DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Interfaces en Android

Eventos

Menús

Navigation

Animaciones

Barra de app

Imagen Asset

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. 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. 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**.

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()**.

Método tratamiento de eventos

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

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:

Métodos asociados a eventos		
Método	**Descripción**	**Evento (escuchador)**
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 tecla en un teclado **hardware** físico **y el foco está situado sobre un elemento** que implementa este método en su escucha.	OnKeyListener

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:

- 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**:

Descripción de dependencias

```
dependencies {
    def nav_version = "2.3.0"
    // Java language implementation
    implementation "androidx.navigation:navigation-fragment:$nav_version"
    implementation "androidx.navigation:navigation-ui:$nav_version"
    // Kotlin
    implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
    implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
    // Feature module Support
    implementation "androidx.navigation:navigation-dynamic-features-fragment:$nav_version"
    // Testing Navigation
    androidTestImplementation "androidx.navigation:navigation-testing:$nav_version"
}
```

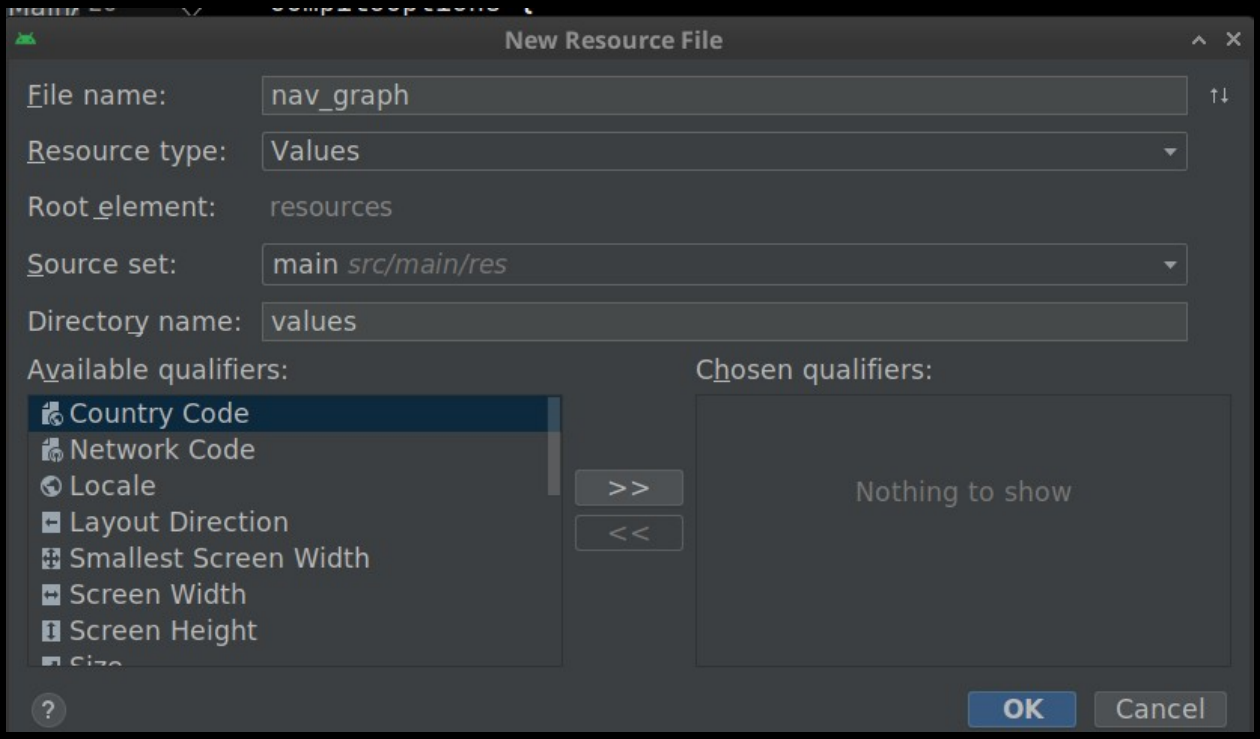
Creación Navigation

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.

