

I.E.S GRAN CAPITÁN

CICLO FORMATIVO DE GRADO SUPERIOR EN DESARROLLO DE
APLICACIONES MULTIPLATAFORMA

PROYECTO INTEGRADO

AnunciaYa



Departamento de Informática

[Manual Técnico](#)

Carlos Murillo Pérez

Manuel González Pérez

ÍNDICE DE CONTENIDOS

| | |
|---|-----------|
| 1 - Sobre éste proyecto | 4 |
| 1.1 - Control de versiones | 4 |
| 1.2 - Licencia de uso | 5 |
| 2 - Análisis del problema | 5 |
| 2.1 - Introducción al problema | 5 |
| 2.2 - Antecedentes | 5 |
| 2.3 - Objetivos | 5 |
| 2.4 - Requisitos | 6 |
| 2.4.1 - Funcionales | 6 |
| 2.4.2 - No funcionales | 7 |
| 2.5 - Recursos | 7 |
| 2.5.1 - Software | 7 |
| 2.5.2 - Hardware | 7 |
| 3 - Diseño de la solución software | 8 |
| 3.1 - Modelados | 8 |
| 3.1.1 - Casos de uso | 8 |
| 3.1.2 - [Interacción] | 9 |
| 3.1.3 - [Estado] | 9 |
| 3.1.4 - [Actividad] | 9 |
| 3.2 - Prototipado gráfico | 9 |
| 3.2.1 - Escritorio | 9 |
| 3.2.2 - Smartphones | 9 |
| 3.3 - Base de datos | 10 |
| 3.3.1 - Diseño Conceptual (ER) | 10 |
| 3.3.2 - Diseño lógico (tablas normalizadas) | 11 |
| 4 - Implementación | 12 |
| 4.1 - Codificación | 12 |
| 4.1.1 - [Usabilidad] | 12 |
| 4.1.2 - Backend | 12 |
| 4.1.2.1 Servidor PHP | 12 |
| usuario.php | 13 |
| categoria.php | 18 |
| pedido.php | 22 |
| anuncio.php | 24 |
| auth.php | 31 |
| conexion.php | 33 |
| municipios.xml | 34 |
| ConfigServer.php | 34 |
| index.php | 35 |
| 4.1.2.2 Cliente Android | 36 |
| InicioFragment | 38 |
| ListAdapter | 41 |

| | |
|--|-----|
| UserFragment | 45 |
| InfoAnuncio | 47 |
| AnuncioUsuario | 49 |
| DialogoCompra | 51 |
| ImageAdapter | 53 |
| Pedidos | 54 |
| ListAnuncios | 55 |
| ImageView | 56 |
| Add Fragment | 56 |
| Envios | 57 |
| Anuncio | 58 |
| BundleRecovery | 59 |
| EditarAnuncio | 60 |
| Login | 63 |
| ServerComunication | 64 |
| Usuario | 69 |
| Register | 71 |
| Metodos | 74 |
| AddAnuncio | 75 |
| MainActivity | 78 |
| 4.1.2.3 Cliente Administrador Escritorio | 78 |
| HelloApplication | 80 |
| Login | 80 |
| UserController | 81 |
| AnuncioController | 83 |
| CarrouselController | 86 |
| CategoriaController | 87 |
| InsertCategoController | 89 |
| InsertUserController | 90 |
| Pedido Controller | 91 |
| Home | 92 |
| LoginController | 95 |
| Pedido | 96 |
| Util | 97 |
| Anuncio | 98 |
| Categoria | 99 |
| User | 99 |
| DataStore | 100 |
| Server Communication | 100 |
| Métodos | 101 |
| 4.1.2.4 Sentencias SQL | 103 |
| Creación tabla usuario. | 103 |
| Creación tabla categorías. | 103 |
| Creación de tabla anuncios. | 104 |

| | |
|--|------------|
| Creación de la tabla pedidos. | 104 |
| 4.1.3 - Frontend | 105 |
| 4.1.3.1 Cliente Android | 105 |
| activity_login.xml | 106 |
| activity_register.xml | 107 |
| fragment_inicio.xml | 108 |
| fragment_user.xml | 109 |
| activity_pedidos.xml | 109 |
| activity_envios.xml | 110 |
| activity_add_anuncio.xml | 110 |
| dialog_comprar.xml | 111 |
| activity_anuncio.xml | 111 |
| activity_anuncio_usuario.xml | 112 |
| list_anuncios.xml | 112 |
| list_pedidos.xml | 113 |
| 4.1.3.2 Cliente Administrador Escritorio | 113 |
| login.fxml | 113 |
| home.fxml | 114 |
| graphics.fxml | 114 |
| usuarios.fxml | 115 |
| anuncios.fxml | 115 |
| categorias.fxml | 116 |
| pedidos.fxml | 116 |
| insert_usuario.fxml | 117 |
| insert_categoria.fxml | 117 |
| imagenes_anuncio.fxml | 118 |
| 4.2 - [Pruebas] | 119 |
| 5 - Documentación | 119 |
| 5.1 - Empaquetado / Distribución | 119 |
| 5.2 - Instalación | 120 |
| Instalación del Cliente Android | 120 |
| 5.3 - Manual de Usuario / Referencia | 122 |
| 6 - Conclusiones | 122 |
| 7 - Bibliografía | 122 |

1 - Sobre éste proyecto

1.1 - Control de versiones

Hemos optado por utilizar para el control de versiones la plataforma de Github, para llevar a cabo el seguimiento de todo el proyecto colaborativo. Es altamente recomendable contar con una plataforma de gestión de versiones para todos los proyectos. A medida que el proyecto crece en complejidad y tamaño, la capacidad de revertir a versiones anteriores se vuelve esencial, especialmente en casos donde la aplicación pueda presentar fallos críticos.

Github → https://github.com/ManuelGonzalez709/Final_Proyect

1.2 - Licencia de uso

La licencia de uso que tendrá nuestra aplicación será libre, siendo totalmente gratuita sin ningún tipo de coste, además de tener la posibilidad de modificar su código fuente y posterior distribución.

2 - Análisis del problema

2.1 - Introducción al problema

En respuesta a las dificultades de compra y venta de productos de segunda mano en Córdoba, hemos decidido desarrollar una aplicación móvil de compra y venta de productos de segunda mano especializada para los municipios de la provincia de Córdoba. Nuestro objetivo es brindar a los usuarios una herramienta confiable que les permita buscar productos de forma más precisa, considerando su ubicación geográfica. A través de esta iniciativa, buscamos satisfacer las necesidades de los usuarios, promover una cultura de consumo consciente y fortalecer la economía local. Creemos que esta aplicación mejorará significativamente la experiencia de compra y venta de productos de segunda mano en la provincia de Córdoba, al tiempo que enriquece el tejido social de la comunidad.

2.2 - Antecedentes

Nos hemos encontrado los siguientes antecedentes a la hora de realizar el desarrollo del proyecto:

- Inicialmente, comenzamos el desarrollo del servidor utilizando Spring Boot junto con el lenguaje de programación Java. Tras realizar diversas pruebas y evaluar los métodos implementados, decidimos migrar a un servidor Apache con PHP para el envío de datos estructurados en formato JSON, el cual ha demostrado ser altamente eficiente.
- Durante el desarrollo del servidor identificamos problemas con la base de datos, por lo que optamos por consolidar el diseño y la posterior traducción antes de continuar con el desarrollo del servidor PHP.
- No estaba habilitado la lectura de directorios mediante url en el servidor apache, por lo que tuvimos que activarlo creando el archivo de configuración .htaccess y añadir *Options +Indexes*.

2.3 - Objetivos

Nuestra aplicación, AnunciaYa, está principalmente enfocada en la plataforma Android. No obstante, también contará con un cliente multiplataforma para su gestión, permitiendo así la compra y venta de productos de segunda mano.

El objetivo principal de nuestro proyecto será el de conseguir implementar mediante herramientas de programación, una aplicación en Java utilizando el IDE IntelliJ, destinada a la gestión administrativa de la misma. Paralelamente, se empleará Android Studio para el desarrollo de la aplicación principal denominada AnunciaYa.

2.4 - Requisitos

2.4.1 - Funcionales

- Usuario comprador y vendedor (cliente móvil)
 - Realizar búsquedas de anuncios filtradas por categoría, ubicación y título.
 - Subir un artículo con fotos.
 - Modificar un artículo.
 - Eliminar un artículo.
 - Comprar artículos.
 - Ver registro artículos comprados.
 - Vender artículos.
 - Ver registro de los artículos vendidos.
 - Crear una cuenta para un nuevo usuario.
 - Modificar datos del usuario logeado.
 - Autologger.
- Usuario administrador (cliente multiplataforma)
 - Ver información detallada del servidor (uso de la app, accesos, peticiones ok).
 - Ver registros totales de los usuarios.
 - Crear nuevos usuarios.
 - Eliminar usuarios.
 - Ver registros totales de las categorías.
 - Crear nuevas categorías.
 - Modificar el nombre de las categorías.
 - Eliminar categorías.
 - Ver registros totales de los anuncios.
 - Ver fotos asociadas a un anuncio.
 - Eliminar anuncios.
 - Ver registros totales de los pedidos.
- Servidor Base de Datos MySQL
 - Gestionará la información de la aplicación con las siguientes tablas:
 - Usuario
 - Categoría
 - Anuncio
 - Pedido
- Servidor Apache PHP
 - Gestión de peticiones HTTP mediante JSON.
 - Procesamiento de solicitudes.
 - Generación de respuestas.
 - Envío de respuestas.
 - Creación de logs.
 - Guardado de contraseñas de usuario Cifradas

2.4.2 - No funcionales

- Rendimiento:
 - La aplicación debe ser capaz de manejar al menos 1000 solicitudes simultáneas.
 - Las búsquedas de anuncios deben completarse en menos de 2 segundos.
- Escalabilidad:
 - La aplicación debe ser escalable para soportar un aumento en el número de usuarios y transacciones sin pérdidas de rendimiento.
- Seguridad:
 - Los datos del usuario deben de ser guardados de manera segura.
- Disponibilidad:
 - La aplicación debe tener una disponibilidad del 99.9% durante el tiempo de operación.
- Usabilidad:
 - La interfaz de usuario debe ser intuitiva y fácil de usar para todos los tipos de usuarios.
 - Debe haber documentación y ayuda accesible para los usuarios.
- Mantenibilidad:
 - El código debe estar bien documentado y seguir las mejores prácticas de desarrollo para facilitar el mantenimiento y la actualización.
- Compatibilidad:
 - La aplicación móvil debe ser compatible con las versiones más recientes de Android.
 - La aplicación multiplataforma debe ser accesible desde gran parte de ordenadores

2.5 - Recursos

2.5.1 - Software

Para el desarrollo o el mantenimiento de la aplicación deberemos de tener:

- Java JDK 21.
- JavaFX.
- IntelliJ IDE (o variantes) para el Administrador.
- Postman (pruebas).
- Cliente BD (phpmyadmin o HeidiSQL).
- Android Studio para el cliente Android.
- Visual Studio Code (o variantes) para el servidor php.
- Github para control de versiones del código.

2.5.2 - Hardware

Para la ejecución de la App en el cliente Android o en Dispositivos de escritorio necesitaremos :

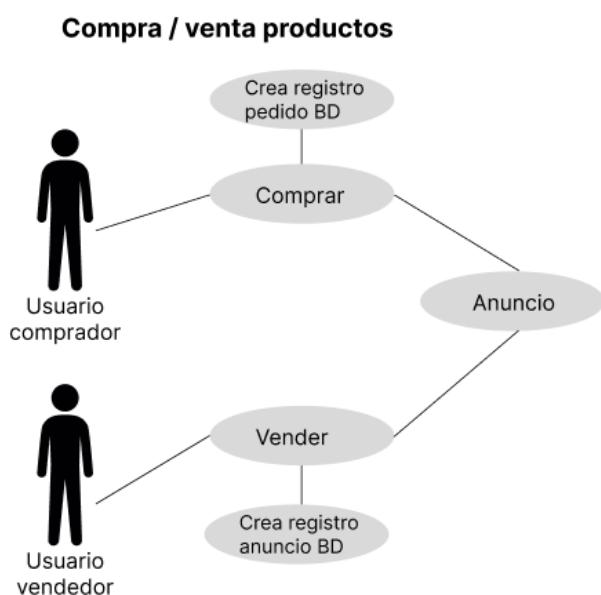
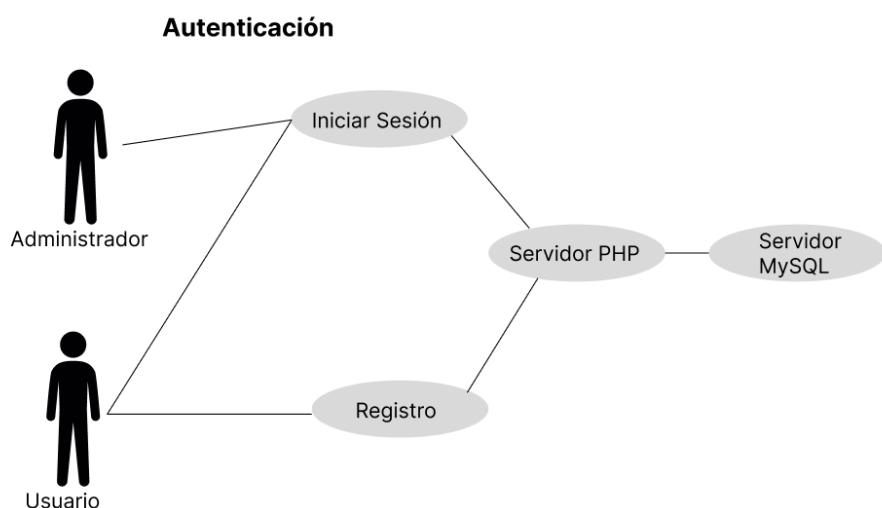
- Cliente Administrador:
 - **S.O:** Windows 7 o superiores / Linux 16:04 o superiores
 - **RAM:** 500 mb de RAM
 - **Procesador:** CPU de 1 GHz o superiores.
 - **Almacenamiento:** 30 MB de almacenamiento.
 - Conectividad a Internet

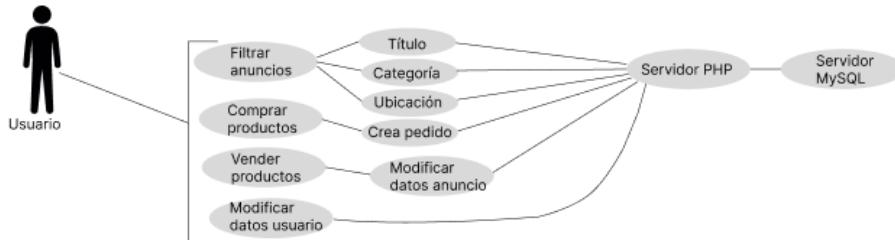
- Cliente Android:
 - **S.O:** Android 7.0 o superiores
 - **Procesador:** Qualcomm Snapdragon 660 o superiores
 - **RAM:** 1 GB de Ram
 - **Almacenamiento:** 50 MB de Almacenamiento
 - Conectividad a Internet

3 - Diseño de la solución software

3.1 - Modelados

3.1.1 - Casos de uso



Aplicación Administrador Escritorio**Aplicación Android****3.1.2 - [Interacción]****3.1.3 - [Estado]****3.1.4 - [Actividad]****3.2 - Prototipado gráfico****3.2.1 - Escritorio**

Para el prototipado de escritorio, elaboramos todo el diseño utilizando la aplicación web Figma, la cual resulta muy útil para la creación de prototipos.

Hemos llevado a cabo la elaboración de todos los diseños de las diversas pantallas, con el objetivo de obtener una visión clara de la apariencia final de la aplicación cliente bajo los permisos de administrador.

Todas las escenas excluyendo al Home son idénticas, ya que se incluirán tablas con todos los registros de la aplicación (usuarios, anuncios, categorías y pedidos). En el Home mostraremos unos gráficos circulares con la actividad de la aplicación.

[Link Diseño Figma](#)

3.2.2 - Smartphones

Para el prototipado móvil hemos hecho exactamente igual que con el de escritorio, simplemente cambiando el diseño, adaptado a vista móvil.

[Link Diseño Figma](#)

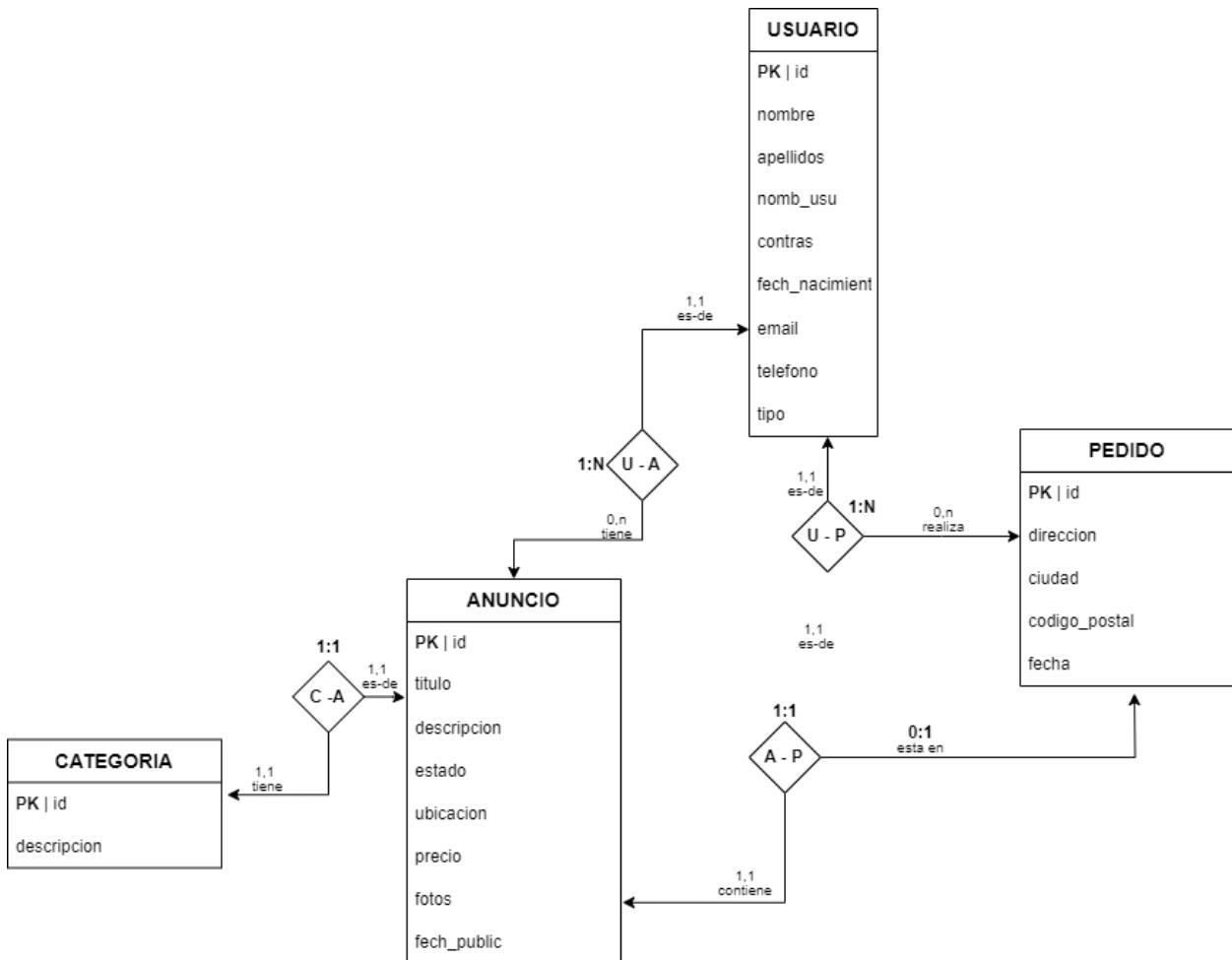
3.3 - Base de datos

Para la gestión de los datos en nuestra aplicación, hemos seleccionado MySQL v8.0.35 como nuestro Sistema Gestor de Base de Datos (SGBD) principal. Esta elección se basa en la amplia adopción y popularidad de MySQL en la comunidad de desarrollo de software, así como en su confiabilidad, rendimiento y escalabilidad comprobados en entornos de producción.

MySQL ofrece una amplia gama de características que se alinean estrechamente con los requisitos de nuestra aplicación, incluida la compatibilidad con múltiples plataformas, seguridad robusta y una sólida comunidad de desarrollo y soporte.

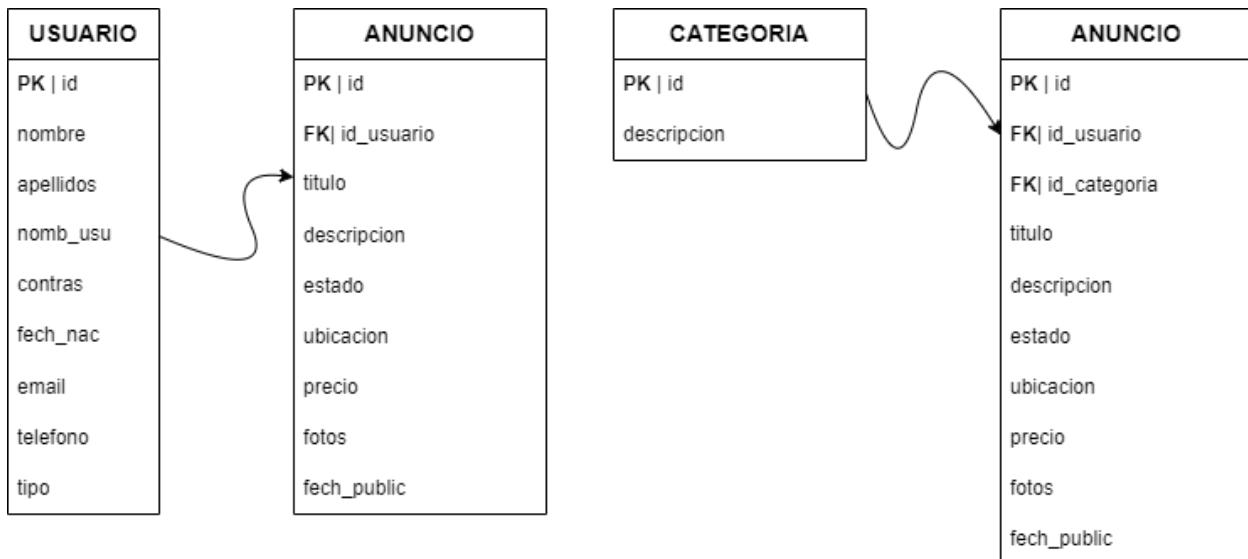
3.3.1 - Diseño Conceptual (ER)

En el siguiente modelo se definen las entidades, relaciones y cardinalidades del diseño entidad relación.

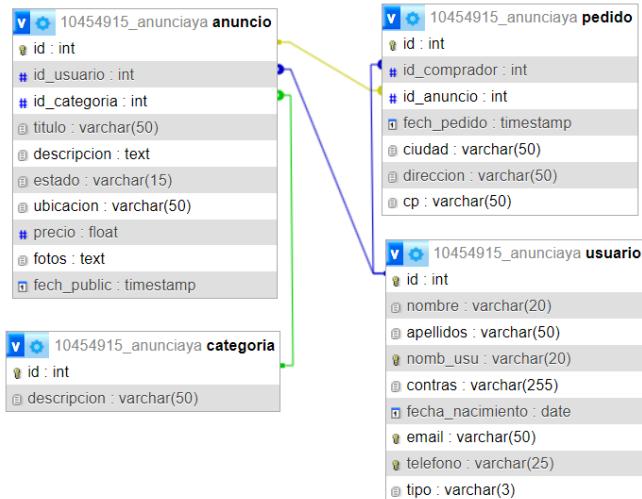


3.3.2 - Diseño lógico (tablas normalizadas)

Una vez realizado el modelo entidad relación deberemos de realizar la traducción a modelo relacional para a posteriori crear las sentencias SQL para importar el diseño directamente en MySQL.



La estructura de tablas y sus respectivas relaciones en nuestra base de datos MySQL quedaría de la siguiente manera.



4 - Implementación

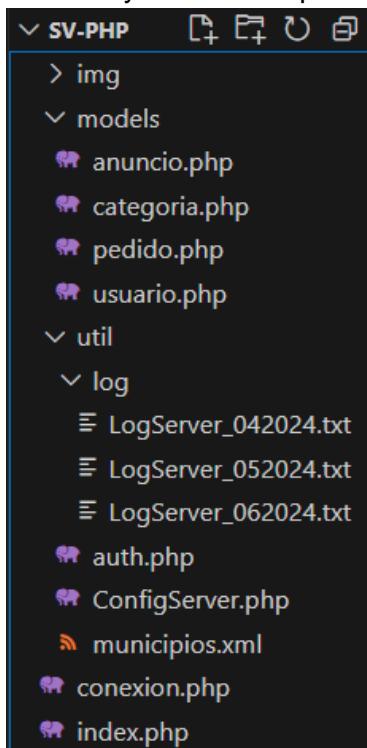
4.1 - Codificación

4.1.1 - [Usabilidad]

4.1.2 - Backend

4.1.2.1 Servidor PHP

El servidor PHP será el encargado de enviar y recibir peticiones a los clientes. La estructura de archivos y directorios que sigue el servidor php es la siguiente.



- El directorio **img** será donde se guarden las imágenes de los anuncios creados en la

aplicación android por los usuarios.

- En el directorio **models** encontramos los modelos .php que contendrán los métodos necesarios.
 - *anuncio.php* → representa el modelo anuncios de la base de datos.
 - *categoria.php* → representa el modelo categorías de la base de datos.
 - *pedido.php* → representa el modelo pedidos de la base de datos.
 - *usuario.php* → representa el modelo usuarios de la base de datos.
- En el directorio **util** encontramos:
 - Directorio **log** con todos los registros del servidor.
 - *auth.php* → contiene los métodos necesarios para la autenticación de los usuarios.
 - *ConfigServer.php* → contiene métodos que guarda el log del servidor.
 - *municipios.xml* → lista con todos los municipios de la provincia de Córdoba.
- *conexión.php* → crear la conexión con la base de datos.
- *index.php* → gestiona las peticiones de los clientes y devuelve el método solicitado por el cliente a partir de clase y método.

usuario.php

Esta clase php gestiona todos los métodos para la tabla usuario de la base de datos.

La función *getUsuarioDataId* obtiene todos los datos de un usuario a partir del *id_usuario* pasado como parámetro en la función.

```
public function getUsuarioDataId($id_usuario) {
    $sql = "SELECT * FROM usuario WHERE id = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener el primer usuario encontrado
    $usuario = $result->fetch_assoc();

    // Cerrar la conexión y devuelve el usuario
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($usuario);
}
```

La función *insertUsuario* devuelve true o false dependiendo de si la inserción del nuevo usuario

cliente android ha sido exitosa o no, a partir de los datos del usuario pasados como parámetro.

```
public function insertUsuario($nombre, $apellidos, $nombr_usu, $contras, $fecha_nacimiento, $email, $telefono){
    $contrasena_cifrada = Auth::hashContras($contras);

    $sql = "INSERT INTO usuario (nombre, apellidos, nombr_usu, contras, fecha_nacimiento, email, telefono)
VALUES (?, ?, ?, ?, ?, ?, ?)";

    $conexion = new Conexion();
    $conexion->conectar();

    $stmt = $conexion->obtenerConexion()->prepare($sql);

    $stmt->bind_param("sssssss", $nombre, $apellidos, $nombr_usu, $contrasena_cifrada,
$fecha_nacimiento, $email, $telefono);

    $stmt->execute();

    // Verifica si la inserción fue exitosa
    $insercion_exitosa = $stmt->affected_rows > 0;

    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se insertó correctamente, de lo contrario, retorna false
    return json_encode($insercion_exitosa);
}
```

La función *AdminInsertUsuario* devuelve true o false dependiendo de si la inserción del nuevo usuario cliente administrador multiplataforma ha sido exitosa o no, a partir de los datos del usuario pasados como parámetro.

```
public function AdminInsertUsuario($nombre, $apellidos, $nombr_usu, $contras, $fecha_nacimiento,
$email, $telefono,$tipo){

    $contrasena_cifrada = Auth::hashContras($contras);

    $sql = "INSERT INTO usuario (nombre, apellidos, nombr_usu, contras, fecha_nacimiento, email, telefono,tipo)
VALUES (?, ?, ?, ?, ?, ?, ?,?)";

    $conexion = new Conexion();
    $conexion->conectar();

    $stmt = $conexion->obtenerConexion()->prepare($sql);

    $stmt->bind_param("ssssssss", $nombre, $apellidos, $nombr_usu, $contrasena_cifrada, $fecha_nacimiento,
$email, $telefono,$tipo);

    $stmt->execute();

    // Verifica si la inserción fue exitosa
    $insercion_exitosa = $stmt->affected_rows > 0;

    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se insertó correctamente, de lo contrario, retorna false
    return json_encode($insercion_exitosa);
}
```

La función *updateUsuario* devuelve true o false dependiendo de si la actualización del usuario

cliente (android o multiplataforma) ha sido exitosa o no, a partir de los datos del usuario pasados como parámetro.

```
public function updateUsuario($id_usuario, $nombre, $apellidos, $telefono, $email, $fecha_nac, $nombr_usu){
    try{
        $sql = "UPDATE usuario SET nombre = ?, apellidos = ?, telefono = ?, email = ?,
        fecha_nacimiento = ?, nombr_usu = ? WHERE id = ?";
        $conexion = new Conexion();
        $conexion->conectar();

        $stmt = $conexion->obtenerConexion()->prepare($sql);

        $stmt->bind_param("sssssi", $nombre, $apellidos, $telefono, $email, $fecha_nac, $nombr_usu, $id_usuario);

        $stmt->execute();

        // Verifica si la actualización fue exitosa
        $filas_afectadas = $stmt->affected_rows;

        $stmt->close();
        $conexion->cerrarConexion();

        // Retorna true si se actualizó al menos una fila, de lo contrario, retorna false
        return json_encode($filas_afectadas > 0);
    } catch(mysqli_sql_exception $exception){
        echo $exception;

        // Definir las cadenas que identifican las violaciones de restricciones únicas
        $unique_nombr_usu = "Duplicate entry 'usuario777' for key 'UNIQUE_nombr_usu'";
        $unique_email = "Duplicate entry 'nuevo@correo.com' for key 'UNIQUE_email'";

        // Arreglo para almacenar los mensajes de error
        $error_message = "excepcion";

        // Si la excepción indica una violación de la restricción de nombre de usuario único, añadir al arreglo de
        if(strpos($exception->getMessage(), $unique_nombr_usu) !== false) {
            $error_message[0] = "unique_nombr_usu"; // Error constraint
        }

        // Si la excepción indica una violación de la restricción de correo electrónico único, añadir al arreglo de
        if(strpos($exception->getMessage(), $unique_email) !== false) {
            $error_message[1] = "unique_email"; // Error constraint
        }

        return json_encode(["error" => $error_message]);
    }
}
```

La función *updateContras* devuelve true o false dependiendo si la actualización de la contraseña del usuario pasado como parámetro ha sido exitosa o no a partir de la nueva contraseña pasada como parámetro.

```
public function updateContras($id_usuario, $nueva_contrasena) {
    // Hashear la nueva contraseña antes de actualizarla en la base de datos
    $contrasena_hasheada = Auth::hashContras($nueva_contrasena);

    $sql = "UPDATE usuario SET contras = ? WHERE id = ?";

    $conexion = new Conexion();
    $conexion->conectar();

    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Enlaza los parámetros con los valores proporcionados
    $stmt->bind_param("si", $contrasena_hasheada, $id_usuario);

    // Ejecuta la consulta
    $stmt->execute();

    // Verifica si la actualización fue exitosa
    $filas_afectadas = $stmt->affected_rows;

    // Cierra la conexión y el statement
    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se actualizó al menos una fila, de lo contrario, retorna false
    return json_encode($filas_afectadas > 0);
}
```

La función *getIdUserByEmail* devuelve el id del usuario dependiendo el email que le hayamos

pasado como parámetro, en caso de que no exista devuelve null.

```
public function getIdUserByEmail($correo){
    $sql = "SELECT id FROM usuario WHERE email = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("s", $correo);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener el primer usuario encontrado
    $usuario = $result->fetch_assoc();

    // Cerrar la conexión y devuelve el usuario
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($usuario);
}
```

La función *getIdUser* devuelve el id del usuario dependiendo el nombre usuario que le hayamos pasado como parámetro, en caso de que no exista devuelve null.

```
public function getIdUser($nombre_usuario){
    $sql = "SELECT id FROM usuario WHERE nombr_usu = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("s", $nombre_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener el primer usuario encontrado
    $usuario = $result->fetch_assoc();

    // Cerrar la conexión y devuelve el usuario
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($usuario);
}
```

La función *getUsers* devuelve todos los usuarios registrados distinto del id usuario pasado como parámetro.

```

public function getUsers($idU){
    $sql = "SELECT * FROM usuario WHERE id <> ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    //pasamos el dato
    $stmt->bind_param("s", $idU);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los usuarios
    $usuarios = array();

    // Verificar si se encontraron usuarios
    if ($result->num_rows > 0) {
        // Obtener todos los usuarios encontrados
        while ($row = $result->fetch_assoc()) {
            $usuarios[] = $row;
        }
    } else {
        // No se encontraron usuarios, devolver null
        $usuarios = null;
    }

    // Cerrar la conexión y devolver los usuarios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($usuarios);
}

```

La función *borrarUsuario* devuelve true o false dependiendo si el borrado del id usuario pasado como parámetro ha sido exitosa o no.

```

public function borrarUsuario($idUsuario){
    $sql = "DELETE FROM usuario WHERE id = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $idUsuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Verifica si la inserción fue exitosa
    $insercion_exitosa = $stmt->affected_rows > 0;

    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se insertó correctamente, de lo contrario, retorna false
    return json_encode($insercion_exitosa);
}

```

categoría.php

Esta clase php gestiona todos los métodos para la tabla categoría de la base de datos.

