

Se borran los campos de las etiquetas e imágenes obtenidas como se detalla en la Figura 3.9.

Figura 3.9. Limpieza de campos del módulo de consulta de imágenes

Módulo subir imágenes

Prueba: Selección de imágenes.

Procedimiento:

1. Presionar el botón `SELECCIONAR IMÁGENES`.
2. Se abre el gestor de archivos multimedia.
3. Seleccionar las imágenes que se desean cargar desde el gestor de archivos multimedia del dispositivo móvil.

Resultado: Se debe mostrar en la aplicación las imágenes seleccionadas en miniatura como lo indica la Figura 3.10. En caso de no seleccionar ninguna imagen muestra el mensaje de error: No ha seleccionado ninguna imagen.

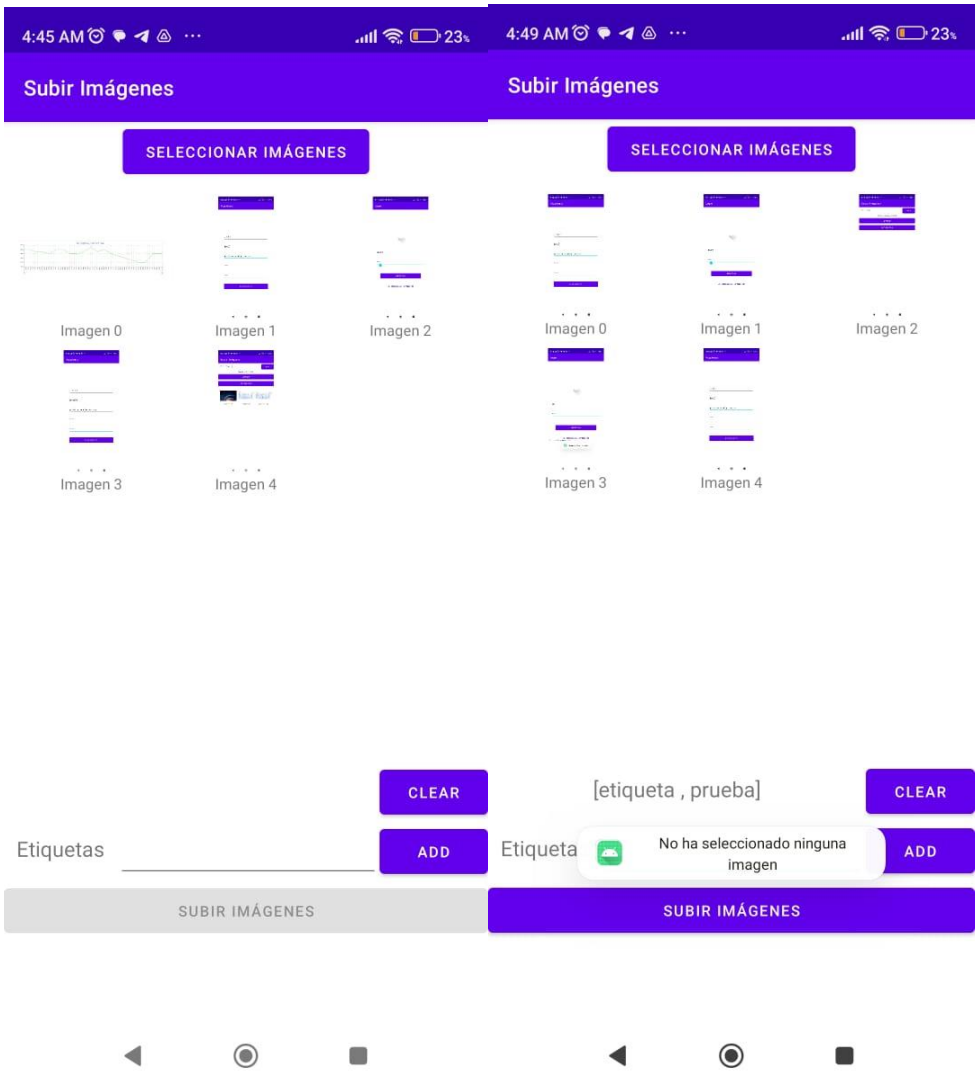


Figura 3.10. Prueba de funcionamiento de selección de imágenes

Prueba: Agregar etiquetas a las imágenes seleccionadas.

