MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE INTERFACES
TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Desarrollo de interfaces en Android (II)

14

ÍNDICE

OBJETIVOS

Crear menús que se ajusten a los estándares.

Crear menús contextuales cuya estructura y contenido sigan los estándares establecidos.

Distribuir acciones en menús, barras de herramientas, botones de comando, entre otros, siguiendo un criterio coherente.

Distribuir adecuadamente los controles en la interfaz de usuario.

Utilizar el tipo de control más apropiado en cada caso.

/ 1. Introducción y contextualización práctica

En este capítulo, vamos a continuar con el desarrollo de interfaces para dispositivos móviles con Android. En el capítulo anterior, analizamos aspectos relativos al diseño de la interfaz en cuanto **aspecto**. En este caso, nos vamos a centrar en otros puntos importantes, como es el caso de la **detección de eventos.**

Los **eventos** son determinantes para garantizar el correcto funcionamiento de una interfaz y, por tanto, la satisfacción de un usuario sobre una aplicación.

Además, analizamos en profundidad el componente Navigation que permite a un desarrollador diseñar de forma ágil y sencilla la ruta de navegación que puede seguir un usuario a través de las pantallas de la aplicación.

Finalmente, nos detendremos en el funcionamiento de la barra de aplicaciones, y a través de casos prácticos, se crearán iconos personalizados utilizando la herramienta Android Studio.

Escucha el siguiente audio donde planteamos la contextualización práctica de este tema, encontrarás su resolución en el apartado Resumen y resolución del caso práctico.

Fig. 1. Smartphone. Dispositivo con SO Android e iconos de aplicaciones.

Audio intro. "Aplicación móvil de alojamientos turísticos"
https://bit.ly/3lFMhgu

/ 2. Eventos

Hasta ahora, hemos analizado aspectos propios del diseño de la interfaz en cuanto al **aspecto**. Al igual que ocurre en el caso de las interfaces de aplicaciones de otro tipo de proyectos, la implementación relativa a la **detección de eventos** son determinantes para garantizar el correcto funcionamiento de una interfaz y, por tanto, la satisfacción de un usuario sobre una aplicación.

De nada sirve que el funcionamiento interno de una aplicación sea extraordinario si el diseño tanto en aspecto como en interacción con la interfaz de la aplicación no es óptimo.

La ocurrencia y posterior tratamiento de un evento se basa en la detección del mismo, habitualmente, acontecida por la pulsación del usuario sobre uno de los elementos mostrados en la pantalla del dispositivo.

La implementación para la detección de eventos se sustenta en los siguientes pasos, utilizando como ejemplo la detección de la pulsación de un botón como evento:

1. En primer lugar, es necesario implementar el método ***setOnClickListener*** que recibe por parámetro una nueva instancia del objeto ***ClickListener***. Este método queda asociado al elemento sobre el que se focaliza la escucha, por ejemplo, un botón.

2. A continuación, se implementa el código que va a ser ejecutado cuando se detecta el evento. Para ello, se crea un nuevo método que será lanzado ante la ocurrencia del evento.

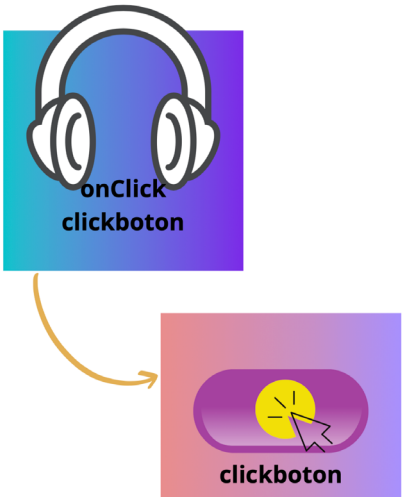

Fig. 2. Funcionamiento de un evento.

2.1. Método y escuchadores

Para implementar el código de eventos, en primer lugar, se debe incluir un elemento de escucha vinculado al método que va a tratar la acción realizada.

