Apuntes

Tema 4. Material Design. Patrones de diseño básicos.

Descargar estos apuntes pdf o html

Índice

Introducción

Material Design se presentó durante la Google I/O 2014, y comprende unas guías o pautas que tienen que seguir las aplicaciones del ecosistema de Google, ya sean para Android como Web. Las características que definen Material Design son: diseño limpio, colorido (através de paletas de colores ya definidas), con efectos bidimensionales (a través de efectos de iluminación y sombra) y el uso de animaciones.

Patrones de diseño de la interfaz de Android

Todos los sistemas operativos proponen diferentes formas de interactuar con los elementos en pantalla. Conocer la diferencia entre ellos y utilizar elementos familiares para el usuario, asegura que se sienta cómodo y seguro usando la aplicación. Las características más comunes son: simplicidad (contar con pocos elementos, pero sobre todo, que aquellos presentes en la interfaz tengan una función bien definida que contribuya a cumplir el objetivo de la app y ayude al usuario), consistencia (el usuario espera que las aplicaciones se comporten de la misma manera) y navegación intuitiva.

Patrones MD

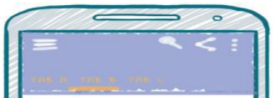
ActionBar y ToolBar

La ActionBar es un elemento visible en la parte superior de la pantalla de una aplicación Android. En ella se recoge entre otras cosas el nombre de la aplicación o de la activity en uso. Además puede incluir el icono de la aplicación, menú overflow y botones de menú.

La ActionBar fue lanzada por Google en 2013 con el lanzamiento de Android 3.0 (API 11). Todas las aplicaciones que utilizan el tema predeterminado proporcionado por Android (Theme.AppCompat.Light.DarkActionBar), contienen una ActionBar de forma predeterminada. Sin embargo, los desarrolladores pueden personalizarlo de varias formas según sus necesidades.
La ToolBar es un elemento de vista que se integra en los diseños XML de una activity. Fue introducido por el equipo de Google con el lanzamiento de Android Lollipop (API 21). Es mucho más flexible y personalizable, y a diferencia e la ActionBar no está necesariamente fijada en la parte superior de la aplicación, es más podemos colocar más de una.

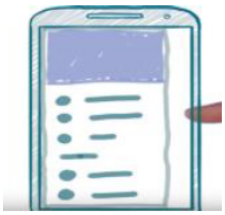
Tabs o pestañas

Permiten la navegación entre diferentes secciones de la aplicación. Se agrupan con AppBarLayout en la ToolBar. Se puede cambiar de una a otra pestaña a través de gestos o pulsando sobre cada uno de ellas.



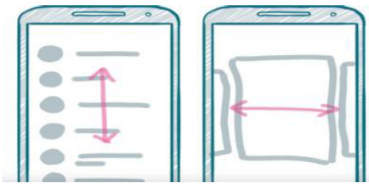
Navigation Drawer

Menú que se desliza desde la parte izquierda de la pantalla y contiene el nivel más alto de opciones de navegación.



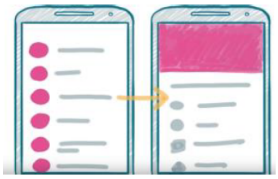
Scrolling y Paging

Un ScrollView permitirá visualizar de forma cómoda la información cuando esta no cabe totalmente en la pantalla. Si la información puede visualizarse de forma completa no aparece el scroll y sino aparecerá de forma automática.
El ViewPager nos va a permitir disponer de varias páginas que serán accesibles con gestos de izquierda y derecha o arriba y abajo.



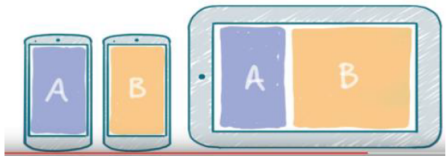
Listas

Es muy común el uso de este patrón de diseño en las aplicaciones para móviles. Cuando mostramos una lista de elementos y pulsamos sobre uno de ellos, aparece una segunda ventana donde se detalla la información del elemento pulsado.



Multipanel

Es una concreción del patrón anterior, de forma que en dispositivos que dispongan de pantallas grandes o dependiendo de si la pantalla la colocamos horizontal, visualizaremos dos paneles o más en la pantalla, obteniendo una visualización optimizada de la vista y de su contenido.