Creación de gráfico de navegación



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 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**.

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:

- 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:

XML Pantalla + acción

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

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.

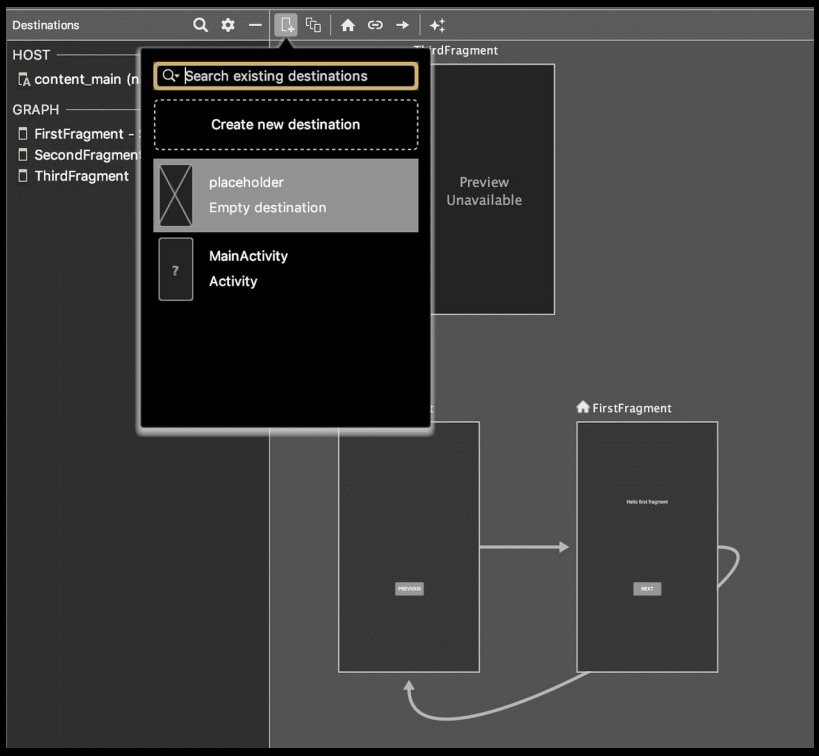
Nueva pantalla-fragment

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

</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:

Creación de pantallas



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

Código inicial de creación de pantalla

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

Para añadir un nombre específico, basta con añadir el atributo name:

```
android:name="com.example.myapplication.NombreDelFragment".
```

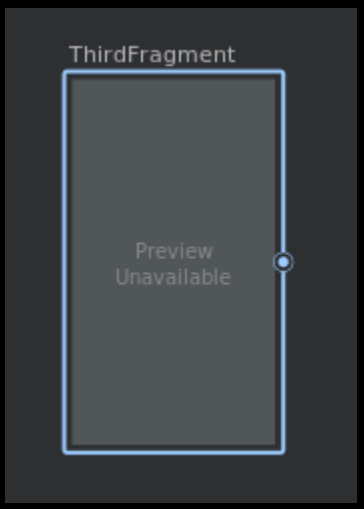
Creación de conexiones entre pantallas

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***:

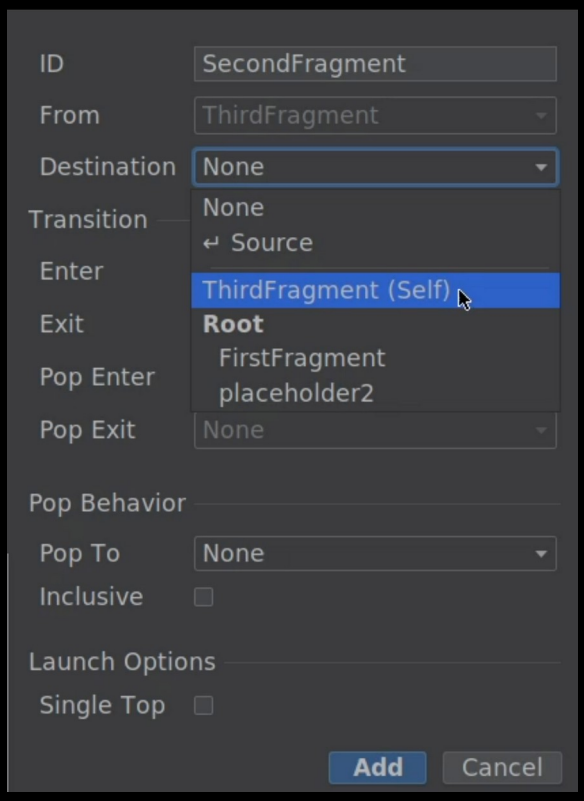
1. La primera opción consiste en seleccionar el punto que aparece en el borde de la pantalla y arrastrar la **flecha hasta el destino**. Esta opción resulta adecuada cuando el **número de pantallas es relativamente pequeño** y quedan todas a la vista.

Pantalla + elemento de conexión



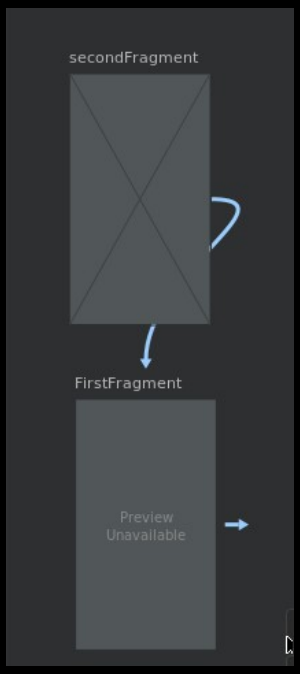
2. La segunda opción es pulsando con el **botón derecho sobre la pantalla origen** que va a ser conectada y se selecciona **Add Action**. En el menú, se indica el **nombre de la conexión** y **Destination** muestra en una lista de valores todos los **posibles destinos**.

Add Action configuración



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

Add Action configuración



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.

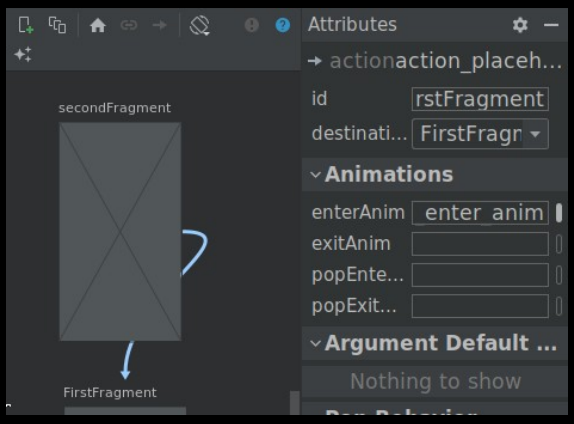
Para concluir este bloque, veremos algunas propiedades interesantes que dotan de cierta animación a la transición entre pantallas:

Propiedades de animación	
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**

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.

Atributo de animación



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

Código de animación de relación

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

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

Navigation es un componente que nos va a permitir implementar cualquier diseño de navegación por una aplicación personalizándolo hasta el nivel en el que nosotros así lo queramos y por lo tanto nos estamos centrando en diseñar las interfaces relativas a la programación, no tanto la funcionalidad que eso se ocuparía en otro momento, en otros módulos, sino que aquí lo que nos centramos es en el **diseño de la propia interfaz** que al final va a ser un elemento clave para que el usuario quiera seguir utilizando nuestra aplicación y que además esa experiencia de navegación sea óptima.

En primer lugar, vamos a saber cómo llegar a crearnos una interfaz en la que nos aparezcan las pantallas, donde nos aparezca el sistema de Navigation.