Por ejemplo, si el evento que se desea tratar es relativo a hacer clic sobre un elemento, necesitaremos utilizar el escuchador ***OnClickListener*** y su método ***onClick***. De esta forma, cuando se detecta la pulsación, se ejecutará el código contenido en el método onClick().

```
private OnClickListener Listener1 = new OnClickListener() {
   public void onClick(View v) {
     ...
   }
};
```

Código 1. Método tratamiento de eventos.

Como se puede ver en la tabla 1, existen multitud de métodos para el tratamiento de eventos y son específicos de cada caso concreto. Asimismo, cada uno de estos métodos presentará su propio escuchador.

En la siguiente tabla, se exponen algunos de los métodos de detección de eventos más comunes, y el elemento escuchador asociado:

MÉTODO	DESCRIPCIÓN	EVENTO
onClick()	Este método se invoca cuando se pulsa sobre un elemento de la interfaz.	onClickListener
onLongClick()	Se invoca cuando se mantiene pulsado un elemento de la interfaz.	onLongClickListener
onFocusChange()	Se invoca cuando el usuario se coloca sobre el elemento donde está realizando la escucha.	OnFocusChangeListener
onTouch()	Elementos "extra" que permiten personalizar la interfaz de desarrollo (calendarios, barras de progreso...).	OnTouchListener
onKey()	Este método se invoca cuando se pulsa sobre una tecla en un teclado hardware y el foco está situado sobre un elemento que implementa este método en su escucha.	OnKeyListener

Tabla 1. Métodos asociados a eventos.

/ 3. *Navigation*

Se trata de un componente que permite implementar cualquier tipo de diseño de navegación a través una aplicación, aportando un alto grado de personalización al diseño de la interfaz. Es decir, este componente se utiliza para diseñar la secuencia de pasos en la navegación entre pantallas de una misma aplicación.

El funcionamiento se basa en indicarle a ***NavController*** la ruta concreta a la que se desea ir, de las recogidas en el gráfico de navegación u otro destino concreto. Este destino será mostrado en el contenedor llamado ***NavHost***.

La implementación de este componente está basada en tres elementos:

Fig. 3. Proceso de instalación. Última pantalla.

- **Gráfico de navegación**: es el código XML en el cual queda reflejada toda la información de navegación.
- ***NavHost***: contenedor vacío utilizado para colocar los destinos hacia los que apunta el gráfico de navegación. Este elemento permite que los destinos vayan modificándose según el usuario navegue a través de la aplicación.
- ***NavController***: este elemento se encarga de la administración de la navegación del *NavHost*.

Para la implementación de *Navigation*, será necesario añadir el siguiente código de dependencias en el fichero **build.gradle**:

Código 2. Descripción de dependencias.

```
dependencies {
    def nav_version = “2.3.0”
    implementation “androidx.navigation:navigation-fragment:$nav_version”
    implementation “androidx.navigation:navigation-ui:$nav_version”

    implementation “androidx.navigation:navigation-fragment-ktx:$nav_version”
    implementation “androidx.navigation:navigation-ui-ktx:$nav_version”

    implementation “androidx.navigation:navigation-dynamic-features-fragment:$nav_version”

    androidTestImplementation “androidx.navigation:navigation-testing:$nav_version”
}
```

3.1. Creación *Navigation*

Un gráfico de navegación es una representación visual del conjunto de “pantallas” que forman una aplicación y de las acciones asociadas a la ocurrencia de cada una de esas pantallas, que quedan representadas con una flecha. Ahora bien, **¿cómo se añade un gráfico de navegación?**

La creación de un nuevo gráfico de navegación se lleva a cabo ubicándonos en la carpeta ***res***. A continuación, se pulsa con el botón derecho sobre esta capeta y se selecciona la opción ***New y New Android Resource File***.

Finalmente, se introduce en la caja de texto *File name* “nav_graph” y pulsamos OK.