Paleta de colores

El sistema de color de Material Design te ayuda a aplicar color a tu interfaz de usuario de una manera simplificada. Los temas de color están diseñados para ser armoniosos, garantizar texto accesible y distinguir elementos y superficies de la interfaz de usuario fácilmente.

Como vimos en un tema anterior, los colores se gestionan desde el archivo `colors.xml` y su contenido al generar el proyecto es similar a:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Aquí tenemos un enlace donde se encuentra información sobre la gestión de colores que ofrece Material Design

En este enlace tenemos una herramienta donde probar las distintas paletas y descargar la configuración de colores elegida.

Una vez descargado el archivo hay que copiarlo a la carpeta `res/values` sustituyendo el archivo de configuración de colores predeterminado por el nuevo.

Es necesario reasiganar los `name` a los colores para hacerlos coincidir en archivo `res/values/styles.xml`

Layouts

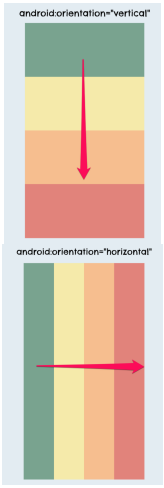
Los layouts son elementos no visibles destinados a controlar la distribución, posición y dimensiones de los controles que se insertan en su interior. Estos elementos extienden de la clase ViewGroup.

La definición de la interfaz gráfica de una aplicación se definirá a través de archivos XML o a través de código de forma programática.

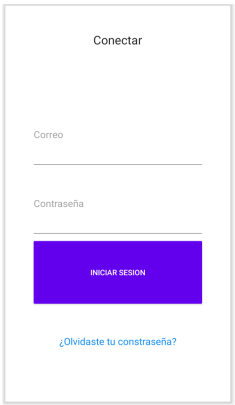
> 📌 Existen distintos tipos de layouts: FrameLayout, LinearLayout, RelativeLayout, GridLayout, ConstraintLayouy y CardView.

LinearLayout

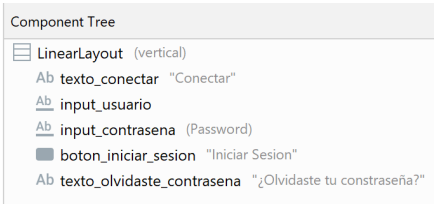
Un `LinearLayout` es un view group que distribuye sus hijos en una sola dimensión establecida. Es decir, o todos organizados en una sola columna (vertical) o en una sola fila (horizontal). La orientación puedes elegirla a través del atributo `android:orientation`.



Veamos un ejemplo, vamos a diseñar la vista que aparece en la siguiente imagen:



Para ello creamos un nuevo proyecto, que llamaremos **EjemploLinearLayout**. Los componentes que vamos a colocar son los siguientes:



El archivo XML correspondiente al diseño sería:

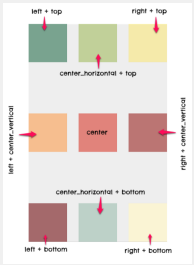
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="48dp">

    <TextView
        android:id="@+id/texto_conectar"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:text="Conectar"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/input_usuario"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:hint="Correo" />

    <EditText
        android:id="@+id/input_contrasena"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:ems="10"
        android:hint="Contraseña"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/boton_iniciar_sesion"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:text="Iniciar Sesion" />

    <TextView
        android:id="@+id/texto_olvidaste_contrasena"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:gravity="center_vertical"
        android:text="¿Olvidaste tu contraseña?"
```