Funcionamiento:

1. Después de cargar las imágenes en la aplicación.
2. Escribir las etiquetas que se desean agregar a las imágenes.
3. Seleccionar el botón `ADD`.

Resultado: se muestra el arreglo de etiquetas ingresadas en la pantalla. En caso de intentar presionar el botón `ADD` con el campo de etiquetas vacío muestra el mensaje de error como muestra la Figura 3.11.

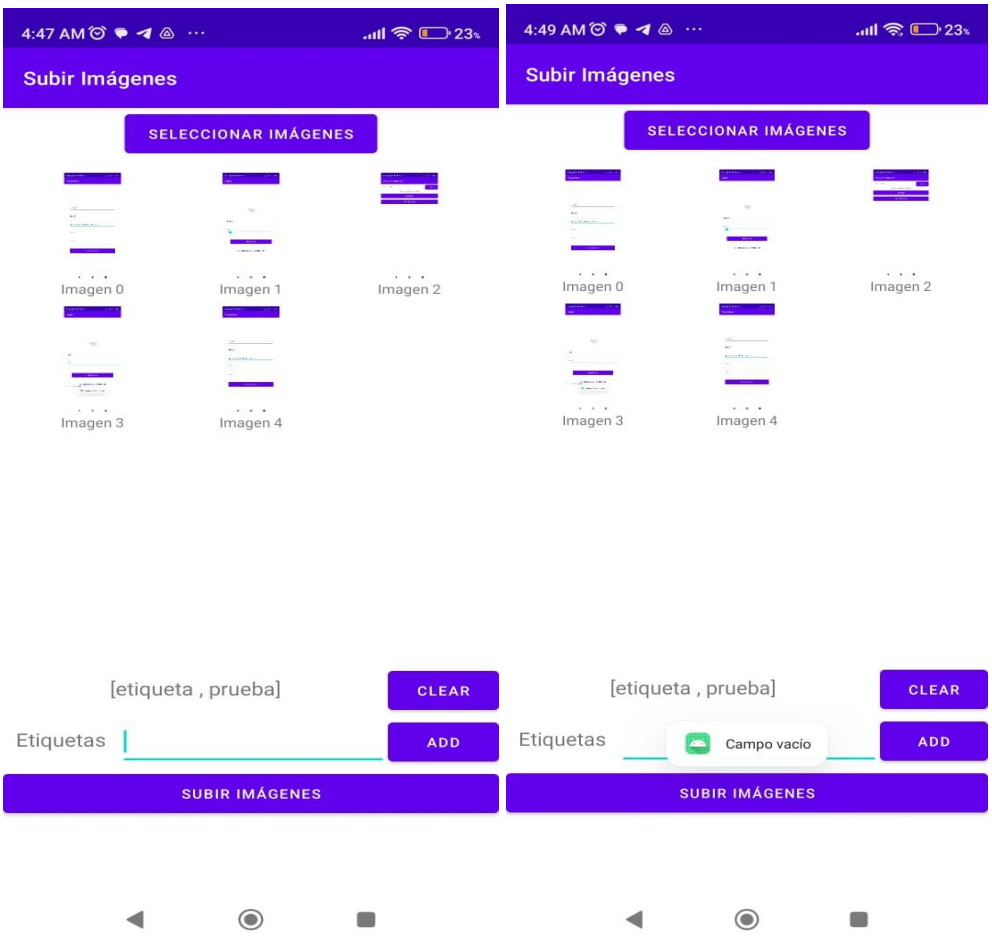


Figura 3.11. Agregar etiquetas a las imágenes seleccionadas

Prueba: Subir imágenes.

Funcionamiento:

1. Después de seleccionar las imágenes y agregar etiquetas, se activa el botón `SUBIR IMÁGENES`.
2. Se presiona el botón `SUBIR IMÁGENES`.
3. Se muestra las imágenes guardadas y se limpian todos los campos como se muestra en Figura 3.12.

Resultado: En el *stub* se guardan las imágenes en la carpeta `public/images` (ver Figura 3.13) y se imprime el *token* y etiquetas recibidas como lo indica la Figura 3.14.