La función *getCategoriaDescripcion* devuelve la descripción de una categoría a partir del id

categoría pasado como parámetro.

```
public function getCategoriaDescripcion($id_categoria){
    $sql = "SELECT descripcion FROM categoria WHERE id = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_categoria);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener la descripción de la categoría
    $descripcion = $result->fetch_assoc();

    // Cerrar la conexión
    $stmt->close();
    $conexion->cerrarConexion();

    // Verificar si se encontró la categoría y devolver la descripción o null
    if ($descripcion) {
        return json_encode($descripcion);
    } else {
        return json_encode(array("success" => false, "data" => "null"));
    }
}
```

La función `getCategoriaId` devuelve el id categoría de una categoría a partir de la descripción de una categoría pasada como parámetro.

```
public function getCategoriaId($descripcion_categoria){
    $sql = "SELECT id FROM categoria WHERE descripcion = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("s", $descripcion_categoria);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener el ID de la primera categoría encontrada
    $categoria = $result->fetch_assoc();

    // Cerrar la conexión
    $stmt->close();
    $conexion->cerrarConexion();

    // Convertir el array asociativo a formato JSON y devolverlo
    return json_encode($categoria);
}
```

La función `getCategorias` devuelve todas las categorías registradas.

```

public function getcategorias(){
    $sql = "SELECT * FROM categoria";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar las categorías
    $categorias = array();

    // Obtener todas las categorías encontradas
    while ($row = $result->fetch_assoc()) {
        $categorias[] = $row;
    }

    // Cerrar la conexión y devolver las categorías como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($categorias);
}

```

La función *insertCategoria* devuelve true o false dependiendo de si la inserción de la nueva categoría ha sido exitosa o no, a partir de los datos de una descripción pasada como parámetro.

```

public function insertCategoria($descripcion){
    $sql = "INSERT INTO categoria (descripcion) VALUES (?)";

    $conexion = new Conexion();
    $conexion->conectar();

    $stmt = $conexion->obtenerConexion()->prepare($sql);

    $stmt->bind_param("s", $descripcion);

    $stmt->execute();

    // Verifica si la inserción fue exitosa
    $insercion_exitosa = $stmt->affected_rows > 0;

    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se insertó correctamente, de lo contrario, retorna false
    return json_encode($insercion_exitosa);
}

```

La función *tieneAnunciosCategoria* devuelve true o false dependiendo de existen anuncios con el *id_categoria* pasado como parámetro.

```
public function tieneAnunciosCategoria($id_categoria){  
    $sql = "SELECT COUNT(*) AS count FROM anuncio WHERE id_categoria = ?";  
  
    // Crear una instancia de la clase Conexion  
    $conexion = new Conexion();  
    $conexion->conectar();  
  
    // Preparar la consulta  
    $stmt = $conexion->obtenerConexion()->prepare($sql);  
  
    // Enlazar el parámetro  
    $stmt->bind_param("i", $id_categoria);  
  
    // Ejecutar la consulta  
    $stmt->execute();  
  
    // Obtener el resultado  
    $result = $stmt->get_result();  
  
    // Obtener la fila del resultado  
    $row = $result->fetch_assoc();  
  
    // Verificar si hay más de 0 anuncios con el id_categoria  
    $hay_anuncios = $row['count'] > 0;  
  
    // Cerrar el statement y la conexión  
    $stmt->close();  
    $conexion->cerrarConexion();  
  
    // Retornar true si hay más de 0 anuncios con el id_categoria, y false si no hay ninguno  
    return json_encode($hay_anuncios);  
}
```

La función *updateCategoria* devuelve true o false dependiendo de si se ha actualizado una categoría a partir del id_categoria y la nueva descripción pasada como parámetro.

```
public function updateCategoria($id_categoria, $descripcion){  
    $sql = "UPDATE categoria SET descripcion = ? WHERE id = ?";  
  
    $conexion = new Conexion();  
    $conexion->conectar();  
  
    $stmt = $conexion->obtenerConexion()->prepare($sql);  
  
    // Cambiar "is" a "si" para vincular el primer parámetro como string y el segundo como entero  
    $stmt->bind_param("si", $descripcion, $id_categoria);  
  
    $stmt->execute();  
  
    // Verifica si la actualización fue exitosa  
    $filas_afectadas = $stmt->affected_rows;  
  
    $stmt->close();  
    $conexion->cerrarConexion();  
  
    // Retorna true si se actualizó al menos una fila, de lo contrario, retorna false  
    return json_encode($filas_afectadas > 0);  
}
```

La función *deleteCategoria* devuelve true o false dependiendo de si se ha eliminado una categoría a partir del id_categoria pasada como parámetro.

```
public function deleteCategoria($id_categoria) {  
    $sql = "DELETE FROM categoria WHERE id = ?";  
  
    $conexion = new Conexion();  
    $conexion->conectar();  
  
    $stmt = $conexion->obtenerConexion()->prepare($sql);  
  
    $stmt->bind_param("i", $id_categoria);  
  
    $stmt->execute();  
  
    // Verificar si la eliminación fue exitosa  
    $filas_afectadas = $stmt->affected_rows;  
  
    $stmt->close();  
    $conexion->cerrarConexion();  
  
    // Retorna true si se eliminó al menos una fila, de lo contrario, retorna false  
    return json_encode($filas_afectadas > 0);  
}
```

pedido.php

Esta clase php gestiona todos los métodos para la tabla pedido de la base de datos.

La función getPedidos devuelve todas los anuncios que un usuario ha comprado a partir del id_usuario pasado como parámetro.

```
public function getPedidos($id_usuario) {
    $sql = "SELECT titulo, precio, direccion, ciudad, fotos FROM anuncio
JOIN pedido ON pedido.id_anuncio = anuncio.id WHERE pedido.id_comprador = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}
```

La función *insertPedido* devuelve true o false dependiendo de si la inserción del nuevo pedido ha sido exitoso o no, a partir del id_comprador, id_anuncio, dirección, ciudad y cp pasados como parámetro.

```

public function insertPedido($id_comprador, $id_anuncio, $direccion, $ciudad, $cp){
    // Comprobar si ya existe el pedido en la bd
    $sql_check = "SELECT COUNT(*) AS total FROM pedido WHERE id_anuncio = ?";

    $conexion = new Conexion();
    $conexion->conectar();

    $stmt_check = $conexion->obtenerConexion()->prepare($sql_check);

    $stmt_check->bind_param("i", $id_anuncio);

    // Ejecutar la consulta
    $stmt_check->execute();

    // Obtener el resultado de la consulta
    $resultado = $stmt_check->get_result();
    $fila = $resultado->fetch_assoc();

    // Verificar si ya existe un pedido con los mismos id_comprador e id_anuncio
    if ($fila['total'] > 0) {
        // Cerrar la consulta
        $stmt_check->close();
        // Cerrar la conexión
        $conexion->cerrarConexion();
        // Retornar false indicando que no se insertó el pedido porque ya existe
        return json_encode(false);
    }

    // Si no existe, proceder con la inserción del nuevo pedido
    $sql_insert = "INSERT INTO pedido (id_comprador, id_anuncio, direccion, ciudad, cp) VALUES (?, ?, ?, ?, ?)";

    $stmt_insert = $conexion->obtenerConexion()->prepare($sql_insert);

    $stmt_insert->bind_param("iisss", $id_comprador, $id_anuncio, $direccion, $ciudad, $cp);

    $stmt_insert->execute();

    // Verificar si la inserción fue exitosa
    $insercion_exitosa = $stmt_insert->affected_rows > 0;

    $stmt_insert->close();

    $conexion->cerrarConexion();

    return json_encode($insercion_exitosa);
}

```

La función getAllPedidos devuelve todos los pedidos registrados en la base de datos, para visualizarlos desde el cliente administrador multiplataforma.

```

public function getAllPedidos() {
    $sql = "SELECT p.id, u.nomb_usu, p.id_anuncio, a.titulo, p.direccion, p.ciudad, p.cp
    FROM pedido p INNER JOIN anuncio a ON p.id_anuncio = a.id
    INNER JOIN usuario u ON p.id_comprador = u.id;";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

anuncio.php

Esta clase php gestiona todos los métodos para la tabla anuncio de la base de datos.

La función getAllAnuncios devuelve todas los anuncios registrados en la base de datos, para visualizarlos desde el cliente administrador multiplataforma.

```

public function getAllAnuncios(){
    $sql = "SELECT a.id, u.nomb_usu, titulo, c.descripcion as categoria, a.descripcion, estado, ubicacion, precio, fotos, fech_public
        FROM anuncio a
        INNER JOIN usuario u ON a.id_usuario = u.id
        INNER JOIN categoria c ON a.id_categoria = c.id";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

La función *getIdNewAnuncioUsuario* devuelve el último id del anuncio creado por el *id_usuario* pasado como parámetro.

```

public function getIdNuevoAnuncioUsuario($id_usuario) {
    $sql = "SELECT MAX(id) AS ultimo_id FROM anuncio WHERE id_usuario = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener el primer resultado como un array asociativo
    $anuncio = $result->fetch_assoc();

    // Cerrar la conexión y devuelve el resultado
    $stmt->close();
    $conexion->cerrarConexion();

    // Retornar el resultado como JSON
    return json_encode($anuncio);
}

```

La función *getAnuncioId* devuelve todos los datos de un anuncio a partir del id_anuncio pasado como parámetro.

```

public function getAnuncioId($id_anuncio){
    $sql = "SELECT * FROM anuncio WHERE id = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_anuncio);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Obtener el primer anuncio encontrado
    $anuncio = $result->fetch_assoc();

    // Cerrar la conexión y devuelve el anuncio
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncio);
}

```

La función *getAnunciosExcepIdUsuarioAndPedido* devuelve todos los anuncios siempre y cuando no sean anuncios creados por el id_usuario pasado como parámetro ni que los anuncios estén comprados (es decir registrados en la tabla pedido).

```

public function getAnunciosExcepIdUsuarioAndPedido($id_usuario) {
    $sql = "SELECT * FROM anuncio WHERE id_usuario != ? AND id NOT IN (SELECT id_anuncio FROM pedido)";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

La función *getAnunciosUbicacion* devuelve los datos básicos de los anuncios filtrados por ubicación a partir del *id_usuario* y la ubicación pasadas como parámetro.

```

public function getAnunciosUbicacion($ubicacion, $id_usuario){[
    $sql = "SELECT id, titulo, precio, ubicacion, fotos
           |   FROM anuncio WHERE id_usuario != ? AND ubicacion = ?";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    $stmt->bind_param("is",$id_usuario, $ubicacion);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();
    |

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
]

```

La función *getAnunciosUbicacionCategoria* devuelve los datos básicos de los anuncios filtrados por ubicación y categoría a partir del *id_usuario*, la ubicación y el *id_categoria* pasadas como parámetro.

```

public function getAnunciosUbicacionCategoria($ubicacion, $id_categoria, $id_usuario){
    $sql = "SELECT id, titulo, precio, ubicacion, fotos
           |   FROM anuncio WHERE id_usuario != ? AND ubicacion = ? AND id_categoria = ?";
    
    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();
    
    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);
    
    $stmt->bind_param("isi",$id_usuario, $ubicacion, $id_categoria);
    
    // Ejecutar la consulta
    $stmt->execute();
    
    // Obtener el resultado de la consulta
    $result = $stmt->get_result();
    
    // Crear un array para almacenar los anuncios
    $anuncios = array();
    
    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }
    
    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

La función `getAnunciosIdCategoria` devuelve los datos básicos de los anuncios filtrados por categoría a partir del `id_usuario` y el `id_categoria` pasadas como parámetro.

```

public function getAnunciosIdCategoria($id_categoria, $id_usuario){
    $sql = "SELECT id, titulo, precio, ubicacion, fotos
           |   FROM anuncio WHERE id_usuario != ? AND id_categoria = ?";
    
    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();
    
    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);
    
    $stmt->bind_param("ii",$id_usuario, $id_categoria);
    
    // Ejecutar la consulta
    $stmt->execute();
    
    // Obtener el resultado de la consulta
    $result = $stmt->get_result();
    
    // Crear un array para almacenar los anuncios
    $anuncios = array();
    
    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }
    
    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

La función `getEnvios` devuelve todos los datos de los anuncios que el usuario ha vendido a partir del `id_usuario`.

```

public function getEnvios($id_usuario) {
    $sql = "SELECT anuncio.* FROM usuario JOIN anuncio ON anuncio.id_usuario = usuario.id JOIN pedido
    ON pedido.id_anuncio = anuncio.id WHERE usuario.id = ?;";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Verificar si se encontraron anuncios
    if ($result->num_rows > 0) {
        // Obtener todos los anuncios encontrados
        while ($row = $result->fetch_assoc()) {
            $anuncios[] = $row;
        }
    } else {
        // No se encontraron anuncios, devolver null
        $anuncios = null;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

La función *getAnunciosIdUsuario* devuelve todos los datos de los anuncios que tiene un usuario pasado como parámetro.

```

public function getAnunciosIdUsuario($id_usuario) {
    $sql = "SELECT * FROM anuncio WHERE id_usuario = ? AND id NOT IN (SELECT id_anuncio FROM pedido)";

    // Obtener la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Preparar la sentencia
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Vincular los parámetros
    $stmt->bind_param("i", $id_usuario);

    // Ejecutar la consulta
    $stmt->execute();

    // Obtener el resultado de la consulta
    $result = $stmt->get_result();

    // Crear un array para almacenar los anuncios
    $anuncios = array();

    // Obtener todos los anuncios encontrados
    while ($row = $result->fetch_assoc()) {
        $anuncios[] = $row;
    }

    // Cerrar la conexión y devolver los anuncios como JSON
    $stmt->close();
    $conexion->cerrarConexion();
    return json_encode($anuncios);
}

```

La función *insertAnuncio* devuelve true o false dependiendo de si la inserción de un anuncio nuevo ha sido exitosa o no, a partir de los datos del anuncio pasados como parámetro.

```

public function insertAnuncio($id_usuario, $id_categoria, $titulo, $descripcion, $estado, $ubicacion, $precio, $fotos) {
    // Query SQL para insertar un nuevo anuncio en la base de datos
    $sql = "INSERT INTO anuncio (id_usuario, id_categoria, titulo, descripcion, estado, ubicacion, precio, fotos) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

    // Obtiene la conexión a la base de datos
    $conexion = new Conexion();
    $conexion->conectar();

    // Prepara la sentencia SQL
    $stmt = $conexion->obtenerConexion()->prepare($sql);

    // Enlaza los parámetros con los valores proporcionados
    $stmt->bind_param("iisssds", $id_usuario, $id_categoria, $titulo, $descripcion, $estado, $ubicacion, $precio, $fotos);

    // Ejecuta la consulta
    $stmt->execute();

    // Verifica si la inserción fue exitosa
    $insercion_exitosa = $stmt->affected_rows > 0;

    // Cierra la sentencia y la conexión
    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se insertó correctamente, de lo contrario, retorna false
    return json_encode($insercion_exitosa);
}

```

La función *subirFotoBaseDatos* devuelve true o false dependiendo de si la actualización del campo fotos en la base de datos añadiendo la nueva url de las imágenes asociadas al anuncio ha sido exitosa o no, a partir de la url de las imágenes y el id_anuncio pasados como parámetro.

```

public function subirFotoBaseDatos($Fotos, $idAnuncio){
    $sql = "UPDATE anuncio SET fotos = ? WHERE id = ?";

    $conexion = new Conexion();
    $conexion->conectar();

    $stmt = $conexion->obtenerConexion()->prepare($sql);

    $stmt->bind_param("si",$Fotos,$idAnuncio);

    $stmt->execute();

    $filas_afectadas = $stmt->affected_rows;

    $stmt->close();
    $conexion->cerrarConexion();

    // Retorna true si se actualizó al menos una fila, de lo contrario, retorna false
    return json_encode($filas_afectadas > 0);
}

```

La función *updateAnuncio* devuelve true o false dependiendo si la actualización del anuncio ha sido exitosa o no a partir de los datos del nuevo anuncio modificado.

```
public function updateAnuncio($id_anuncio, $titulo, $descripcion, $estado, $ubicacion,$precio, $fotos, $id_categoria){  
    $sql = "UPDATE anuncio SET titulo = ?, descripcion = ?, estado = ?, ubicacion = ?, precio = ?, fotos = ?, id_categoria = ? WHERE id = ?";  
  
    $conexion = new Conexion();  
    $conexion->conectar();  
  
    $stmt = $conexion->obtenerConexion()->prepare($sql);  
  
    $stmt->bind_param("ssssdsii", $titulo, $descripcion, $estado,$ubicacion, $precio, $fotos, $id_categoria, $id_anuncio);  
  
    $stmt->execute();  
  
    $filas_afectadas = $stmt->affected_rows;  
  
    $stmt->close();  
    $conexion->cerrarConexion();  
  
    // Retorna true si se actualizó al menos una fila, de lo contrario, retorna false  
    return json_encode($filas_afectadas > 0);  
}
```

La función *deleteAnuncio* devuelve true o false dependiendo si la eliminación del anuncio pasado como parámetro como *id_anuncio* ha sido exitosa o no.

```
public function deleteAnuncio($id_anuncio) {  
    $sql = "DELETE FROM anuncio WHERE id = ?";  
  
    $conexion = new Conexion();  
    $conexion->conectar();  
  
    $stmt = $conexion->obtenerConexion()->prepare($sql);  
  
    $stmt->bind_param("i", $id_anuncio);  
  
    $stmt->execute();  
  
    $filas_afectadas = $stmt->affected_rows;  
  
    $stmt->close();  
    $conexion->cerrarConexion();  
  
    return json_encode($filas_afectadas > 0);  
}
```

auth.php

Esta clase php gestiona todos los métodos para la autenticación de los usuarios clientes.

La función *verificarAuthClient* devuelve true o false dependiendo si la verificación de cuenta del cliente android ha sido exitosa o no, a partir del nombre de usuario y contraseña pasadas como parámetro.

```
public function verificarAuthClient($nomb_usuario, $contrasena) {
    $sql = "SELECT contras FROM usuario WHERE nomb_usu = ? AND tipo = 'STD' ";

    // Prepara la consulta
    $stmt = $this->conexion->obtenerConexion()->prepare($sql);

    // Vincula el parámetro
    $stmt->bind_param("s", $nomb_usuario);

    $stmt->execute();

    // Obtiene el resultado de la consulta
    $result = $stmt->get_result();

    // Verifica si se encontraron resultados
    if ($result->num_rows == 1) {
        // Obtiene los datos del usuario
        $row = $result->fetch_assoc();

        // Obtener la contraseña almacenada en la base de datos
        $contrasena_almacenada = $row['contras'];

        // Verificar si la contraseña coincide utilizando Bcrypt
        if (password_verify($contrasena, $contrasena_almacenada)) {
            // La contraseña coincide, la autenticación es exitosa
            $stmt->close();
            return true;
        } else {
            // La contraseña no coincide, la autenticación falló
            $stmt->close();
            return false;
        }
    } else {
        // No se encontró el usuario, la autenticación falló
        $stmt->close();
        return false;
    }
}
```

La función `verificarAuthAdmin` devuelve true o false dependiendo si la verificación de cuenta del cliente administrador multiplataforma ha sido exitosa o no, a partir del nombre de usuario y contraseña pasadas como parámetro.

```
public function verificarAuthAdmin($email, $contrasena) {
    $sql = "SELECT contras FROM usuario WHERE email = ? AND tipo = 'ADM' ";

    // Prepara la consulta
    $stmt = $this->conexion->obtenerConexion()->prepare($sql);

    // Vincula el parámetro
    $stmt->bind_param("s", $email);

    $stmt->execute();

    // Obtiene el resultado de la consulta
    $result = $stmt->get_result();

    // Verifica si se encontraron resultados
    if ($result->num_rows == 1) {
        // Obtiene los datos del usuario
        $row = $result->fetch_assoc();

        // Obtener la contraseña almacenada en la base de datos
        $contrasena_almacenada = $row['contras'];

        // Verificar si la contraseña coincide utilizando Bcrypt
        if (password_verify($contrasena, $contrasena_almacenada)) {
            // La contraseña coincide, la autenticación es exitosa
            $stmt->close();
            return true;
        } else {
            // La contraseña no coincide, la autenticación falló
            $stmt->close();
            return false;
        }
    } else {
        // No se encontró el usuario, la autenticación falló
        $stmt->close();
        return false;
    }
}
```

La función `hashContras` realiza un hash de la contraseña pasada como parámetro y la devuelve.

```
public static function hashContras($password) {
    // Utiliza password_hash para hashear la contraseña
    $hashedPassword = password_hash($password, PASSWORD_DEFAULT);
    return $hashedPassword;
}
```

conexion.php

Esta clase php gestiona todos los métodos para la conexión del servidor php con la base de datos.

La función conectar crea una conexión a la base de datos con las credenciales necesarias.

```
public function conectar() {
    $this->conexion = new mysqli($this->host, $this->usuario, $this->contrasena, $this->basedatos);

    // Verifica si hay errores en la conexión
    if ($this->conexion->connect_error) {
        die("Error de conexión: " . $this->conexion->connect_error);
    }

    // Codificación de caracteres en UTF-8
    $this->conexion->set_charset("utf8");
}
```

La función obtenerConexion devuelve un objeto conexión de la base de datos.

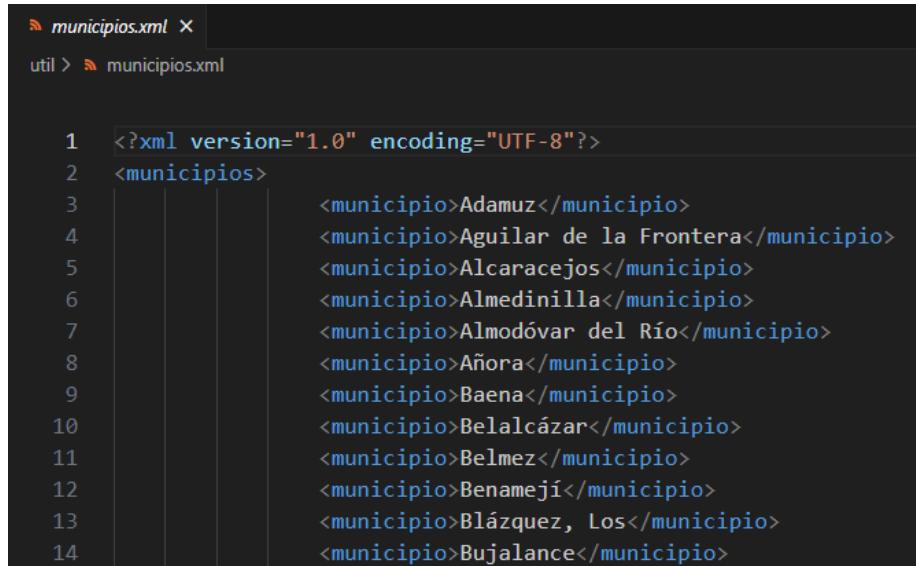
```
public function obtenerConexion() {
    return $this->conexion;
}
```

La función cerrarConexion cierra la conexión con la base de datos.

```
public function cerrarConexion() {
    if ($this->conexion) {
        $this->conexion->close();
    }
}
```

municipios.xml

Este archivo xml contiene todos los municipios de la provincia de Córdoba.



```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <municipios>
3     <municipio>Adamuz</municipio>
4     <municipio>Aguilar de la Frontera</municipio>
5     <municipio>Alcaracejos</municipio>
6     <municipio>Almedinilla</municipio>
7     <municipio>Almodóvar del Río</municipio>
8     <municipio>Añora</municipio>
9     <municipio>Baena</municipio>
10    <municipio>Belalcázar</municipio>
11    <municipio>Belmez</municipio>
12    <municipio>Benamejí</municipio>
13    <municipio>Blázquez, Los</municipio>
14    <municipio>Bujalance</municipio>

```

ConfigServer.php

Esta clase php gestiona los métodos necesarios para crear un log de las peticiones realizadas por el servidor.

La función `escribirLog` se encarga de escribir el log del servidor de todas las peticiones enviadas y recibidas para a posteriori ser usadas en la aplicación cliente administrador para ver la actividad del servidor.

```

class ConfigServer {
    public function escribirLog($MetodoEjecutado, $resultadoConsulta) {
        $nombreArchivo = "./util/log/LogServer_".date("mY").".txt";
        $nombreAlmacenDatosGraficos = "./util/log/Data.txt";
        $fecha_actual = date("d/m/Y::H:i:s");
        $ipPeticion = $_SERVER['REMOTE_ADDR'];
        // Contenido a escribir en el archivo
        if(($resultadoConsulta == "null" || $resultadoConsulta == "")&& $resultadoConsulta == FALSE)
            $contenido = $fecha_actual."|".$ipPeticion."|".$MetodoEjecutado."|FAILED";
        else $contenido = $fecha_actual."|".$ipPeticion."|".$MetodoEjecutado."|OK";

        // Intentar abrir el archivo en modo append (si no existe, se creará)
        if ($archivo = fopen($nombreArchivo, "a")) {
            // Escribir el contenido en el archivo
            if (fwrite($archivo, $contenido . PHP_EOL) === false) {
                echo "ServerError: No se pudo escribir en el archivo: " . $nombreArchivo;
            }
            // Cerrar el archivo
            fclose($archivo);
        } else {
            echo "ServerError: No se pudo abrir el archivo: " . $nombreArchivo;
        }
    }
}
?>

```

index.php

Esta clase php gestiona las peticiones de entrada de los clientes que siguen la siguiente estructura:

```
{
    "class": "",
    "method": "",
    "params": [
        ...
    ]
}
```

- class → clase a llamar.
- method → método a llamar.
- params → parámetros de las funciones.

```
<?php

include 'models/categoría.php';
include 'models/pedido.php';
include 'models/anuncio.php';
include 'models/usuario.php';
include 'util/ConfigServer.php';

$data = json_decode(file_get_contents('php://input'), true);

if (isset($data['class']) && isset($data['method']) && isset($data['params'])) {
    // Obtiene la clase y el método solicitado
    $class = $data['class'];
    $method = $data['method'];

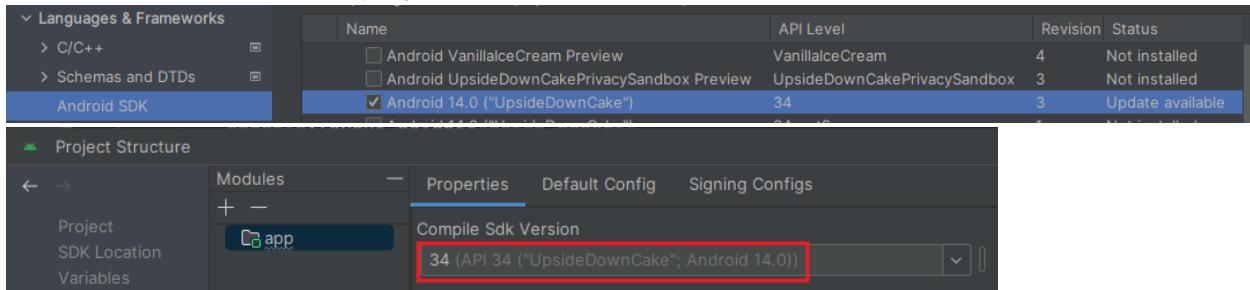
    // Verifica que clase se solicita
    switch ($class) {
        case 'Categoria':
            $obj = new Categoria();
            break;
        case 'Pedido':
            $obj = new Pedido();
            break;
        case 'Anuncio':
            $obj = new Anuncio();
            break;
        case 'Usuario':
            $obj = new Usuario();
            break;
        case 'Auth':
            $obj = new Auth();
            break;
        default:
            echo json_encode(['success' => false, 'error' => 'Clase no encontrada']);
            exit(); // Salir del script si la clase no es válida
    }
}
```

4.1.2.2 Cliente Android

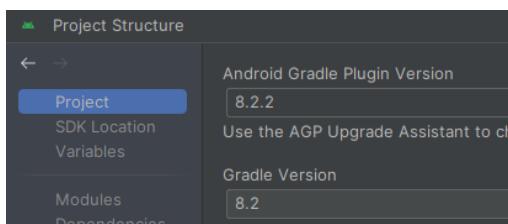
El cliente Android será la aplicación principal de nuestro proyecto, en la cual se podrán realizar las funciones anteriores citadas en resumen la compra venta de productos de segunda mano.

El desarrollo se ha realizado usando el IDE (Entorno de Desarrollo Integrado) Android Studio en su versión Hedgehog 2023.1.1 Patch 2.

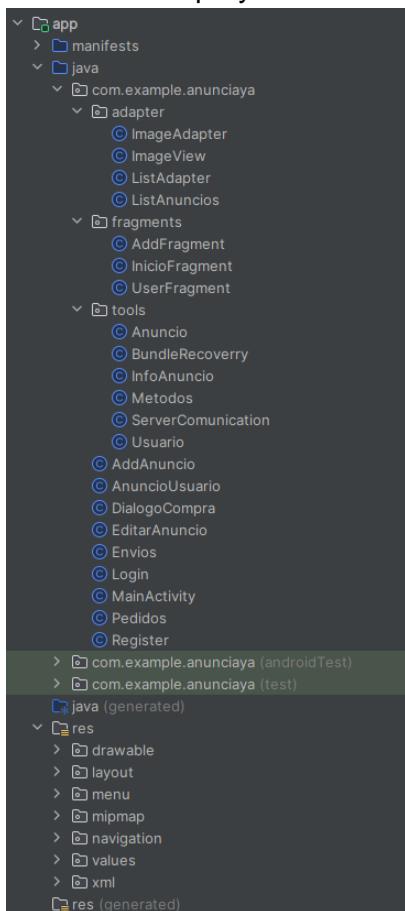
El SDK utilizado en todo el proyecto es el Android 14.0 con un nivel API 34.



Versión del Gradle 8.2



Estructura del proyecto.



Dependencias utilizadas a lo largo del proyecto.

```

dependencies {
    implementation("com.github.bumptech.glide:glide:4.16.0")
    implementation("commons-net:commons-net:3.8.0")
    annotationProcessor("com.github.bumptech.glide:compiler:4.12.0")
    implementation("com.github.denzcoskun:ImageSlideshow:0.1.2")
    implementation("com.github.dhaval2404:imagepicker:2.1")
    implementation("androidx.navigation:navigation-fragment:2.7.7")
    implementation("androidx.navigation:navigation-ui:2.7.7")
    implementation("com.fasterxml.jackson.core:jackson-databind:2.13.0")
    implementation("androidx.appcompat:appcompat:1.7.0")
    implementation("com.google.android.material:material:1.12.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
}

```

- Paquete **adapter**
 - ImageAdapter → adaptador creado para el carrousel de fotos.
 - ImageView → clase necesaria para el carrousel de fotos.
 - ListAdapter → clase que controla el recyclerview de los anuncios.
 - ListAnuncios → clase de tipo *ListAnuncios* para el recyclerview con las propiedades idAnuncio, título, precio, ubicación, dirección, ciudad y foto.
- Paquete **fragments** → contiene las clases que definen y gestionan los fragments.
 - AddFragment → fragment que añade un nuevo anuncio.
 - InicioFragment → fragment de inicio que contiene todos los anuncios para proceder con compra.
 - UserFragment → fragment que muestra información del usuario logueado.
- Paquete **tools**
 - Anuncio → clase que representa a los anuncios.
 - BundleRecovery → clase que guarda ids de inicio de sesión.
 - InfoAnuncio → clase que gestiona toda la información detallada de un anuncio.
 - Metodos → clase que contiene todos los métodos usados en el proyecto.
 - ServerComunication → clase que gestiona la comunicación del servidor php.
 - Usuario → clase que representa a los usuarios.
- Paquete principal (**aunciaya**)
 - AddAnuncio → clase que añade un nuevo anuncio.
 - AnuncioUsuario → clase que contiene toda la información detallada de un anuncio que pertenece a un usuario, el cual se puede eliminar o editar.
 - DialogoCompra → clase que controla la compra de un producto.
 - EditarAnuncio → clase que gestiona la edición de los datos de un anuncio.
 - Envios → clase que administra los productos vendidos a otros usuarios.
 - Login → clase que controla el inicio de sesión de los usuarios.
 - MainActivity → clase Principal encargada de verificar si la sesión ha sido iniciada y, en caso afirmativo, ejecutar el navigation.
 - Pedidos → clase que administra los productos comprados a otros usuarios.
 - Register → clase que controla el registro de nuevos usuarios.

