**Proyecto de desarrollo de aplicaciones multiplataforma**

**TÍTULO DEL PROYECTO**

CICLO FORMATIVO DE GRADO SUPERIOR

**Desarrollo de Aplicaciones Multiplataforma (IFCS02)**

**Curso 2022-23**

Autor/a/es:

**Carlos Moreno Luengos**

Tutor/a:

**Nombre y Apellidos**

Departamento de Informática y Comunicaciones

**I.E.S. Luis Vives**

índice

[2 INTRODUCCIÓN 3](#_Toc115806331)

[2.1 OBJETIVO 3](#_Toc115806332)

[2.2 ALCANCE 3](#_Toc115806333)

[2.3 JUSTIFICACIÓN 3](#_Toc115806334)

[3 IMPLEMENTACIÓN 3](#_Toc115806335)

[3.1 ANÁLISIS DE LA APLICACIÓN 4](#_Toc115806336)

[3.2 DISEÑO 6](#_Toc115806337)

[3.3 IMPLEMENTACIÓN 9](#_Toc115806338)

[3.4 IMPLANTACIÓN 9](#_Toc115806339)

[3.5 DOCUMENTACIÓN 9](#_Toc115806340)

[4 RESULTADOS Y DISCUSIÓN 9](#_Toc115806341)

[5 TRABAJO FUTURO (Opcional) 9](#_Toc115806342)

[6 CONCLUSIONES 10](#_Toc115806343)

[7 BIBLIOGRAFÍA 10](#_Toc115806344)

[ANEXOS 10](#_Toc115806345)

[I. ANEXO 1 10](#_Toc115806346)

[II. ANEXO 2… 10](#_Toc115806347)

# INTRODUCCIÓN

Se elaborará una red social que estará enfocada a la fotografía y al video, para que personas profesionales de este ámbito puedan mostrar y compartir sus proyectos.

## OBJETIVO

Estoy interesado en crear aplicaciones multiplataforma (Dispositivos domésticos y móviles) en la parte de la interfaz y en la parte servidor se realizará una API Rest para almacenar la información y poder consultarla. La parte servidor se elaborará con Spring Boot y la parte cliente se desarrollará con el lenguaje Dart, con Flutter.

## ALCANCE

Respecto a gestión de usuarios, se permitirá que un usuario pueda registrarse e iniciar sesión, haciendo especial hincapié en una correcta gestión de contraseñas. Se permitirá la subida de ficheros multimedia y aplicar seguridad en la aplicación.

Como posibles mejoras el usuario podrá comunicarse vía mensaje con otros usuarios y la creación de un entorno virtualizado para que la comunicación entre la parte cliente y servidor no se quede en la LAN, si no que la comunicación sea vía WAN.

## JUSTIFICACIÓN

Se elaborará una red social que estará enfocada a la fotografía y al video, para que personas profesionales de este ámbito puedan mostrar y compartir sus proyectos.

La mayor parte de las redes sociales están orientadas a aspectos lúdicos, esta aplicación se enfoca a profesionales del mundo audiovisual con el fin de mostrar la evolución de sus trabajos.

# IMPLEMENTACIÓN

## ANÁLISIS DE LA APLICACIÓN

### SERVIDOR

#### JAVA

Es el lenguaje que se va a utilizar para poder llevar a cabo el BackEnd. Java es un lenguaje de programación orientado a objetos. Casi todo exceptuando en algún caso, es un objeto.

Está inspirado en el lenguaje de programación C y C++.

#### SPRING BOOT

Spring Boot será el framework elegido. Spring es todo un ecosistema de proyectos que nos dan soporte para el desarrollo de aplicaciones empresariales robustas.

### CLIENTE

#### DART

Es el lenguaje de programación que se utilizará para realizar la parte FrontEnd. Dart es un lenguaje de programación de código abierto, desarrollado por Google. Dart nació para ofrecer una alternativa más moderna. Esta orientado a objetos y tiene una máquina virtual como Java.