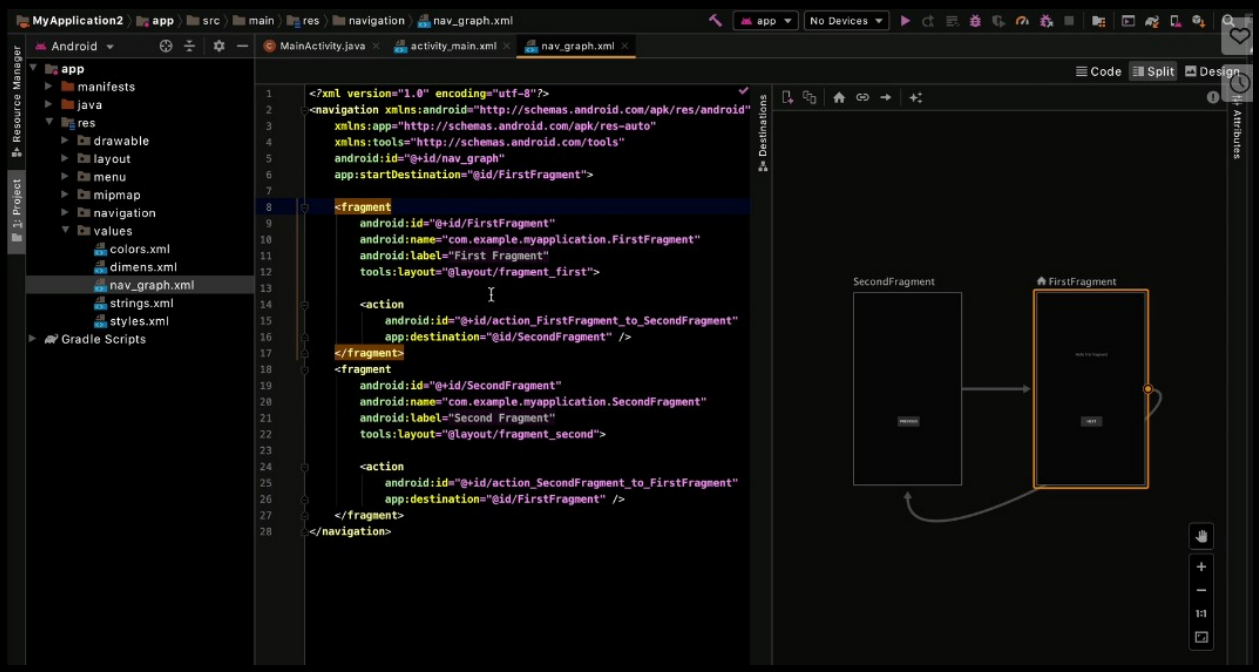
Lo que tenemos que hacer es desde la **carpeta del proyecto**, desde RES, pulsamos el botón derecho, ahora New y a continuación lo que vamos a seleccionar es Android Resource File. Vemos que nos va a aparecer esta pantalla que tenemos por aquí, lo que tenemos que pulsar es lo siguiente nav_graph y pulsamos OK. Y como vemos, pues nos habría abierto el programa.

Ya lo tendríamos implementado de antes...

Lo que nos aparece por defecto siempre es esta pantalla que tenemos por aquí. Muy importante, puede que al abrirlo la primera vez solo nos aparezca el código y al igual que ocurría, veíamos en el capítulo anterior, si pulsamos a Design nos va a aparecer solamente la parte más gráfica y si queremos ver código y parte gráfica lo que hacemos es dividir la pantalla. A este lado nos quedaría el código en XML que define la interfaz y aquí lo que vamos a tener colocados son las diferentes pantallas.

Estas serían las partes clave. Vamos a distinguir también para que nos quede claro cuáles son los elementos. Aquí equivale un elemento de la derecha con su respectivo elemento de la izquierda. ¿Qué hacemos? Lo pulsamos cuando este queda marcado de color azul. Es el código que además se nos resalta al mismo tiempo en XML. Este es fragment. **Cada una de las pantallas aparecen identificadas inequívocamente** con un fragment y vemos que dentro de este fragmento de código vamos a tener diferentes elementos.