InicioFragment

Esta clase java gestiona todos los métodos para que funcione correctamente todas las funcionalidades del inicio fragment. Muestra todos los anuncios registrados, además de poder

filtrar por categoría, ubicación y título.

Propiedades que conforman la clase `InicioFragment`.

```
private AutoCompleteTextView actvUbicacion;
private TextView tvNoAnuncios;
private SearchView searchView;
private Spinner spinner;
private View view;
private ListAdapter listAdapter;
private int idUsuario;
private String categoSelect;
private Metodos m;
```

El método sobrescrito `onCreateView` es el primer método que se ejecuta al llamar al `InicioFragment`.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    // Inflar el layout del fragmento
    view = inflater.inflate(R.layout.fragment_inicio, container, attachToRoot: false);
    initComponents();
    searchView.clearFocus();
    autocompletarUbicacion();
    tvNoAnuncios.setVisibility(View.GONE);
```

El siguiente evento se desencadena al ingresar texto en el `SearchView` de la pantalla de inicio. Al introducir texto, se invoca el método `filter` del `ListAdapter`, el cual se encarga de filtrar los anuncios según su título.

```
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    @Override
    public boolean onQueryTextSubmit(String query) { return false; }

    @Override
    public boolean onQueryTextChange(String newText) {
        String txtMinuscula = newText.toLowerCase(); // Convertir texto a minúsculas
        listAdapter.filter(txtMinuscula); // Aplicar el filtro con el texto en minúsculas
        return false;
    }
});
```

El siguiente evento se ejecuta al seleccionar un ítem en el Spinner de categorías. Dependiendo de la categoría seleccionada, se mostrarán los anuncios correspondientes a dicha categoría. Si la categoría seleccionada es None se mostrarán todos los anuncios sin filtrado de categoría..

```
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        categoSelect = (String) parent.getItemAtPosition(position);
        actvUbicacion.setText("");
        actvUbicacion.clearFocus();
        if (categoSelect.equals("None")) {
            listAdapter.updateData(m.getAnunciosInicio(new String[]{String.valueOf(idUsuario)}));
            updateNoAnuncios();
        } else {
            int idCategoria = m.getCategoriaId(new String[]{categoSelect});
            listAdapter.updateData(m.getAnunciosIdCategoria(new String[]{String.valueOf(idCategoria), String.valueOf(idUsuario)}));
            updateNoAnuncios();
        }
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {}
});
```

El siguiente evento se ejecutará cuando ingresemos en el componente *AutoCompleteTextView* algún municipio de la provincia de Córdoba, si tenemos alguna categoría seleccionada que sea diferente a None, filtrará por ubicación y categoría.

```
actvUbicacion.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        String textoSeleccionado = actvUbicacion.getText().toString().trim();

        // Si el texto está vacío, muestra los anuncios actuales
        if (textoSeleccionado.isEmpty()) {
            actualizarLista();
        }
        // Hay una categoría seleccionada para filtrarla también por ubicación
        String ubiSeleccionada = (String) parent.getItemAtPosition(position);
        if(!categoSelect.equals("None")){
            // Si se selecciona un elemento de la lista, filtrar los anuncios por el elemento seleccionado
            int idCategoria = m.getCategoriaId(new String[]{categoSelect});
            List<ListAnuncios> anuncios = m.getAnunciosUbicacionCategoria(new String[]{ubiSeleccionada,
                String.valueOf(idCategoria), String.valueOf(idUsuario)});

            // Verifica si hay anuncios para la ubicación seleccionada
            if (anuncios != null && !anuncios.isEmpty()) {
                listAdapter.updateData(anuncios);
            } else {listAdapter.clearData();}
            // Actualiza el TextView indicando si hay o no anuncios
        } else {
            // Si se selecciona un elemento de la lista, filtrar los anuncios por el elemento seleccionado
            List<ListAnuncios> anuncios = m.getAnunciosUbicacion(new String[]{ubiSeleccionada, String.valueOf(idUsuario)});

            // Verifica si hay anuncios para la ubicación seleccionada
            if (anuncios != null && !anuncios.isEmpty()) {
                listAdapter.updateData(anuncios);
            } else {listAdapter.clearData();}
            // Actualiza el TextView indicando si hay o no anuncios
        }
        updateNoAnuncios();
    }
});
```

El evento siguiente se ejecutará cuando pierda del foco el componente *AutoCompleteTextView*, si el texto de los municipios está vacío mostrará todos los anuncios por defecto.

```
actvUbicacion.setOnFocusChangeListener(new View.OnFocusChangeListener() {
    @Override
    public void onFocusChange(View v, boolean hasFocus) {
        if (!hasFocus) {
            // Si el campo pierde el foco (es decir, el usuario sale del campo de texto)
            String texto = actvUbicacion.getText().toString().trim();
            if (texto.isEmpty()) {
                // Si la categoría es None (ninguna) se mostrarán todos los anuncios
                if(!categoSelect.equals("None")){
                    int idCategoria = m.getIdCategoria(new String[]{categoSelect});
                    listAdapter.updateData(m.getAnunciosIdCategoria(
                        new String[]{String.valueOf(idCategoria),
                        String.valueOf(idUsuario)}));
                    updateNoAnuncios();
                    // Si no se mostrarán los anuncios filtrados por la categoría seleccionada
                } else{
                    actualizarLista();
                    updateNoAnuncios();
                }
            }
        }
    }
});
```

El método *autocompletarUbicación* se encarga de asignar al componente *AutoCompleteTextView*, los municipios de la provincia de Córdoba guardados en un xml en el servidor php.

```
private void autocompletarUbicacion(){
    ServerCommunication communication = new ServerCommunication();
    try{
        String[] Municipios = communication.getMunicipios().split(regex: ";");
        ArrayAdapter<String> adapter = new ArrayAdapter<>(requireActivity().getApplicationContext(), R.layout.auto_municipios_rojo3, Municipios);
        adapter.setDropDownViewResource(R.layout.auto_municipios_rojo3);
        // android.R.layout.simple_spinner_dropdown_item
        actvUbicacion.setAdapter(adapter);
    }catch (Exception e){
        Log.i( tag: "Error_Ubicacion",e.toString());
    }
}
```

El método sobrescrito *onResume* se ejecutará cuando se vuelva al InicioFragment para que muestre la lista de anuncios actualizada por defecto.

```
@Override
public void onResume() {
    super.onResume();
    actualizarLista();
    spinner.setSelection(m.getCategorias().indexOf("None"));
    actvUbicacion.setText("");
    actvUbicacion.clearFocus();
}
```

El método *actualizarLista* actualiza el recyclerview con los anuncios obtenidos del servidor php.

```
private void actualizarLista() {
    List<ListAnuncios> nuevaLista = m.getAnunciosInicio(new String[]{Integer.toString(idUsuario)});
    listAdapter.updateData(nuevaLista);
}
```

El método *updateNoAnuncios* hace visible o no un TextView indicando que no hay anuncios actualmente.

```
private void updateNoAnuncios() {
    if (listAdapter.getItemCount() == 0) {
        tvNoAnuncios.setVisibility(View.VISIBLE);
    } else {
        tvNoAnuncios.setVisibility(View.GONE);
    }
}
```

Actualmente no hay anuncios disponibles...

El método *initComponents* inicializa todos los componentes del InicioFragment.

```
private void initComponents() {
    m = new Metodos(); // Inicializar clase Metodos

    actvUbicacion = view.findViewById(R.id.actvUbicacion);
    tvNoAnuncios = view.findViewById(R.id.tvNoAnunciosCategoria);
    searchView = view.findViewById(R.id.searchView);
    searchView.clearFocus();
    spinner = view.findViewById(R.id.spCategorias);
    RecyclerView recyclerView = view.findViewById(R.id.rvAnunciosHome);

    Context context = requireActivity().getApplicationContext();
    SharedPreferences sharedPreferences = context.getSharedPreferences("MisDatos", Context.MODE_PRIVATE);
    BundleRecovery almacen = new BundleRecovery(sharedPreferences);
    idUsuario = almacen.recuperarInt("loginId"); // idUsuario logeado

    listAdapter = new ListAdapter(m.getAnunciosInicio(new String[]{Integer.toString(idUsuario)}),
        requireContext(), fragment: this, R.layout.list_anuncios);
    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager(requireContext()));
    recyclerView.setAdapter(listAdapter);

    // Crear y configurar el adaptador
    ArrayAdapter<String> adapter = new ArrayAdapter<>(requireContext(), R.layout.spinner_categorias_rojo3, m.getCategorias());
    adapter.setDropDownViewResource(R.layout.spinner_categorias_rojo3);

    // Asignar el adaptador al Spinner
    spinner.setAdapter(adapter);

    spinner.setSelection(m.getCategorias().indexOf("None")); // Selecciona por defecto la categoria ninguna
}
```

ListAdapter

Esta clase java gestiona todos los métodos para que funcione correctamente el adapter del recyclerview con todos los anuncios en nuestra aplicación android.

Propiedades que conforman la clase InicioFragment.

```
private List<ListAnuncios> mDatos;
private final List<ListAnuncios> mDatosOriginal;
private final LayoutInflater mInflater; // Describe de donde viene el layout
private final Context context; // De que clase llamamos el adaptador
private Fragment fragment; // Fragment donde se ejecuta el recyclerview
private int layoutResourceId; // Id del layout a usar en el recycler view
```

El método *updateData* actualiza el recyclerview asignando los nuevos anuncios a partir de un

ArrayList de ListAnuncios pasado como parámetro.

```
public void updateData(List<ListAnuncios> newAnuncios) {
    if (newAnuncios != null) {
        // Limpiar la lista antes de agregar los nuevos anuncios
        this.mDatos.clear();
        // Agregar los nuevos anuncios a la lista existente
        this.mDatos.addAll(newAnuncios);
        // Notificar al adaptador que los datos han cambiado
        notifyDataSetChanged();
    }
}
```

El método *clearData* elimina todos los anuncios actuales del recyclerview.

```
public void clearData(){this.mDatos.clear();}
```

El método *filter* filtra los anuncios por su título, para a posteriori usarse en un evento en el componente SearchView.

```
public void filter(String strSearch){
    if(strSearch.length() == 0){
        mDatos.clear();
        mDatos.addAll(mDatosOriginal);
    } else{
        List<ListAnuncios> collect = mDatos.stream().filter(i -> i.getTitulo().toLowerCase().
            contains(strSearch)).collect(Collectors.toList());
        mDatos.clear();
        mDatos.addAll(collect);
    }
    notifyDataSetChanged();
}
```

El método sobrescrito *getItemCount* devuelve el tamaño de elementos de la lista de anuncios.

```
@Override
public int getItemCount() { return mDatos.size(); } // Tamaño de elementos de la lista
```

El método *onCreateViewHolder* tiene la función de crear nuevas instancias de ViewHolder, que representarán cada ítem en la lista.

```
@NonNull
@Override
public ListAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = mInflater.inflate(layoutResourceId, parent, attachToRoot: false); // Usar el nuevo layout
    return new ListAdapter.ViewHolder(view);
}
```

El método *onBindViewHolder* cumple dos funciones principales: enlaza los datos a las vistas del ViewHolder y configura la información correspondiente a la posición específica del ítem en la

lista. Además, establece un OnClickListener para manejar las interacciones del usuario con los ítems de la lista. Dependiendo de en qué fragment se ejecute se mandará una información del anuncio u otra mediante los intent.

```

@Override
public void onBindViewHolder(@NonNull ListAdapter.ViewHolder holder, int position) {
    ListAnuncios anuncio = mDatos.get(position);
    holder.bindData(anuncio);

    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Metodos m = new Metodos();
            int idAnuncio = anuncio.getIdAnuncio(); // Obtiene el id anuncio seleccionado
            // Fragmento actual ejecutado
            Fragment currentFragment = fragment;

            // Si se ejecuta el recycler view desde InicioFragment
            if (currentFragment instanceof InicioFragment) {
                Intent intent = new Intent(context, InfoAnuncio.class);

                intent.putExtra( name: "idAnuncio", String.valueOf(idAnuncio));
                Anuncio a = m.getAnuncioId(new String[]{String.valueOf(idAnuncio)});
                intent.putExtra( name: "a_fotos", a.getFotos());
                intent.putExtra( name: "a_titulo", a.getTitulo());
                intent.putExtra( name: "a_descripcion", a.getDescripcion());
                intent.putExtra( name: "a_precio", a.getPrecio());
                intent.putExtra( name: "a_ubicacion", a.getUbicacion());
                intent.putExtra( name: "a_descripCategoria",
                    m.getCategoriaDescripcion(new String[]{String.valueOf(a.getIdCategoria())}));
                intent.putExtra( name: "a_estado", a.getEstado());

                Usuario u = m.getUsuarioDataId(new String[]{String.valueOf(a.getIdUsuario())});
                intent.putExtra( name: "a_nombCompUsu", value: u.getNombre() + " " + u.getApellidos());

                context.startActivity(intent);
            }
            // Si se ejecuta el recycler view desde UserFragment
        } else if (currentFragment instanceof UserFragment) {
            Intent intent = new Intent(context, AnuncioUsuario.class);

            intent.putExtra( name: "idAnuncio", String.valueOf(idAnuncio));
            Anuncio a = m.getAnuncioId(new String[]{String.valueOf(idAnuncio)});
            intent.putExtra( name: "a_fotos", a.getFotos());
            intent.putExtra( name: "a_titulo", a.getTitulo());
            intent.putExtra( name: "a_descripcion", a.getDescripcion());
            intent.putExtra( name: "a_precio", a.getPrecio());
            intent.putExtra( name: "a_ubicacion", a.getUbicacion());
            intent.putExtra( name: "a_descripCategoria",
                m.getCategoriaDescripcion(new String[]{String.valueOf(a.getIdCategoria())}));
            intent.putExtra( name: "a_estado", a.getEstado());

            Usuario u = m.getUsuarioDataId(new String[]{String.valueOf(a.getIdUsuario())});
            intent.putExtra( name: "a_nombCompUsu", value: u.getNombre() + " " + u.getApellidos());

            context.startActivity(intent);
        }
    });
}

```

Propiedades y constructor de la clase ViewHolder, la cual asignará los componentes del recyclerview.

```
public class ViewHolder extends RecyclerView.ViewHolder {
    ImageSlider iconImagen;
    TextView tvTitulo;
    TextView tvPrecio;
    TextView tvUbicacion;
    TextView tvDireccion;
    TextView tvCiudad;

    ViewHolder(View itemView) {
        super(itemView);
        tvTitulo = itemView.findViewById(R.id.tvTitulo);
        tvPrecio = itemView.findViewById(R.id.tvPrecio);
        tvUbicacion = itemView.findViewById(R.id.tvUbicacion);
        iconImagen = itemView.findViewById(R.id.isFotosAnuncioPrev);
        tvDireccion = itemView.findViewById(R.id.tvDireccion);
        tvCiudad = itemView.findViewById(R.id.tvCiudad);
    }
}
```

El método *bindData* vincula los datos de un elemento de la lista con las vistas correspondientes en el ViewHolder. Se encarga de configurar las vistas según los datos del elemento y el layout utilizado en el RecyclerView.

```
void bindData(final ListAnuncios item) {
    ArrayList<SlideModel> imageList = new ArrayList<>();
    // Obtiene todas las fotos y con el separados ; usa la primera
    String[] fotos = item.getFoto().split( regex: ";" );

    tvTitulo.setText(item.getTitulo());
    tvPrecio.setText(item.getPrecio());

    if(layoutResourceId == R.layout.list_anuncios){tvUbicacion.setText(item.getUbicacion());}
    if(layoutResourceId == R.layout.list_pedidos){
        tvDireccion.setText(item.getDireccion());
        tvCiudad.setText(item.getCiudad());
    }

    if (fotos != null && fotos.length > 0) {
        for (String foto : fotos) {
            if (!foto.isEmpty()) {
                imageList.add(new SlideModel(foto, ScaleTypes.CENTER_CROP));
            }
        }
    }

    // Asignar la lista de imágenes al ImageSlider
    iconImagen.setImageList(imageList);
}
```

UserFragment

Esta clase java gestiona todos los métodos para que funcione correctamente todas las funcionalidades del usuario fragment.

Propiedades que conforman la clase UserFragment.

```
private ImageView logout;
private SharedPreferences sharedpreferences ;
private int idUsuario;
private View view;
private BundleRecovery almacen;
private Context context;
private ListAdapter listAdapter;
private RecyclerView recyclerView;
private Metodos m;
private TextView nombreUsu,correoUsu,telefonoUsu;
private Usuario usuario;
private CardView btAjustesUsuario, btPedidosUsuario, btEnviosUsuario;
```

El método sobrescrito `onCreateView` es el primer método que se ejecuta al llamar al `UserFragment`. En este método, se definen los eventos de los botones.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    view = inflater.inflate(R.layout.fragment_user, container, attachToRoot: false);
    context = requireActivity().getApplicationContext();
    sharedpreferences = context.getSharedPreferences( name: "MisDatos", Context.MODE_PRIVATE);
    almacen = new BundleRecovery(sharedpreferences);
    m = new Metodos();

    btAjustesUsuario = view.findViewById(R.id.btUserAjustes);
    btEnviosUsuario = view.findViewById(R.id.btUserEnvios);
    btPedidosUsuario = view.findViewById(R.id.btUserPedidos);
    logout = view.findViewById(R.id.btLogout);

    logout.setOnClickListener(view -> Logout());
    btAjustesUsuario.setOnClickListener(view -> LanzarAjustesUsuario());
    btPedidosUsuario.setOnClickListener(view -> LanzarMisPedidos());
    btEnviosUsuario.setOnClickListener(view -> LanzarMisEnvios());

    recyclerView = view.findViewById(R.id.rvAnunciosUsuario); // Declaramos cual es el recyclerView
    nombreUsu = view.findViewById(R.id.userNombreUsuario);
    telefonoUsu = view.findViewById(R.id.userTelefonoUsuario);
    correoUsu = view.findViewById(R.id.userCorreoUsuario);

    setAnunciosUser();
    CargarDatosUsuario();
    return view;
}
```

Los siguientes dos métodos inician una nueva activity.

```

private void LanzarMisEnvios(){
    Intent intent = new Intent(getActivity(), Envios.class);
    startActivity(intent);
}

private void LanzarMisPedidos(){
    Intent intent = new Intent(getActivity(), Pedidos.class);
    startActivity(intent);
}

```

El método *LanzarAjustesUsuario* inicia una nueva actividad y, además, envía valores desde la actividad actual a la nueva actividad.

```

private void LanzarAjustesUsuario(){
    Intent intent = new Intent(getActivity(), Register.class);
    intent.putExtra( name: "fromMainActivity", value: true);
    startActivity(intent);
}

```

El método *Logout* cierra la sesión actual del usuario y vuelve a la actividad de inicio de sesión (*LoginActivity*) para que el usuario pueda iniciar sesión nuevamente.

```

private void Logout(){
    almacen.guardarInt( clave: "logginId", valor: -1);
    Intent intent = new Intent(getActivity(), Login.class);
    startActivity(intent);
    getActivity().finish();
}

```

El método *setAnunciosUser* asigna los anuncios de un usuario para mostrarlos en un recyclerview.

```

private void setAnunciosUser(){
    idUsuario = almacen.recuperarInt( clave: "logginId");
    listAdapter = new ListAdapter(m.getAnunciosIdUsuario(new String[]{Integer.toString(idUsuario)}),
        requireContext(), fragment: this, R.layout.list_anuncios);
    recyclerView.setHasFixedSize(true);
    recyclerView.setLayoutManager(new LinearLayoutManager(requireContext()));
    recyclerView.setAdapter(listAdapter);
}

```

El método sobrescrito *onResume* se ejecuta al regresar al UserFragment, mostrando así la lista de anuncios del usuario y sus datos.

```

@Override
public void onResume() {
    super.onResume();
    setAnunciosUser();
    CargarDatosUsuario();
}

```

InfoAnuncio

Esta clase Java gestiona toda la información detallada de un anuncio, así como el proceso de su

compra.

Propiedades que conforman la clase InfoAnuncio.

```
private TextView tvTitulo;
private TextView tvDescripcion;
private TextView tvPrecio;
private TextView tvNombreCompletoUsuario;
private TextView tvEstado;
private TextView tvCategoria;
private TextView tvUbicacion;
private ImageSlider isFotosAnuncio;
private Button btComprar;
private String idAnuncio;
```

El método sobrescrito *onCreate* es el primer método que se ejecuta al llamar a la activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_anuncio);
    initComponents();
    addDatosAnuncio();
```

El siguiente evento se ejecutará al pulsar el botón de comprar. Se enviarán el título, estado, precio e ID del anuncio a otra actividad para mostrar un resumen de la compra. Se mostrará un diálogo con toda la información de compra.

```
btComprar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        DialogoCompra dialogo = new DialogoCompra();
        Bundle bundle = new Bundle();
        bundle.putString("ac_titulo", tvTitulo.getText().toString());
        bundle.putString("ac_estado", tvEstado.getText().toString());
        bundle.putString("ac_precio", tvPrecio.getText().toString());
        bundle.putString("ac_idAnuncio", String.valueOf(idAnuncio));
        dialogo.setArguments(bundle);
        dialogo.show(getSupportFragmentManager(), tag: "DialogoCompra");
    }
});
```

El método *initComponents* inicializa los componentes.

```
private void initComponents() {  
    tvTitulo = findViewById(R.id.tvTituloArt);  
    tvDescripcion = findViewById(R.id.tvDescripcionArt);  
    tvPrecio = findViewById(R.id.tvPrecioArt);  
    tvEstado = findViewById(R.id.tvEstadoArt);  
    tvCategoria = findViewById(R.id.tvCategoriaArt);  
    tvUbicacion = findViewById(R.id.tvUbicacionArt);  
    tvNombrCompletoUsuario = findViewById(R.id.tvNombrCompleto);  
    isFotosAnuncio = findViewById(R.id.isFotosAnuncio);  
    btComprar = findViewById(R.id.btComprar);  
}
```

El método *addDatosAnuncio* asigna los datos del anuncio y del propietario obtenidos a través de un Intent para mostrarlos.

```
private void addDatosAnuncio() {  
    ArrayList<SlideModel> imageList = new ArrayList<>();  
  
    Intent intent = getIntent();  
    idAnuncio = intent.getStringExtra( name: "idAnuncio");  
    tvTitulo.setText(intent.getStringExtra( name: "a_titulo"));  
    tvDescripcion.setText(intent.getStringExtra( name: "a_descripcion"));  
    tvPrecio.setText(intent.getStringExtra( name: "a_precio"));  
    tvEstado.setText(intent.getStringExtra( name: "a_estado"));  
    tvCategoria.setText(intent.getStringExtra( name: "a_descripCategoria"));  
    tvUbicacion.setText(intent.getStringExtra( name: "a_ubicacion"));  
    tvNombrCompletoUsuario.setText(intent.getStringExtra( name: "a_nombrCompUsu"));  
    String fotos = intent.getStringExtra( name: "a_fotos");  
  
    // Separar la ruta de la foto usando ";" como separador  
    String[] fotoSplit = fotos.split( regex: ";");  
  
    // Agregar cada foto individualmente al ArrayList  
    for (String rutaFoto : fotoSplit) {  
        if (!rutaFoto.isEmpty()) {  
            imageList.add(new SlideModel(rutaFoto, ScaleTypes.CENTER_CROP));  
        }  
    }  
    // Asignar la lista de imágenes al ImageSlider  
    isFotosAnuncio.setImageList(imageList);  
}
```

AnuncioUsuario

Esta clase Java gestiona toda la información detallada de un anuncio, así como su modificación o eliminación.

Propiedades que conforman la clase AnuncioUsuario.

```
private TextView tvTitulo;
private TextView tvDescripcion;
private TextView tvPrecio;
private TextView tvEstado;
private TextView tvCategoria;
private TextView tvUbicacion;
private ImageSlider isFotosAnuncio;
private Button btEditarAnuncio;
private Button btBorrarAnuncio;
private String idAnuncio;
private String fotos;
```

El método sobrescrito `onCreate` es el primer método que se ejecuta al llamar a la activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_anuncio_usuario);
    initComponents();
    addDatosAnuncio();
```

El siguiente evento se ejecuta al pulsar el botón de eliminar el anuncio actual. Se mostrará un diálogo preguntando al usuario si está seguro de que desea eliminar el anuncio. Si el usuario pulsa en aceptar, el anuncio se eliminará.

```
btBorrarAnuncio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Metodos m = new Metodos();
        AlertDialog.Builder builder = new AlertDialog.Builder(context: AnuncioUsuario.this);

        builder.setMessage("¿Estás seguro de que deseas eliminar este anuncio?")
            .setPositiveButton("Confirmar", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    if(m.deleteAnuncio(new String[]{idAnuncio}).equals("true")){
                        Toast.makeText(context: AnuncioUsuario.this, text: "El anuncio ha sido eliminado correctamente!", Toast.LENGTH_SHORT)
                            .show();
                        finish();
                    } else if (m.deleteAnuncio(new String[]{idAnuncio}).equals("\\"err_const\"")) {
                        Toast.makeText(context: AnuncioUsuario.this, text: "No se puede eliminar un anuncio que ha sido comprado", Toast.LENGTH_SHORT)
                            .show();
                    } else{
                        Toast.makeText(context: AnuncioUsuario.this, text: "Ha ocurrido un error al intentar eliminar el anuncio", Toast.LENGTH_SHORT)
                            .show();
                    }
                }
            })
            .setNegativeButton("Cancelar", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int id) {
                    dialog.dismiss();
                }
            });
        AlertDialog dialog = builder.create();
        dialog.show();
    }
});
```

El siguiente evento se ejecutará al pulsar el botón de editar el anuncio actual. Se llevará al usuario

a otra activity para editar los datos del anuncio. Los datos del anuncio se enviarán a la otra actividad mediante un Intent.

```
btEditarAnuncio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(packageContext: AnuncioUsuario.this, EditarAnuncio.class);
        intent.putExtra(name: "au_idAnuncio", idAnuncio);
        intent.putExtra(name: "au_titulo", tvTitulo.getText());
        intent.putExtra(name: "au_descripcion", tvDescripcion.getText());
        intent.putExtra(name: "au_precio", tvPrecio.getText());
        intent.putExtra(name: "au_estado", tvEstado.getText());
        intent.putExtra(name: "au_categoria", tvCategoria.getText());
        intent.putExtra(name: "au_ubicacion", tvUbicacion.getText());
        intent.putExtra(name: "au_fotos", fotos);
        startActivity(intent);
        finish();
    }
});
```

El método *initComponents* inicializa los componentes.

```
private void initComponents() {
    tvTitulo = findViewById(R.id.tvTituloArt);
    tvDescripcion = findViewById(R.id.tvDescripcionArt);
    tvPrecio = findViewById(R.id.tvPrecioArt);
    tvEstado = findViewById(R.id.tvEstadoArt);
    tvCategoria = findViewById(R.id.tvCategoriaArt);
    tvUbicacion = findViewById(R.id.tvUbicacionArt);
    isFotosAnuncio = findViewById(R.id.isFotosAnuncio);
    btEditarAnuncio = findViewById(R.id.btEditarAnuncio);
    btBorrarAnuncio = findViewById(R.id.btBorrarAnuncio);
}
```

El método `addDatosAnuncio` asigna los datos del anuncio y del propietario obtenidos a través de un Intent para mostrarlos.

```
private void addDatosAnuncio() {
    ArrayList<SlideModel> imageList = new ArrayList<>();

    Intent intent = getIntent();
    idAnuncio = intent.getStringExtra( name: "idAnuncio");
    tvTitulo.setText(intent.getStringExtra( name: "a_titulo"));
    tvDescripcion.setText(intent.getStringExtra( name: "a_descripcion"));
    tvPrecio.setText(intent.getStringExtra( name: "a_precio"));
    tvEstado.setText(intent.getStringExtra( name: "a_estado"));
    tvCategoria.setText(intent.getStringExtra( name: "a_descripCategoria"));
    tvUbicacion.setText(intent.getStringExtra( name: "a_ubicacion"));
    fotos = intent.getStringExtra( name: "a_fotos");

    // Separar la ruta de la foto usando ";" como separador
    String[] fotoSplit = fotos.split( regex: ";" );

    // Agregar cada foto individualmente al ArrayList
    for (String rutaFoto : fotoSplit) {
        if (!rutaFoto.isEmpty()) {
            imageList.add(new SlideModel(rutaFoto, ScaleTypes.CENTER_CROP));
        }
    }
    // Asignar la lista de imágenes al ImageSlider
    isFotosAnuncio.setImageList(imageList);
}
```

DialogoCompra

Esta clase Java gestiona la compra de un producto especificando la dirección de envío. Propiedades que conforman la clase DialogoCompra.

```
private EditText etDireccionCompra;
private AutoCompleteTextView actvCiudadCompra;
private EditText etCPCompra;
private TextView tvTituloCompra;
private TextView tvEstadoCompra;
private TextView tvPrecioCompra;
private Button btComprarComp;
private String idAnuncio;
```

El método sobrescrito `onCreateView` se ejecuta al iniciar el DialogFragment del DialogoCompra. En este método se añade la información del resumen de la compra del artículo, así como un evento que, al pulsar el botón de compra, ejecuta la inserción de un nuevo pedido.

```
@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.dialog_compra, container, attachToRoot: false);
    initComponents(view);
    addDatosCompra();
    autocompletarUbicacion();

    btComprarComp.setOnClickListener(v -> {
        if(!etDireccionCompra.getText().toString().isEmpty() && !actvCiudadCompra.getText().toString().isEmpty()
            && !etCPCompra.getText().toString().isEmpty()){
            // El cp no puede tener más de cinco nº
            if (etCPCompra.getText().Length() > 5){
                Toast.makeText(getContext(), text: "El código postal no puede tener más de cinco números", Toast.LENGTH_SHORT).show();
            } else{
                Metodos m = new Metodos();
                SharedPreferences sharedpreferences = requireContext().getSharedPreferences(name: "MisDatos", MODE_PRIVATE);
                BundleRecovery dataRecovery = new BundleRecovery(sharedpreferences);
                int idUsuario = dataRecovery.recuperarInt(clave: "loginId");

                if(m.insertPedido(new String[]{String.valueOf(idUsuario), idAnuncio, etDireccionCompra.getText().toString(),
                    actvCiudadCompra.getText().toString(), etCPCompra.getText().toString() }) ){
                    Toast.makeText(getContext(), text: "Artículo comprado exitosamente!", Toast.LENGTH_SHORT).show();
                    dismiss(); // Cerrar ventana diálogo
                    getActivity().finish(); // Cerrar activity y volver al fragment inicio
                } else{
                    Toast.makeText(getContext(), text: "Ha ocurrido un error al realizar la compra", Toast.LENGTH_SHORT).show();
                }
            }
        } else{
            Toast.makeText(getContext(), text: "Por favor, completa todos los campos obligatorios", Toast.LENGTH_SHORT).show();
        }
    });
    return view;
}
```

El método `initComponents` inicializa los componentes.

```
private void initComponents(View view){
    etDireccionCompra = view.findViewById(R.id.etDireccionCompra);
    actvCiudadCompra = view.findViewById(R.id.actvCiudadCompra);
    etCPCompra = view.findViewById(R.id.etCPCompra);
    tvTituloCompra = view.findViewById(R.id.tvTituloCompra);
    tvEstadoCompra = view.findViewById(R.id.tvEstadoCompra);
    tvPrecioCompra = view.findViewById(R.id.tvPrecioCompra);
    btComprarComp = view.findViewById(R.id.btComprarComp);
}
```

El método addDatosCompra asigna el resumen de los datos de compra del artículo, los cuales son obtenidos del activity anterior.

```
public void addDatosCompra(){
    Bundle args = getArguments();
    if (args != null) {
        String titulo = args.getString(key: "ac_titulo");
        String estado = args.getString(key: "ac_estado");
        String precio = args.getString(key: "ac_precio");
        idAnuncio = args.getString(key: "ac_idAnuncio");

        tvTituloCompra.setText(titulo);
        tvEstadoCompra.setText(estado);
        tvPrecioCompra.setText(precio);
    }
}
```

ImageAdapter

Esto es una clase que se encarga de Preparar el funcionamiento del Carrousel para el apartado de Añadir Anuncio

El método onBindViewHolder se encarga de preparar los Datos de la Clase ViewHolder, la clase viewHolder tiene un constructor que se encarga de preparar los datos para ser mostrados en el ViewHolder.

```
@Override
public void onBindViewHolder(@NonNull ViewHolder holder, @SuppressLint("RecyclerView") int position) {
    Glide.with(context).load(arrayList.get(position)).into(holder.imageView);
    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            | onItemClickListener.onClick(holder.imageView, arrayList.get(position));
        }
    });
}

@Override
public int getItemCount() { return arrayList.size(); }

4 usages
public static class ViewHolder extends RecyclerView.ViewHolder{
    3 usages
    ImageView imageView;
    1 usage
    public ViewHolder(@NonNull View itemView){
        super(itemView);
        imageView = itemView.findViewById(R.id.list_item_img);
    }
}
2 usages
public void setOnItemClickListener(OnItemClickListener onItemClickListener){
    this.onItemClickListener = onItemClickListener;
}
3 usages 1 implementation
public interface OnItemClickListener{
    1 usage 1 implementation
    void onClick(ImageView imageView, String url);
}
```

Pedidos

Esta clase Java gestiona los productos que un usuario ha comprado.