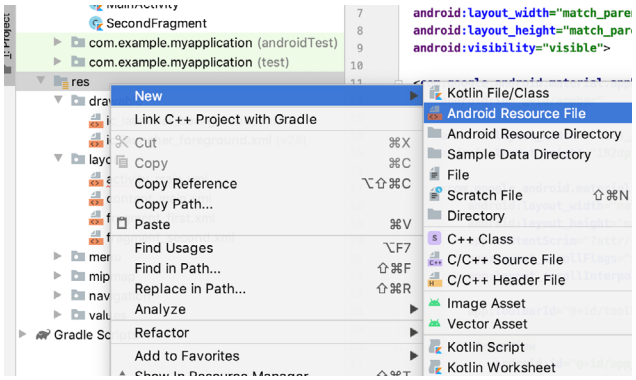

Fig.4. Selección en Android Resource File.

Tras la creación, se abre por defecto el fichero **nav_graph.xml** de la carpeta *values*, pero para poder acceder al editor que se analizará en el siguiente apartado, se

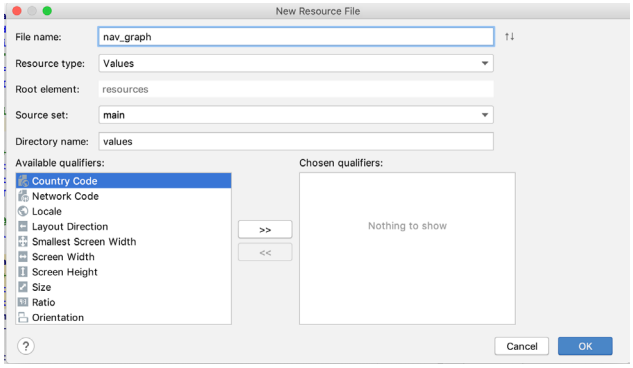

Fig. 5. Creación de gráfico de navegación.

debe localizar el fichero ***nav_graph*** de la carpeta *Navigation* y ejecutarlo, es decir, hacer doble clic sobre él para que se abra el gráfico de navegación:

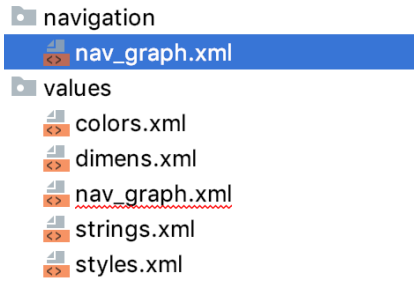

Fig. 6. Ruta ficheros nav_graph.xml.

3.2. Análisis de la interfaz

Tras la creación del gráfico de navegación, se accede al **editor de *Navigation*.** Como se puede ver en la siguiente imagen, la interfaz queda dividida en tres secciones claramente diferenciadas:

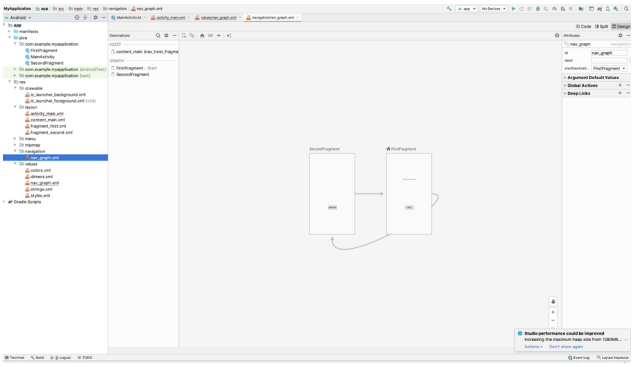

Fig. 7. Interfaz del editor de Navigation.

- **Panel de destinos (zona izquierda)**: similar a los exploradores de proyectos en los que aparecen todos los ficheros que forman parte de una misma carpeta. En este caso, se muestran todos los destinos accesibles.

- **Graph Editor (zona central)**: es la zona en la que se realiza la representación visual, bien en forma de código (XML) desde la pestaña Text, o bien de forma visual desde la pestaña *Design*.

- **Atributos (zona derecha)**: se recogen todos los atributos del elemento que se está analizando en cada momento.

En este caso, también será posible personalizar la vista de Android Studio, conmutando entre las diferentes opciones que aparecen en la parte superior derecha de la herramienta.