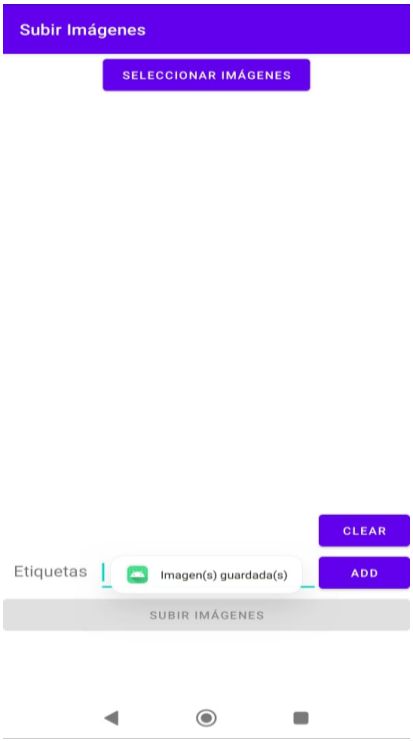


Figura 3.12. Subir imágenes al *stub*

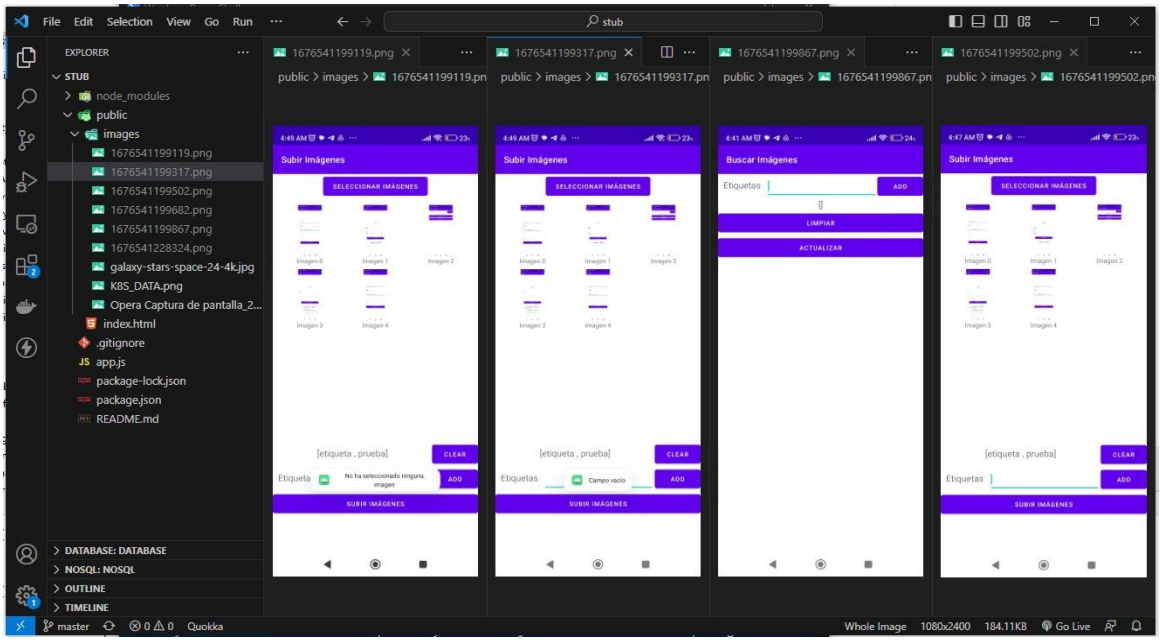


Figura 3.13. Guardado de imágenes seleccionadas en la carpeta `public/images`

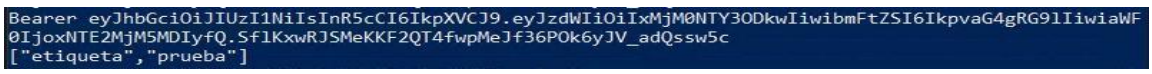


Figura 3.14. Impresión del *token* y etiquetas recibidas en el *stub*

3.1.2 PRUEBAS DE VALIDACIÓN

Para las pruebas de validación del subsistema se pidió a cuatro usuarios que prueben la aplicación Android, para posteriormente llenar la encuesta. El formato de la encuesta y sus resultados se encuentran en el Anexo II. Un resumen de los resultados de la encuesta de validación se encuentra tabulados en la Tabla 3.1 y Tabla 3.2.

La encuesta realizada consta de 18 preguntas, las cuales se basaron en los requerimientos funcionales para cada módulo del subsistema.