Propiedades que conforman la clase Pedidos.

```
private TextView tvNoAnunComprad;
private BundleRecoveryy almacen;
private int IdUser;
private ListAdapter listAdapter ;
private RecyclerView recyclerView;
```

El método sobrescrito `onCreate` es el primer método que se ejecuta al llamar a la activity.

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_pedidos);

    initComponents();
    cargarAnuncios();
}
```

El método `initComponents` inicializa los componentes.

```
private void initComponents(){
    recyclerView = findViewById(R.id.rvPedidos);
    tvNoAnunComprad = findViewById(R.id.tvNoAnunComprad);
    SharedPreferences sharedPreferences = getSharedPreferences( name: "MisDatos", MODE_PRIVATE);
    almacen = new BundleRecoveryy(sharedPreferences);
}
```

El método `cargarAnuncios` se encarga de asignar los datos de los anuncios al recyclerview.

```
private void cargarAnuncios(){
    Metodos m = new Metodos();
    IdUser = almacen.recuperarInt( clave: "logginId"); // idUsuario Logeado
    List<ListAnuncios> pedidos = m.getPedidos(new String[]{Integer.toString(IdUser)});
    if (pedidos != null) {
        listAdapter = new ListAdapter(pedidos, getApplicationContext(), fragment: null, R.layout.list_pedidos);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
        recyclerView.setAdapter(listAdapter);
        if (listAdapter.getItemCount() > 0) {
            tvNoAnunComprad.setVisibility(View.GONE);
        } else {
            tvNoAnunComprad.setVisibility(View.VISIBLE);
        }
    }
}
```

ListAnuncios

Esto es una clase que usa el Recyclerview para la creación del Recyclerview

Capturas de los set/get y el constructor.

```

public ListAnuncios(String titulo, String precio, String direccion, String ciudad, String foto) {
    this.titulo = titulo;
    this.precio = precio;
    this.direccion = direccion;
    this.ciudad = ciudad;
    this.foto = foto;
}
1 usage
public int getIdAnuncio() {return idAnuncio;}
public String getTitulo() {return titulo;}
1 usage
public String getPrecio() {return precio;}
1 usage
public String getUbicacion() {return ubicacion;}
1 usage
public String getDireccion() {return direccion;}
1 usage
public String getCiudad() {return ciudad;}
public String getFoto() {return foto;}
no usages
public void setIdAnuncio(int idAnuncio) {this.idAnuncio = idAnuncio;}
public void setTitulo(String titulo) {this.titulo = titulo;}
no usages
public void setPrecio(String precio) {this.precio = precio;}
no usages
public void setUbicacion(String ubicacion) {this.ubicacion = ubicacion;}
no usages
public void setDireccion(String direccion) {this.direccion = direccion;}
no usages
public void setCiudad(String ciudad) {this.ciudad = ciudad;}
public void setFoto(String foto) {this.foto = foto;}

public class ListAnuncios implements Serializable {
    3 usages
    private int idAnuncio;
    4 usages
    private String titulo;
    4 usages
    private String precio;
    3 usages
    private String ubicacion;
    3 usages
    private String direccion;
    3 usages
    private String ciudad;
    4 usages
    private String foto;

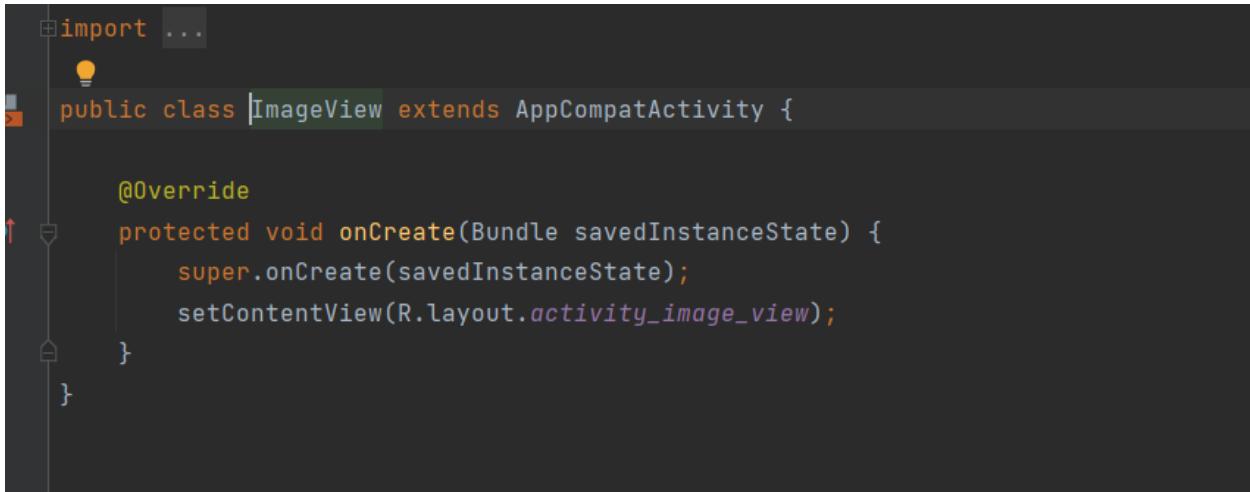
    6 usages
    public ListAnuncios(int idAnuncio, String titulo, String precio, String ubicacion, String foto) {
        this.idAnuncio = idAnuncio;
        this.titulo = titulo;
        this.precio = precio;
        this.ubicacion = ubicacion;
        this.foto = foto;
    }
}

```

ImageView

Es una clase que era necesaria para los Carrousel del Añadir Anuncio, por lo que solo nos importa

el .xml que genera



```

import ...

public class ImageView extends AppCompatActivity {

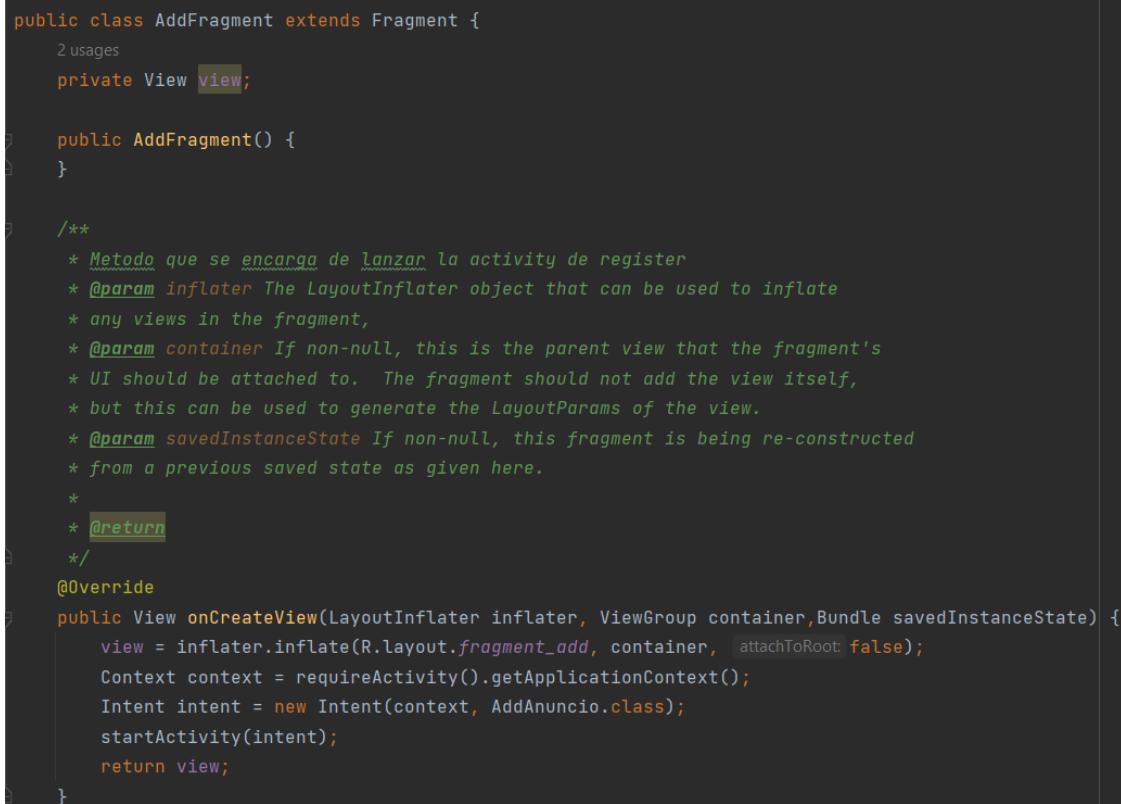
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_image_view);
    }
}

```

Add Fragment

Clase que controla el Fragment Add , que al hacer clic , activa la actividad de Crear Anuncio , actividad que puede ser rehusada por el editar anuncio

El método onCreateView se encarga de que la primera vez que se arranca este Fragment lo que hace es lanzar la actividad de AñadirAnuncio , que contiene todos los datos necesarios para la inserción de Anuncios



```

public class AddFragment extends Fragment {
    2 usages
    private View view;

    public AddFragment() {
    }

    /**
     * Metodo que se encarga de lanzar la activity de register
     * @param inflater The LayoutInflater object that can be used to inflate
     * any views in the fragment,
     * @param container If non-null, this is the parent view that the fragment's
     * UI should be attached to. The fragment should not add the view itself,
     * but this can be used to generate the LayoutParams of the view.
     * @param savedInstanceState If non-null, this fragment is being re-constructed
     * from a previous saved state as given here.
     *
     * @return
     */
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        view = inflater.inflate(R.layout.fragment_add, container, false);
        Context context = requireActivity().getApplicationContext();
        Intent intent = new Intent(context, AddAnuncio.class);
        startActivity(intent);
        return view;
    }
}

```

El método OnResume se encarga de controlar cuando la actividad anterior ha sido borrada , por lo que te envía al fragment Home

```

/*
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
    view = inflater.inflate(R.layout.fragment_add, container, attachToRoot: false);
    Context context = requireActivity().getApplicationContext();
    Intent intent = new Intent(context, AddAnuncio.class);
    startActivity(intent);
    return view;
}

/**
 * Método que se encarga de capturar cuando el fragment ha sido pausado
 */
@Override
public void onPause() { super.onPause(); }

/**
 * Método que captura cuando el fragment vuelve a ser reanudado
 */
@Override
public void onResume() {
    super.onResume();
    NavController navController = Navigation.findNavController(requireActivity(), R.id.nav_hostfragment);
    navController.navigate(R.id.page_inicio); // Reemplaza "homeFragment" con el ID de tu fragmento de inicio
}

```

Envíos

Esta clase Java gestiona los productos que un usuario ha vendido.

Propiedades que conforman la clase Envíos.

```

private TextView tvNoAnunVend;
private RecyclerView recyclerView;
private BundleRecovery almacen;
private int IdUser;
private ListAdapter listAdapter ;

```

El método sobrescrito `onCreate` es el primer método que se ejecuta al llamar a la activity.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_envios);

    initComponents();
    cargarAnuncios();
}

```

El método *initComponents* inicializa los componentes.

```
public void initComponents(){
    tvNoAnunVend = findViewById(R.id.tvNoAnunVend);
    recyclerView = findViewById(R.id.rvEnvios);
    SharedPreferences sharedpreferences = getSharedPreferences( name: "MisDatos", MODE_PRIVATE);
    almacen = new BundleRecovery(sharedpreferences);
}
```

El método cargarAnuncios se encarga de asignar los datos de los anuncios al recyclerview.

```
private void cargarAnuncios(){
    Metodos m = new Metodos();
    IdUser = almacen.re recuperarInt( clave: "logginId"); // idUsuario logeado
    List<ListAnuncios> pedidos = m.getEnvios(new String[]{Integer.toString(IdUser)});
    if (pedidos != null) {
        listAdapter = new ListAdapter(pedidos, getApplicationContext(), fragment: null, R.layout.list_anuncios);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
        recyclerView.setAdapter(listAdapter);
        if (listAdapter.getItemCount() > 0) {
            tvNoAnunVend.setVisibility(View.GONE);
        } else {
            tvNoAnunVend.setVisibility(View.VISIBLE);
        }
    }
}
```

Anuncio

Clase usada para la creación de Anuncios

Capturas de los set/get y el constructor

```
public class Anuncio {
    3 usages
    private int id;
    3 usages
    private int idUsuario;
    3 usages
    private int idCategoria;
    3 usages
    private String titulo;
    3 usages
    private String descripcion;
    3 usages
    private String estado;
    3 usages
    private String ubicacion;
    3 usages
    private String precio;
    3 usages
    private String fotos;

    public Anuncio(){}

    public Anuncio(int id, int idUsuario, int idCategoria, String titulo, String descripcion,
                  String estado, String ubicacion, String precio, String fotos) {
        this.id = id;
        this.idUsuario = idUsuario;
        this.idCategoria = idCategoria;
        this.titulo = titulo;
        this.descripcion = descripcion;
        this.estado = estado;
        this.ubicacion = ubicacion;
        this.precio = precio;
        this.fotos = fotos;
    }
}
```

```

public int getId() {return id;}
2 usages
public int getIdUsuario() {return idUsuario;}
2 usages
public int getidCategoria() {return idCategoria;}
public String getTitulo() {return titulo;}
3 usages
public String getDescripcion() {return descripcion;}
3 usages
public String getEstado() {return estado;}
3 usages
public String getUbicacion() {return ubicacion;}
3 usages
public String getPrecio() {return precio;}
3 usages
public String getFotos() {return fotos;}

public void setId(int id) {this.id = id;}
2 usages
public void setIdUsuario(int idUsuario) {this.idUsuario = idUsuario;}
2 usages
public void setidCategoria(int idCategoria) {this.idCategoria = idCategoria;}
public void setTitulo(String titulo) {this.titulo = titulo;}
2 usages
public void setDescripcion(String descripcion) {this.descripcion = descripcion;}
2 usages
public void setEstado(String estado) {this.estado = estado;}
2 usages
public void setUbicacion(String ubicacion) {this.ubicacion = ubicacion;}
2 usages
public void setPrecio(String precio) {this.precio = precio;}
2 usages
public void setFotos(String fotos) {this.fotos = fotos;}

```

BundleRecovery

Es una clase que permite guardar datos en Caché del Móvil , así como el id del usuario para el inicio de Sesión

El método de GuardarString , al pasarle una Clave / valor guarda un String

El metodo GuardarInt , al pasarle una Clave / valor guarda un Int

Los metodos recuperarInt/recuperarString retornan valores int/String segun Clave

```

import android.content.SharedPreferences;

public class BundleRecovery {

    5 usages
    private final SharedPreferences preferencias;
    public BundleRecovery(SharedPreferences preferencias){this.preferencias = preferencias;}
    no usages
    public void guardarString(String clave, String valor){
        SharedPreferences.Editor editor = preferencias.edit();
        editor.putString(clave, valor);
        editor.apply();
    }
    3 usages
    public void guardarInt(String clave, int valor){
        SharedPreferences.Editor editor = preferencias.edit();
        editor.putInt(clave,valor);
        editor.apply();
    }
    public int recuperarInt(String clave) { return preferencias.getInt(clave, defaultValue: -1); }
    no usages
    public String recuperarString(String clave) { return preferencias.getString(clave, defaultValue: null); }
}

```

EditarAnuncio

Esta clase Java permite la edición de anuncios existentes de un usuario.

Propiedades que conforman la clase EditarAnuncio.

```
private EditText etTituloMod;
private EditText etDescripMod;
private EditText etPrecioMod;
private Spinner spEstadoMod;
private Spinner spCategoriaMod;
private Button btModAnuncio;
private AutoCompleteTextView actvUbiMod;
private String idAnuncio;
ArrayList<String> estados;
private ImageAdapter adapter;
private RecyclerView recyclerView;
private ArrayList<String> fotos = new ArrayList<>();
private BundleRecovery almacenDatos;
```

El método onCreate se encarga de preparar los datos principales ya que como hemos abierto un anuncio para ser editado , tenemos que tener los datos cargados para borrarlos .

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_editar_anuncio);
    initComponents();
    addDatosFormularioAnuncio();
    autocompletarUbicacion();
    findViewById(R.id.btremoveFoto).setOnClickListener(view->borrarUltFoto());
    findViewById(R.id.btaddFoto).setOnClickListener(view -> ImagePicker.with( activity: this).crop().maxResultSize( width: 480, height: 320).start());
    adapter.setOnItemClickListener(new ImageAdapter.OnItemClickListener() {
        @Override
        public void onClick(ImageView imageView, String path) {
            startActivityForResult(new Intent( packageContext: EditarAnuncio.this, ImageView.class).putExtra( name: "image", path),
                ActivityOptions.makeSceneTransitionAnimation( activity: EditarAnuncio.this, imageView, sharedElementName: "image").toBundle());
        }
    });
    btModAnuncio.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Metodos m = new Metodos();
            ArrayList<String> Salvadas = new ArrayList<>();
            String fotosSubidas = "";
            for(int i = 0; i< fotos.size();i++){
                if(fotos.get(i).toString().contains("http://") || fotos.get(i).toString().contains("https://")){
                    String[] fotoSplitted = fotos.get(i).split( regex: "/");
                    Salvadas.add(fotoSplitted[fotoSplitted.length-1].toString());
                }
            }
            if(m.borrarFotosIdAnuncio(idAnuncio,Salvadas)){//Si Fotos Anuncio Borradas
                for(int i = 0;i<fotos.size();i++){
                    if(!fotos.get(i).contains("http://") && !fotos.get(i).contains("https://")){
                        fotosSubidas+=m.subirFotoServer(fotos.get(i),almacenDatos.recuperarInt( clave: "loginId"),Integer.parseInt(idAnuncio));
                    }else fotosSubidas+=fotos.get(i);
                    if(i<fotos.size())fotosSubidas+=";";
                }
            }
        }
    });
}
```

El btModAnuncioOnclick se encarga de guardar el Anuncio con los datos existentes en ese

momento

```

btModAnuncio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Metodos m = new Metodos();
        ArrayList<String> Salvadas = new ArrayList<String>();
        String fotosSubidas = "";
        for(int i = 0; i< fotos.size();i++){
            if(fotos.get(i).toString().contains("http://") || fotos.get(i).toString().contains("https://")){
                String[] fotoSplitted = fotos.get(i).split( regex: "/");
                Salvadas.add(fotoSplitted[fotoSplitted.length-1].toString());
            }
        }
        if(m.borrarFotosIdAnuncio(idAnuncio,Salvadas)){//Si Fotos Anuncio Borradas
            for(int i = 0;i<fotos.size();i++){
                if(!fotos.get(i).contains("http://") && !fotos.get(i).contains("https://")){
                    fotosSubidas+=m.subirFotoServer(fotos.get(i),almacenDatos.recuperarInt( clave: "loginId"),Integer.parseInt(idAnuncio));
                }else fotosSubidas+=fotos.get(i);
                if(i-1<fotos.size())fotosSubidas+=";";
            }
        }
        String [] params2 = {spCategoriaMod.getSelectedItem().toString()};
        int idcategoria = m.getCategoriaId(params2);
        String []params = {idAnuncio,
                           etTituloMod.getText().toString(),
                           etDescripMod.getText().toString(),
                           spEstadoMod.getSelectedItem().toString(),
                           actvUbiMod.getText().toString(),
                           etPrecioMod.getText().toString(),
                           fotosSubidas,Integer.toString(idcategoria)};
        if(m.updateAnuncio(params))Toast.makeText(context: EditarAnuncio.this, text: "El anuncio actualizado con éxito!", Toast.LENGTH_SHORT).show();
        else Toast.makeText(context: EditarAnuncio.this, text: "No se modificó ningún campo", Toast.LENGTH_SHORT).show();
        finish();
    }
});
```

El metodo OnActibityResult captura al cerrarse el Selector de fotos y detecta si la foto ha sido seleccionada o no

```

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(data.getData()!= null){
        String urlFoto = data.getData().getPath();
        fotos.add(urlFoto);
        recyclerView.setAdapter(adapter);
    }
}
```

El metodo addDatosFormularioAnuncio se encarga de cargar los datos principales del anuncio al abrir el editor por primera vez

```

/**
 * Método que asigna los datos del anuncio al formulario
 * a partir del intent de la activity anterior
 */
1 usage
private void addDatosFormularioAnuncio(){
    Metodos m = new Metodos();
    Intent intent = getIntent();
    estados = new ArrayList<>();
    estados.add("Muy bueno");
    estados.add("Usado");
    estados.add("Nuevo");

    idAnuncio = intent.getStringExtra( name: "au_idAnuncio");
    etTituloMod.setText(intent.getStringExtra( name: "au_titulo"));
    etDescripMod.setText(intent.getStringExtra( name: "au_descripcion"));
    etPrecioMod.setText(intent.getStringExtra( name: "au_precio"));
    setSpinner(spCategoriaMod, m.getcategorias());
    spCategoriaMod.setSelection(m.getcategorias().indexOf(intent.getStringExtra( name: "au_categoria")));
    setSpinner(spEstadoMod, estados);
    spEstadoMod.setSelection(estados.indexOf(intent.getStringExtra( name: "au_estado")));
    actvUbiMod.setText(intent.getStringExtra( name: "au_ubicacion"));
    String []fotosArray = intent.getStringExtra( name: "au_fotos").split( regex: ";");
    for(int i = 0; i<fotosArray.length;i++)fotos.add(fotosArray[i]);
    recyclerView.setAdapter(adapter);
}

```

El metodo SetSpinner se encarga de cargar los datos de un spinner una vez pasado el id del spinner

```

2 usages
public static void setSpinner(Spinner spinner,ArrayList<String>estados){
    ArrayAdapter<String> adapter = new ArrayAdapter<>(spinner.getContext(), android.R.layout.simple_spinner_item, estados);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    // Asignar el adaptador al Spinner
    spinner.setAdapter(adapter);
}

```

El metodo initComponents se encarga de obtener el id de los componentes de la Aplicacion

```

private void initComponents(){
    etTituloMod = findViewById(R.id.etTituloMod);
    etDescripMod = findViewById(R.id.etDescripMod);
    etPrecioMod = findViewById(R.id.etPrecioMod);
    spEstadoMod = findViewById(R.id.spEstadoMod);
    spCategoriaMod = findViewById(R.id.spCategoriaMod);
    actvUbiMod = findViewById(R.id.actvUbiMod);
    btModAnuncio = findViewById(R.id.btModAnuncio);

    recyclerView = findViewById(R.id.recyclerEditar);
    adapter = new ImageAdapter( context: this, fotos);

    SharedPreferences sharedPreferences = getSharedPreferences( name: "MisDatos", MODE_PRIVATE);
    almacenDatos = new BundleRecovery(sharedPreferences);

}

```

Este método lo que hace es preparar el selector de Ubicacion

```

private void autocompletarUbicacion(){
    ServerCommunication communication = new ServerCommunication();
    try{
        String[]Municipios = communication.getMunicipios().split( regex: ":" );
        ArrayAdapter<String> adapter =
            new ArrayAdapter<~>( context: EditarAnuncio.this, android.R.layout.simple_dropdown_item_1line, Municipios);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        actvUbiMod.setAdapter(adapter);
    }catch (Exception e){
        Log.i( tag: "Error", e.toString());
    }
}

```

Este metodo lo que hace es borrar la ultima foto del Carrousel

```

private void borrarUltFoto(){
    if(fotos.size()>0){
        fotos.remove( index: fotos.size()-1);
        recyclerView.setAdapter(adapter);
    }else Toast.makeText(getApplicationContext(), text: "No hay fotos disponibles para borrar", Toast.LENGTH_SHORT).show();
}

```

Login

Esta clase Java gestiona el inicio de sesión de los usuarios para loguearse en la aplicación.

Propiedades que conforman la clase Login.

```

private TextView registerButton;
private BundleRecovery dataRecovery;
private Metodos metodos;
private Button IniciarSesion;
private EditText loginNombreUsuario;
private EditText loginContraseña;

```

El método sobreescrito `onCreate` es el primer método que se ejecuta al iniciar la activity.

En él tenemos eventos si se pulsa en el botón de registro para que redireccione al usuario a otra activity con el formulario de registro o el botón iniciar sesión para que compruebe las credenciales. Si es exitoso el inicio de sesión guarda el id del usuario para toda la sesión.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);
    SharedPreferences sharedpreferences = getSharedPreferences( name: "MisDatos", MODE_PRIVATE);
    metodos = new Metodos();
    dataRecovery = new BundleRecovery(sharedpreferences);
    // comprueba si has iniciado sesion
    // si loginId != -1 ha iniciado sesion
    //dataRecovery.guardarInt("loginId", -1);
    if(dataRecovery.recuperarInt( clave: "loginID")!= -1)LanzarMain();

    registerButton = findViewById(R.id.btLoginRegistrate);
    loginContraseña = findViewById(R.id.loginContraseña);
    loginNombreUsuario = findViewById(R.id.LoginNoUsu);
    IniciarSesion = findViewById(R.id.IniciarSesion);

    registerButton.setOnClickListener(new View.OnClickListener() // Te lleva a la ventana de Inicio de Sesión
    {public void onClick(View v) {abrirRegister();}}); // Te lleva a la ventana de Inicio de Sesión
    IniciarSesion.setOnClickListener(new View.OnClickListener(){
        public void onClick(View v) {
            if(loginNombreUsuario.getText().toString().compareTo("")!= 0 & loginContraseña.getText().toString().compareTo("")!= 0){
                String[] p1 = {loginNombreUsuario.getText().toString(), loginContraseña.getText().toString()};
                if(metodos.verificarAuthCliente(p1)){
                    String[]p2 = {loginNombreUsuario.getText().toString()};
                    int idUser = metodos.getIdUser(p2);
                    dataRecovery.guardarInt( clave: "loginId",idUser);
                    LanzarMain();
                }else Toast.makeText(getApplicationContext(), text: "Usuario o Contraseña Incorrectos", Toast.LENGTH_SHORT).show();
            }else Toast.makeText(getApplicationContext(), text: "Los campos no pueden estar vacíos", Toast.LENGTH_SHORT).show();
        }
    });
}
}

```

Los siguientes métodos lanzan una nueva activity.

```

private void LanzarMain(){
    Intent intent = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent);
    finish();
}

/*Metodo que se encarga de abrir la ventana de Registro*/
private void abrirRegister(){
    Intent intent = new Intent( packageContext: this, Register.class);
    startActivity(intent);
    finish();
}

```

ServerCommunication

Es una clase que contiene los llamadores al servidor , además es la clase encargada de encender el hilo para la comunicación con el servidor , además de otros llamamientos necesarios para el tratamiento de las imágenes de los anuncios del servidor

Aquí podemos ver los Constructores y el método getResultadoServer que retorna el resultado del Servidor

```
+4 usages
public class ServerCommunication {
    4 usages
    private String urlServer = "http://192.168.18.5/sv-php";
    6 usages
    private String resultadoServer = "";
    2 usages
    private String urlServerFtp = "192.168.18.5";
    2 usages
    private int PuertoFTP = 21;
    2 usages
    private String userFtp = "sv-anunciaya";
    2 usages
    private String passFtp = "00&67mG{oE";
    2 usages
    private String rutaFtp = "/";

    no usages
    public ServerCommunication(String server){urlServer = server;}

    5 usages
    public ServerCommunication(){}
    1 usage
    public String getResultadoServer() {return resultadoServer;}
    1 usage
```

Este es el Método de Comunicacion , que pasado una Url , una clase, un metodo y unos parametros se encarga de establecer comunicacion con los metodos del servidor y dicha respuesta la guarda en respuestaServer , esta respuesta es dada con Json es por eso que depende del metodo que se ejecuta en el servidor necesitamos parsear unos datos u otros.

```

private String comunicacion(String urlServer, String clase, String metodo, String[] params) {
    try {
        // URL del servidor PHP
        URL url = new URL(urlServer + "/index.php");

        // Abrir conexión
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setDoOutput(true);

        String data = "{\"class\": \"" + clase + "\", \"method\": \"" + metodo + "\", \"params\": " + getParamsParsed(params) + "}";

        // Escribir datos en el cuerpo de la solicitud
        OutputStream os = conn.getOutputStream();
        os.write(data.getBytes());
        os.flush();

        // Leer respuesta del servidor
        BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        StringBuilder response = new StringBuilder();
        String output;
        while ((output = br.readLine()) != null) {
            response.append(output);
        }
        conn.disconnect();
        return response.toString();
    } catch (Exception e) {
        Log.e("tag: ErrorServer", "msg: Error al comunicarse con el servidor", e);
        return null;
    }
}

```

El método *getParamsParsed* se encarga de meter los parámetros entre corchetes[] y por cada parámetro introduce una coma ej [dato1,dato2...]