#### FLUTTER

Es un SDK desarrollado por Google para poder crear aplicaciones móviles tanto para Android como para IOS, pero hoy en día también sirve para crear aplicaciones de escritorio en sistemas operativos como Linux y Windows. Compila de manera nativa, la creación de interfaces es muy flexible.

### HERRAMIENTAS

* POSTMAN: Cliente que nos ayuda al desarrollo de APIs REST.
* DRAWIO: Herramienta para realizar los diagramas.
* INTELIJ IDEA: Entorno de desarrollo para implementar el BackEnd
* VISUAL STUDIO: Entorno de desarrollo para implementar la parte FronEnd.
* ANDROID STUDIO: Entorno de desarrollo para implementar la parte FrontEnd.
* GIT: Sistema de control de versiones distribuido.
* GITHUB: Sistema para alojar el proyecto de forma distribuida.
* ADOBE XD: Software utilizado para crear el prototipo de la aplicación.
* WAKATIME: Plugin para controlar el tiempo de desarrollo del proyecto, te indica las horas programadas al día, diferenciando el lenguaje y los IDE que se hayan utilizado.
* DuckDNS: Servicio gratuito, encargado de dirigir las direcciones IP a subdominios del tipo duckdns.org

## DISEÑO

### DIAGRAMA DE CLASES UML

Diagrama

Descripción generada automáticamente

### ARQUITECTURA DEL SERVIDOR

### Diagrama Descripción generada automáticamente

### DIAGRAMA ENTIDAD RELACIÓN

### Interfaz de usuario gráfica, Texto Descripción generada automáticamente

### DIAGRAMA CASOS DE USO

### Diagrama, Esquemático Descripción generada automáticamente

### DISEÑO APLICACIÓN MÓVIL

Inicio de aplicación

Gráfico

Descripción generada automáticamente con confianza media

Inicio de sesión

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Registro de usuario

Tabla

Descripción generada automáticamente con confianza baja

Apartado Home

Interfaz de usuario gráfica

Descripción generada automáticamente

Perfil de usuario

Una captura de pantalla de un celular con texto e imágenes

Descripción generada automáticamente con confianza media

Buscar usuarios

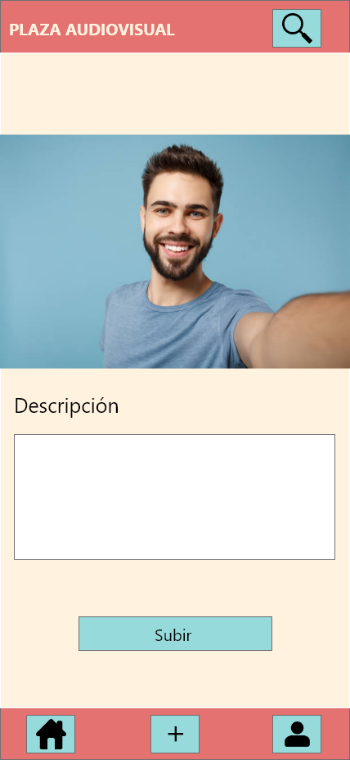


Añadir contenido

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Subir contenido



## IMPLEMENTACIÓN

### SERVIDOR

### Entry Points

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

### Paquetes servidor

**Config**

Este paquete contiene las clases y paquete relacionados sobre la configuración del servidor.

**Config > Security**

Paquete que contiene las clases y paquetes que están relacionadas con la seguridad del servidor.

**Config > Security > JWT**

Paquete que contiene las clases sobre la configuración JWT Token.

**Config > Security > Model**

Paquete que contiene las clases relacionadas con la respuesta del token del usuario y el logueo.

**Config > Security > Password**

Paquete que contiene la clase para poder codificar la contraseña del usuario.

**Controller**

Paquete que contiene las clases, con sus respectivos entry points para que el usuario pueda atacar a la API.

**DTO**

Paquete que contiene los objetos de transferencia de datos los cuales son los que llegarán al usuario.

**Exceptions**

Paquete que contiene las clases por si ocurre un error, personalizarlo para tener más información sobre el mismo.

**Mapper**

Paquete que contiene las clases que se encargan de transformar tipos de objetos a otros.