Los resultados indican que el 50% de las personas presentan inconvenientes al agregar etiquetas a las imágenes seleccionadas, esto corresponde al módulo de subida de imágenes, en los comentarios de la encuesta los usuarios indicaron que el teclado del teléfono bloquea visualmente el campo donde se ingresan las etiquetas. Otro inconveniente presentando en las pruebas de validación corresponde al subir las imágenes al *stub*, de igual manera esto corresponde al módulo de subida de imágenes, una de las cuatro personas indicó que al subir una fotografía la aplicación se colgó por unos segundos y no verificó si la imagen se cargó correctamente en el *stub*. No se presentaron inconvenientes en los demás módulos del subsistema.

Tabla 3.1. Resumen de los resultados de la encuesta de validación (parte 1 de 2)

No	Pregunta	Respuesta %
1	**¿Inició sesión correctamente?**	
	Si	100
	No	0
2	**Si presentó algún problema en el inicio de sesión, descríbalo**	
	Sin respuestas	
3	**¿Pudo crear registrarse como un usuario nuevo?**	
	Si	100
	No	0
4	**Si presentó algún problema al registrarse como nuevo usuario, descríbalo**	
	Sin respuestas	
5	**¿Los botones del menú funcionan correctamente?**	
	Si	100
	No	0
6	**Si presentó algún problema en el menú de la aplicación, descríbalo**	
	Sin respuestas	
7	**¿Pudo ingresar correctamente las etiquetas a buscar?**	
	Si	100
	No	0
8	**Si presentó algún problema al ingresar las etiquetas a buscar, descríbalo**	
	Sin respuestas	

Tabla 3.2. Resumen de los resultados de la encuesta de validación (parte 2 de 2)

No	Pregunta	Respuesta %
9	**Al momento de ingresar las etiquetas, ¿Se desplegaron imágenes?**	
	Si	100
	No	0
10	**Si presentó algún problema el despliegue de las imágenes, descríbalo**	
	Sin respuestas	
11	**Al buscar imágenes al pulsar "LIMPIAR", ¿se vaciaron las imágenes y etiquetas?**	
	Si	100
	No	0
12	**Si presentó algún problema al momento de presionar "Limpiar", descríbalo**	
	Sin respuestas	
13	**Al subir imágenes pudo seleccionar los archivos multimedia que se encontraban en su teléfono**	
	Si	100
	No	0
14	**Si presentó algún problema al momento de subir imágenes, descríbalo**	
	Sin respuestas	
15	**¿Pudo agregar correctamente las etiquetas a las imágenes seleccionadas**	
	Si	50
	No	50
16	**Si presentó algún problema al momento de asignar etiquetas a las imágenes, descríbalo**	
	El teclado del teléfono no permite visualizar las etiquetas ingresadas El teclado bloquea la parte donde se ingresa las etiquetas	
17	**¿Pudo subir las imágenes al *stub*?**	
	Si	75
	No	25
18	**Si presentó problemas al subir imágenes al *stub*, descríbalo**	
	Al subir una fotografía la aplicación se colgó por unos segundos	

3.1.3 CORRECCIÓN DE ERRORES ENCONTRADOS

Como se evidenció en la pregunta 16 de la Tabla 3.2 el teclado del teléfono al momento de ingresar las etiquetas seleccionadas cubre los campos de las etiquetas y no permite su visualización, como se muestra en la Figura 3.15. Para corregir este error se edita el archivo `AndroidManifest.xml`, en la etiqueta `activity` se agrega la propiedad `android:windowSoftInputMode="adjustPan"` como se puede observar en la línea 28 de la Figura 3.16. Esta propiedad evita que se oculte el componente `EditText` en el cual se está ingresando texto bajo el teclado.