El metodo *lanzarPeticion* se encarga de lanzar un hilo para llamar al metodo de Comunicacion para poder recibir la respuesta ya que android studio no deja realizar peticiones con el mismo hilo principal

```

private static String getParamsParsed(String[] params) {
    String retornador = "[";
    for (int i = 0; i < params.length; i++) {
        if (i != params.length - 1) {
            retornador += "\"" + params[i] + "\" , ";
        } else {
            retornador += "\"" + params[i] + "\" ]";
        }
    }
    return retornador;
}
23 usages
public Boolean LanzarPeticion(String clase, String metodo, String[] parametros) {
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() { resultadoServer = comunicacion(urlServer, clase, metodo, parametros); }
    });
    // lanzamos el hilo y esperamos a que termine
    try { thread.start(); thread.join(); return true; }
    catch (Exception e) { return false; }
}

```

El método *borrarFoto server* se encarga de establecer conexión con el servidor FTP para borrar las fotos del servidor que tienen que ser borradas

```

public Boolean borrarFotoServer(String idAnuncio,ArrayList<String>Salvadas){
    FTPClient ftpClient2 = new FTPClient();
    try {
        ftpClient2.connect(urlServerFtp, PuertoFTP);
        ftpClient2.login(userFtp, passFtp);
        ftpClient2.enterLocalPassiveMode();
        ftpClient2.setFileType(FTP.BINARY_FILE_TYPE);

        // Cambiamos al directorio donde están los archivos
        ftpClient2.changeWorkingDirectory(rutaFtp);

        // Obtenemos la lista de nombres de archivos
        String[] archivos = ftpClient2.listNames();

        // Iteramos sobre los archivos y eliminamos aquellos que comiencen con "15_"
        if (archivos != null) {
            for (String archivo : archivos) {
                if (archivo.startsWith(idAnuncio+"_") && !Salvadas.contains(archivo)) {
                    ftpClient2.deleteFile(archivo);
                }
            }
        }
        return true;
    } catch (IOException ex) {
        ex.printStackTrace();return false;
    } finally {
        try {
            if (ftpClient2.isConnected()) {ftpClient2.logout();ftpClient2.disconnect();}
        } catch (IOException ex) {ex.printStackTrace();}
    }
}
}

```

El metodo subir foto server se encarga de realizar una petición FTP para subir una Imagen al servidor, con un nombre en concreto

```

public String subirFotoServer(String rutaFoto,int idUsuario,int idAnuncio){
    FTPClient ftpClient = new FTPClient();
    //String serverFTPSubida = "http://sv-anunciaya.000webhostapp.com/sv-php";
    try {
        ftpClient.connect(urlServerFtp, PuertoFTP);
        ftpClient.login(userFtp, passFtp);
        ftpClient.enterLocalPassiveMode();
        ftpClient.setFileType(FTP.BINARY_FILE_TYPE);

        File file = new File(rutaFoto);
        String fileName = idAnuncio+"_"+idUsuario+"_"+file.getName();
        FileInputStream inputStream = new FileInputStream(file);

        boolean uploaded = ftpClient.storeFile( remote: rutaFtp + "/" + fileName, inputStream);
        inputStream.close();
        if (uploaded) {
            return urlServer+"/img/"+fileName;
        } else {
            return null;
        }
    } catch (IOException ex) {
        Log.i( tag: "ErrorFtp",ex.toString());
        return null;
    }finally {
        try {
            if (ftpClient.isConnected()) {ftpClient.logout();ftpClient.disconnect();}
        } catch (IOException ex) {ex.printStackTrace();}
    }
}

```

El metodo *borrarFotos* es el encargado de Arrancar el hilo para que funcione el *borrarFotoServer*, porque android Studio no deja realizar peticiones con el Hilo principal

El metodo *subirFoto* es el encargado de Arrancar el hilo para que funcione el subirFotoServer, porque android Studio no deja realizar peticiones con el Hilo principal

```
1 usage
public Boolean borrarFotos(String id, ArrayList<String>Salvadas){
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {borrarFotoServer(id,Salvadas);}
    });
    // lanzamos el hilo y esperamos a que termine
    try{thread.start();thread.join(); return true;}
    catch (Exception e){return false;}
}

1 usage
public String subirFoto(String url, int idUsuario,int idAnuncio){
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {resultadoServer = subirFotoServer(url,idUsuario,idAnuncio);}
    });
    // lanzamos el hilo y esperamos a que termine
    try{thread.start();thread.join(); return resultadoServer;}
    catch (Exception e){return null;}
}
```

El método de *ObtenerMunicipios* se encarga de realizar una petición al servidor ya que en el servidor hay un xml que contiene los municipios que se muestran en apartado de insertar Anuncio y en el buscador

```
private String obtenerMunicipios(String urlServer){
    try {
        String respuesta = "";
        // Crear una URL y establecer la conexión HTTP
        URL url = new URL(spec urlServer + "/util/municipios.xml");
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        conn.connect();

        // Leer la respuesta de la solicitud HTTP
        InputStream inputStream = conn.getInputStream();

        // Parsear el XML
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document doc = builder.parse(inputStream);

        // Crear un objeto XPath
        XPath xpath = XPathFactory.newInstance().newXPath();
        // Compilar la expresión XPath
        XPathExpression expr = xpath.compile( expression: "//municipio");
        // Evaluar la expresión XPath para obtener el resultado
        NodeList nodeList = (NodeList) expr.evaluate(doc, XPathConstants.NODESET);
        // Recorrer los nodos y obtener el texto de cada nodo
        for (int i = 0; i < nodeList.getLength(); i++) {
            Element element = (Element) nodeList.item(i);
            respuesta+=element.getTextContent()+"";
        }
        return respuesta;

    } catch (Exception e) {
        e.printStackTrace(); return null;
    }
}
```

Este método *getMunicipios* se encarga de lanzar el hilo que es necesario para el metodo de Obtener Municipios y retorna el resultado

```
4 usages
public String getMunicipios(){
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() { resultadoServer = obtenerMunicipios(urlServer); }
    });
    // lanzamos el hilo y esperamos a que termine
    try{thread.start();thread.join();return resultadoServer;}
    catch (Exception e){return null;}
}
```

Usuario

Clase usada para el manejo de los datos del usuario

En esta clase podremos ver los get/set y el constructor del usuario

```
public class Usuario {
    private int id;
    private String nombre;
    private String apellidos;
    private String nombUsu;
    private String contras;
    private String fechaNacimiento;
    private String email;
    private String telefono;
    private String tipo;
    private String foto;

    public Usuario(){}
}
```

```
public Usuario(int id, String nombre, String apellidos, String nombUsu,
              String contras, String fechaNacimiento, String email, String telefono,
              String tipo, String foto) {
    this.id = id;
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.nombUsu = nombUsu;
    this.contras = contras;
    this.fechaNacimiento = fechaNacimiento;
    this.email = email;
    this.telefono = telefono;
    this.tipo = tipo;
    this.foto = foto;
}

public int getId() {return id;}
4 usages
public String getNombre() {return nombre;}
3 usages
public String getApellidos() {return apellidos;}
1 usage
public String getNombUsu() {return nombUsu;}
no usages
public String getContras() {return contras;}
1 usage
public String getFechaNacimiento() {return fechaNacimiento;}
2 usages
public String getEmail() {return email;}
2 usages
public String getTelefono() {return telefono;}
no usages
public String getTipo() {return tipo;}
public String getFoto() {return foto;}

public void setId(int id) {this.id = id;}
1 usage
public void setNombre(String nombre) {this.nombre = nombre;}
1 usage
public void setApellidos(String apellidos) {this.apellidos = apellidos;}
1 usage
public void setNombUsu(String nombUsu) {this.nombUsu = nombUsu;}
1 usage
public void setContras(String contras) {this.contras = contras;}
1 usage
public void setFechaNacimiento(String fechaNacimiento) {this.fechaNacimiento = fechaNacimiento;}
1 usage
public void setEmail(String email) {this.email = email;}
1 usage
public void setTelefono(String telefono) {this.telefono = telefono;}
1 usage
public void setTipo(String tipo) {this.tipo = tipo;}
public void setFoto(String foto) {this.foto = foto;}
```

Register

Esta clase Java gestiona el registro de un nuevo usuario.

Propiedades que conforman la clase Register.

```
private TextView tvTitReg;
private TextView tvTitActDat;
private DatePickerDialog datePickerDialog;
private Button dateButton;
private Button registerButton;
private EditText nombre, apellidos,nomb_usu,email,telf,contras;
private BundleRecovery dataRecovery;
private TextView iniciaSesion;
private Boolean lanzadaMain = false;
private Usuario user ;
private ImageView imagen;
```

El método sobreescrito *onCreate* se ejecuta al iniciar la activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_register);
    initComponents();
    initDatePicker();

    lanzadaMain = getIntent().getBooleanExtra("fromMainActivity", false);
    if (lanzadaMain) PrepararDatos();

    iniciaSesion.setOnClickListener(v -> LanzarLoggin());
    registerButton.setOnClickListener(v -> onclickInsertarUsuario());
}
```

El método *initComponents* inicializa los componentes.

```
private void initComponents(){
    SharedPreferences sharedpreferences = getSharedPreferences("MisDatos", MODE_PRIVATE);
    dataRecovery = new BundleRecovery(sharedpreferences);

    tvTitReg = findViewById(R.id.tvTitReg);
    tvTitReg.setVisibility(View.VISIBLE);
    tvTitActDat = findViewById(R.id.tvTitActDat);
    tvTitActDat.setVisibility(View.GONE);
    nombre = findViewById(R.id.registerNombre);
    apellidos = findViewById(R.id.registerApellidos);
    nomb_usu = findViewById(R.id.registerNombreUsuario);
    email = findViewById(R.id.registerEmail);
    telf = findViewById(R.id.registerTelefono);
    contras = findViewById(R.id.registerContrasena);
    iniciaSesion = findViewById(R.id.btRegistrateIniciaSesion);
    imagen = findViewById(R.id.imageView);

    dateButton = findViewById(R.id.registerEdad);
    dateButton.setText(getFecha());
    registerButton = findViewById(R.id.Registrar);
}
```

El método *PrepararDatos* recupera la información del usuario actualmente autenticado para

emplearla en la modificación de sus datos en la configuración del usuario. Este método es una reutilización de la clase register.

```
private void PrepararDatos(){
    Metodos metodos = new Metodos();
    try{
        String[]params = {Integer.toString(dataRecovery.recuperarInt( clave: "logginId"))};
        user = metodos.getUsuarioDataId(params);
        if(user != null){
            nombre.setText(user.getNombre());
            apellidos.setText(user.getApellidos());
            nomb_usu.setText(user.getNombUsu());
            dateButton.setText(user.getFechaNacimiento());
            email.setText(user.getEmail());
            telf.setText(user.getTelefono());
            registerButton.setText("Guardar Cambios");
            imagen.setImageResource(R.drawable.user_logo_black);
            tvTitReg.setVisibility(View.GONE);
            tvTitActDat.setVisibility(View.VISIBLE);

            RelativeLayout preguntarCuenta = findViewById(R.id.tienesCuentaAsk);
            preguntarCuenta.setVisibility(View.GONE);

        }else{
            Toast.makeText(getApplicationContext(), text: "Error al Cargar los datos del Usuario", Toast.LENGTH_SHORT).show();
            finish();
        }
    }catch (Exception e){
        Log.i( tag: "Error:",e.toString());
    }
}
```

El método *onclickInsertarUsuario* registra un nuevo usuario en la base de datos.

```
private void onclickInsertarUsuario(){
    Metodos metodos = new Metodos();
    if(nombre.getText().toString().compareTo("") != 0){
        if(apellidos.getText().toString().compareTo("")!= 0){
            if(nomb_usu.getText().toString().compareTo("")!= 0){
                if(email.getText().toString().compareTo("")!= 0){
                    if(telf.getText().toString().compareTo("")!= 0){
                        if(contras.getText().toString().compareTo("")!= 0 || lanzadaMain){
                            if(dateButton.getText().toString().compareTo("")!= 0 || lanzadaMain){
                                if(lanzadaMain){
                                    String[] params = {
                                        Integer.toString(dataRecovery.recuperarInt( clave: "logginId")), // id_usuario
                                        nombre.getText().toString(), // nombre
                                        apellidos.getText().toString(), // apellido
                                        telf.getText().toString(), // telefono
                                        email.getText().toString(), // email
                                        dateButton.getText().toString(), // fecha_nac
                                        nomb_usu.getText().toString(), // nomb_usu
                                    };
                                    if(contras.getText().toString().compareTo("")!= 0){
                                        String[]contra = { Integer.toString(dataRecovery.recuperarInt( clave: "logginId")),contras.getText().toString()};
                                        if(!metodos.ActualizarContasena(contra))
                                            Toast.makeText(getApplicationContext(), text: "Error al Actualizar la contraseña", Toast.LENGTH_SHORT).show();
                                    }
                                    if(metodos.ActualizarUsuario(params))Toast.makeText(getApplicationContext(), text: "Usuario Actualizado", Toast.LENGTH_SHORT).show();
                                    finish();
                                }else{
                                    String [] p1 = {nombre.getText().toString(),apellidos.getText().toString(),nomb_usu.getText().toString(),contras.getText().toString()};
                                    int idUser = metodos.insertUsuario(p1);
                                    ifidUser!= -1{
                                        dataRecovery.guardarInt( clave: "logginId",idUser);LanzarMain();
                                    }else Toast.makeText(getApplicationContext(), text: "Usuario Existente", Toast.LENGTH_SHORT).show();
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        }else Toast.makeText(getApplicationContext(), text: "El teléfono no puede estar vacío", Toast.LENGTH_SHORT).show();
    }else Toast.makeText(getApplicationContext(), text: "El email no puede estar vacío", Toast.LENGTH_SHORT).show();
    }else Toast.makeText(getApplicationContext(), text: "El nombre de usuario no puede estar vacío", Toast.LENGTH_SHORT).show();
    }else Toast.makeText(getApplicationContext(), text: "Los apellidos no pueden estar vacíos", Toast.LENGTH_SHORT).show();
    }else Toast.makeText(getApplicationContext(), text: "El nombre no puede estar vacío", Toast.LENGTH_SHORT).show();
}
}

```

Los siguientes métodos lanzan una nueva activity.

```

private void LanzarLogin(){
    Intent intent = new Intent( packageContext: this, Login.class);
    startActivity(intent);
    this.finish();
}

private void LanzarMain(){
    Intent intent = new Intent( packageContext: this, MainActivity.class);
    startActivity(intent);
    this.finish();
}
}

```

El método `getFecha` devuelve la fecha actual con la clase Calendar.

```

private String getFecha() {
    Calendar cal = Calendar.getInstance();
    int año = cal.get(Calendar.YEAR);
    int mes = cal.get(Calendar.MONTH);
    mes = mes+1;
    int dia = cal.get(Calendar.DAY_OF_MONTH);
    return makeDateString(dia,mes,año);
}
}

```

El método `initDatePicker` inicializa el componente DatePicker.

```

private void initDatePicker() {
    DatePickerDialog.OnDateSetListener dateSetListener = (view, año, mes, dia) -> {
        mes = mes+1;
        String date = makeDateString(dia,mes,año);
        dateButton.setText(date);
    };
    Calendar cal = Calendar.getInstance();
    int año = cal.get(Calendar.YEAR);
    int mes = cal.get(Calendar.MONTH);
    int dia = cal.get(Calendar.DAY_OF_MONTH);

    int style = AlertDialog.THEME_HOLO_LIGHT;

    datePickerDialog = new DatePickerDialog( context: this,style,dateSetListener,año,mes,dia);
}
}

```

El método `makeDateString` genera el formato de fecha.

```

private String makeDateString(int dia, int mes, int año) {return año+"-"+mes+"-"+dia;}
}

```

El método `openDatePicker` inicia una instancia de datePicker y lo abre.

```

public void openDatePicker(View view){datePickerDialog.show();}
}

```

Metodos

Esta clase Java gestiona todos los métodos del servidor para que el cliente pueda usarlos de una forma más óptima.

Propiedades y constructor que conforman la clase Metodos.

```
private static ServerCommunication communication;

public Metodos() {communication = new ServerCommunication();}
```

Los métodos que componen esta clase son similares al siguiente ejemplo. Se realiza una solicitud HTTP al servidor PHP y, dependiendo del caso, se devuelve un valor booleano, entero, cadena de texto, ArrayList, entre otros.

```
public int insertUsuario(String[]args){
    try{
        if(communication.LanzarPeticion( clase: "Usuario", metodo: "insertUsuario",args)){
            JSONObject jsonObject = new JSONObject(getRespuestaServer());
            if(jsonObject.getBoolean( name: "data")){ // si sale true o se ha insertado
                String []nomb_usu = {args[2]};
                return getIdUser(nomb_usu);
            }else return -1;
        }else return -1;
    }catch (Exception e){return -1;}
}
```

Disponemos de otro tipo de método, como el siguiente ejemplo, en el cual la información obtenida se asigna a un ArrayList y se devuelve. Este es el caso del método utilizado para obtener las categorías.

```
public ArrayList<String> getCategorias() {
    try {
        communication.LanzarPeticion( clase: "Categoria", metodo: "getcategorias", new String[]{""});
        ObjectMapper objectMapper = new ObjectMapper();
        JsonNode jsonNode = objectMapper.readTree(getRespuestaServer());

        // Verificar si el nodo "data" existe y es una cadena JSON válida
        if (jsonNode.has( fieldName: "data")) {
            // Obtener la cadena JSON del nodo "data"
            String dataJsonString = jsonNode.get("data").asText();

            // Parsear la cadena JSON para obtener las descripciones de categorías
            ArrayList<String> categorias = new ArrayList<>();
            JsonNode dataNode = objectMapper.readTree(dataJsonString);
            for (JsonNode node : dataNode) {
                String descripcion = node.get("descripcion").asText();
                categorias.add(descripcion);
            }
            categorias.add("None");

            // Devolver la lista de categorías
            return categorias;
        } else {
            // Manejar el caso cuando no hay datos disponibles
            return new ArrayList<>();
        }
    } catch (Exception e) {
        // Manejar cualquier excepción y devolver null
        return null;
    }
}
```

AddAnuncio

Es la clase necesaria para lanzar la vista de Añadir Anuncio , además es la que controla la

Inserciones de Anuncio

```
public class AddAnuncio extends AppCompatActivity {
    3 usages
    private Spinner spinnerEstado,spinnerCategoria;
    4 usages
    private AutoCompleteTextView ubicacion;
    2 usages
    private ServerCommunication communication;
    3 usages
    private RecyclerView recyclerView;
    9 usages
    private ArrayList<String> fotos = new ArrayList<>();
    4 usages
    private ImageAdapter adapter;
    7 usages
    private Metodos metodos;
    2 usages
    private Button boton;
    3 usages
    private EditText titulo,Descripcion,Precio;
    2 usages
    private BundleRecoveryy almacenDatos;
    2 usages
}
```

El método `onCreate` se encarga de Inicializar todos los datos necesarios que tienen que estar precargados para la hora de insertar anuncios , como por ejemplo los spinner de los estados , la configuración necesaria para el selector de provincias , la configuración del carrousel de imágenes.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_anuncio);
    communication = new ServerCommunication();
    metodos = new Metodos();
    String[]estados = {"Muy bueno","Usado","Nuevo"};
    spinnerCategoria = findViewById(R.id.newCategoria);setSpinner(spinnerCategoria,metodos.getcategorias());
    spinnerEstado = findViewById(R.id.newEstado);setSpinner(spinnerEstado,estados);
    ubicacion = findViewById(R.id.newUbicacion);autocompleteUbicacion();
    recyclerView = findViewById(R.id.recycler);
    titulo = findViewById(R.id.newTitulo);
    Descripcion = findViewById(R.id.newDescripcion);
    Precio = findViewById(R.id.newPrecio);

    SharedPreferences sharedpreferences = getSharedPreferences( name: "MisDatos", MODE_PRIVATE);
    almacenDatos = new BundleRecoveryy(sharedpreferences);
    adapter = new ImageAdapter( context: AddAnuncio.this, fotos);

    findViewById(R.id.btremoveFoto).setOnClickListener(view->borrarUltFoto());
    findViewById(R.id.btaddFoto)
        .setOnClickListener(view ->ImagePicker.with( activity: AddAnuncio.this).crop().maxResultSize( width: 480, height: 320).start());
    adapter.setOnItemClickListener((imageView, path) -> startActivity(new Intent( packageContext: AddAnuncio.this, ImageView.class).putExtra(
        ActivityOptions.makeSceneTransitionAnimation( activity: AddAnuncio.this, imageView, sharedElementName: "image")
        .toBundle()));
    boton = findViewById(R.id.btnewCrearAnuncio);
    boton.setOnClickListener(view -> CrearAnuncio());
}
```

El metodo de `CrearAnuncio` se encarga de Controlar la creación del Anuncio , los datos insertados en este (si son válidos o no), además si estos datos son válidos, el anuncio se insertará

```

    ...
    private void CargarAnuncio(){
        try{
            String[] args = {spinnerCategoria.getSelectedItem().toString()};
            if(args[0].toLowerCase().compareTo("none")!=0){
                int idCategoria = metodos.getCategoridaId(args);
                int idUsuario = almacenDatos.recuperarInt(clave: "loginId");
                if(titulo.getText().toString().compareTo("")!= 0){
                    if(Descripcion.getText().toString().compareTo("")!=0){
                        if(Precio.getText().toString().compareTo("")!= 0){
                            if(Ubicacion.getText().toString().compareTo("")!= 0){
                                if(fotos.size()>0){
                                    Anuncio anuncioCreado = new Anuncio();
                                    anuncioCreado.setTitulo(titulo.getText().toString());
                                    anuncioCreado.setDescripcion(Descripcion.getText().toString());
                                    anuncioCreado.setPrecio(Precio.getText().toString());
                                    anuncioCreado.setUbicacion(Ubicacion.getText().toString());
                                    anuncioCreado.setidCategoria(idCategoria);
                                    anuncioCreado.setEstado(spinnerEstado.getSelectedItem().toString());
                                    anuncioCreado.setFotos("");anuncioCreado.setIdUsuario(idUsuario);
                                    String [] parametros = {
                                        Integer.toString(idUsuario),Integer.toString(idCategoria),
                                        anuncioCreado.getTitulo(),anuncioCreado.getDescripcion(),
                                        anuncioCreado.getEstado(),anuncioCreado.getUbicacion(),
                                        anuncioCreado.getPrecio(),anuncioCreado.getFotos()};
                                    if(metodos.insertarAnuncio(parametros)){
                                        String[] paramsNuevos = {Integer.toString(idUsuario)};
                                        int idAnuncioCreado = metodos.getIdNewAnuncioUsuario(paramsNuevos);
                                        String fotosSubidas = "";
                                        for(int i = 0;i<fotos.size();i++){
                                            fotosSubidas+=metodos.subirFotoServer(fotos.get(i),idUsuario,idAnuncioCreado);
                                            if(i-1<fotos.size())fotosSubidas+=";";
                                            anuncioCreado.setFotos(anuncioCreado.getFotos());
                                        }
                                        if(metodos.insertarAnuncio(parametros)){
                                            String[] paramsNuevos = {Integer.toString(idUsuario)};
                                            int idAnuncioCreado = metodos.getIdNewAnuncioUsuario(paramsNuevos);
                                            String fotosSubidas = "";
                                            for(int i = 0;i<fotos.size();i++){
                                                fotosSubidas+=metodos.subirFotoServer(fotos.get(i),idUsuario,idAnuncioCreado);
                                                if(i-1<fotos.size())fotosSubidas+=";";
                                            }
                                            String[] argumentosUploadImagen = {fotosSubidas,Integer.toString(idAnuncioCreado)};
                                            metodos.updateAnuncioUploadImagen(argumentosUploadImagen);
                                            Toast.makeText(getApplicationContext(), text: "Anuncio Creado Correctamente",Toast.LENGTH_SHORT).show();
                                            finish();
                                        }else Toast.makeText(getApplicationContext(), text: "No se pudo insertar el anuncio", Toast.LENGTH_SHORT).show();
                                    }else Toast.makeText(getApplicationContext(), text: "Al menos añade una Foto", Toast.LENGTH_SHORT).show();
                                }else Toast.makeText(getApplicationContext(), text: "Introduzca la Ubicacion", Toast.LENGTH_SHORT).show();
                            }else Toast.makeText(getApplicationContext(), text: "Introduzca el Precio", Toast.LENGTH_SHORT).show();
                        }else Toast.makeText(getApplicationContext(), text: "Introduzca la Descripcion", Toast.LENGTH_SHORT).show();
                    }else Toast.makeText(getApplicationContext(), text: "Introduzca el Titulo del Anuncio", Toast.LENGTH_SHORT).show();
                }else Toast.makeText(getApplicationContext(), text: "Elija una Categoria valida", Toast.LENGTH_SHORT).show();
            }catch (Exception e){Toast.makeText(getApplicationContext(), text: "Error: "+e.toString(), Toast.LENGTH_SHORT).show();}
        }
    }
}

```

El metodo *onActivityResult* se encarga de Controlar cuando el desplegador de seleccionador de imagenes para subir ha sido cerrado , para añadir las imágenes seleccionadas al carrousel
 El metodo *borrarUltFoto* , se encarga de borrar la ultima foto para cuando damos en el botón de

borrar foto

El metodo de *autocompletearUbicacion* se encarga de preparar los datos del selector de provincias.

```
/*Metodo que se encarga de controlar cuando se ha cerrado el ImagePicker*/
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(data.getData()!= null){
        String urlFoto = data.getData().getPath();
        fotos.add(urlFoto);
        recyclerView.setAdapter(adapter);
    }
}
/*Metodo que borra la ultima foto*/
1 usage
private void borrarUltFoto(){
    if(fotos.size()>0){
        fotos.remove( index: fotos.size()-1 );
        recyclerView.setAdapter(adapter);
    }else Toast.makeText(getApplicationContext(), text: "No hay fotos disponibles para borrar", Toast.LENGTH_SHORT).show();
}
/*Metodo que se encarga de cargar el buscador de */
1 usage
private void autocompleteUbicacion(){
    try{
        String[]Municipios = communication.getMunicipios().split( regex: ";" );
        ArrayAdapter<String> adapter = new ArrayAdapter<>( context: AddAnuncio.this, R.layout.auto_municipios_rojo2, Municipios );
        ubicacion.setAdapter(adapter);
    }catch (Exception e){
        Log.i( tag: "Error", e.toString());
    }
}
```

Los metodos *setSpinner* , pasado un Spinner se encarga de preparar sus datos.

```
1
1 usage
public static void setSpinner(Spinner spinner, String[]estados){
    ArrayAdapter<String> adapter = new ArrayAdapter<>(spinner.getContext(), R.layout.spinner_categorias_rojo2, estados);
    adapter.setDropDownViewResource(R.layout.spinner_categorias_rojo2);
    spinner.setAdapter(adapter);
}
1 usage
public static void setSpinner(Spinner spinner,ArrayList<String>estados){
    ArrayAdapter<String> adapter = new ArrayAdapter<>(spinner.getContext(), R.layout.spinner_categorias_rojo2, estados);
    adapter.setDropDownViewResource(R.layout.spinner_categorias_rojo2);
    // Asignar el adaptador al Spinner
    spinner.setAdapter(adapter);
}
```

MainActivity

Esta clase Java gestiona todos los métodos del servidor para que el cliente pueda usarlos de una forma más óptima.

Propiedades que conforman la clase MainActivity.

```
private BundleRecovery dataRecovery;
private int IdUser;
```

El método sobrescrito `onCreate` es el primer método que se ejecuta al llamar a la activity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    SharedPreferences sharedPreferences = getSharedPreferences( name: "MisDatos", MODE_PRIVATE);
    dataRecovery = new BundleRecovery(sharedPreferences);
    IdUser = ComprobarLoggin();
    setupNavegacion();
}
```

El método `setupNavegacion` configura la navegación de la aplicación utilizando una barra de navegación inferior (BottomNavigationView).

```
private void setupNavegacion() {
    BottomNavigationView bottomNavigationView = findViewById(R.id.bottomNavigationView);
    NavHostFragment navHostFragment = (NavHostFragment) getSupportFragmentManager().findFragmentById(R.id.nav_hostfragment);
    NavigationUI.setupWithNavController(bottomNavigationView,navHostFragment.getNavController());
}
```

El método `ComprobarLoggin` retorna el id del usuario almacenado, en caso de que no haya nada es -1.

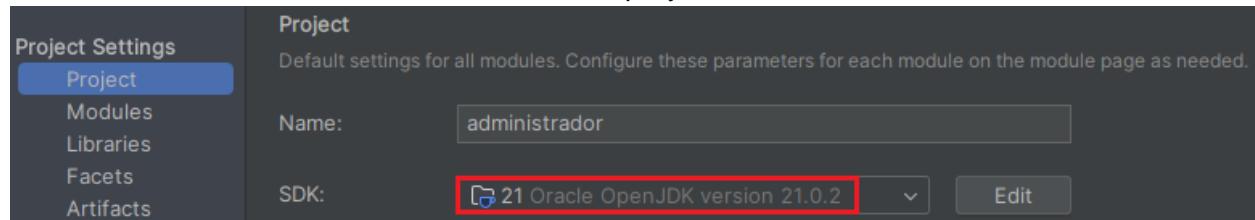
```
private int ComprobarLoggin() { return dataRecovery.recuperarInt( clave: "loginId"); }
```

4.1.2.3 Cliente Administrador Escritorio

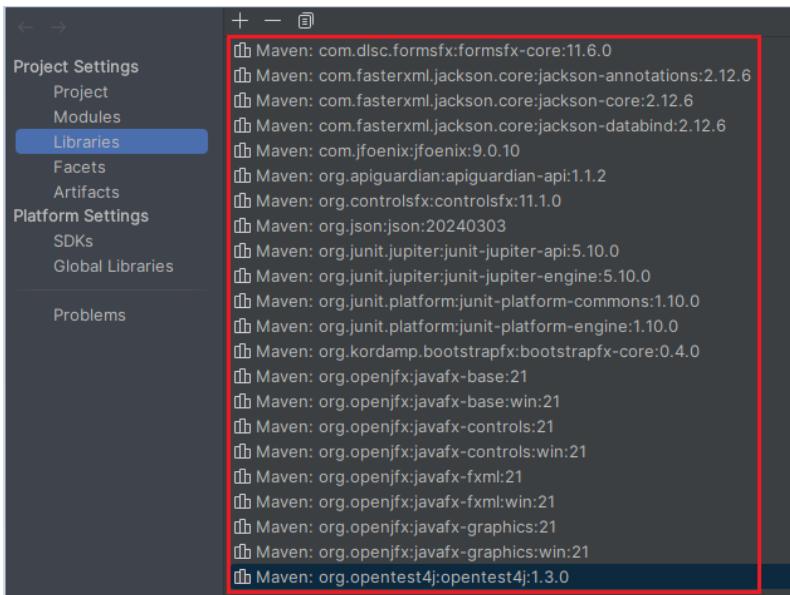
El administrador multiplataforma ha sido desarrollado utilizando la tecnología JavaFX, presentando una apariencia visualmente atractiva. Este administrador ha sido creado con el propósito de gestionar la información del cliente android, tales como usuarios, anuncios, categorías y pedidos registrados.

El desarrollo de la aplicación se ha llevado a cabo empleando el Entorno de Desarrollo Integrado (IDE) IntelliJ IDEA, en su versión 2023.3.4.

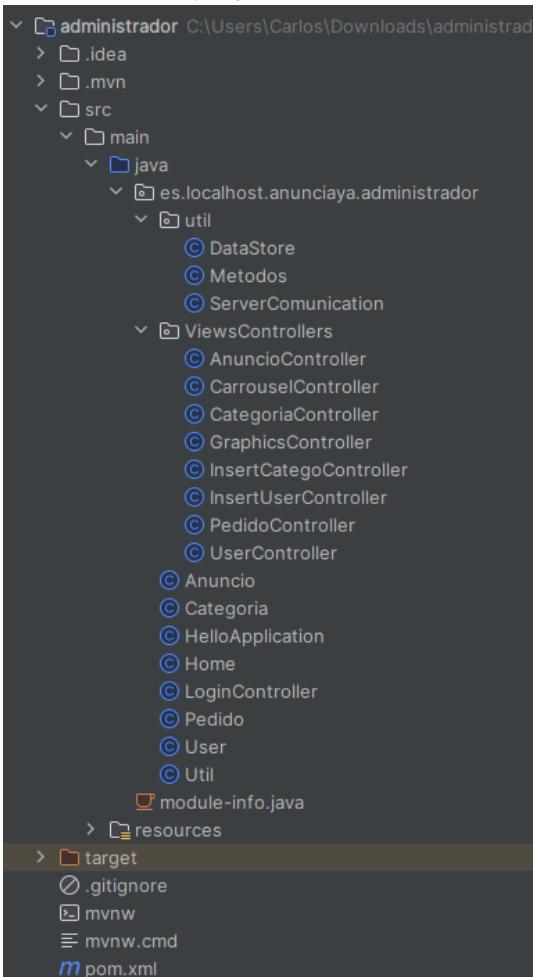
Hemos usado la versión SDK 21 de Java en el proyecto.



Librerías utilizadas a lo largo del desarrollo.



Estructura del proyecto.



- Paquete **util**
 - DataStore → Es una clase que se usa para el tema de guardar datos en cache.
 - Metodos → clase que contiene todos los métodos usados en el proyecto.
 - ServerComunication → clase que gestiona la comunicación del servidor php.
- Paquete **ViewsControllers**
 - AnuncioController → clase que gestiona los anuncios de la bd.
 - CarrouselController → clase que visualiza las imágenes que corresponden a un anuncio existente.
 - CategoriaController → clase que gestiona las categorías de la bd.
 - GraphicsController → clase que obtiene la información del log del servidor.
 - InsertCategoController → clase que añade una nueva categoría a la bd.
 - InsertUserController → clase que añade un nuevo usuario a la bd.
 - PedidoController → clase que gestiona los pedidos de la bd.
 - UserController → clase que gestiona los usuarios de la bd.
- Paquete principal (**administrador**)
 - Anuncio → clase que representa a los anuncios.
 - Categoria → clase que representa a las categorías.
 - HelloApplication → clase principal de la aplicación (llamada a login.fxml).
 - Home → clase que visualiza los gráficos circulares con los logs del servidor.
 - LoginController → clase que verifica la autenticación del usuario administrador.
 - Pedido → clase que representa a los pedidos.
 - User → clase que representa a los usuarios.
 - Util → clase que contiene métodos a reusar.

HelloApplication

Esta clase Java se encarga de iniciar la aplicación llamando al login.fxml.

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("login.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), 558, 406);
        stage.setTitle("Login | AnunciaYa");
        stage.setResizable(false);
        stage.setScene(scene);
        stage.show();
    }

    public static void main(String[] args) { launch(); }
}
```

Login

Esta clase Java gestiona la autenticación de los usuarios administradores.

Propiedades que conforman la clase Login.

```
@FXML private TextField tfEmail;
@FXML private PasswordField pfPasswd;
private Metodos m;
```

El evento *onLoginClick* se ejecutará cuando se pulse en el botón iniciar sesión. Comprobará que las credenciales proporcionadas están registradas en alguna cuenta administrador en la base de datos.

```
@FXML
private void onLoginClick(){
    if(!tfEmail.getText().isEmpty() && !pfPasswd.getText().isEmpty()){
        String[]params = {tfEmail.getText(),pfPasswd.getText()};
        if(m.AuthAdmin(params)){
            DataStore.getInstance().setData(m.getIdUserByEmail(tfEmail.getText()));
            Util.mostrarDialogo( titulo: "MENSAJE_INFO", textoEncab: "Autenticación Exitosa.",
                contenido: """
                    |¡Bienvenido de vuelta al panel de administración!
                    |Ahora tienes acceso completo a todas las funciones
                    |administrativas.""",
                Alert.AlertType.INFORMATION);

            Util.loadNewScene( nScene: "home.fxml", tfEmail, title: "Home | AnunciaYa");
        } else{
            Util.mostrarDialogo( titulo: "MENSAJE_ERROR", textoEncab: "Autenticación Fallida.",
                contenido: "Error al autenticar. Verifica que el correo electrónico y la contraseña sean correctos. ",
                Alert.AlertType.ERROR);
        }
    } else{
        Util.mostrarDialogo( titulo: "MENSAJE_WARNING", textoEncab: "Campos Vacíos",
            contenido: "Por favor, completa todos los campos requeridos para iniciar sesión correctamente.", Alert.AlertType.WARNING);
    }
}
```

El siguiente método sobrescrito *initialize* se ejecuta al iniciar la escena login.

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) { m = new Metodos(); }
```

UserController

Esta clase Java gestiona los usuarios registrados en la base de datos.

Propiedades que conforman la clase UserController.