En función de las pantallas que se creen y de las relaciones entre las mismas de forma gráfica, se creará el código XML asociado. Para acceder a este código, se selecciona la opción *Code*.

Los principales elementos que se observan en el código generado son:

- ***navigation***: es el elemento principal de un fichero XML bajo el que aparecen el resto de elementos (pantallas como fragment y relaciones como action).

- ***fragment***: representa cada una de las pantallas colocadas en el editor.

- ***action***: indica el flujo de relaciones que existen entre las pantallas.

En el siguiente código, se muestra una primera pantalla y la relación de esta con la segunda.

```
<fragment
    android:id="@+id/FirstFragment"
    android:name="com.example.myapplication.FirstFragment"
    android:label="@string/first_fragment_label"
    tools:layout="@layout/fragment_first">
    <action
        android:id="@+id/action_FirstFragment_to_SecondFragment"
        app:destination="@id/SecondFragment"
    />
</fragment>
```

Código 3. XML Pantalla + acción.

3.3. Creación de un nuevo esquema de pantallas en Navigation

Cuando se crea un nuevo gráfico de navegación, este aparece configurado por defecto como se muestra en la figura del apartado anterior. Habitualmente con dos elementos de tipo *fragment* y la relación que une a ambos. Ahora bien, un desarrollador podrá modificar este esquema todo lo necesario, creando así una interfaz propia de la aplicación y mejorando la experiencia de navegación de los usuarios de la misma.

Para añadir nuevas pantallas, es posible hacerlo a través del código XML o del editor gráfico.

Si se realiza de la primera forma, bastará con insertar el siguiente código donde se indica el nombre de la nueva pantalla.

```
<fragment
    android:id="@+id/ThirdFragment"
    android:name="com.example.myapplication.SecondFragment"
</fragment>
```

Código 4. Nueva pantalla-fragment.

Si se desea hacer de forma gráfica, una vez seleccionado el tipo de pantalla de entre las que se muestran, se pulsa el botón con un rectángulo y un símbolo + de color verde.

En cualquiera de los casos, el resultado será como el que se muestra en la siguiente imagen.

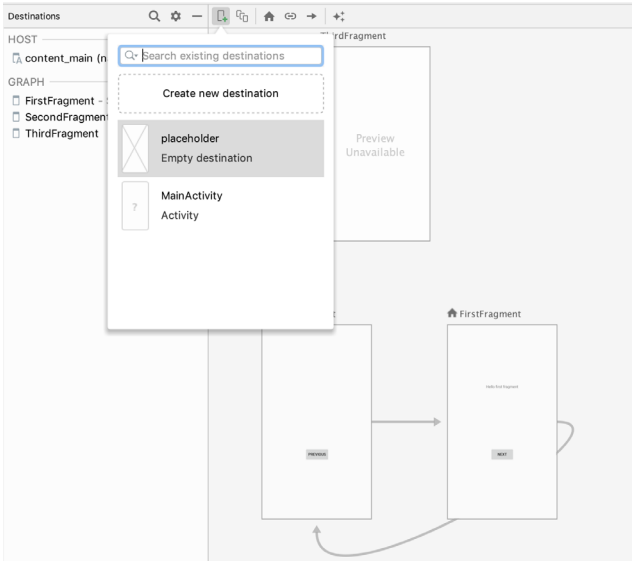

Fig. 8. Creación de pantallas.

En este segundo caso, el código XML generado solo incluye un atributo **id**.

```
<fragment android:id="@+id/placeholder" />
```

Código 5. Código inicial de creación de pantalla..

Para añadir un nombre específico, basta con añadir el atributo name, como aparece en el Código 4.

Vídeo 1. "Creación de un componente Navigation. Análisis de la interfaz y análisis del código"

https://bit.ly/35EDV31

3.4. Creación de conexiones entre pantallas

Como se ha visto anteriormente, las pantallas quedan conectadas mediante relaciones, es decir, la acción efectuada sobre una determinada interfaz tiene como resultado una nueva, ya sea de tipo específico o general (como el regreso a la pantalla de inicio).