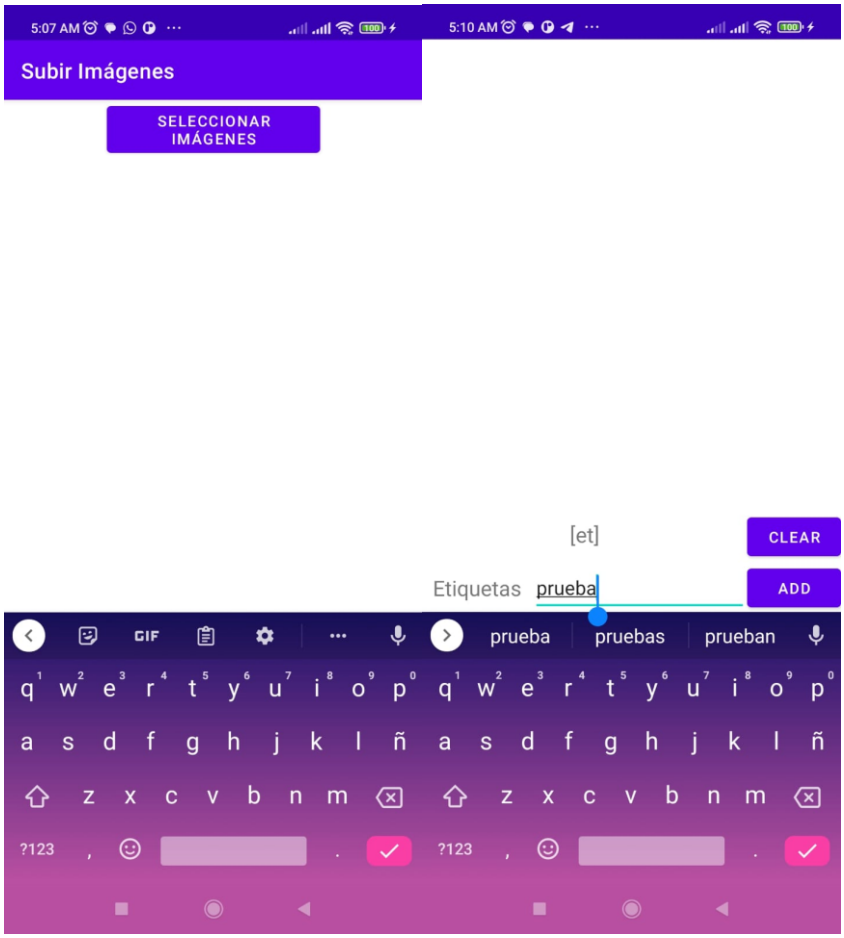


Figura 3.15. Antes y después del error encontrado.

```
25      <activity
26          android:name=".UploadActivity"
27          android:exported="false"
28          android:windowSoftInputMode="adjustPan"
29          android:label="Subir Imágenes" />
```

Figura 3.16. Corrección del error encontrado

En el componente para seleccionar imágenes, al seleccionar imágenes de un tamaño superior a 10MB la aplicación parece congelada y el usuario no tenía una experiencia adecuada porque la respuesta del servidor tardaba algunos segundos. Para mejorar esta característica de la aplicación al presionar en el botón `SUBIR IMÁGENES` se muestra un componente `ProgressDialog` con el mensaje `Subiendo imágenes` y se oculta cuando se recibe una respuesta por parte del *stub,* como lo muestra la Figura 3. 17 y Figura 3.18.

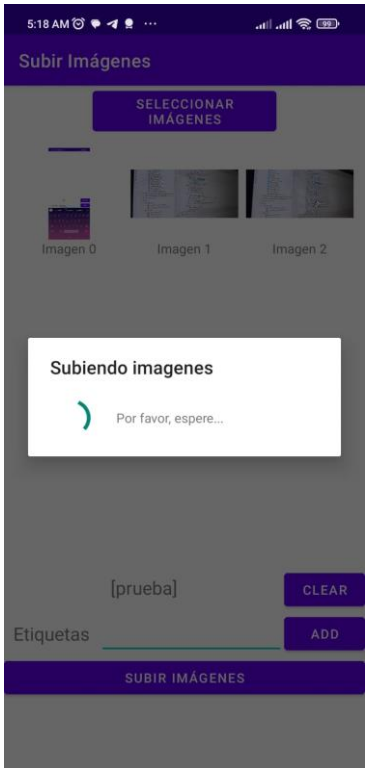


Figura 3. 17. Componente `ProgressDialog` mientras se cargan las imágenes en el *stub*

```
144    private void uploadImage() {
145        progressDialog = ProgressDialog.show( context: this,  title: "Subiendo imagenes",  message: "Por favor, espere...");
146        String url = "http://" + Backend.servername() + ":" + Backend.port() + "/upload";
147        VolleyMultipartRequest volleyMultipartRequest = new VolleyMultipartRequest(Request.Method.POST, url,
148                response -> {
149                    progressDialog.dismiss();
150                    try {
151                        JSONObject jsonObject = new JSONObject(new String(response.data));
152                        Toast.makeText( context: this, jsonObject.getString( name: "message"), Toast.LENGTH_SHORT).show();
153                        limpiar();
154                    } catch (JSONException e) {
155                        Log.e( tag: "JsonError", e.getMessage());
156                    }
157                },
158                error -> {
159                    progressDialog.dismiss();
160                    Toast.makeText( context: this,  text: "Error al subir imagen", Toast.LENGTH_SHORT).show();
161                }) {
```

Figura 3.18. Código del `ProgressDialog`

3.2 CONCLUSIONES

- Se desarrolló un subsistema de gestión de fotografías, el cual permite subir fotografías y asignarle etiquetas, y también realizar consultas de fotografías de acuerdo con las etiquetas asignadas. La aplicación se desarrolló para el sistema operativo Android y se utilizó la metodología Kanban para su desarrollo.