```
@FXML private TableView<User> tbUsuarios;
@FXML private TableColumn<User, Integer> idColumn;
@FXML private TableColumn<User, String> nombreColumn;
@FXML private TableColumn<User, String> apellidosColumn;
@FXML private TableColumn<User, String> nombrUsuColumn;
@FXML private TableColumn<User, String> fechaNacimientoColumn;
@FXML private TableColumn<User, String> emailColumn;
@FXML private TableColumn<User, String> telefonoColumn;
@FXML private TableColumn<User, String> tipoColumn;
private Metodos me;
private String IdUsuario ;
private ObservableList<User> userList;
```

El siguiente método sobrescrito *initialize* se ejecuta al iniciar la escena login.

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    IdUsuario = DataStore.getInstance().getData();
    me = new Metodos();
    PrepararTabla();
    CargarDatosTabla();
}
```

El método *CargarDatosTabla* asigna los datos de los usuarios de la base de datos obtenidos con

el método `getAllUsers` y los asigna a la tabla.

```
public void CargarDatosTabla(){
    String[]params = {IdUsuario};
    userList = me.getAllUsers(params);
    tbUsuarios.setItems(userList);
}
```

El método `PrepararTabla` configura las columnas de la tabla para asignar los datos correspondientes a cada campo de la base de datos. Además, establece eventos en las columnas de la tabla para permitir la edición del campo al hacer doble clic.

```
private void PrepararTabla(){
    idColumn.setCellValueFactory(new PropertyValueFactory<>("Id"));idColumn.setResizable(false);
    nombreColumn.setCellValueFactory(new PropertyValueFactory<>("Nombre"));nombreColumn.setResizable(false);
    apellidosColumn.setCellValueFactory(new PropertyValueFactory<>("Apellidos"));apellidosColumn.setResizable(false);
    nombUsuColumn.setCellValueFactory(new PropertyValueFactory<>("nomb_usu"));nombUsuColumn.setResizable(false);
    fechaNacimientoColumn.setCellValueFactory(new PropertyValueFactory<>("fecha_nacimiento"));fechaNacimientoColumn.setResizable(false);
    emailColumn.setCellValueFactory(new PropertyValueFactory<>("Email"));emailColumn.setResizable(false);
    telefonoColumn.setCellValueFactory(new PropertyValueFactory<>("Telefono"));telefonoColumn.setResizable(false);
    tipoColumn.setCellValueFactory(new PropertyValueFactory<>("Tipo"));tipoColumn.setResizable(false);

    // Configurar la tabla y las columnas como editables
    tbUsuarios.setEditable(true);
    fechaNacimientoColumn.setEditable(false);tipoColumn.setEditable(false);

    // Configurar CellFactory para permitir edición
    nombreColumn.setCellFactory(TextFieldTableCell.forTableColumn());
    apellidosColumn.setCellFactory(TextFieldTableCell.forTableColumn());
    nombUsuColumn.setCellFactory(TextFieldTableCell.forTableColumn());
    emailColumn.setCellFactory(TextFieldTableCell.forTableColumn());
    telefonoColumn.setCellFactory(TextFieldTableCell.forTableColumn());

    // Agregar manejadores de commit para actualizar el modelo de datos
    nombreColumn.setOnEditCommit(event -> {User user = event.getRowValue();user.setNombre(event.getNewValue()); Actualizar(user)});
    apellidosColumn.setOnEditCommit(event -> {User user = event.getRowValue();user.setApellidos(event.getNewValue()); Actualizar(user)};
    nombUsuColumn.setOnEditCommit(event -> {User user = event.getRowValue();user.setNomb_usu(event.getNewValue()); Actualizar(user)};
    emailColumn.setOnEditCommit(event -> {User user = event.getRowValue();user.setEmail(event.getNewValue()); Actualizar(user)});
    telefonoColumn.setOnEditCommit(event -> {User user = event.getRowValue();user.setTelefono(event.getNewValue()); Actualizar(user)}
}
```

El evento `btInsertar` abre una ventana emergente para proceder a la inserción de un nuevo usuario.

```
@FXML
public void btInsertar() {
    try{
        FXMLLoader loader = new FXMLLoader(getClass().getResource("name: /es/localhost/anunciaya/administrador/insert_usuario.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        InsertUserController insertUserController = loader.getController();
        insertUserController.setUserController(this);
        Stage currentStage = (Stage) tbUsuarios.getScene().getWindow();
        Stage dialogStage = new Stage();
        dialogStage.setTitle("Insertar Usuario");
        dialogStage.setScene(scene);
        dialogStage.initModality(Modality.WINDOW_MODAL);
        dialogStage.initOwner(currentStage);
        dialogStage.showAndWait();
    }catch (Exception e){ System.out.println(e);}
}
```

El evento *btBorrar* elimina el usuario actual seleccionado tanto de la tabla como de la base de datos.

```
@FXML
public void btBorrar(){
    User selectedUser = tbUsuarios.getSelectionModel().getSelectedItem();
    if(selectedUser != null){
        // Alert de confirmación
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmar eliminación");
        alert.setHeaderText("¿Estás seguro de que deseas eliminar el usuario?");
        alert.setContentText("Esta acción no se puede deshacer.");

        // Cargar la hoja de estilo
        String css = Util.class.getResource(name: "/es/localhost/anunciaya/administrador/estilos/alert.css").toExternalForm();
        alert.getDialogPane().getStylesheets().add(css);

        // Aplicar una clase CSS personalizada al DialogPane
        alert.getDialogPane().getStyleClass().add("custom-alert");

        Optional<ButtonType> result = alert.showAndWait();

        // Si se pulso en OK
        if (result.isPresent() && result.get() == ButtonType.OK) {
            try{
                String[]params= {Integer.toString(tbUsuarios.getSelectionModel().getSelectedItem().getId())};
                if(me.borrarUsuario(params)) Util.mostrarDialogo(titulo: "Server Message", textoEncab: "Borrado de Usuario", contenido: "El usuario ha sido borrado con éxito.");
                else Util.mostrarDialogo(titulo: "Server Message", textoEncab: "Borrado de Usuario", contenido: "El borrado no se pudo llevar a cabo.");
                CargarDatosTabla();
            }catch (Exception e){Util.mostrarDialogo(titulo: "App Message", textoEncab: "Borrado de Usuario", contenido: "Error en el Borrado");}
        } else{
            Util.mostrarDialogo(titulo: "MENSAJE_INFO", textoEncab: "Eliminación cancelada", contenido: "La eliminación del usuario ha sido cancelada.");
        }
    } else {
        Util.mostrarDialogo(titulo: "MENSAJE_WARNING", textoEncab: "No se seleccionó ningún usuario", contenido: "Por favor, selecciona un usuario para borrar.");
    }
}
```

El método *Actualizar* se encarga de actualizar los datos del usuario al editarlo de la tabla.

```
public void Actualizar(User usuario){
    try{
        String[]params = {Integer.toString(usuario.getId()),usuario.getNombre(),usuario.getApellidos(),usuario.getTelefono(),usuario.getEmail()};
        if(!me.actualizarUsuario(params))Util.mostrarDialogo(titulo: "App Message", textoEncab: "Actualizado de Usuario", contenido: "Error en el Actualizado.");
        CargarDatosTabla();
    }catch (Exception e){Util.mostrarDialogo(titulo: "App Message", textoEncab: "Actualizado de Usuario", contenido: "Error en el Actualizado");}
}
```

AnuncioController

Esta clase Java gestiona los anuncios registrados en la base de datos.

Propiedades que conforman la clase AnuncioController.

```
private ObservableList<Anuncio> anunciosList;
private List<Image> images = new ArrayList<>();

@FXML private TableView<Anuncio> tbAnuncios;
@FXML private TableColumn<Anuncio, Integer> tcIdAnuncio;
@FXML private TableColumn<Anuncio, String> tcNombUsu;
@FXML private TableColumn<Anuncio, String> tcCategoria;
@FXML private TableColumn<Anuncio, String> tcTitulo;
@FXML private TableColumn<Anuncio, String> tcDescripcion;
@FXML private TableColumn<Anuncio, String> tcEstado;
@FXML private TableColumn<Anuncio, String> tcUbicacion;
@FXML private TableColumn<Anuncio, Double> tcPrecio;
@FXML private TableColumn<Anuncio, String> tcFotos;
@FXML private TableColumn<Anuncio, String> tcFechPubl;
```

El siguiente método sobrescrito *initialize* se ejecuta al iniciar la escena del anuncio.

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    PrepararTabla();
    CargarDatosTabla();
}
```

El método *CargarDatosTabla* asigna los datos de los anuncios de la base de datos obtenidos con el método *getAllAnuncios* y los asigna a la tabla.

```
private void CargarDatosTabla() {
    Metodos me = new Metodos();
    anunciosList = me.getAllAnuncios(new String[]{""});
    tbAnuncios.setItems(anunciosList);
}
```

El método *PrepararTabla* configura las columnas de la tabla para asignar los datos correspondientes a cada campo de la base de datos.

```
private void PrepararTabla() {
    tcIdAnuncio.setCellValueFactory(new PropertyValueFactory<>( s: "id"));
    tcNombUsu.setCellValueFactory(new PropertyValueFactory<>( s: "nombUsu"));
    tcCategoria.setCellValueFactory(new PropertyValueFactory<>( s: "categoria"));
    tcTitulo.setCellValueFactory(new PropertyValueFactory<>( s: "titulo"));
    tcDescripcion.setCellValueFactory(new PropertyValueFactory<>( s: "descripcion"));
    tcEstado.setCellValueFactory(new PropertyValueFactory<>( s: "estado"));
    tcUbicacion.setCellValueFactory(new PropertyValueFactory<>( s: "ubicacion"));
    tcPrecio.setCellValueFactory(new PropertyValueFactory<>( s: "precio"));
    tcFotos.setCellValueFactory(new PropertyValueFactory<>( s: "fotos"));
    tcFechPubl.setCellValueFactory(new PropertyValueFactory<>( s: "fechPubl"));
}
```

El evento *onVerRegClick* muestra una ventana emergente donde se mostrarán las imágenes asociadas al anuncio seleccionado en la tabla.

```
@FXML
private void onVerRegClick() {
    Anuncio selectedAnuncio = tbAnuncios.getSelectionModel().getSelectedItem();
    if (selectedAnuncio != null) {
        List<String> urls = obtenerUrlsImagenes(selectedAnuncio.getFotos());

        // Cargar imágenes desde las URLs
        images.clear();
        for (String url : urls) {
            images.add(new Image(url));
        }
        showImageCarouselDialog();
    } else {
        Util.mostrarDialogo( titulo: "MENSAJE_WARNING", textoEncab: "No se seleccionó ningún anuncio", contenido: "Por favor, selecciona un anuncio");
    }
}
```

El evento `onBorrarRegClick` elimina el anuncio seleccionado en la tabla.

```

@FXML
private void onBorrarRegClick() {
    Metodos m = new Metodos();
    Anuncio selectedAnuncio = tbAnuncios.getSelectionModel().getSelectedItem();

    if (selectedAnuncio != null) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmar eliminación");
        alert.setHeaderText("Estás seguro de que deseas eliminar el anuncio?");
        alert.setContentText("Esta acción no se puede deshacer.");

        // Cargar la hoja de estilo
        String css = Util.class.getResource(name: "/es/localhost/anunciaya/administrador/estilos/alert.css").toExternalForm();
        alert.getDialogPane().getStylesheets().add(css);

        // Aplicar una clase CSS personalizada al DialogPane
        alert.getDialogPane().getStyleClass().add("custom-alert");

        Optional<ButtonType> result = alert.showAndWait();
        if (result.isPresent() && result.get() == ButtonType.OK) {
            if (m.deleteAnuncio(new String[]{String.valueOf(selectedAnuncio.getId())})) {
                tbAnuncios.getItems().remove(selectedAnuncio);
                tbAnuncios.refresh();
                Util.mostrarDialogo(titulo: "MENSAJE_INFO", textoEncab: "Anuncio eliminado", contenido: "El anuncio ha sido eliminado correctamente");
            } else {
                Util.mostrarDialogo(titulo: "MENSAJE_ERROR", textoEncab: "No se pudo eliminar el anuncio", contenido: "El anuncio no pudo ser eliminado");
            }
        } else {
            Util.mostrarDialogo(titulo: "MENSAJE_INFO", textoEncab: "Eliminación cancelada", contenido: "La eliminación del anuncio ha sido cancelada");
        }
    } else {
        Util.mostrarDialogo(titulo: "MENSAJE_WARNING", textoEncab: "No se seleccionó ningún anuncio", contenido: "Por favor, selecciona un anuncio");
    }
}

```

El método `obtenerUrlsImagenes` parsea y devuelve una lista de string con todas las urls de las imágenes del anuncio pasadas como parámetro.

```

private List<String> obtenerUrlsImagenes(String cadenaUrls) {
    List<String> urls = new ArrayList<>();
    if (cadenaUrls != null && !cadenaUrls.isEmpty()) {
        String[] partes = cadenaUrls.split(regex: ";");
        for (String parte : partes) {
            urls.add(parte.trim());
        }
    }
    return urls;
}

```

El método `showImageCarouselDialog` ejecuta la clase `CarouselController` y muestra las imágenes asociadas a un anuncio.

```
private void showImageCarouselDialog() {
    try {
        Anuncio selectedAnuncio = tbAnuncios.getSelectionModel().getSelectedItem();
        // Cargar el archivo FXML desde los recursos
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/es/localhost/anunciaya/administrador/imagenes_anuncio.fxml"));
        Parent root = loader.load();

        // Obtener el controlador del FXML y configurar las imágenes
        CarouselController controller = loader.getController();
        controller.setImages(images);

        // Crear y configurar el Stage para el diálogo modal
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setTitle("Imágenes Anuncio " + selectedAnuncio.getId());

        DialogPane dialogPane = new DialogPane();
        dialogPane.setContent(root);
        dialogPane.setPrefWidth(420);
        dialogPane.setPrefHeight(387.2);
        stage.setScene(new Scene(dialogPane));
        stage.setResizable(false);

        // Establecer la ventana principal como propietario de la ventana emergente
        Stage primaryStage = (Stage) tbAnuncios.getScene().getWindow(); // Reemplaza 'tbAnuncios' con cualquier nodo de tu ventana principal
        stage.initOwner(primaryStage);

        // Mostrar la ventana emergente y esperar hasta que se cierre
        stage.showAndWait();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

CarouselController

Esta clase Java gestiona los eventos que se producen en la ventana emergente de las imágenes de los anuncios.

Propiedades que conforman la clase `CarouselController`.

```
@FXML private ImageView imageView;
private List<Image> images;
private int currentIndex;
```

El método `setImages` asigna una lista de imágenes al componente de vista de imágenes.

```
public void setImages(List<Image> images) {
    this.images = images;
    currentIndex = 0;
    updateImageView();
}
```

El método `showPreviousImage` muestra la imagen anterior en la lista de imágenes, si hay una disponible.

```
@FXML
private void showPreviousImage() {
    if (currentIndex > 0) {
        currentIndex--;
        updateImageView();
    }
}
```

El método `showNextImage` muestra la siguiente imagen en la lista de imágenes, si hay una

disponible

```
@FXML
private void showNextImage() {
    if (currentIndex < images.size() - 1) {
        currentIndex++;
        updateImageView();
    }
}
```

El método *updateImageView* actualiza el componente ImageView con la imagen correspondiente al índice actual

```
private void updateImageView() {
    if (images != null && !images.isEmpty()) {
        imageView.setImage(images.get(currentIndex));
    }
}
```

CategoríaController

Esta clase Java gestiona las categorías registradas en la base de datos.

Propiedades que conforman la clase CategoríaController.

```
private ObservableList<Categoria> categoriasList;
@FXML private TableView<Categoria> tbCategorias;
@FXML private TableColumn<Categoria, Integer> tcIdCategoria;
@FXML private TableColumn<Categoria, String> tcDescrip;
```

El siguiente método sobrescrito *initialize* se ejecuta al iniciar la escena login.

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    PrepararTabla();
    CargarDatosTabla();
}
```

El método *CargarDatosTabla* asigna los datos de las categorías de la base de datos obtenidos con el método *getAllCategorias* y los asigna a la tabla.

```
public void CargarDatosTabla(){
    Metodos me = new Metodos();

    categoriasList = me.getAllCategorias(new String[]{""});
    tbCategorias.setItems(categoriasList);
}
```

El método *PrepararTabla* configura las columnas de la tabla para asignar los datos correspondientes a cada campo de la base de datos. Además, establece eventos en las columnas de la tabla para permitir la edición del campo al hacer doble clic.

```
private void PrepararTabla(){
    Metodos me = new Metodos();
    tcIdCategoria.setCellValueFactory(new PropertyValueFactory<>("id")); tcIdCategoria.setResizable(false);
    tcDescrip.setCellValueFactory(new PropertyValueFactory<>("descripcion")); tcDescrip.setResizable(false);

    // Hacer la tabla y la columna de descripción editables
    tbCategorias.setEditable(true);
    tcDescrip.setCellFactory(TextFieldTableCell.forTableColumn(new DefaultStringConverter()));

    // Manejar el evento de edición de la celda
    tcDescrip.setOnEditCommit(event -> {
        Categoría categoria = event.getRowValue();
        String nuevaDescripcion = event.getNewValue();

        if (categoria != null) {
            categoria.setDescripcion(nuevaDescripcion);

            // Actualizar la base de datos
            if (me.updateCategoria(new String[]{String.valueOf(categoria.getId()), nuevaDescripcion})) {
                // Actualizar la tabla si la actualización en la base de datos fue exitosa
                tbCategorias.refresh();
                Util.mostrarDialogo(título: "MENSAJE_INFO", textoEncab: "Categoria actualizada", contenido: "Se ha actualizado la descripción de la categoría");
            } else {
                // Manejar el error (puedes mostrar un diálogo de error aquí)
                Util.mostrarDialogo(título: "MENSAJE_ERROR", textoEncab: "No se pudo actualizar la categoría", contenido: "La descripción de la categoría no se ha actualizado");
            }
        }
    });
}
```

El método *onAddCatClick* abre una ventana emergente para proceder a la inserción de una nueva categoría.

```
@FXML private void onAddCatClick(){
    try {
        // Cargar el archivo FXML desde los recursos
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/es/localhost/anunciaya/administrador/insert_categoria.fxml"));
        Parent root = loader.load();

        // Obtener el controlador del FXML y configurar las imágenes
        InsertCategoController controller = loader.getController();
        controller.setCategoController(this);

        // Crear y configurar el Stage para el diálogo modal
        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        stage.setTitle("Insertar Categoría");
        stage.setScene(new Scene(root));
        stage.setResizable(false);

        // Establecer la ventana principal como propietario de la ventana emergente
        Stage primaryStage = (Stage) tbCategorias.getScene().getWindow();
        stage.initOwner(primaryStage);

        // Mostrar la ventana emergente y esperar hasta que se cierre
        stage.showAndWait();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

El método *onBorrarCatClick* elimina la categoría seleccionada en la tabla.

```

@FXML private void onBorrarCatClick() {
    Metodos m = new Metodos();
    Categoria selectedCategoria = tbCategorias.getSelectionModel().getSelectedItem();

    if (selectedCategoria != null) {
        // Alert de confirmación
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmar eliminación");
        alert.setHeaderText("¿Estás seguro de que deseas eliminar la categoría?");
        alert.setContentText("Esta acción no se puede deshacer.");

        // Cargar la hoja de estilo
        String css = Util.class.getResource(name: "/es/localhost/anunciaya/administrador/estilos/alert.css").toExternalForm();
        alert.getDialogPane().getStylesheets().add(css);

        // Aplicar una clase CSS personalizada al DialogPane
        alert.getDialogPane().getStyleClass().add("custom-alert");

        Optional<ButtonType> result = alert.showAndWait();

        // Si se pulsó en OK
        if (result.isPresent() && result.get() == ButtonType.OK) {
            if(!m.tieneAnunciosCategoria(new String[]{String.valueOf(selectedCategoria.getId())})){
                if (m.deleteCategoria(new String[]{String.valueOf(selectedCategoria.getId())})) {
                    // Eliminar la categoría de la lista observable
                    tbCategorias.getItems().remove(selectedCategoria);
                    // Refrescar el TableView para reflejar los cambios
                    tbCategorias.refresh();
                    Util.mostrarDialogo(título: "Mensaje_INFO", textoEncab: "Categoría eliminada", contenido: "La categoría ha sido eliminada");
                } else {
                    // Manejar el caso en que la categoría no pudo ser eliminado
                    Util.mostrarDialogo(título: "Mensaje_WARNING", textoEncab: "No se pudo eliminar la categoría", contenido: "La categoría no se pudo eliminar");
                }
            } else{
                Util.mostrarDialogo(título: "Operación no permitida", textoEncab: "Eliminación de categoría no permitida", contenido: "La categoría ya tiene anuncios asociados");
            }
        } else {
            // Se hizo click en Cancel o cerró el diálogo
            Util.mostrarDialogo(título: "Mensaje_INFO", textoEncab: "Eliminación cancelada", contenido: "La eliminación de la categoría ha sido cancelada");
        }
    } else {
        Util.mostrarDialogo(título: "Mensaje_WARNING", textoEncab: "No se seleccionó ninguna categoría", contenido: "Por favor, seleccione una categoría");
    }
}

```

InsertCategoController

Esta clase Java gestiona la inserción de nuevas categorías en la base de datos.

Propiedades que conforman la clase InsertCategoController.

```

private CategoriaController categoriaController;
@FXML private TextField tfNombrCategoria;
@FXML private Button btCancelar;

```

El método `setCategoController` establece la referencia del controlador padre.

```

public void setCategoController(CategoriaController categoController) {
    this.categoriaController = categoController;
}

```

El evento *onAceptarClick* se activará al presionar el botón de insertar categoría en el formulario de creación de nuevas categorías.

```
@FXML private void onAceptarClick() {
    Metodos m = new Metodos();
    if (!tfNomCategoría.getText().isEmpty()) {
        if(m.insertCategoria(new String[]{tfNomCategoría.getText()})){
            Util.mostrarDialogo( titulo: "MENSAJE_INFO", textoEncab: "Categoria añadida", contenido: "Se ha registrado la nueva categoria con exito.", categoriaController.CargarDatosTabla());
            Cerrar();
        } else{
            Util.mostrarDialogo( titulo: "MENSAJE_ERROR", textoEncab: "No se pudo añadir la categoria", contenido: "Ha surgido un error con la insercion de la categoria");
        }
    } else {
        Util.mostrarDialogo( titulo: "MENSAJE_WARNING", textoEncab: "Hay campos vacíos", contenido: "Por favor comprueba que todos los campos estan llenos");
    }
}
```

El evento *btCancelar* cierra la ventana emergente del formulario en el método *Cerrar*.

```
public void btCancelar(){Cerrar();}
private void Cerrar(){
    categoriaController.CargarDatosTabla();
    Stage stage = (Stage) btCancelar.getScene().getWindow();
    stage.close();
}
```

InsertUserController

Esta clase Java gestiona la inserción de nuevos usuarios en la base de datos.

Propiedades que conforman la clase *InsertUserController*.

```
@FXML private Button btCancelar;
@FXML TextField Nombre,Apellidos,NombreUsuario,Email,Telefono;
@FXML private DatePicker dateFechaNacimiento;
@FXML private SplitMenuItem sliTipo;
@FXML private PasswordField tbContrasena;
private Metodos m ;
private UserController userController;
```

El método sobrescrito *initialize* se ejecuta al iniciar la escena *insertusercontroller*.

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) { m = new Metodos(); }
```

El evento *btAceptar* se activará al presionar el botón de insertar usuario en el formulario de creación de nuevos usuarios.

```
public void btAceptar(){
try{
    if(!Nombre.getText().isEmpty() && !Apellidos.getText().isEmpty() && !NombreUsuario.getText().isEmpty()){
        if(!Email.getText().isEmpty() && !Telefono.getText().isEmpty() && !tbContrasena.getText().isEmpty()){
            try{
                Integer.parseInt(Telefono.getText().toString());
                String fechaNacimiento = dateFechaNacimiento.getValue().format(DateTimeFormatter.ofPattern("yyyy-MM-dd"));
                String []params = {Nombre.getText(),Apellidos.getText(),NombreUsuario.getText(),tbContrasena.getText(),fechaNacimiento};
                if(m.insertarUsuario(params)){
                    Util.mostrarDialogo( titulo: "Mensaje del Server", textoEncab: "Insercción de Usuario", contenido: "Usuario Insertado con exito.", Cerrar());
                }else Util.mostrarDialogo( titulo: "Mensaje del Server", textoEncab: "Insercción de Usuario", contenido: "El Usuario no pudo ser insertado.");
            }catch (Exception e){Util.mostrarDialogo( titulo: "Telefono Erroneo", textoEncab: "Número de Tf Erroneo", contenido: "Número de telefono incorrecto");}
            }else Util.mostrarDialogo( titulo: "Campos Erróneos", textoEncab: "Campos Incompletos", contenido: "Rellena los campos correctamente");
        }else Util.mostrarDialogo( titulo: "Campos Erróneos", textoEncab: "Campos Incompletos", contenido: "Rellena los campos correctamente");
    }catch (Exception e) {Util.mostrarDialogo( titulo: "Server_Error", textoEncab: "Autenticación Fallida.", contenido: "Ocurrio un error en el servidor");}
}
```

El evento *btCancelar* cierra la ventana emergente del formulario en el método *Cerrar*.

```

public void btCancelar(){Cerrar();}
private void Cerrar(){
    userController.CargarDatosTabla();
    Stage stage = (Stage) btCancelar.getScene().getWindow();
    stage.close();
}

```

El método *EstablecerTextSelector* establece por defecto los tipos de usuarios que se pueden crear con perfil administrador o sin permisos.

```

public void EstablecerTextoSelector(javafx.event.ActionEvent event){
    MenuItem menuItem = (MenuItem) event.getSource();
    slTipo.setText(menuItem.getText());
}

```

Pedido Controller

La clase *PedidoController* se encarga de Controlar los pedidos de la App, aqui podemos ver cuales son sus atributos

```

public class PedidoController implements Initializable {
    private ObservableList<Pedido> pedidosList;  2 usages
    @FXML
    private TableView<Pedido> tbPedidos;
    @FXML private TableColumn<Pedido, Integer> tcIdPedido;
    @FXML private TableColumn<Pedido, String> tcNombUsu;
    @FXML private TableColumn<Pedido, Integer> tcIdAnuncio;
    @FXML private TableColumn<Pedido, String> tcTitulo;
    @FXML private TableColumn<Pedido, String> tcDireccion;
    @FXML private TableColumn<Pedido, String> tcCiudad;
    @FXML private TableColumn<Pedido, String> tcCp;
}

```

Luego tenemos el método initialize que es el primer método que se ejecuta al abrir la app

```

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    PrepararTabla();
    CargarDatosTabla();
}

```

El método CargarDatosTabla se encarga de llamar al método necesario de la clase Metodos para retornar todos los pedidos de la base de datos y luego lo asigna a la tabla.

```
private void CargarDatosTabla(){ 1 usage
    Metodos me = new Metodos();

    pedidosList = me.getAllPedidos(new String[]{" "});
    tbPedidos.setItems(pedidosList);
}
```

El método Preparar Tabla se encarga de preparar cada tipo de dato que va a contener cada encabezado y si se puede modificar o no.

```
private void PrepararTabla(){ 1 usage
    tcIdPedido.setCellValueFactory(new PropertyValueFactory<>( s: "id")); tcIdPedido.setResizable(false);
    tcNomUsu.setCellValueFactory(new PropertyValueFactory<>( s: "nomUsu")); tcNomUsu.setResizable(false);
    tcIdAnuncio.setCellValueFactory(new PropertyValueFactory<>( s: "idAnuncio")); tcIdAnuncio.setResizable(false);
    tcTitulo.setCellValueFactory(new PropertyValueFactory<>( s: "titulo")); tcTitulo.setResizable(false);
    tcDireccion.setCellValueFactory(new PropertyValueFactory<>( s: "direccion")); tcDireccion.setResizable(false);
    tcCiudad.setCellValueFactory(new PropertyValueFactory<>( s: "ciudad")); tcCiudad.setResizable(false);
    tcCp.setCellValueFactory(new PropertyValueFactory<>( s: "cp")); tcCp.setResizable(false);
}
```

Home

La clase Home es el encargado de navegar sobre las vistas y cargar nuevas vistas sobre la vista de Home , de manera que tendremos el menú operativo siempre de Fondo

Como podremos ver en la siguiente Imagen , vemos que el cargarGraphics llama al método de cargarFXML con el nombre de graphics.fxml abriendo así esta ventana sobre la vista actual.

Los métodos de CargarUsuario, CargarAnuncio, CargarCategoria, CargarPedidos, CargarLogin hacen todos las mismas cosas, cargan vistas fxml en la ventana de Home

```
public class Home implements Initializable {
    @FXML private AnchorPane opacityPane,drawerPane,panelFondo;
    @FXML private ImageView drawerImage;
    private int Abierto = 0 ; 4 usages

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        MenuLateral();
        CargaGraphics();
    }
    public void CargarGraphics(){ 2 usages
        cargarFXML( NombreView: "graphics.fxml");clickOpacityPannel();
    }
    public void CargarUsuarios(){ 1 usage
        cargarFXML( NombreView: "usuarios.fxml");clickOpacityPannel();
    }
    public void CargarAnuncios(){ 1 usage
        cargarFXML( NombreView: "anuncios.fxml");clickOpacityPannel();
    }
    public void CargarCategorias(){ 1 usage
        cargarFXML( NombreView: "categorias.fxml");clickOpacityPannel();
    }
    public void CargarPedidos(){ 1 usage
        cargarFXML( NombreView: "pedidos.fxml");clickOpacityPannel();
    }
    public void CargarLogin(){ 1 usage
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Terminar Sesión");
        alert.setHeaderText("Estás apunto de cerrar sesión");
        alert.setContentText("Si confirmas el cerrar sesión deberás de volver a iniciar la sesión.");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.isPresent() && result.get() == ButtonType.OK) {
            Util.mostrarDialogo( titulo: "MENSAJE_INFO", textoEncab: "Sesión Cerrada", contenido: "Has cerrado sesión exitosamente.", Alert.AlertType.INFORMATION);
            cargarEscenaPrincipal( NombreView: "login.fxml");
        }else{
            Util.mostrarDialogo( titulo: "MENSAJE_INFO", textoEncab: "Operación cancelada", contenido: "La acción de cerrar sesión ha sido cancelada.", Alert.AlertType.INFORMATION);
        }
    }
}
```

El método cargarFXML es la encargada de Cargar vistas fxml en la vista actual(Home)

```
private void cargarFXML(String NombreView){ 5 usages
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource(NombreView));
        AnchorPane content = loader.load();
        panelFondo.getChildren().setAll(content);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

El método de *cargarEscenaPrincipal* se encarga de cargar la escenaPrincipal, o lo que es lo mismo , cargar una scene reemplazando la scene anterior , como bien puede ser el tema de volver al login, impidiendo que se vea la scene de home.

```
private void cargarEscenaPrincipal(String NombreView) { 1 usage
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource(NombreView));
        Pane content = loader.load(); // Usar Pane en lugar de AnchorPane para mayor flexibilidad

        // Obtener el stage principal
        Stage stage = (Stage) panelFondo.getScene().getWindow();

        // Crear una nueva escena con el contenido cargado
        Scene newScene = new Scene(content);

        // Establecer la nueva escena en el stage principal
        stage.setScene(newScene);

        // Mostrar el stage principal con la nueva escena
        stage.show();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassCastException e) {
        System.err.println("Error al convertir el layout: " + e.getMessage());
    }
}
```

La clase *Menu Lateral*, son métodos necesarios para que funcione el Menu Lateral, es por eso que este método se encarga de preparar los datos necesarios para el funcionamiento del Menu lateral.

```
private void MenuLateral(){ 1 usage
    opacityPane.setVisible(false);

    FadeTransition fadeTransition=new FadeTransition(Duration.seconds( v: 0.5 ),opacityPane);
    fadeTransition.setFromValue(1);
    fadeTransition.setToValue(0);
    fadeTransition.play();

    TranslateTransition translateTransition=new TranslateTransition(Duration.seconds( v: 0.5 ),drawerPane);
    translateTransition.setByX(-604.5);
    translateTransition.play();

    opacityPane.setOnMouseClicked(event -> clickOpacityPannel());
    drawerImage.setOnMouseClicked(event -> CerrarMenuLateral());

}

private void clickOpacityPannel(){ 6 usages
```

El método de *clickOpacityPannel* se encarga de capturar cuando hacemos click fuera del menu para controlar que hemos hecho click fuera del Menu , para asi cerrar el menu lateral

```
private void clickOpacityPannel(){ 6 usages
    if(Abierto != 0){
        FadeTransition fadeTransition1=new FadeTransition(Duration.seconds( v: 0.5 ),opacityPane);
        fadeTransition1.setFromValue(0.15);
        fadeTransition1.setToValue(0);
        fadeTransition1.play();

        fadeTransition1.setOnFinished(event1 -> {
            opacityPane.setVisible(false);
        });

        TranslateTransition translateTransition1=new TranslateTransition(Duration.seconds( v: 0.5 ),drawerPane);
        translateTransition1.setByX(-604.5);
        translateTransition1.play();
        Abierto = 0;
    }
}
```

El método de *CerrarMenuLateral* se encarga de cerrar el menu Lateral en caso de que este abierto

```
private void CerrarMenuLateral(){ 1 usage
    if(Abierto != 1){
        opacityPane.setVisible(true);

        FadeTransition fadeTransition1=new FadeTransition(Duration.seconds( v: 0.5),opacityPane);
        fadeTransition1.setFromValue(0);
        fadeTransition1.setToValue(0.15);
        fadeTransition1.play();

        TranslateTransition translateTransition1=new TranslateTransition(Duration.seconds( v: 0.5),drawerPane);
        translateTransition1.setByX(+604.5);
        translateTransition1.play();
        Abierto = 1;
    }
}
```

LoginController

Es la clase encargada de controlar la clase de login , estos son los atributos de la clase controladora

```
public class LoginController implements Initializable {
    @FXML private TextField tfEmail;
    @FXML private PasswordField pfPasswd;
    private Metodos m; 3 usages
```

El método *onLoginCLick* , sirve para capturar la pulsacion del Boton de login , ya que al hacer click en este necesitamos comprobar si el usuario y la contraseña han sido insertados , y en este caso de que sea así , preguntar al server si las credenciales son válidas, de ser asi lanzará el Home.

```
@FXML
private void onLoginClick(){
    if(!tfEmail.getText().isEmpty() && !pfPasswd.getText().isEmpty()){
        String[]params = {tfEmail.getText(),pfPasswd.getText()};
        if(m.AuthAdmin(params)){
            DataStore.getInstance().setData(m.getIdUserByEmail(tfEmail.getText()));
            Util.mostrarDialogo( titulo: "MENSAJE_INFO", textoEncab: "Autenticación Exitosa.",
                contenido: """
                    ¡Bienvenido de vuelta al panel de administración!
                    Ahora tienes acceso completo a todas las funciones
                    administrativas."",
                Alert.AlertType.INFORMATION);

            Util.loadNewScene( nScene: "home.fxml", tfEmail, title: "Home | AnunciaYa");
        } else{
            Util.mostrarDialogo( titulo: "MENSAJE_ERROR", textoEncab: "Autenticación Fallida.",
                contenido: "Error al autenticar. Verifica que el correo electrónico y la contraseña sean correctos. ",
                Alert.AlertType.ERROR);
        }
    } else{
        Util.mostrarDialogo( titulo: "MENSAJE_WARNING", textoEncab: "Campos Vacíos",
            contenido: "Por favor, completa todos los campos requeridos para iniciar sesión correctamente. ", Alert.AlertType.WARNING);
    }
}
```

El método Initialize es el primer método que se lanza al abrir la app , es por ello que lo que haremos será crear el objeto métodos para poder tener comunicación con el servidor

```
@Override  
public void initialize(URL url, ResourceBundle resourceBundle) {  
    m = new Metodos();  
}
```

Pedido

La clase Pedido es una clase utilizada para cuando tenemos que ver los pedidos de la base de datos en la app , en esta captura podemos ver los atributos , constructor, gettes y settes.

```
public class Pedido {  
    private int id; 3 usages  
    private String nombUsu; 3 usages  
    private int idAnuncio; 3 usages  
    private String titulo; 3 usages  
    private String direccion; 3 usages  
    private String ciudad; 3 usages  
    private String cp; 3 usages  
  
    public Pedido(int id, String nombUsu, int idAnuncio, String titulo, String direccion, String ciudad, String cp) { 1 usage  
        this.id = id;  
        this.nombUsu = nombUsu;  
        this.idAnuncio = idAnuncio;  
        this.titulo = titulo;  
        this.direccion = direccion;  
        this.ciudad = ciudad;  
        this.cp = cp;  
    }  
  
    public int getId() {return id;}  
    public String getNombUsu() {return nombUsu;} no usages  
    public int getIdAnuncio() {return idAnuncio;} no usages  
    public String getTitulo() {return titulo;} no usages  
    public String getDireccion() {return direccion;} no usages  
    public String getCiudad() {return ciudad;} no usages  
    public String getCp() {return cp;} no usages  
  
    public void setId(int id) {this.id = id;}  
    public void setNombUsu(String nombUsu) {this.nombUsu = nombUsu;} no usages  
    public void setIdAnuncio(int idAnuncio) {this.idAnuncio = idAnuncio;} no usages  
    public void setTitulo(String titulo) {this.titulo = titulo;} no usages  
    public void setDireccion(String direccion) {this.direccion = direccion;} no usages  
    public void setCiudad(String ciudad) {this.ciudad = ciudad;} no usages  
    public void setCp(String cp) {this.cp = cp;} no usages  
}
```

Util

La clase Util contiene métodos que pueden ser utilizados por cualquier clase , es por ello que son útiles o comunes.