Por lo tanto, las acciones representan las rutas que se van a seguir en la aplicación producto de la acción del usuario sobre la misma. A continuación, se indican varias formas para crear estas conexiones, ahora bien, debemos recordar que se está creando la conexión, pero el código que implementa la acción del cambio de una pantalla a la otra se realiza a través de los ficheros **.java**:

1. La primera opción consiste en **seleccionar el punto** que aparece en el borde de la pantalla y arrastrar la flecha hasta el destino. Esta opción resulta adecuada cuando el número de pantallas es relativamente pequeño y quedan todas a la vista.

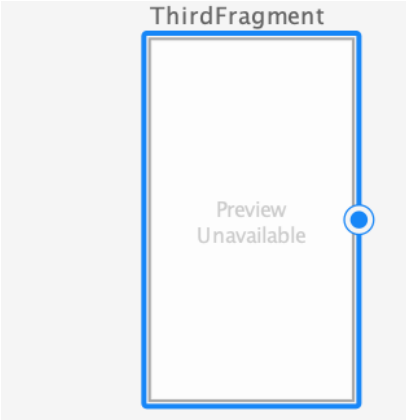

Fig. 9. Pantalla + elemento de conexión.

2. La segunda opción **es pulsando con el botón derecho** sobre la pantalla **origen** que va a ser conectada y se selecciona ***Add Action***. En el menú, se indica el **nombre de la conexión** y ***Destination*** muestra en una lista de valores todos los posibles destinos.

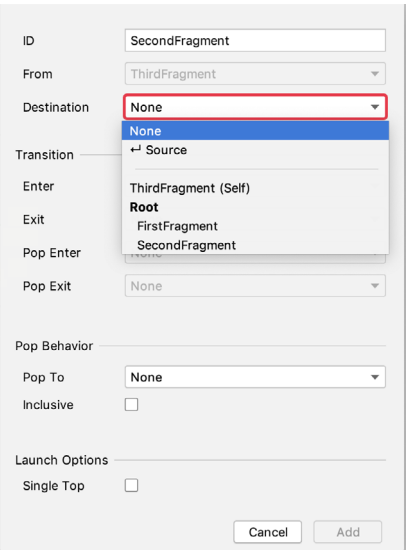

Fig. 10. Add Action configuración.

Cualquiera de las dos opciones tiene como resultado la conexión de pantalla ***ThirdFragment*** con ***SecondFragment***:

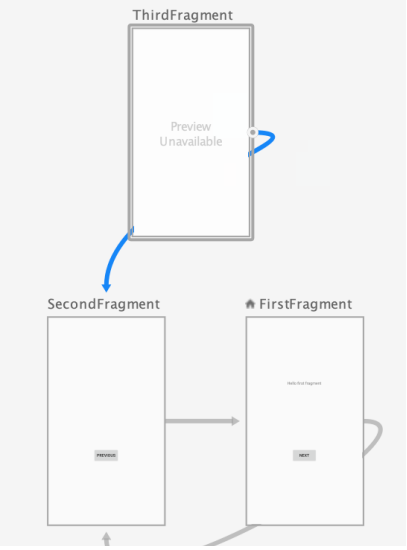

Fig. 11. Add Action configuración.

3.5. Transacciones entre pantallas con animación

El manejo de *Navigation* requiere de práctica, por lo que se aconseja visitar el sitio oficial de desarrollo de Android para seguir ampliando y actualizando el contenido sobre sus componentes y el resto de los utilizados por Android.

Para concluir este bloque, veremos algunas propiedades interesantes que dotan de cierta animación a la transición entre pantallas:

app:enterAnim	Animación asociada a la entrada en una pantalla
app:exitAnim	Animación asociada a la salida de una pantalla
app:popEnterAnim	Animación asociada a la entrada en una pantalla a través de una acción emergente
app:popExitAnim	Animación asociada a la salida de una pantalla a través de una acción emergente

Tabla 2. Propiedades de animación.

Para indicar el tipo de animación que va a presentar cada relación, se debe seleccionar y, a continuación, en el menú **Atributos**, escoger la acción sobre la que se va a indicar un nuevo tipo de animación para, después, pulsar el **botón que aparece a la derecha**.