- Se analizó las herramientas necesarias para el manejo de fotografías para el sistema operativo Android para lo cual se investigó el uso de la clase `ImageView` para la muestra de archivos multimedia tipo imagen en una aplicación Android. Adicional se analizó el uso de la librería *Glide*, la cual proporciona a Android una API para una visualización más suave y amigable con el usuario de una lista de imágenes. También se investigó el uso de *Volley* Client para construir el cliente REST de la aplicación.

- Se diseñó el prototipo para que permita cargar fotografías con etiquetas y realizar consulta; para lo cual en primer lugar se elaboró la lista de requerimientos funcionales y no funcionales del sistema lo cual permitió establecer las tareas y el flujo del tablero Kanban y los diagramas UML correspondientes. El uso del tablero Kanban permitió asignar cada una de las tareas al ciclo de vida del *software*, facilitando el desarrollo, implementación y pruebas del prototipo y en consecuencia tener entregables funcionales de cada módulo del prototipo.

- Al implementar el subsistema de adquisición de fotografías de acuerdo con el diseño realizado se utilizó las herramientas consultadas en la primera fase del desarrollo del proyecto. Se cumplió con la implementación de las tres vistas establecidas en el alcance: vista de acceso, vista de selección y etiquetado de imágenes y la vista de búsqueda de imágenes. Pero, para una mejor experiencia del usuario se incrementó una vista para la creación de un nuevo usuario y un menú interactivo para poder ingresar a las diferentes opciones de la aplicación.

- Los resultados arrojados al ejecutar las pruebas de funcionamiento y validación del subsistema permitieron: en primer lugar, comprobar el funcionamiento de los tres módulos principales de la aplicación móvil; y, además, corregir y mejorar la experiencia del usuario al momento de ingresar etiquetas y al instante de cargar imágenes en el *stub*.

3.3 RECOMENDACIONES

- Se puede desarrollar la aplicación Android utilizando otras tecnologías como *React Native* o *Flutter* permitiendo el desarrollo multiplataforma (Android & iOS) para utilizar el mismo código.
- Para seleccionar las imágenes del dispositivo se utilizó el método `startActivityForResult` el cual se está dejando de utilizar gracias al desarrollo del nuevo método `ActivityResultLauncher` disponible para Android Studio, el cual se puede utilizar en el desarrollo de futuros de *software*.
- En el subsistema desarrollado para establecer las visualizaciones de las imágenes se utilizó el recurso `GridView`. Pero, otra opción es el elemento `RecyclerView` que facilita la observación de los ítems en listas o cuadriculas de forma dinámica. Además, con `RecyclerView` se tiene la posibilidad de mostrar los elementos de la lista de una forma más eficiente y personalizable; porque se puede reciclar los recursos que se utilizan para realizar el trabajo.
- El código desarrollado en Android se utilizó el lenguaje de programación Java el cual está siendo sustituido por *Kotlin* ya que nos proporciona y código más limpio y escalable. Se recomienda para trabajos futuros utilizar *Kotlin* como lenguaje de programación para aplicaciones Android.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] M. Tapia, «ESTUDIO Y DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES ANDROID,» Noviembre 2013. [En línea]. Disponible en: http://repositorio.utn.edu.ec/bitstream/123456789/2614/1/04%20ISC%20284%20TESIS.pdf. [Último acceso: 29 Enero 2023].

[2] J. Enriquez y S. Casas, «Usabilidad en aplicaciones móviles,» 11 Junio 2014. [En línea]. Disponible en: https://doi.org/10.22305/ict-unpa.v5i2.71. [Último acceso: 15 Enero 2023].

[3] O. Constantin, M. Novac, C. Gordan, T. Berczes y G. Bujdosó, «Comparative study of Google Android, Apple iOS and Microsoft Windows Phone mobile operating systems,» 17 Julio 2017. [En línea]. Disponible en: 10.1109/EMES.2017.7980403. [Último acceso: 15 Enero 2023].