Contiene dos métodos:

MostrarDialogo → es un método que se encarga de mostrar mensajes por pantalla

```
public static void mostrarDialogo(String titulo, String textoEncab, String contenido, Alert.AlertType tipoDialogo) { 46 usages
    Alert alert = new Alert(tipoDialogo);
    alert.setTitle(titulo);
    alert.setHeaderText(textoEncab);
    alert.setContentText(contenido);

    // Cargar la hoja de estilo
    String css = Util.class.getResource(name: "/es/localhost/anunciaya/administrador/estilos/alert.css").toExternalForm();
    alert.getDialogPane().getStylesheets().add(css);

    // Aplicar una clase CSS personalizada al DialogPane
    alert.getDialogPane().getStyleClass().add("custom-alert");

    alert.showAndWait();
}
```

LoadNewScene → es un método encargado de abrir escenas nuevas sobreponiendo la escena anterior

```
public static void loadNewScene(String nScene, TextField textField, String title) { 1 usage
    try {
        FXMLLoader loader = new FXMLLoader(Util.class.getResource(nScene));
        Parent root = loader.load();

        // Obtiene la escena actual
        Stage stage = (Stage) textField.getScene().getWindow();
        stage.setTitle(title);

        Scene scene = new Scene(root);

        // Asigna la Stage a la escena
        stage.setScene(scene);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Anuncio

Es una clase usada para el apartado de Mostrar anuncios en la App.

```
public class Anuncio {
    private int id;  3 usages
    private String nombUsu;  3 usages
    private String categoria;  3 usages
    private String titulo;  3 usages
    private String descripcion;  3 usages
    private String estado;  3 usages
    private String ubicacion;  3 usages
    private double precio;  3 usages
    private String fotos;  3 usages
    private String fechPubl;  3 usages
```

Estos son los constructores , getters y setters de la Clase

```
public Anuncio(){}
public Anuncio(int id, String nombUsu, String categoria, String titulo, String descripcion, String estado, String ubicacion,
              double precio, String fotos, String fechPubl) {
    this.id = id;
    this.nombUsu = nombUsu;
    this.categoria = categoria;
    this.titulo = titulo;
    this.descripcion = descripcion;
    this.estado = estado;
    this.ubicacion = ubicacion;
    this.precio = precio;
    this.fotos = fotos;
    this.fechPubl = fechPubl;
}
public int getId() {return id;}
public String getNombUsu() {return nombUsu;}  no usages
public String getCategoría() {return categoria;}  no usages
public String getTitulo() {return titulo;}  no usages
public String getDescripcion() {return descripcion;}  no usages
public String getEstado() {return estado;}  no usages
public String getUbicacion() {return ubicacion;}  no usages
public double getPrecio() {return precio;}  no usages
public String getFotos() {return fotos;}  1 usage
public String getFechPubl() {return fechPubl;}  no usages
public void setId(int id) {this.id = id;}
public void setNombUsu(String nombUsu) {this.nombUsu = nombUsu;}  no usages
public void setCategoría(String categoria) {this.categoría = categoria;}  no usages
public void setTitulo(String titulo) {this.titulo = titulo;}  no usages
public void setDescripcion(String descripcion) { this.descripcion = descripcion;}  no usages
public void setEstado(String estado) {this.estado = estado;}  no usages
public void setUbicacion(String ubicacion) {this.ubicacion = ubicacion;}  no usages
public void setPrecio(double precio) {this.precio = precio;}  no usages
public void setFotos(String fotos) {this.fotos = fotos;}  no usages
public void setFechPubl(String fechPubl) {this.fechPubl = fechPubl;}  no usages
```

Categoría

Es una clase usada para mostrar las Categorías de la base de datos.

Estas son sus propiedades ,constructores ,gettes y settes.

```
public class Categoria {
    private int id; 3 usages
    private String descripcion; 3 usages

    public Categoria(int id, String descripcion) { 2 usages
        this.id = id;
        this.descripcion = descripcion;
    }

    public int getId() {return id;}
    public String getDescripcion() {return descripcion;} no usages

    public void setId(int id) {this.id = id;}
    public void setDescripcion(String descripcion) {this.descripcion = descripcion;} 1 usage
}
```

User

La clase User es una clase usada para mostrar los usuarios de la App.

Aquí podemos ver sus atributos , constructores, setter y getter.

```
public class User {
    private int Id; 3 usages
    private String Nombre, Apellidos, nomb_usu, fecha_nacimiento, email, telefono, tipo; 3 usages

    public User(int id, String nombre, String apellidos, String nomb_usu, String fecha_nacimiento, String email, String telefono, String tipo) {
        Id = id;
        Nombre = nombre;
        Apellidos = apellidos;
        this.nomb_usu = nomb_usu;
        this.fecha_nacimiento = fecha_nacimiento;
        this.email = email;
        this.telefono = telefono;
        this.tipo = tipo;
    }

    public int getId() { return Id; }
    public void setId(int id) { Id = id; }
    public String getNombre() { return Nombre; }
    public void setNombre(String nombre) { Nombre = nombre; }
    public String getApellidos() { return Apellidos; }
    public void setApellidos(String apellidos) { Apellidos = apellidos; }
    public String getNomb_usu() { return nomb_usu; }
    public void setNomb_usu(String nomb_usu) { this.nomb_usu = nomb_usu; }
    public String getFecha_nacimiento() { return fecha_nacimiento; }
    public void setFecha_nacimiento(String fecha_nacimiento) { this.fecha_nacimiento = fecha_nacimiento; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getTelefono() { return telefono; }
    public void setTelefono(String telefono) { this.telefono = telefono; }
    public String getTipo() { return tipo; }
    public void setTipo(String tipo) { this.tipo = tipo; }
}
```

DataStore

La clase DataStore es usada para almacenar información temporal (En caché) para ser usada desde cualquier parte de la app , como bien puede ser , almacenar el id del usuario que se ha logueado

```
public class DataStore {
    private static DataStore instance = new DataStore(); 1 usage
    private String data; 2 usages

    private DataStore() {} 1 usage

    public static DataStore getInstance() { return instance; }

    public String getData() { return data; }

    public void setData(String data) { this.data = data; }
}
```

Server Communication

Es una clase usada para establecer comunicación con el servidor y dicha respuesta retornarla al servidor que ha hecho la petición.

El método comunicación realiza la petición por json al servidor

```
public class ServerCommunication {
    public static String comunicacion(String urlServer, String clase, String metodo, String[] params){ 14 usages
        try {
            URL url = new URL(spec: urlServer + "/index.php");

            // Abrir conexión
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setDoOutput(true);

            String data = "{\"class\":\"" + clase + "\", \"method\":\"" + metodo + "\", \"params\":[" + getParamsParsed(params) + "]";

            // Escribir datos en el cuerpo de la solicitud
            OutputStream os = conn.getOutputStream();
            os.write(data.getBytes());
            os.flush();

            // Leer respuesta del servidor
            BufferedReader br = new BufferedReader(new InputStreamReader((conn.getInputStream())));
            StringBuilder response = new StringBuilder();
            String output;
            while ((output = br.readLine()) != null) {
                response.append(output);
            }
            conn.disconnect();
            return response.toString();
        } catch (Exception e) {return null;}
    }
}
```

El metodo `getParamsParsed` se encarga de construir los parametros en petición json.

```
private static String getParamsParsed(String[] params) { 1 usage
    String retornador = "[";
    for (int i = 0; i < params.length; i++) {
        if (i != params.length - 1) retornador += "\"" + params[i] + "\"+" ;
        else retornador += "\"" + params[i] + "\"]";
    }
    return retornador;
}
```

Métodos

La clase `Métodos` es la encargada de llamar al server communication , pero llama al Server Communication con métodos que se encarga de parsear o interpretar la respuesta del servidor recibida como JSON.

```
public class Metodos {
    private int IdUser = 0; 1 usage
    private String urlServer = "https://40945016.servicio-online.net/sv-php/"; 18 usages
```

Esto es un ejemplo de un método , como bien puede ser el método de actualizarUsuario, vemos que los parámetros pasados a `ServerCommunication` son valores como la url del server la clase y el método junto con los parámetros,además podemos ver como el dato que retornamos es “data” devuelto por el json

```
public Boolean actualizarUsuario(String[]params){ 1 usage
    try{
        String resultadoServer = ServerCommunication.comunicacion(urlServer, clase:"Usuario", metodo:"updateUsuario",params);
        JSONObject jsonObject = new JSONObject(resultadoServer);
        return jsonObject.getBoolean(key:"data");
    }catch (Exception e){
        Util.mostrarDialogo(titulo:"Server_Error", textoEncab:"Actualizacion Fallida.", contenido:"Ocurrio un error en el servidor: "+e,Alert.AlertType.INFORMATION);
        return false;
    }
}
```

Además tenemos métodos un tanto diferentes usados para los gráficos, como el siguiente , que es el encargado de obtener todos los nombres de los ficheros de logs que hay para luego ser analizados

```
public ArrayList<String> obtenerLogs() { 1 usage
    try {
        // Crear una instancia de URL
        URL url = new URL(spec:urlServer + "util/log/");
        URLConnection conn = url.openConnection();
        // Obtener el flujo de entrada para leer la respuesta
        BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));

        ArrayList<String> lista = new ArrayList<>();
        String linea;
        while ((linea = reader.readLine()) != null) {
            // Filtrar las líneas que contienen nombres de archivos
            if (linea.contains("<a href=") && linea.contains(".txt")) {
                lista.add(obtenerHrefs(linea));
            }
        }
        reader.close();
        return lista;
    } catch (IOException e) {...}
}
```

Esta clase anterior usa la siguiente clase usada para obtener de una linea HTML el nombre del

fichero .txt

```
private static String obtenerHrefs(String html) { 1 usage
    Pattern pattern = Pattern.compile(regex: "<a\\s+href=\\\"([^\"]+)\\\"");
    Matcher matcher = pattern.matcher(html);
    while (matcher.find()) return matcher.group(1);
    return null;
}
```

También un método para obtener el Uso total , contando el número de líneas de cada fichero de cada mes

```
public ObservableList<PieChart.Data> obtenerUsoTotal(ArrayList<String>Logs){ 1 usage
try{
    ObservableList<PieChart.Data>piechartData = FXCollections.observableArrayList();
    for(int i = 0; i<Logs.size();i++){
        URL url = new URL(spec: urlServer + "util/log/" + Logs.get(i));
        URLConnection conn = url.openConnection();
        BufferedReader reader = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        int numeroLineas = 0;
        while (reader.readLine() != null) {
            numeroLineas++;
        }
        String[] ficheroParseado = Logs.get(i).split(regex: "\n");
        piechartData.add(new PieChart.Data(s: ficheroParseado[1].substring(0,2)+"/"+ficheroParseado[1].substring(2,6),numeroLineas));
        reader.close();
    }
    return piechartData;
} catch (Exception e){...}
}
```

También tenemos un método para obtener todas las IP diferentes que hay en el fichero y el número de veces que aparece en el fichero para contar el Número de Interacciones en la App y su uso total.

```
public static ArrayList<String> getAllIp(String log) { 1 usage
    // Usar un HashSet para eliminar duplicados automáticamente
    HashSet<String> uniqueIPsSet = new HashSet<>();

    // Patrón para buscar direcciones IP en el registro
    String regex = "\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(log);

    while (matcher.find()) {
        uniqueIPsSet.add(matcher.group());
    }

    // Convertir el HashSet a ArrayList
    return new ArrayList<>(uniqueIPsSet);
}
```

Los demás métodos que contiene esta clase son los siguientes :

```

    }
    public String getIdUserByEmail(String email){...}
    public Boolean AuthAdmin(String[]params){...}
    public ObservableList<User> getAllUsers(String[] args) {...}
    public ObservableList<PieChart.Data> usoDatosIp(String fichero){...}
    public ObservableList<PieChart.Data> obtenerEstadosPeticiones(String fichero){...}

    public ObservableList<Anuncio> getAllAnuncios(String[] args) {...}

    public boolean deleteAnuncio(String[] args) {...}

    public ObservableList<Categoria> getAllCategorias(String[] args) {...}
    public boolean tieneAnunciosCategoria(String[] args){...}
    public boolean insertCategoria(String[] args) {...}
    public boolean updateCategoria(String[] args) {...}
    public boolean deleteCategoria(String[] args) {...}
    public ObservableList<Pedido> getAllPedidos(String[] args) {...}
    public ObservableList<PieChart.Data> obtenerUsoTotal(ArrayList<String>Logs){...}
    public ArrayList<String> obtenerLogs() {...}
    private static int contarPeticiones(String log, String estado) {...}
    private static String obtenerHrefs(String html) {...}
    public static ArrayList<String> getAllIp(String log) {...}
}

```

4.1.2.4 Sentencias SQL

Creación tabla usuario.

```

CREATE TABLE `usuario` (
  `id` int(5) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(20) NOT NULL,
  `apellidos` varchar(50) NOT NULL,
  `nombr_usu` varchar(20) NOT NULL,
  `contras` varchar(255) NOT NULL,
  `fecha_nacimiento` date NOT NULL,
  `email` varchar(50) NOT NULL,
  `telefono` varchar(25) NOT NULL,
  `tipo` varchar(3) NOT NULL DEFAULT 'STD',
  PRIMARY KEY (`id`),
  UNIQUE KEY `UNIQUE_tlf` (`telefono`),
  UNIQUE KEY `UNIQUE_nombr_usu` (`nombr_usu`) USING BTREE,
  UNIQUE KEY `UNIQUE_email` (`email`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;

```

Creación tabla categorías.

```

CREATE TABLE `categoria` (
  `id` int(5) NOT NULL AUTO_INCREMENT,
  `descripcion` varchar(50) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;

```

Creación de tabla anuncios.

```
CREATE TABLE `anuncio` (
  `id` int(5) NOT NULL AUTO_INCREMENT,
  `id_usuario` int(5) NOT NULL,
  `id_categoria` int(5) NOT NULL,
  `titulo` varchar(50) NOT NULL,
  `descripcion` text NOT NULL,
  `estado` varchar(15) NOT NULL,
  `ubicacion` varchar(50) NOT NULL,
  `precio` float NOT NULL,
  `fotos` text NOT NULL,
  `fech_public` date DEFAULT current_timestamp(),
  PRIMARY KEY (`id`),
  KEY `FK_anuncio_categoria` (`id_categoria`),
  KEY `FK_anuncio_usuario` (`id_usuario`),
  CONSTRAINT `FK_anuncio_categoria` FOREIGN KEY (`id_categoria`) REFERENCES `categoria` (`id`),
  CONSTRAINT `FK_anuncio_usuario` FOREIGN KEY (`id_usuario`) REFERENCES `usuario` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;
```

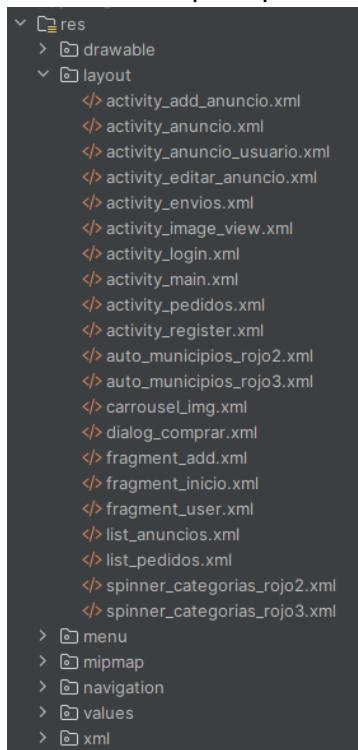
Creación de la tabla pedidos.