Por ejemplo, vamos a tener una ID, una forma de identificar como si fuese el nombre de una variable, identificar cada una de las pantallas. En este caso es secondfragment. Si nos venimos aquí marcaría el primer código. Este sería el firstfragment.



¿Qué más tenemos? La etiqueta. Esto es lo que nos aparece por aquí. Lo que nos aparece por aquí sería este nombre. Al pulsar dos veces lo que nos indica es que en este caso es una etiqueta que recibe el nombre de secondfragment y que es de tipo string. ¿Cuál es el contenido? El que nos aparece aquí arriba. Este es el nombre que aparece, el nombre que se le asigna a cada una de las etiquetas de las pantallas.

¿Qué nos quedan? Las **acciones**, las relaciones. ¿Cómo queda unida una ventana con la otra? Con este sistema de navegación lo que vamos a ir haciendo es definiendo esa **ruta que va a seguir un usuario en capas**. Imaginamos que accedemos a una aplicación, encontramos una primera pantalla de inicio. Imaginamos que es una tienda de ropa. Pulsaríamos la primera opción y escogeríamos la tipología de la ropa faldas. Entraríamos un nivel hacia abajo. Nos llevaría a otra pantalla. A continuación seleccionamos dentro de las faldas si son largas o son cortas. ¿Qué es lo que hace esto? Cuando lo pulsamos nos va a llevar a otra pantalla.

De esta forma lo que estamos construyendo son **sistemas en profundidad** que van a dar al usuario la sensación de estar el mismo diseñando su paseo por la aplicación. En este caso, equiparable a un paseo por una tienda física.

¿Cómo representamos la **unión entre cada una de las pantallas**? Aquí lo tenemos, **action**. Es lo que nos va a definir cómo conectar una con otra. Por ejemplo, en el caso de SecondFragment vendríamos a este fragmento de aquí y lo que nos dice es que se define una relación, que es esta horizontal, que cuando se realiza determinada acción sobre la pantalla número 2 nos va a llevar a la primera. En este caso, que es el que se nos crea por defecto, de nuevo si escogiéramos la FirstFragment vemos que tenemos una acción que nos relaciona, nos vincula directamente esta ventana, esta pantalla, con la segunda.

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.

Código de elemento de 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>
```

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.

Código de animación ejecutada al pulsar la pantalla