[4] J. Cuello y J. Vittone, Diseñando apps para móviles, Catalina Duque Giraldo, 2013.

[5] Developers Android, «Arquitectura de la plataforma,» 07 Mayo 2020. [En línea]. Disponible en: https://developer.android.com/guide/platform?hl=es-419. [Último acceso: 5 Febrero 2023].

[6] Universidad Politécnica de Valencia, «Arquitectura de Android,» 2017. [En línea]. Disponible en: http://www.androidcurso.com/index.php/recursos/31-unidad-1-vision-general-y-entorno-de-desarrollo/99-arquitectura-de-android. [Último acceso: 13 Enero 2023].

[7] P. Blanco, J. Camarero, A. Fumero, A. Werterski y P. Rodríguez, «Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone,» 2016. [En línea]. Disponible en: https://www.researchgate.net/profile/Antonio_Fumero/publication/267795011_Metodologia_de_desarrollo_agil_para_sistemas_moviles_Introduccion_al_desarrollo_con_Android_y_el_iPhone/links/577009d108ae842225aa444b/Metodologia-de-desarrollo-agil-para-sistemas-m.

[8] J. Fernández, Alor, Giner y M. Paredes, «Generación automática de interfaces de usuario de aplicaciones para dispositivos móviles para el ámbito de cuidado de la salud mediante comandos de voz,» 4 DIciembre 2020. [En línea]. [Último acceso: 10 Enero 2023].

[9] J. Gironés, El gran Libro de Android, Bogotá: Marcombo, 2019.

[10] V. González, «El ciclo de vida de una aplicación Android,» Androidsis, 20 Noviembre 2018. [En línea]. Disponible en: https://www.androidsis.com/el-ciclo-de-vida-de-una-aplicacion-de-android/. [Último acceso: 15 Enero 2023].

[11] Google Developers, «Android Studio,» 2023. [En línea]. Disponible en: https://developer.android.com/studio. [Último acceso: 12 Enero 2023].

[12] Java, «¿Cómo puedo obtener Java para dispositivos móviles?,» Oracle, 2022. [En línea]. Disponible en: https://www.java.com/es/download/help/java_mobile_es.html. [Último acceso: 20 Enero 2023].

[13] S. Bose, M. Mukherjee, A. Kundu, Banerjee y Madhurima, «A COMPARATIVE STUDY: JAVA VS KOTLIN PROGRAMMING IN ANDROID APPLICATION DEVELOPMENT,» Mayo 2018. [En línea]. Disponible en: https://pdfs.semanticscholar.org/c0ee/43434064520cdde7222318bf6c4d2db69177.pdf. [Último acceso: 23 Enero 2023].

[14] Google Developers, «ImageView,» 2023. [En línea]. Disponible en: https://developer.android.com/reference/android/widget/ImageView. [Último acceso: 23 Enero 2023].

[15] Glide, «About Glide,» 2023. [En línea]. Disponible en: https://bumptech.github.io/glide/. [Último acceso: 18 Enero 2023].

[16] Google Developers, «Descripción general de Volley,» 2022. [En línea]. Disponible en: https://developer.android.com/training/volley?hl=es-419. [Último acceso: 12 Enero 2023].

[17] Google Developers, «Cómo realizar una solicitud stándar,» 2023. [En línea]. Disponible en: https://developer.android.com/training/volley/request?hl=es-419#java. [Último acceso: 14 Enero 2023].

[18] H. Li, Y. Xu, F. Wu y Y. Changhong, «Research of “Stub” remote debugging technique,» 2009. [En línea]. Disponible en: https://ieeexplore.ieee.org/abstract/document/5228140. [Último acceso: 18 Enero 2023].

[19] «Stub de método,» [En línea]. Disponible en: https://hmong.es/wiki/Stub_(computer_science). [Último acceso: 16 Enero 2023].

[20] L. Castellano, «KANBAN. METODOLOGÍA PARA AUMENTAR LA EFICIENCIA DE LOS PROCESOS.,» Junio 2019. [En línea]. Disponible en: https://web.p.ebscohost.com/abstract?direct=true&profile=ehost&scope=site&authtype=cr

awler&jrnl=22544143&AN=135831578&h=fBOlgJvCASKNB%2fTQluiI7tR6LfQqvxh7pGAHemhIZA9dygvF9Iulf02C8dUe4KxMoSKwqJpdwZm5N8phoQ0IwA%3d%3d&crl=c&resultNs=AdminWebAuth&resultLocal=. [Último acceso: 21 Enero 2023].