**Model**

Paquete que contiene las clases modelo, con sus respectivos atributos para poder obtener los datos de la base de datos.

**Repository**

Paquete que contiene los repositorios, se encargan de realizar consultas a la base de datos.

**Service**

Paquete que contiene los servicios, se comunican con los repositorios para realizar las consultas y contienen la lógica.

### CLIENTE

**my\_api**

Esta clase es utilizada para atacar al servidor, contiene todos los métodos para obtener la información necesaria.

Texto

Descripción generada automáticamenteMétodo para que un usuario se pueda registrar con sus datos correspondientes.

**user**

Clase modelo, contiene los atributos del usuario necesario para poder comunicarse correctamente con el servidor.

Texto

Descripción generada automáticamenteMétodo para poder emparejar los datos que se obtienen en formato json con el modelo.

**register\_page**

Vista para que el usuario se pueda registrar.

Texto

Descripción generada automáticamenteLlama al widget RegisterForm que es el que se encarga de que el usuario pueda insertar sus respectivos datos.

**auth**

Esta clase es implementada para obtener el token de un usuario una vez es logueado, se utiliza la dependencia flutter\_secure\_storage, para poder persistir la sesión, de esta manera si un usuario logueado cierra la aplicación y después la vuelve a abrir no hace falta que se vuelva a loguear, también se puede eliminar la sesión, por ejemplo si el usuario cierra la sesión, esta se elimina. También la sesión se eliminará y el usuario se debe de loguear de nuevo si la sesión expira, ya que tiene un tiempo de vida.

Texto

Descripción generada automáticamente

**dialogs**

Esta clase se encarga de mostrar un diálogo (información extra para que la visualice el usuario), por ejemplo, si un usuario se registra de manera satisfactoria se muestra el dialogo de que se ha registrado.

Texto

Descripción generada automáticamente

También contiene este método que sirve para dar al usuario una sensación de carga, por ejemplo cuando el usuario se loguea aparece una animación personalizada (Típico Progress Dialog).  
  
**extras**

Esta clase utiliza la dependencia image\_picker, sirve para que si el usuario quiere subir una imagen a su perfil o quiere actualizar su avatar, abre la galería del sistema y escoge la imagen que quiere.

Texto

Descripción generada automáticamente

**responsive**

Esta clase es utilizada para que la aplicación sea responsiva y que se vea de una manera bonita independientemente de la resolución del dispositivo.

Texto

Descripción generada automáticamente

**card\_user**

Componente para visualizar en el inicio las imágenes que suben los usuarios. Se comunica con **list\_user**

para tener la información necesaria de los usuarios.

**create\_media**

Componente para subir una imagen y ponerle una descripción.

**list\_user**

Componente para visualizar a todos los usuarios, con sus imágenes.

**login\_form**

Componente donde el usuario se puede loguear, si pulsa el botón ‘Sign In’ atacará al servidor y si los datos son correctos le llevará al inicio de la aplicación, si el proceso es erróneo, se mostrará un dialogo indicando que las credenciales no son correctas.

Texto

Descripción generada automáticamente

También en esta parte de la aplicación existe la posibilidad, si el usuario es nuevo, se pueda registrar pulsando al botón ‘Sign Up’ y le llevará a la vista de registro.

**logout\_button**

Componente para cerrar sesión y eliminar el token almacenado.

**modify\_form**

Componente para que el usuario pueda modificar su información, por ejemplo, nombre de usuario, teléfono, correo electrónico, etc

**register\_form**

Componente para que el usuario pueda registrarse introduciendo sus datos. Si por algún casual, un campo no está rellenado (campo obligatorio) o por ejemplo la confirmación de la contraseña no es la misma que la contraseña, el usuario no podrá registrarse de manera satisfactoria.

**search\_user\_delegate**

Componente utilizado para poder buscar un usuario por su nombre de usuario, filtra el nombre introducido atacando a la base de datos.

**user\_profile**

Componente para visualizar el perfil del usuario logueado, con su foto de perfil y su nombre de usuario, incluyendo una lista de todas las imágenes que ha subido.