En el cuadro de diálogo que aparecerá, se debe seleccionar el tipo de animación que va a presentar la acción.

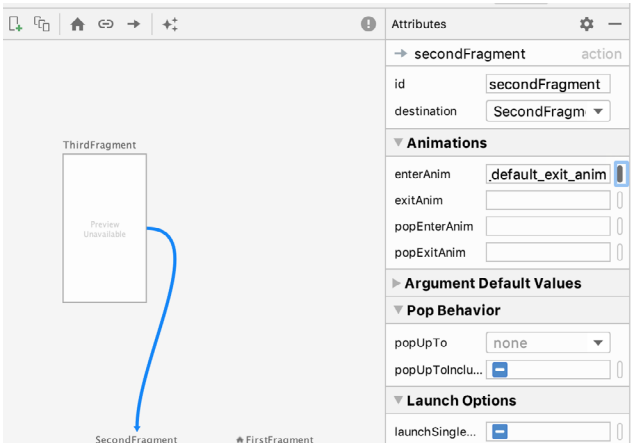

Fig. 12. Atributo de animación.

Desde la pestaña de *Text*, se actualizará automáticamente el código añadiendo los nuevos atributos:

```
<action
    android:id="@+id/secondFragment"
    app:destination="@id/SecondFragment"
    app:enterAnim="@anim/nav_default_exit_anim"
/>
```

Código 6. Código de animación de relación.

/ 4. Animaciones

Las animaciones, al igual que ocurre con el diseño web, aportan un alto grado de dinamismo a la funcionalidad de la aplicación y, además, permiten informar al usuario del estado de la petición, por ejemplo, para indicar que una petición se está procesando o que se está autenticando a un usuario en la aplicación.

En primer lugar, es necesario crear la animación, y para ello se utiliza ***AnimationDrawable***, en concreto, se implementa utilizando un elemento ***animation-list,*** que permite cargar todos los fotogramas que formarán parte de la animación.

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/rocket_thrust1" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust2" android:duration="200" />
    <item android:drawable="@drawable/rocket_thrust3" android:duration="200" />
</animation-list>
```

Código 7. Código de elemento de animación.

El atributo ***oneshot*** permite indicar si la animación se reproduce una sola vez (*true*) o de forma indefinida (*false*).

Los códigos de creación de animaciones se colocan en la ruta **res/drawable/**. Para ser utilizados, serán llamados desde el código Java correspondiente.

```
AnimationDrawable rocketAnimation;
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    ImageView rocketImage = (ImageView) findViewById(R.id.rocket_image);    rocketImage.
    setBackgroundResource(R.drawable.rocket_thrust);
    rocketAnimation = (AnimationDrawable) rocketImage.getBackground();
    rocketImage.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        rocketAnimation.start();
    }
});
```

Código 8. Animación ejecutada al pulsar la pantalla.

Audio 1. "El sistema de color de Material Design"

https://bit.ly/35EnzHV

/ 5. Agregar un barra de *app*

Uno de los elementos presentes habitualmente en cualquier aplicación es la **barra de acciones o barra de *app***. En el capítulo anterior ya se expuso cómo colocar un elemento de este tipo sobre la interfaz, pero ahora analizaremos en profundidad su comportamiento y la personalización posible de la misma.

Las barras de acciones tienen un espacio limitado, por lo que solo se podrán colocar algunos accesos rápidos. La elección de estos resultará clave para mejorar la experiencia de navegación del usuario.

Para incorporar nuevos botones a esta barra, se debe crear un nuevo archivo en la ruta **res/menu** y, a continuación, crear un nuevo elemento ítem para cada uno de los componentes que se quieran incluir en la barra.

Se debe prestar especial atención al atributo ***showAsAction***, puesto que indica cómo quedaría el icono de acción representado:

- ***ifRoom***: queda representada con un icono si hay espacio suficiente y en el menú ampliado si no lo hay.
- ***never***: por el contrario, si se indica directamente el valor never, solo será mostrado el elemento en el menú ampliado (se accede a través de los tres puntos en vertical).

En el siguiente código, es posible ver un elemento con cada uno de los valores indicados.

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="com.example.myapplication.MainActivity">
<item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never"
    />
</menu>
```

Código 9. Creación y configuración de la barra de app.