[21] G. Rocha, «Cómo hacer un tablero Kanban en 5 pasos con un ejemplo de software,» 8 Nobiembre 2021. [En línea]. Disponible en: https://profile.es/blog/como-hacer-tablero-kanban-ejemplo/. [Último acceso: 26 Enero 2023].

[22] L. Carvajal, Metodología de la Investgación Científica. Curso general y aplicado, 28 ed., Santiago de Cali: U.S.C., 2006, p. 139.

[23] F. Redrován, N. Loja y K. Correa, «COMPARISON OF QUALITY METRICS FOR WEB APPLICATION DEVELOPMENT,» 09 Marzo 2018. [En línea]. Disponible en: https://dialnet.unirioja.es/servlet/articulo?codigo=6551746. [Último acceso: 12 Enero 2023].

[24] B. Delgado y M. Vargas, «Descubriendo la anatomía de una aplicación sobre Android,» 2012. [En línea]. Disponible en: https://doi.org/10.5377/nexo.v25i2.685. [Último acceso: 10 Enero 2023].

[25] A. Baz, I. Ferreira, M. Álvarez y R. García, «Dispositivos móviles,» [En línea]. Disponible en: http://isa.uniovi.es/docencia/SIGC/pdf/telefonia_movil.pdf.

[26] C. Rodriguez, «apliint,» Principales tecnologias frontend, 15 Marzo 2022. [En línea]. Disponible en: https://apliint.com/2022/03/15/10-principales-tecnologias-frontend-para-usar-en-2022/. [Último acceso: 19 Enero 2023].

5 ANEXOS

5.1 ANEXO I. TABLERO KANBAN

El *link* para visualizar el tablero Kanban en la aplicación Trello es el siguiente: https://trello.com/invite/b/faWc1X7b/ATTI2d310ec66569875b6832463d259d8d51820BAAF2/tesis-jenifer-jacome

5.2 ANEXO II. RESULTADOS ENCUESTA DE VALIDACIÓN

Los resultados otorgados desde la aplicación *GoogleForms* se especifican en la Tabla 5.1.

Tabla 5.1. Resultados de la encuesta de validación

Marca temporal	**2/15/2023 13:09:25**	**2/15/2023 13:10:35**	**2/15/2023 13:10:53**	**2/15/2023 13:11:08**
¿Inició sesión correctamente?	Si	Si	Si	Si
Si presentó algún problema en el inicio de sesión, descríbalo				
¿Pudo crear registrarse como un usuario nuevo?	Si	Si	Si	Si
Si presentó algún problema al registrarse como nuevo usuario, descríbalo				
¿Los botones del menú funcionan correctamente?	Si	Si	Si	Si
Si presentó algún problema en el menú de la aplicación, descríbalo				
¿Pudo ingresar correctamente las etiquetas a buscar?	Si	Si	Si	Si
Si presentó algún problema al ingresar las etiquetas a buscar, descríbalo				
Al momento de ingresar las etiquetas, ¿Se desplegaron imágenes?	Si	Si	Si	Si
Si presentó algún problema el despliegue de las imágenes, descríbalo				
Al buscar imágenes al pulsar "LIMPIAR", ¿se vaciaron las imágenes y etiquetas?	Si	Si	Si	Si
Si presentó algún problema al momento de presionar "Limpiar", descríbalo				
Al subir imágenes pudo seleccionar los archivos multimedia que se encontraban en su teléfono	Si	Si	Si	Si
Si presentó algún problema al momento de subir imágenes, descríbalo				
¿Pudo agregar correctamente las etiquetas a las imágenes seleccionadas	No	No	Si	Si
Si presentó algún problema al momento de asignar etiquetas a las imágenes, descríbalo	El teléfono del teclado no permite visualizar las etiquetas ingresadas	El teclado bloquea la parte donde se ingresan las etiquetas		
¿Pudo subir las imágenes al *stub*?	No	Si	Si	Si
Si presentó problemas al subir imágenes al *stub*, descríbalo	Al subir una fotografía la aplicación se colgó por unos segundos			

5.3 ANEXO III. CÓDIGO DESARROLLADO

Todo el proyecto desarrollado en Android estudio se encuentra en un CD adjunto.