## IMPLANTACIÓN

**SERVIDOR**

El servidor esta alojado en una máquina virtual de Ubuntu, la configuración de la red esta en modo NAT.

**Servicio DNS**

Para que el cliente pueda comunicarse con el servidor se ha configurado un servicio DNS, de esta manera no tenemos que conocer la IP pública y si cambiase en algún momento no habría problema.

Para este caso se ha utilizado Duck DNS.

Para instalarlo utilizamos los siguientes comandos:

* sudo apt-get install zenity cron curl
* chmod +x duck-setup-gui.sh

Y para levantar el servicio:

* ./duck-setup-gui.sh

Una vez instalado, se nos abrirá una ventana para introducir el nombre de dominio que queremos utilizar. En nuestro caso será carlos.moreno.duckdns.org.

El siguiente paso que tenemos que realizar es registrarnos en su página web y una vez registrados nos mandarán por correo un token que tendremos que introducir.

Para saber que el servicio esta corriendo podemos utilizar el comando: crontab -l

**Apertura de puertos**

Para que el cliente pueda comunicarse con el servidor desde un red WAN hay que exponer puertos.

Para abrir los puertos de nuestro router lo primero que tenemos que hacer es poner la dirección IP de nuestro Gateway, introducir usuario y contraseña

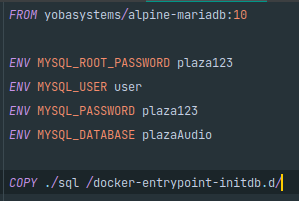
Gráfico

Descripción generada automáticamente con confianza media

**BASE DE DATOS**

Texto

Descripción generada automáticamenteFichero Docker-compose

Fichero Dockerfile

Texto, Aplicación

Descripción generada automáticamente

Fichero init-db.sql, script para tener datos dumi en la base de datosPara iniciar la base de datos hay que utilizar el siguiente comando: sudo docker-compose up -d

**CLIENTE**

Para generar el apk, utilizamos el siguiente comando

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Una vez acabe el proceso se nos generará el siguiente archivo

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## DOCUMENTACIÓN

La documentación del servidor ha sido exportada y parte de ella ha sido adjuntada en el Anexo 1 de este documento (JDOC – documentación Java). La documentación del cliente se adjunta en el Anexo 2.

# RESULTADOS Y DISCUSIÓN

El tiempo de planificación se ha cumplido según las expectativas.

No he tenido muchos problemas a la hora de desarrollar la aplicación ya que hay mucha documentación en internet sobre flutter.

# TRABAJO FUTURO (Opcional)

Para un futuro se quiere añadir la funcionalidad de me gustas, para que un usuario pueda dar me gustas a las Medias de otros usuarios y esos usuarios puedan ver los me gustas que tienen.

También se incluirá la posibilidad de que un usuario pueda tener seguidores y pueda visualizarlos.

También se tiene pensado incluir comentarios a las Medias para que los usuarios puedan dar sus aportaciones a la Media.

# CONCLUSIONES

Me ha gustado realizar este proyecto ya que he asentado conocimientos sobre Spring Boot, también he aprendido una nueva tecnología llamada Flutter, que ha sido bastante interesante de realizar y entretenida.

# BIBLIOGRAFÍA

<https://definicion.de/java/>

<https://es.wikipedia.org/wiki/Dart>

<https://aurestic.es/que-es-flutter/>

<https://stackoverflow.com/>

<https://dart.dev/>

# ANEXOS

## ANEXO 1

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

## ANEXO 2

// Para registrar a un usuario

register({required User user, required BuildContext context})

// Para loguear un usuario

login({required User user, required BuildContext context})

// Para obtener la información del usuario

getUserInfo({required username})

// Para borrar una Media

deleteMedia(BuildContext context, {required String id})

// Para modificar el usuario

updateUser({required User user, required BuildContext context})

// Para obtener las medias medias de los usuarios

getMediaUser()

// Para obtener el nombre de un usuario

getUsername()

// Para establecer el avatar del usuario

setImageAvatar(String path, BuildContext context)

// Para que el usuario pueda subir una Media

setMedia(Uint8List bytes, String path, String description, BuildContext context)