Los elementos colocados en la barra presentarán una acción asociada a través de eventos.

Vídeo 2. "Funcionamiento de la barra de app"
https://bit.ly/3kH5YTE

/ 6. Caso práctico 1: "*Navigation* con dos pantallas"

Planteamiento: Utilizando el componente *Navigation*, vamos a implementar un sistema compuesto por, al menos, dos pantallas que deben cumplir los siguientes criterios:

- La primera ventana recibirá el nombre *FirstFragment* y será principal, desde donde se inicia la ejecución de la aplicación.
- La segunda ventana y siguientes recibirán los nombres *SecondFragment, ThirdFragment*...

Nudo:Para la implementación de este caso, debemos crear un nuevo componente *Navigation* desde el menú *New Android Resource File*.

A continuación, se colocarán las pantallas incluyendo un nuevo fragmento, o desde el menú superior se selecciona el botón con el signo + de color verde y se añaden nuevas pantallas.

Desenlace: El código muestra la implementación de la descripción anterior.

```
<?xml version=”1.0” encoding=”utf-8”?>
<navigation
xmlns:android=“http://schemas.android.com/apk/res/android”
    xmlns:app=”http://schemas.android.com/apk/res-auto”
    xmlns:tools=”http://schemas.android.com/tools”
    android:id=”@+id/nav_graph”
    app:startDestination=”@id/FirstFragment”>
    <fragment
        android:id=”@+id/FirstFragment”
        android:name=”com.example.myapplication.FirstFragment”
        android:label=”@string/first_fragment_label”
        tools:layout=”@layout/fragment_first”>
        <action
            android:id=”@+id/action_FirstFragment_to_SecondFragment”
            app:destination=”@id/SecondFragment” />
    </fragment>
    <fragment
        android:id=”@+id/SecondFragment”
        android:name=”com.example.myapplication.SecondFragment”
        android:label=”@string/second_fragment_label”
        tools:layout=”@layout/fragment_second”>
        <action
            ndroid:id=”@+id/action_SecondFragment_to_FirstFragment”
            pp:destination=”@id/FirstFragment” />
    </fragment>
</navigation>
```

Código 10. Creación de un componente Navigation con dos pantallas y acciones.

/ 7. Caso práctico 2: “Crear iconos y logotipos personalizados para la aplicación”

Planteamiento: El desarrollo de iconos personalizados para cada aplicación es un elemento muy importante para dotar al sitio de una marca propia de identidad. Android Studio incluye herramientas que permite utilizar imágenes y adaptarlas con distintas plantillas a los diferentes elementos visuales que pueden ser utilizados en el desarrollo de una interfaz.

En este caso, se va a desarrollar un conjunto de iconos para una aplicación que permite realizar llamadas telefónicas que van a tener una duración máxima de 10 minutos. Es decir, la funcionalidad interna de la aplicación finaliza la llamada al pasar el período de tiempo indicado en el menú de configuración.

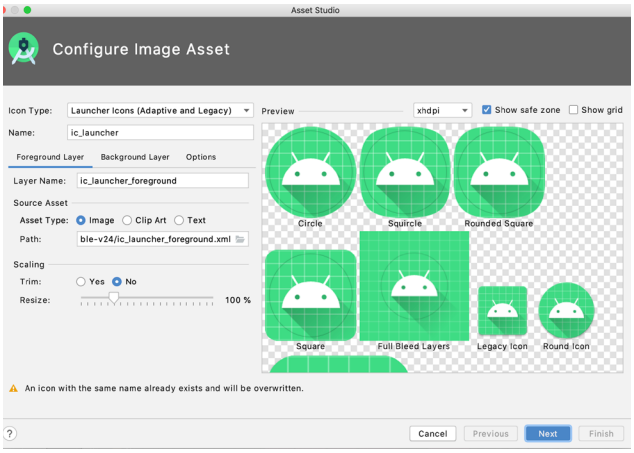

Fig. 13. Configuración Imagen Asset.