```
CREATE TABLE `pedido` (
  `id` int(5) NOT NULL AUTO_INCREMENT,
  `id_comprador` int(5) NOT NULL,
  `id_anuncio` int(5) NOT NULL,
  `fech_pedido` date DEFAULT current_timestamp(),
  `ciudad` varchar(50) DEFAULT NULL,
  `direccion` varchar(50) DEFAULT NULL,
  `cp` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `FK_pedido_usuario` (`id_comprador`),
  KEY `FK_pedido_anuncio` (`id_anuncio`),
  CONSTRAINT `FK_pedido_usuario` FOREIGN KEY (`id_comprador`) REFERENCES `usuario` (`id`),
  CONSTRAINT `FK_pedido_anuncio` FOREIGN KEY (`id_anuncio`) REFERENCES `anuncio` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;
```

4.1.3 - Frontend

4.1.3.1 Cliente Android

Los layouts que conforman el proyecto del cliente android son los siguientes, los cuales se explicarán más detalladamente a continuación indicando los componentes que incluyen y su funcionalidad principal.



activity_login.xml

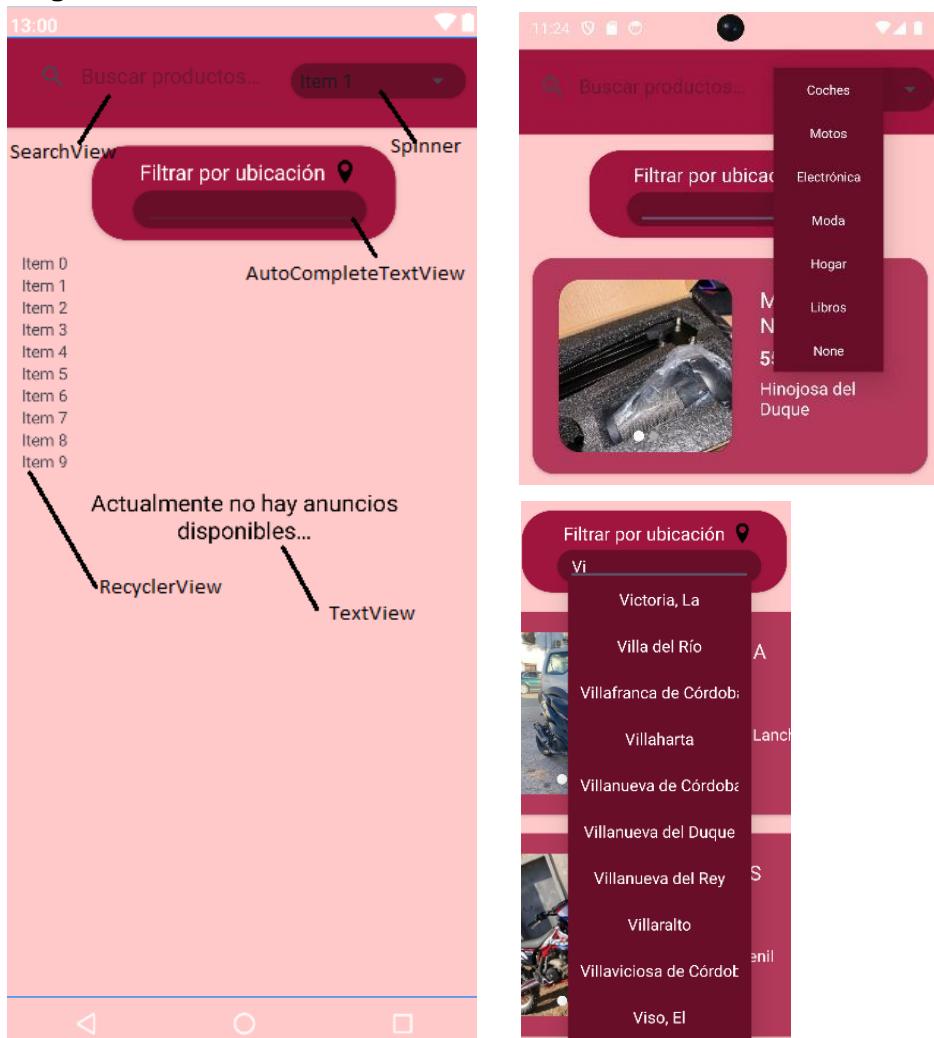
La siguiente activity se encarga de loguear a los usuarios con sus cuentas existentes.

Encontramos:

- EditText para el nombre de usuario.
- EditText de tipo password para la contraseña.
- Button para iniciar sesión con las credenciales introducidas.
- TextView para mostrar textos.

activity_register.xml

Esta Actividad es usada para crear tu usuario en la app, podemos ver que en caso de que ya tengas cuenta , solo tendrías que dar en *inicia sesión*.

fragment_inicio.xml

El siguiente fragment es el principal en el cuál se mostrarán todos los anuncios existentes, los cuales se podrán filtrar por título del anuncio, categoría y ubicación.

Encontramos:

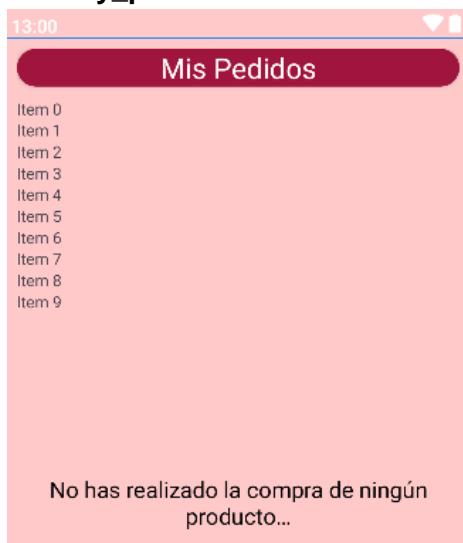
- SearchView para buscar anuncios por título.
- Spinner donde encontramos todas las categorías existentes.
- AutoCompleteTextView autocompleta el texto introducido con los municipios de la provincia de Córdoba.
- RecyclerView contiene una lista con todos los anuncios existentes.

fragment_user.xml

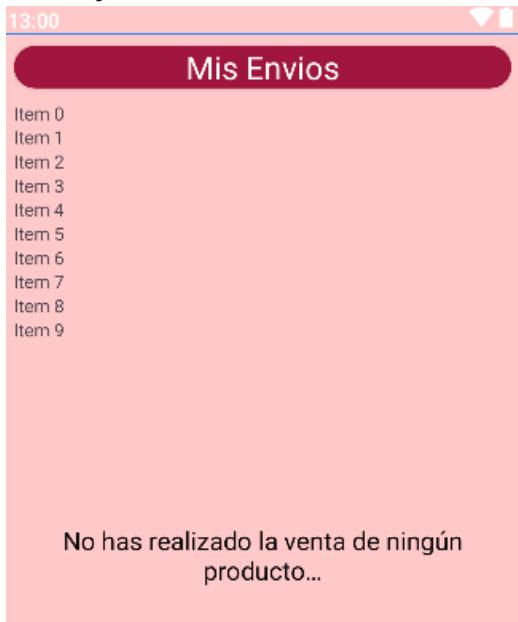
El siguiente fragment contiene toda la información del usuario, tanto los anuncios que tiene asociados como sus productos comprados y vendidos y un apartado de ajustes para modificar sus datos personales.

Encontramos:

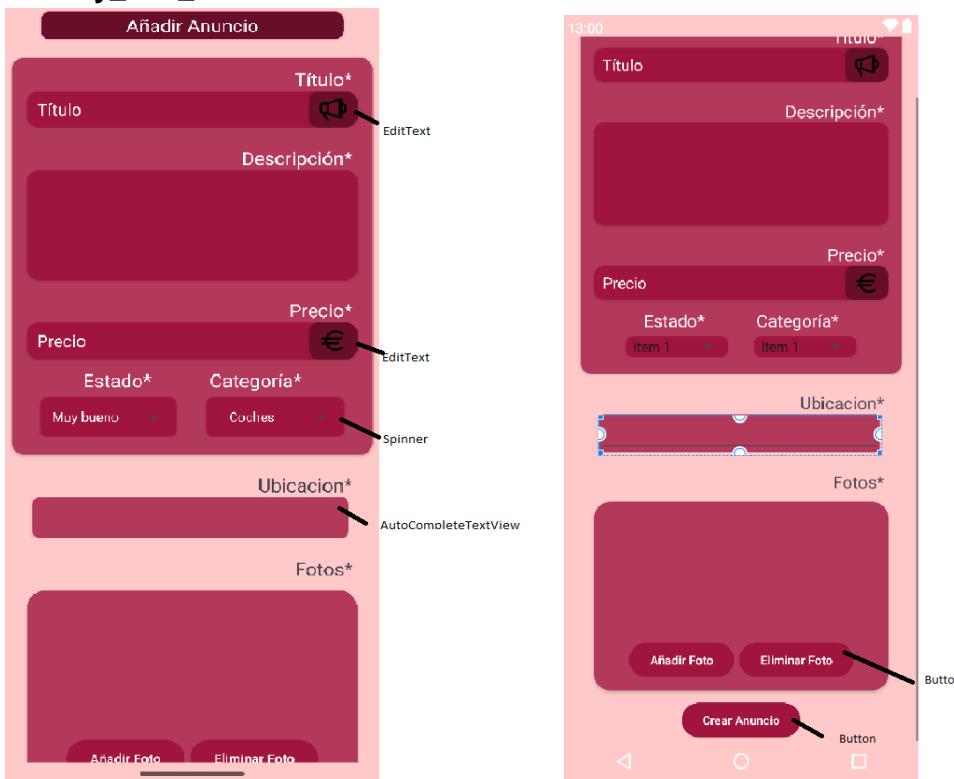
- ImageView con un botón de apagado para cerrar la sesión.
- RecyclerView con una lista de todos los anuncios que pertenecen a ese usuario.

activity_pedidos.xml

La siguiente activity contiene un registro con todos los anuncios que hemos comprado, en el caso de que no hayamos realizado la compra de ninguno nos aparecerá el siguiente mensaje.

activity_envios.xml

La siguiente activity contiene un registro con todos los anuncios que hemos vendido, en el caso de que no hayamos realizado la venta de ninguno nos aparecerá el siguiente mensaje.

activity_add_anuncio.xml

La activity add Anuncio contiene todos los datos necesarios y los parámetros necesarios para la Inserción de Usuario.

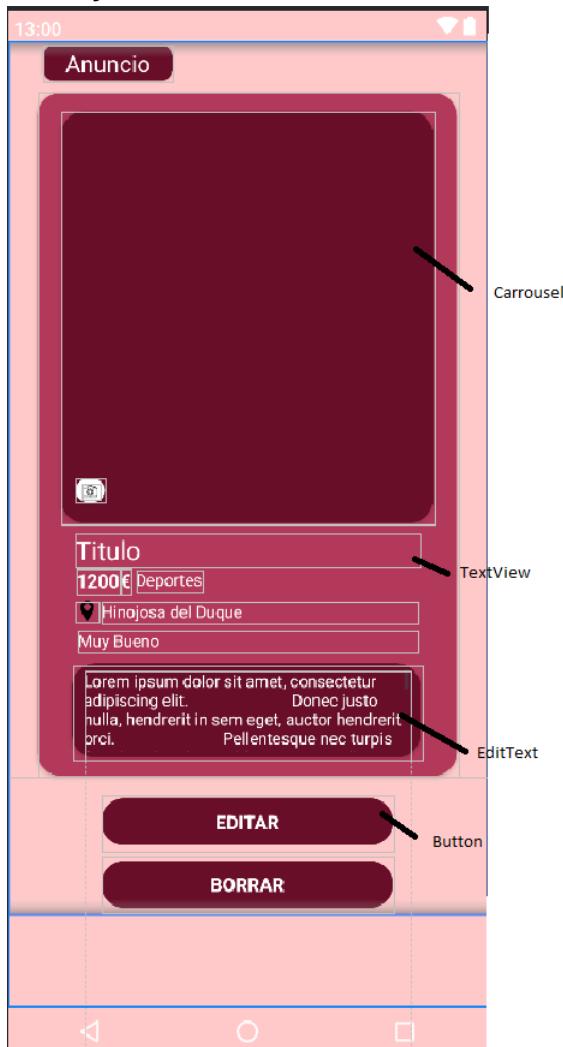
dialog_comprar.xml

El siguiente diálogo contiene la información de envío del producto
Encontramos:

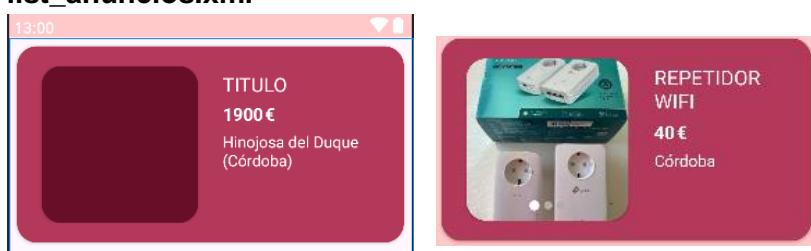
- EditText para añadir la dirección y el código postal.
- AutoCompleteTextView para añadir la ciudad.

activity_anuncio.xml

Esta es la vista que se despliega al abrir un anuncio desde el RecyclerView.

activity_anuncio_usuario.xml

La activity anuncio usuario se encarga de mostrar la información del anuncio seleccionado por el usuario a la hora de dar click en uno de tus anuncios , dando opción a editar el anuncio o borrarlo si lo desea el usuario.

list_anuncios.xml

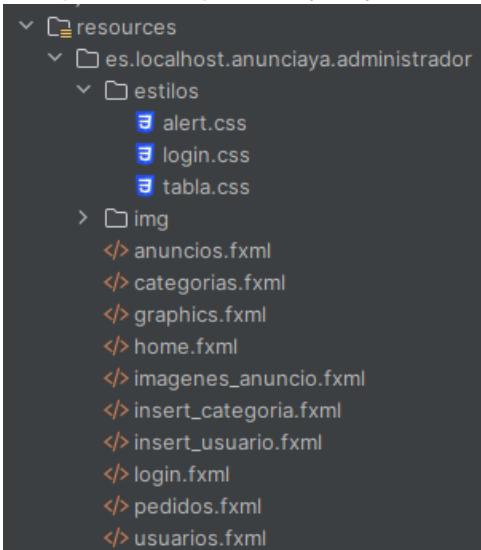
El siguiente diseño xml contiene el diseño que se mostrará en el recyclerview de la lista de anuncios.

list_pedidos.xml

El siguiente diseño xml contiene el diseño que se mostrará en el recyclerview de la lista de anuncios comprados.

4.1.3.2 Cliente Administrador Escritorio

Los diseños de las escenas que conforman el proyecto del cliente administrador de escritorio son los siguientes, los cuales se explicarán más detalladamente a continuación indicando los componentes que incluyen y su funcionalidad principal.

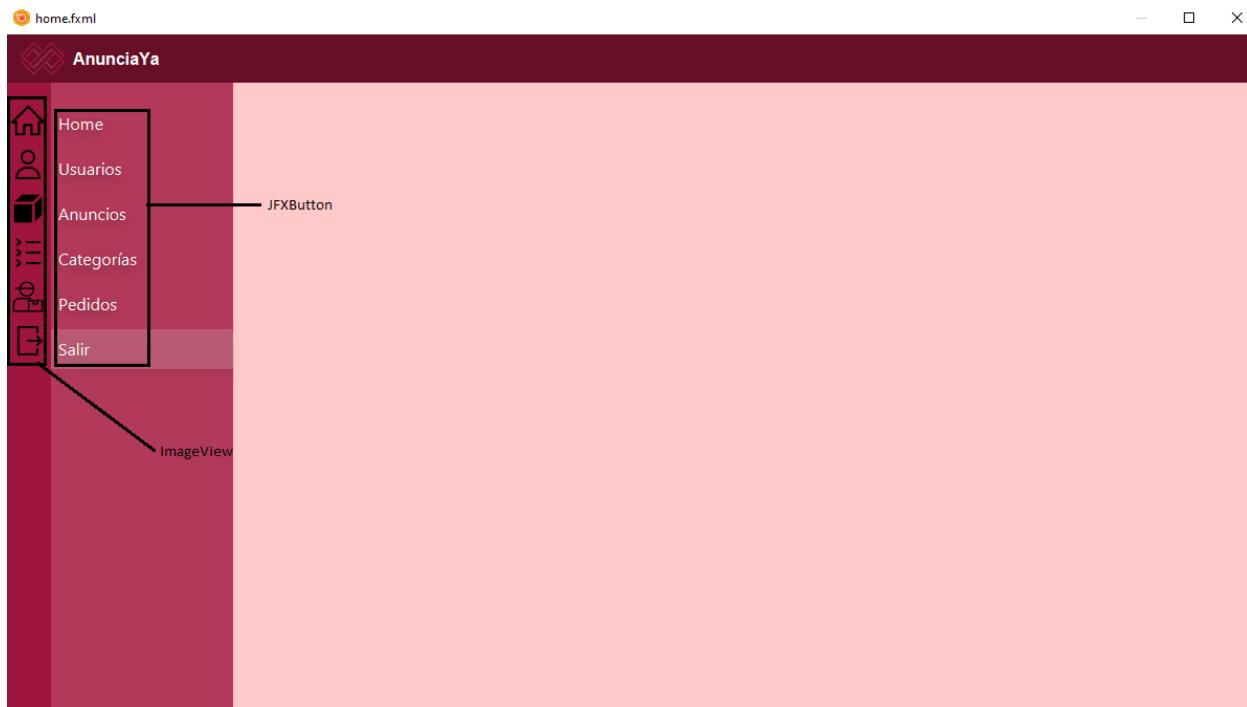
**login.fxml**

La siguiente escena se encarga de logear a los usuarios administradores en la aplicación de escritorio.

Encontramos:

- Label contiene únicamente texto, usado para el título.
- TextField para el email
- PasswordField para la contraseña
- Button para iniciar sesión con las credenciales introducidas.

home.fxml

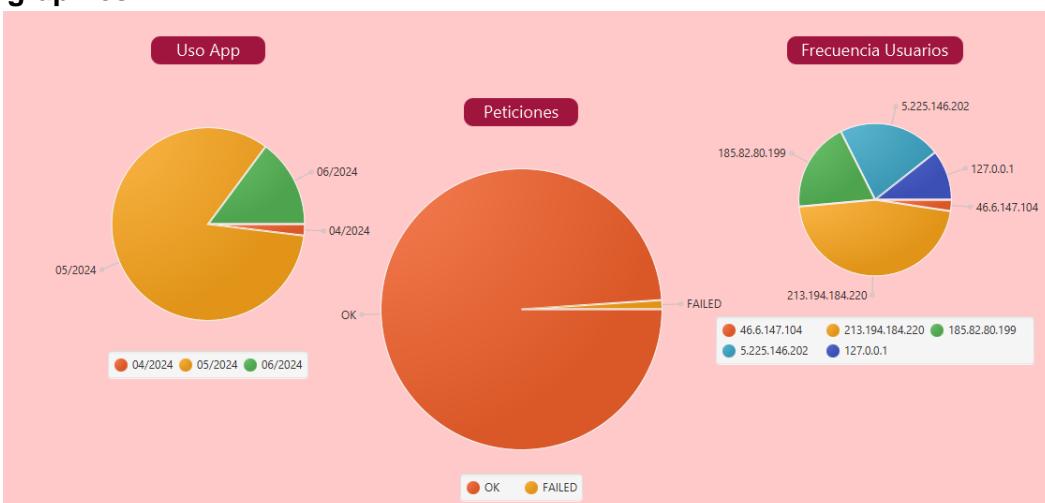


La siguiente escena gestiona el inicio de la aplicación, conteniendo un menú de navegación lateral donde podemos acceder a los distintos apartados de la aplicación de escritorio.

Encontramos:

- JFXButton botones.
- ImageView preview de los botones.

graphics.fxml



La siguiente escena muestra información del uso de la aplicación cliente, las peticiones al servidor y la frecuencia de acceso de los usuarios con la dirección IP Pública.

usuarios.fxml

USUARIOS REGISTRADOS

| ID | Nombre | Apellidos | Nombre Usuario | Fecha Nacimiento | Email | Teléfono | Tipo |
|----|---------|--------------------|----------------|------------------|--------------------------|-------------------|------|
| 22 | María | Martínez Rodríg... | mmartinez | 2002-12-09 | mariamarrodr@gmail.com | +34 789 90 00 763 | STD |
| 23 | Lucas | Murillo Sánchez | lmurillo | 2004-06-28 | lucasmurilsnch@gmail.com | +34 789 90 00 060 | STD |
| 24 | Mario | Ruiz Moyano | mruiz | 1998-02-26 | mario Ruiz my@gmail.com | +34 689 90 00 760 | STD |
| 38 | Claudia | Martínez Rodríg... | cmartinez | 2002-12-09 | mariamarroddr@gmail.com | +34 789 90 00 760 | STD |
| 39 | Juan | López García | jlopez | 1995-05-15 | juanlopez@gmail.com | +34 123 45 67 890 | STD |
| 40 | Ana | González Fernán... | agonzalez | 1988-03-22 | anagonzalez@gmail.com | +34 456 78 90 123 | STD |
| 41 | Carlos | Pérez Sánchez | cperez | 1990-07-30 | carlosperez@gmail.com | +34 789 12 34 567 | STD |

La siguiente escena muestra información de todos los usuarios registrados en la base de datos, teniendo la posibilidad de dar de alta uno nuevo pulsando en la imagen con el ícono de más, o de eliminar un usuario seleccionado en la tabla pulsando en el ícono de la papelera.

Encontramos:

- TableView tabla con la información de todos los usuarios.
- TableColumn columnas que componen la tabla.
- ImageView iconos con para realizar operaciones.

anuncios.fxml

ANUNCIOS REGISTRADOS

| ID | Nombre Usuario | Categoría | Título | Descripción | Estado | Ubicación | Precio | Fotos | Fech |
|-----|----------------|-------------|----------------|--|-----------|---------------------|---------|---|-------|
| 103 | cperez | Coches | Golf 2013 | Volkswagen Golf Mk ... | Muy bueno | Lucena | 16500.0 | https://4094501... | 2024- |
| 104 | cperez | Coches | BMW F30 | En perfecto estado, c... | Muy bueno | Lucena | 16490.0 | https://4094501... | 2024- |
| 105 | cperez | Coches | Range Rover... | IBERICA RETAIL SUR ... | Muy bueno | Lucena | 42990.0 | https://4094501... | 2024- |
| 100 | cperez | Motos | Yamaha yzf ... | vendo Yamaha r1 20... | Usado | Cabra | 6900.0 | https://4094501... | 2024- |
| 101 | cmartinez | Motos | Yamaha Jog r | Vendo yamaha jog r ... | Usado | Fuente la Lancha | 1400.0 | https://4094501... | 2024- |
| 102 | mruiz | Motos | Gasgas 250cc | Se vende gas gas del... | Usado | Puente Genil | 2400.0 | https://4094501... | 2024- |
| 83 | mmartinez | Electrónica | Micro NUEVO | Buenas ! Este micrófo... | Muy bueno | Hinojosa del Du... | 55.0 | https://4094501... | 2024- |
| 84 | mmartinez | Electrónica | Cable carga... | Se vende Cable Carg... | Nuevo | Pozoblanco | 10.0 | https://4094501... | 2024- |
| 85 | lmurillo | Electrónica | Repetidor wifi | TP Link modelo TL-W... | Usado | Córdoba | 40.0 | https://4094501... | 2024- |
| 91 | mmartinez | Moda | Jeans hombre | Vaqueros Energie par... | Usado | Palma del Río | 30.0 | https://4094501... | 2024- |
| 92 | agonzalez | Moda | Camiseta Ni... | Camiseta con logo es... | Usado | Hinojosa del Du... | 15.0 | https://4094501... | 2024- |
| 93 | mmartinez | Moda | Polo LACOS... | Polo azul de algodón... | Usado | Pozoblanco | 28.0 | https://4094501... | 2024- |
| 94 | jlopez | Moda | Nike Airfor... | Se vende porque me ... | Nuevo | Lucena | 75.0 | https://4094501... | 2024- |
| 95 | mruiz | Moda | Air Jordan 4 | Talla 42 Sin usar Tengo la factura de c... | Nuevo | Villafranca de C... | 250.0 | https://4094501... | 2024- |

| Fecha Publicación | Comprado |
|---------------------|----------|
| 2024-05-29 00:00:00 | No |
| 2024-05-29 00:00:00 | Sí |
| 2024-05-29 00:00:00 | No |

La siguiente escena muestra información de todos los anuncios existentes en la base de datos, teniendo la posibilidad de visualizar las imágenes asignadas de un anuncio, o de eliminar un anuncio seleccionado de la tabla.

Encontramos los mismos componentes que en la escena usuarios.fxml.

categorías.fxml

CATEGORÍAS REGISTRADAS

| ID | Descripción Categoría | |
|----|-----------------------|--|
| 1 | Coches | (+) Agregar |
| 2 | Motos | (x) Eliminar |
| 3 | Electrónica | |
| 4 | Moda | |
| 5 | Hogar | |
| 6 | Libros | |

La siguiente escena muestra información de todas las categorías existentes en la base de datos, teniendo la posibilidad de dar de alta una nueva categoría, o de eliminar una categoría seleccionada de la tabla.

Encontramos los mismos componentes que en la escena usuarios.fxml.

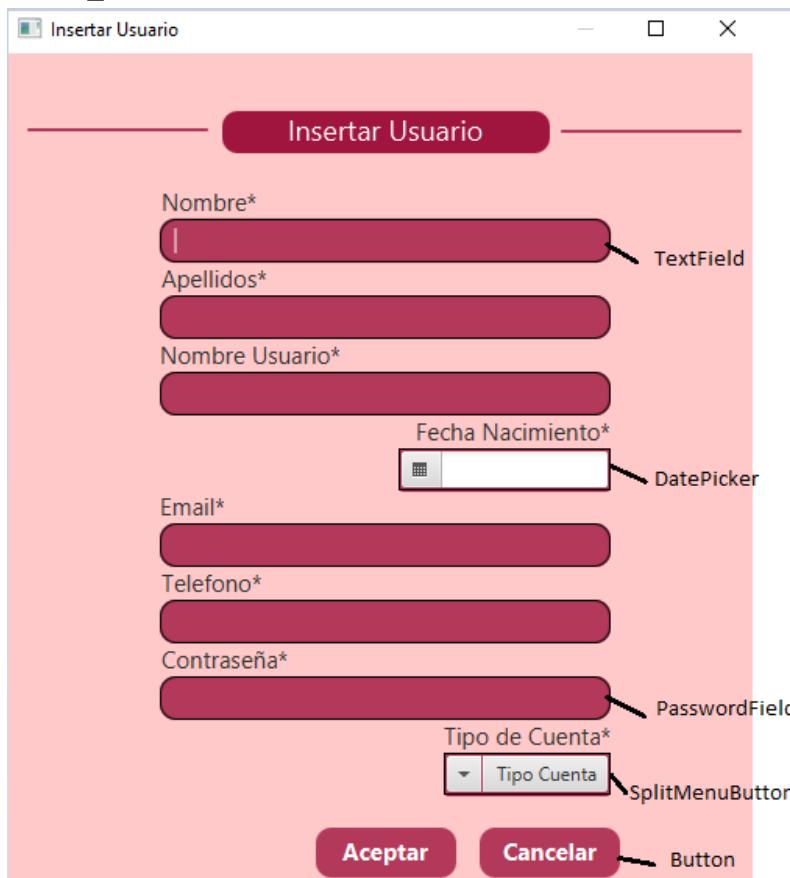
pedidos.fxml

PEDIDOS REGISTRADOS

| ID | Nombre Usuario | Id Anuncio | Título Anuncio | Dirección | Ciudad | CP |
|----|----------------|------------|----------------|-------------------|---------|-------|
| 7 | agonzalez | 92 | Camiseta Nike | Calle Canarias 12 | Córdoba | 14270 |

La siguiente escena muestra información de todos los pedidos creados en la base de datos.

Encontramos los mismos componentes que en la escena usuarios.fxml.

insert_usuario.fxml

La siguiente ventana emergente contiene un formulario para dar de alta a un nuevo usuario.

Encontramos:

- TextField campos de texto para introducir el nombre, apellidos, nombre de usuario, email y teléfono.
- DatePicker para mostrar un calendario para seleccionar la fecha de nacimiento.
- PasswordField para introducir la contraseña.
- SplitMenuButton lista desplegable para marcar el tipo de cuenta ADM administrador o STD estándar.
- Button acepta guardar los cambios o cancelar para cerrar la ventana emergente.

insert_categoria.fxml

La siguiente ventana emergente contiene un pequeño formulario para añadir una nueva categoría.

Encontramos componentes ya citados en el insert_usuario.fxml

imagenes_anuncio.fxml

La ventana emergente mostrará las imágenes del anuncio seleccionado en la tabla. En esta ventana, se proporcionan dos botones para navegar entre las imágenes: uno para avanzar a la siguiente y otro para retroceder a la imagen anterior.

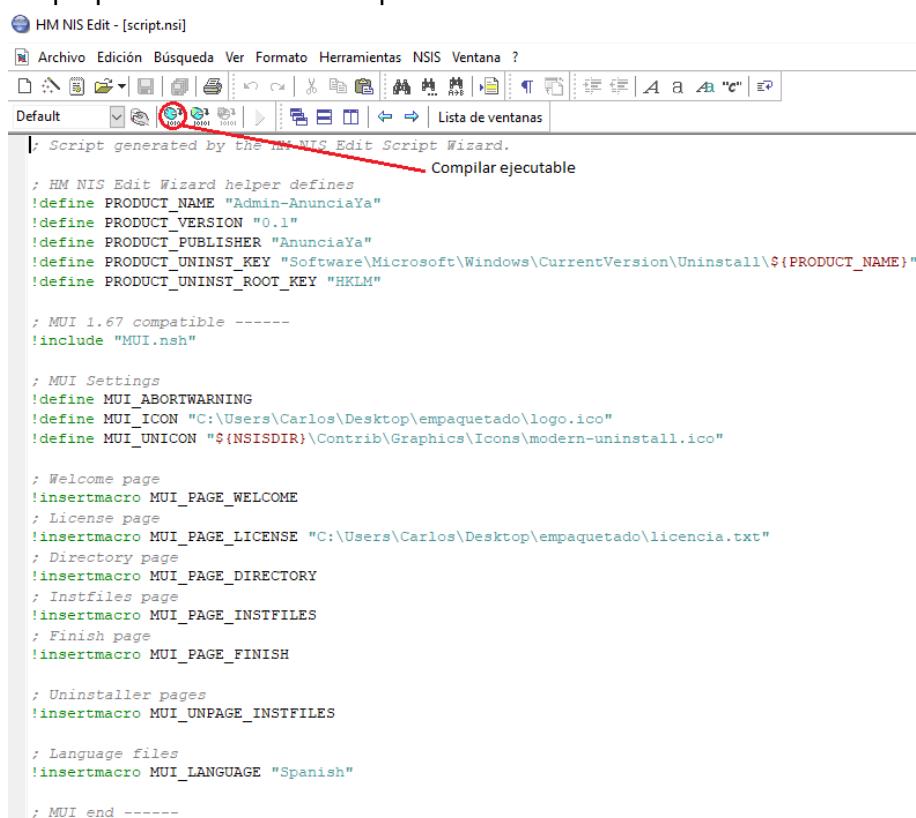
4.2 - [Pruebas]

5 - Documentación

5.1 - Empaquetado / Distribución

La distribución de aplicaciones para clientes finales comienza con el empaquetado, que consiste en reunir todos los archivos necesarios y configurar un instalador. Posteriormente, la aplicación se distribuye a través de canales como tiendas de aplicaciones, sitios web, o sistemas de gestión de versiones, asegurando compatibilidad, seguridad y facilidad de actualización para una experiencia de usuario óptima.

Para el cliente administrador multiplataforma, hemos utilizado HM NIS Edit para crear el paquete de instalación, mientras que para el cliente Android, la aplicación ha sido compilada y empaquetada en el formato por defecto APK.



```

HM NIS Edit - [script.nsi]
Archivo Edición Búsqueda Ver Formato Herramientas NSIS Ventana ?
Default Lista de ventanas
; Script generated by the HM NIS Edit Script Wizard.
; Compilar ejecutable

; HM NIS Edit Wizard helper defines
#define PRODUCT_NAME "Admin-AnunciaYa"
#define PRODUCT_VERSION "0.1"
#define PRODUCT_PUBLISHER "AnunciaYa"
#define PRODUCT_UNINST_KEY "Software\Microsoft\Windows\CurrentVersion\Uninstall\$PRODUCT_NAME"
#define PRODUCT_UNINST_ROOT_KEY "HKLM"

; MUI 1.67 compatible -----
#include "MUI.nsh"

; MUI Settings
#define MUI_ABORTWARNING
#define MUI_ICON "C:\Users\Carlos\Desktop\empaquetado\logo.ico"
#define MUI_UNICON "${NSISDIR}\Contrib\Graphics\Icons\modern-uninstall.ico"

; Welcome page
!insertmacro MUI_PAGE_WELCOME
; License page
!insertmacro MUI_PAGE_LICENSE "C:\Users\Carlos\Desktop\empaquetado\licencia.txt"
; Directory page
!insertmacro MUI_PAGE_DIRECTORY
; Instfiles page
!insertmacro MUI_PAGE_INSTFILES
; Finish page
!insertmacro MUI_PAGE_FINISH

; Uninstaller pages
!insertmacro MUI_UNPAGE_INSTFILES

; Language files
!insertmacro MUI_LANGUAGE "Spanish"

; MUI end -----

```

Si todo ha ido bien y no ha surgido ningún error nos aparecerá el siguiente mensaje

```

Install: 5 pages (320 bytes), 4 sections (2 required) (8288 bytes),
Uninstall: 1 page (128 bytes), 1 section (2072 bytes), 48 instructi
Using zlib compression.

EXE header size:          202752 / 39936 bytes
Install code:              4650 / 29802 bytes
Install data:              14459091 / 15931953 bytes
Uninstall code+data:       11105 / 15420 bytes
CRC (0x6ADDBF9B):          4 / 4 bytes

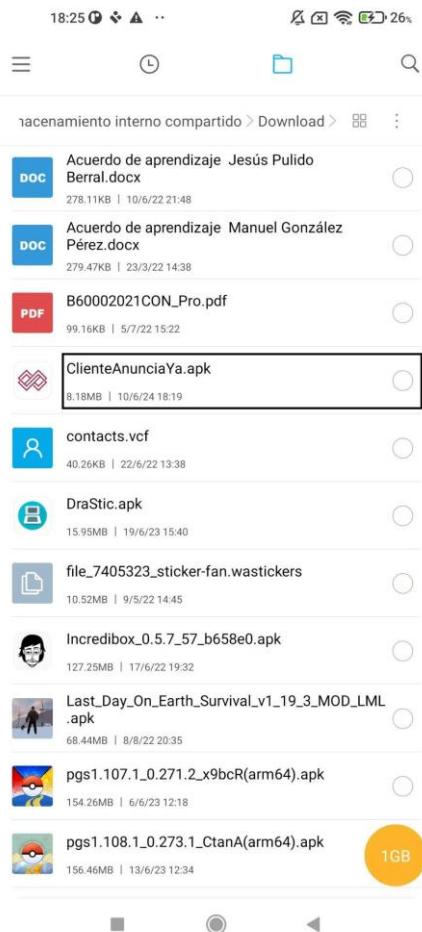
Total size:                14677602 / 16017115 bytes (91.6%)

```

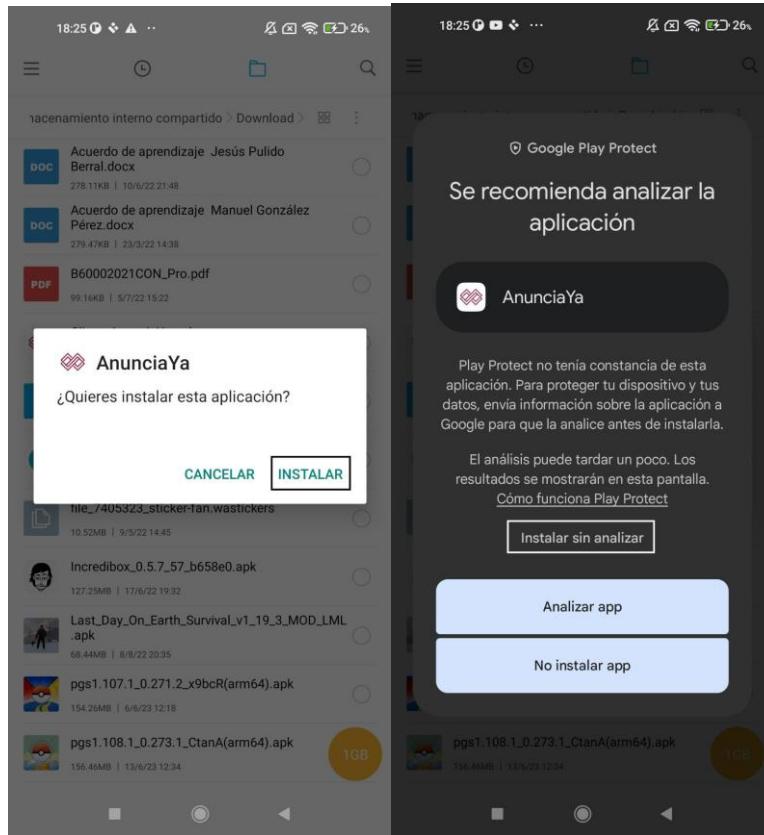
5.2 - Instalación

Instalación del Cliente Android

Primero de todo debemos de dar click en la APK que queremos instalar , que en nuestro caso es ClienteAnunciaYa.apk



Después daremos en Instalar para instalar la aplicación, después de esto nos aparecerá el siguiente mensaje de seguridad [este mensaje aparece ya que no es una app oficial del play store], lo que haremos será dar en Instalar sin Analizar.



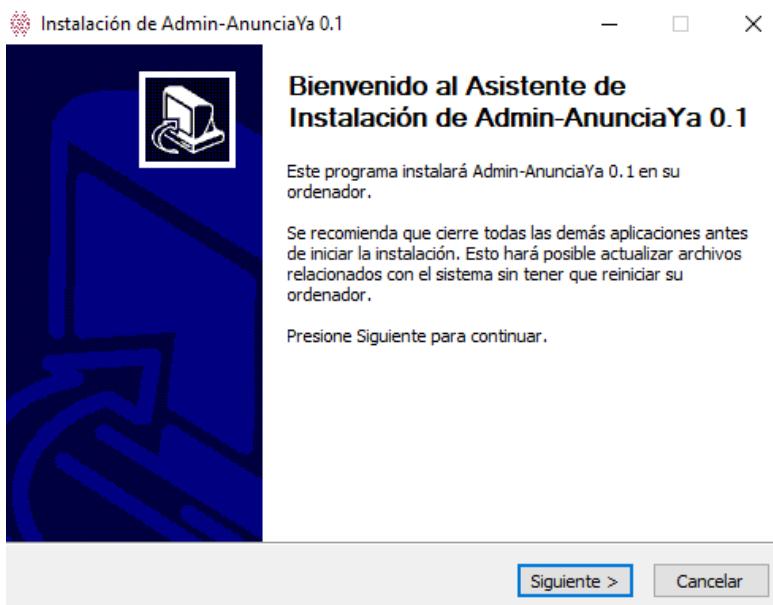
Ahora lo que haremos será esperar a que se termine de instalar y daremos en Abrir



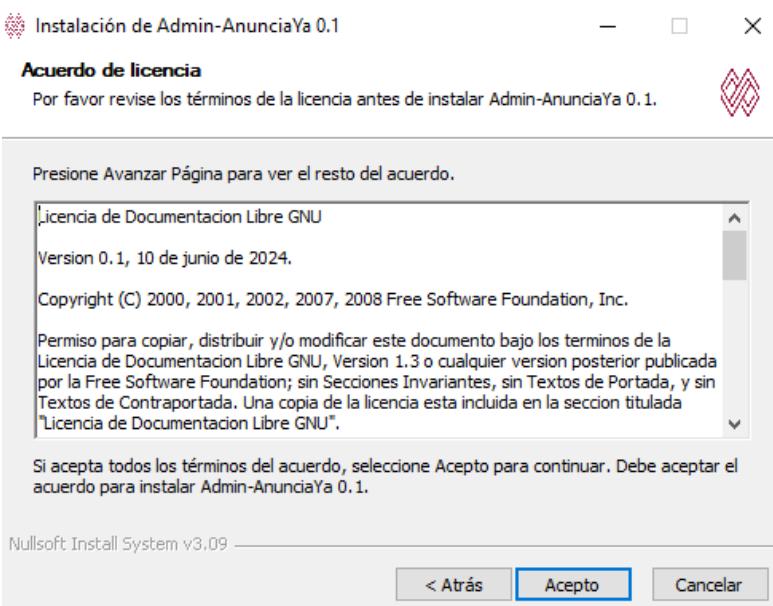
Instalación del Cliente Administrador Multiplataforma

Para la instalación del cliente administrador multiplataforma usaremos el ejecutable generado con HM NIS Edit.

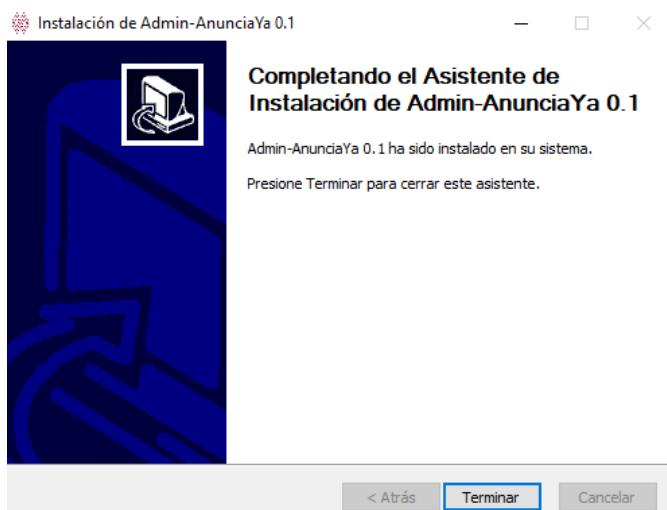
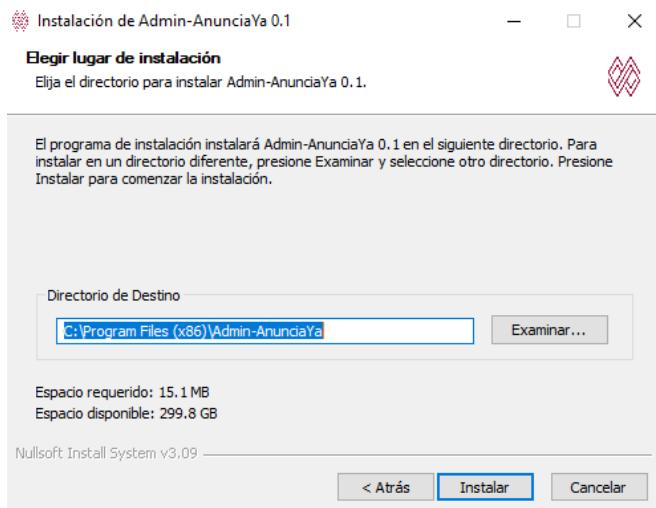
Al iniciar el ejecutable seguiremos el asistente de instalación de la aplicación y al finalizar nos creará un acceso directo en el escritorio para iniciar directamente la aplicación.



Tendremos que aceptar el acuerdo de licencia de la aplicación para continuar con el proceso de instalación de la misma.



Podremos cambiar el lugar de instalación de la aplicación, por defecto la instalará en Archivos de Programa.



5.3 - Manual de Usuario / Referencia

https://docs.google.com/document/d/1WjMb0YoqYCTo7uSURMIQuMbqiUt-T1kVJC-M_eUMpAA/edit#heading=h.jm2she7yuoh

6 - Conclusiones

Una vez acabado el proyecto , hemos visualizado futuras mejoras para este:

- Elaboración de chat.
- Mejora de Ubicación (Por Coordenadas).
- Acceso a la App mediante la API de Google.
- Imágenes en los perfiles de Usuario.
- Reseñas de usuarios a otros usuarios (Sistema de Valoraciones)
- Sistema de Reportes de cuentas (De usuario a usuario) y sistema en Perfil administrador de revisado de estos reportes y poder de Banear ip y borrar cuenta

Aun así esta aplicación es bastante completa y contiene bastantes funciones que haría esta app muy visible en el mercado.

7 - Bibliografía

<https://www.youtube.com/watch?v=H7JG947tLzo>

<https://www.youtube.com/watch?v=frGLzamduv0&t=185s>

<https://www.youtube.com/watch?v=c1rwM1LYjIM&t=283s>

https://www.youtube.com/watch?v=p04_f4mamrA&t=14s

<https://www.youtube.com/watch?v=42ZQ7GJvJho>

<https://www.youtube.com/watch?v=6t97D6ysXJ0&t=485s>

<https://www.youtube.com/watch?v=hxjNYe1CvEo&t=110s>

<https://www.youtube.com/watch?v=tQ7V7iBg5zE>

<https://www.youtube.com/watch?v=yTleUlvuMBk&t=421s>

<https://www.youtube.com/shorts/xX57O5YdH9o>

<https://www.youtube.com/watch?v=PmxZrPy1xyU>

<https://www.vipnet360.com/blog/que-son-las-licencias-de-software-y-que-tipos-existen#:~:text=Licencia%20de%20software%20libre,->

[Las%20licencias%20de&text=Se%20caracteriza%20por%20no%20contar,pueden%20distribuir%20de%20forma%20gratuita.](#)