```
AnimationDrawable rocketAnimation;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ImageView rocketImage = (ImageView) findViewById(R.drawable.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();
        }
    });
```

El sistema de color de Material Design

Para crear una **marca de seña de identidad** es importante tener un sistema de color que lo identifique. Con el sistema de color de Material Design se pueden utilizar **paletas de colores** para la creación de **temas de color**, así como herramientas específicas para la **selección del color adecuado**.

El sistema de color de Material Design permite aplicar color a la interfaz de usuario de una manera muy intuitiva. La clave de este sistema consiste en seleccionar un color primario y uno secundario para representar la marca. Además, existen **variantes oscuras o claras** que se pueden aplicar sobre el color seleccionado.

Los temas de color están diseñados para presentar un **aspecto armónico**, sin grandes excentricidades, garantizando que el **texto sea accesible** y se pueda **distinguir elementos y superficies** de la interfaz de usuario entre sí.

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.

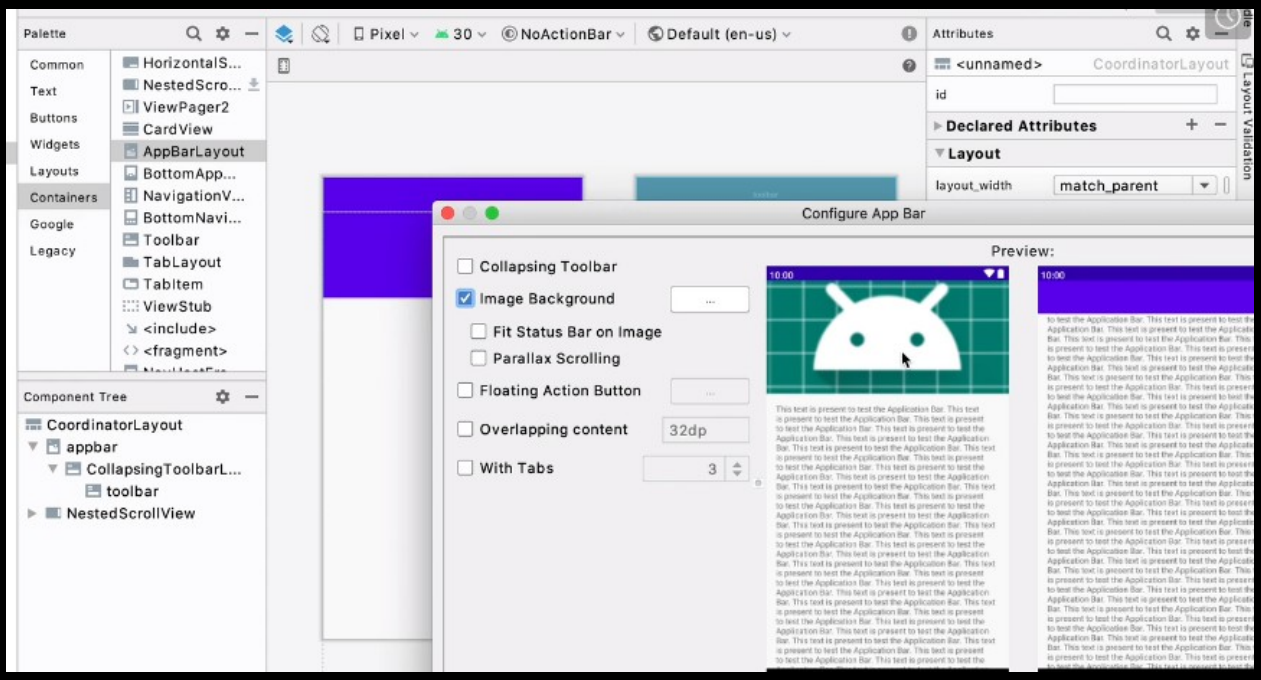
Creación y configuración de la barra de app