Nudo: Para la creación de estos iconos personalizados, se debe acceder a la ventana de configuración mostrada en la Figura 13, desde el res, pulsamos ***New*** y, a continuación, ***Imagen Asset.***

Desenlace: En el menú de la herramienta, es posible modificar todos los atributos de diseño. Por ejemplo, imaginemos que los colores característicos de la aplicación son el naranja y el azul, por lo que accediendo a la paleta de colores, tomaremos el azul para el fondo del icono y el naranja para el símbolo central.

Ahora bien, ¿qué imagen es conveniente utilizar para el icono? Escogemos una imagen que sea fácil de asociar al propósito de la aplicación y que, además, presente líneas sencillas que no resulten difíciles de identificar en dispositivos de menor tamaño.

Un posible resultado sería el siguiente:

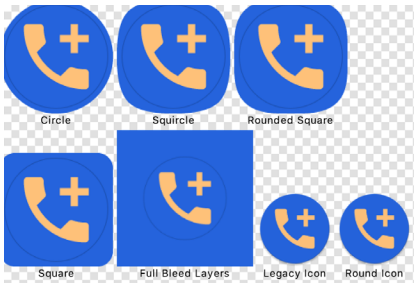

Fig. 14. Diseños de icono.

/ 8. Resumen y resolución del caso práctico de la unidad

En este capítulo, hemos visto uno de los elementos más importantes en cuanto al desarrollo de cualquier aplicación, ya sea de escritorio o para un dispositivo móvil, hablamos de los **eventos**. Recordemos que para la implementación del código de eventos, en primer lugar, se debe incluir un elemento de escucha vinculado al método que va a tratar la acción realizada. Por ejemplo, como método de escucha: *setOnClickListener*, y los escuchadores: *ClickListener*.

También hemos trabajado ampliamente con el componente ***Navitagion***, que permite implementar cualquier tipo de diseño de navegación a través de una aplicación, aportando un alto grado de personalización al diseño de la interfaz. El funcionamiento se basa en indicarle a *NavController* la ruta concreta a la que se desea ir, de las recogidas en el gráfico de navegación u otro destino concreto. Este destino será mostrado en el contenedor llamado *NavHost*.

El paquete *Swing* contiene todas las clases necesarias para programar todo tipo de componentes visuales. *Swing* es una extensión para AWT, un *kit* de herramientas de *widgets* utilizados para el desarrollo de interfaces gráficas en Java.

Resolución del caso práctico inicial

Con los conocimientos adquiridos a lo largo del tema, podemos responder a las preguntas de diseño sobre la interfaz de la aplicación móvil que se propuso al principio del tema.

Para implementar un diseño de una interfaz compuesto por dos pantallas se utilizará el componente *Navigation*.

Como se ha visto durante la unidad, para la implementación de este componente, será necesario acceder al menú *New Android Resource File* de Android Studio y colocar las pantallas incluyendo un nuevo fragmento.

Por otro lado, para el diseño de la aplicación de alojamientos rurales, se crearán una serie de iconos basados en la silueta de una casa con el fondo verde, como se ha representado en la figura anterior, dotando a la aplicación de una marca propia de identidad.

Por último, comentar que Android Studio permite utilizar imágenes y adaptarlas a distintas plantillas obteniendo una interfaz personalizada.

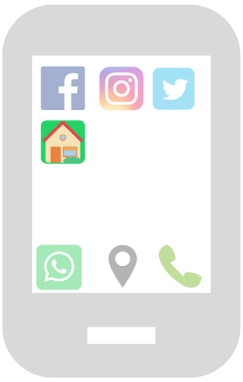

Fig. 15. Icono de interfaz.

/ 9. Bibliografía

Fernández, A.; García-Miguel, B., y García-Miguel, D. (2020). *Desarrollo de Interfaces* (1.a ed.). Madrid, España: Síntesis.

9.1. Webgrafía

Android Developers. Consultado en: https://developer.android.com