```
<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>
```

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

Funcionamiento de la barra de app

Cuando desarrollamos una aplicación para dispositivos móviles con el sistema operativo Android, es importante tener en cuenta que aparecerá una **barra en la parte superior** de estas aplicaciones. Esta barra puede ser llamada de diversas formas, como barra de aplicaciones, barra de app o barra de acciones. En esta barra suelen mostrarse **accesos directos a diferentes funcionalidades** de la herramienta. Además, a menudo encontramos **tres puntos verticales** en el extremo derecho, conocidos como **menú ampliado**, donde se incorporan acciones adicionales que no caben en la barra de acciones principal.



Vamos a analizar cómo se inserta y diseña una interfaz que incluya este elemento. Uno de los atributos principales que vamos a examinar es ‘showAsAction’.

Dentro de Android Studio, en la vista de diseño, podemos insertar esta barra seleccionando '**Container**' y luego 'appBarLayout'. A continuación, podemos elegir entre diferentes opciones de diseño, como 'SmartKey', que proporciona un diseño más amplio, o 'WithTabs', que incluye pestañas dentro de la barra de acciones. Una vez seleccionada la opción deseada, simplemente confirmamos y la barra se insertará en la interfaz.

El código XML es crucial para definir los componentes de la interfaz. En este caso, el 'toolbar' se define con atributos como dimensiones de ancho y alto. Para modificar atributos específicos, como 'showAsAction', es necesario acceder al archivo 'menu_main.xml' dentro de la carpeta 'res/menu' del proyecto.

El atributo 'showAsAction' determina si los botones de la barra de herramientas se mostrarán como iconos directamente en la barra de acciones o en el menú ampliado.

Las opciones incluyen 'ifRoom', que muestra el icono solo si hay suficiente espacio en la barra de acciones, y 'never', que siempre coloca el elemento en el menú ampliado.

Es importante colocar los elementos **directamente en el menú ampliado por defecto** para que los desarrolladores puedan **priorizar** qué elementos deben aparecer en la barra de acciones. Esto evita que la barra de acciones se sature con elementos innecesarios.

Para continuar trabajando en el diseño de la interfaz, volvemos siempre al archivo 'activity_main.xml', que es donde se desarrolla la mayor parte de la interfaz.

Ejemplo de Navigation con dos pantallas

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…

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.

El código muestra la implementación de la descripción anterior:

Creación de un componente Navigation con dos pantallas y acciones

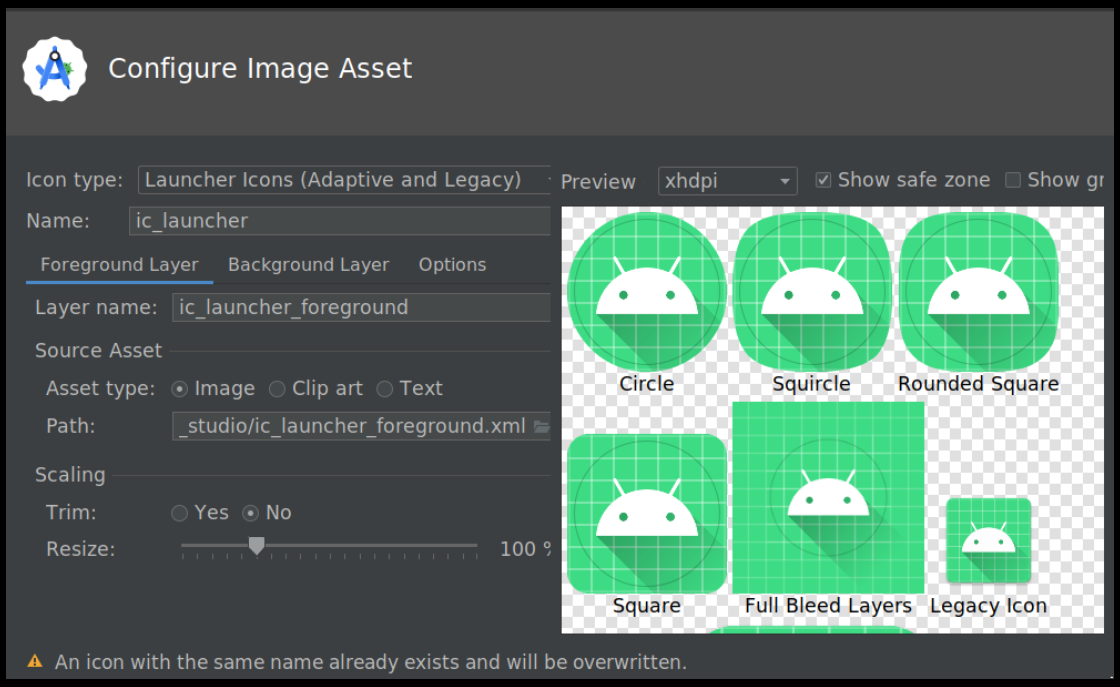
```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/FirstFragment">

    <fragment
        android:id="@+id/FirstFragment"
        android:name="com.example.myapplication.FirstFragment"
        android:label="fragment_first"
        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="fragment_second"
        tools:layout="@layout/fragment_second">

        <action
            android:id="@+id/action_SecondFragment_to_FirstFragment"
            app:destination="@id/FirstFragment" />
    </fragment>

</navigation>
```

Crear iconos y logotipos personalizados para la aplicación

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.

Configuración Imagen Asset



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

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.

https://developer.android.com