```
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#0E8AEE" />
</LinearLayout>
```

📌 Aclaraciones:

- **Líneas 3 y 4**: Se utilizan las propiedades `android:layout_width` y `android:layout_height` para determinar las dimensiones del layout o de cualquier elemento de vista. Los valores que pueden tomar son `match_parent` y `wrap_content` , es decir se ajusta al tamaño del contenedor ocupando todo su espacio, o se ajusta al contenido o tamaño de la vista respectivamente. También se le puede asignar un valor concreto en unidades `dps`
- **Línea 5** con la propiedad `android:orientation` especificamos como se van a distribuir los elementos dentro del contenedor, horizontalmente o verticalmente.
- **Líneas 10-12** Para crear un diseño lineal en el que cada campo secundario use la misma cantidad de espacio en la pantalla, define el `android:layout_height` de cada vista en `0dp` (para un diseño vertical) o el `android:layout_width` de cada vista en `0dp` (para un diseño horizontal). Luego, fija el `android:layout_weight` de cada vista en 1. Si das valores distintos a 1 el tamaño de layout se distribuirá siguiendo esta distribución.

- **Línea 13** con la propiedad `android:layout_gravity` alineamos cada vista dentro del layout.



- **Línea 23** con la propiedad `android:hint` visualiza un texto en el `EditText` y cuando pinchamos en él desaparece.
- **Línea 31** con la propiedad `android:ems` se usa para determinar el tamaño de un carácter según la cantidad de puntos que use la fuente del texto.
- **Línea 33** con la propiedad `android:inputType` se usa para determinar el tipo de caracteres que se permiten como entrada en el `EditText` . Este atributo determina también el tipo de teclado virtual que le aparecerá al usuario para realizar la entrada de caracteres.

- **Línea 50** con la propiedad `android:gravity` se usa para determinar el tipo de alineamiento del texto dentro de una vista. Los valores son idénticos a los de la propiedad `android:layout_gravity`

CardView

Aquí tenemos un enlace donde se encuentra información ofrecida sobre este elemento en Material Design.

Este nuevo componente llamado `CardView` es la implementación que nos proporciona Google del elemento visual en forma de tarjetas de información que tanto utiliza en muchas de sus aplicaciones, entre ellas Google Now.

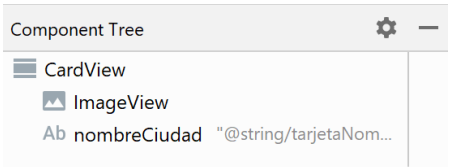
Está incluido en la librería **material**, por lo que hay que agregar esta dependencia a nuestro proyecto:

```
dependencies {
     implementation 'com.google.android.material:material:1.2.1'
}
```

Veamos un ejemplo, vamos a diseñar el cardView que aparece en la siguiente imagen:

Para ello creamos un nuevo proyecto, que llamaremos **EjemploCardView**.

Los componentes que vamos a colocar son los siguientes:



El archivo XML correspondiente al diseño sería:

```xml
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardCornerRadius="8dp"
    card_view:cardElevation="12dp">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/sanfrancisco"
        android:scaleType="centerCrop"
        android:contentDescription="@string/todo" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/nombreCiudad"
        android:padding="10dp"
        android:text="@string/tarjetaNombreCiudad"
        android:layout_gravity="bottom"
        android:background="#8c000000"
        android:textColor="@color/white"
        android:textSize="30sp"
        android:textStyle="bold"/>
</com.google.android.material.card.MaterialCardView>
```

📌 Usa estas propiedades para personalizar la apariencia del widget **`CardView`** :

- Para definir el radio de la esquina de tus diseños, usa el atributo **`card_view:cardCornerRadius`** .
- Para definir el color de fondo de una tarjeta, usa el atributo **`card_view:cardBackgroundColor`** .
- Para efecto de elevación con sombra usa el atributo **`card_view:cardElevation`** .

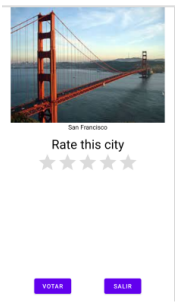
Constraint Layout

Para poder comprender como funciona este layout, primero debemos de conocer como se integran las vistas o componentes en los layouts. Veamos la siguiente imagen:

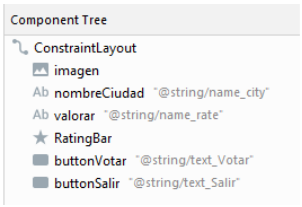
Cada lateral de un componente viene referenciado por un nombre específico que permite hacer referencia al mismo. `Top` es la parte superior, `Bottoom` es la parte inferior, `Start` es la parte izquierda y `End` es la parte derecha.

Para poder añadir componentes debemos restringir su posición definiendo unos anclajes basados en los laterales de cada componente. Son necesarios al menos dos anclajes.

Veamos un ejemplo práctico. Vamos a diseñar la siguiente interfaz:



Creamos un proyecto con el nombre **ejemploConstraintLayout** e integraremos los siguientes componentes:



Como vemos insertamos en el contenedor una `ImageView`, dos `TextView` una `RatingBar` y dos `Button`. No os preocupéis porque más adelante explicaremos con más detalle estos componentes, ahora solamente queremos utilizarlos para explicar `ConstraintLayout`.

El archivo XML correspondiente es el siguiente:

```xml
<?xml version="1.0" encoding="utf-8"? >
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imagen"
        android:layout_width="match_parent"
        android:layout_height="720px"
        android:layout_marginTop="0dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/sanfrancisco" />

    <TextView
        android:id="@+id/nombreCiudad"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/imagen"
        android:text="@string/name_city"
        android:textAlignment="center"
        android:textSize="15dp"
        android:textColor="@color/black"/>

    <TextView
        android:id="@+id/valorar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/nombreCiudad"
        android:text="@string/name_rate"
        android:textAlignment="center"
        android:textSize="30dp"
        android:textColor="@color/black"
        android:layout_marginTop="10dp"/>

    <RatingBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/valorar"
        app:layout_constraintEnd_toEndOf="parent"
        android:numStars="5"/>

    <Button
        android:id="@+id/buttonVotar"
        android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="@string/text_Votar"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toStartOf="@id/buttonSalir"
            android:layout_marginBottom="45dp"/>

        <Button
            android:id="@+id/buttonSalir"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/text_Salir"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@id/buttonVotar"
            android:layout_marginBottom="45dp"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

📌 Aclaraciones:

- **ImageView** líneas 10..16:
  Línea 10 con la propiedad `android:id="@+id/…"` damos un nombre a la vista para posteriormente poder hacer referencia a ella.
  Línea 11 con la propiedad `android:width` indicamos el ancho de esta view, el valor `match_parent` ajusta el ancho al ancho disponible en el contenedor que recibe la view, el valor `wrap_content` ajusta el ancho al tamaño de la imagen.
  Línea 12 con la propiedad `android:height` se define la altura de la view, la definimos en este caso indicando la altura en dp para que puedan reescalarse correctamente en diferentes dispositivos.
  Línea 13 la propiedad `android:marginTop` permite definir el margen superior respecto al contenedor principal, como queremos que esté lo más arriba posible, le quitamos el margen superior.
  Ahora viene la parte que más nos interesa, los anclajes de nuestra imagen.
  Línea 14 con la propiedad `constraintStart_toStartOf:` especificamos que el Start de nuestro componente estará anclado al comienzo de su padre, con eso conseguimos mantenerlo a la izquierda.
  Línea 15 con la propiedad `constraintTop_toTopOf:` especificamos que la parte superior de nuestro componente estará anclada a la parte superior del padre.
  Línea 16 por último, necesitamos cargar la imagen en la view, utilizamos para ello la propiedad `app:srcCompat:` .
- **TextView** líneas 22..27
  En la línea 22 y 23 se definen los anclajes, como se puede apreciar, el texto está de izquierda a derecha así pues, anclaremos el Start al Start del padre, igual que se hizo en

la imagen. La parte superior del texto, tendremos que anclarla a la parte inferior de la imagen para que nos quede justo debajo.
En la propiedad `android:text:` referenciamos un recurso string donde tenemos el texto que queremos que aparezca en el **TextView**.
Con `android:textAlignment:` centramos el texto independientemente de lo grande o pequeña que sea la pantalla del terminal.

- El siguiente **TextView** tiene que ir anclado a su izquierda con el padre y estar por debajo al texto que acabamos de colocar previamente, líneas 33 y 34.
  La única propiedad que no hemos visto anteriormente es `android:marginTop` línea 39 y se utiliza para dejar un margen superior respecto a otros componentes, en este caso, 10dp.
- El siguiente componente es una **RatingBar** tiene que ir anclado a su izquierda con el padre y estar por debajo al **TextView** que tiene como id **valorar** líneas 44..46.
  Con la propiedad `android:numStars:` en la línea 47 elegimos el número de estrellas que deseamos que aparezcan en la **RatingBar**.
  ✋ Con la propiedad `android:stepSize: 0.5` nos permitiría seleccionar media estrella(probar vosotros)
- Veamos el código para incorporar los dos **Button** a la vista.
  Buscaremos que ambos botones estén a la misma altura y lo conseguiremos dando un margen inferior. Además, los colocaremos anclados entre ellos en cuanto al eje horizontal se refiere y anclados a la vez al padre, en este caso cada uno en el lateral que corresponda y respecto a la parte inferior para poder ajustarlo más cómodamente.

Motion Layout

`MotionLayout` es un tipo de diseño que ayuda a administrar las animaciones o el movimiento de las vistas de mi aplicación. Es una subclase de `ConstraintLayout`.

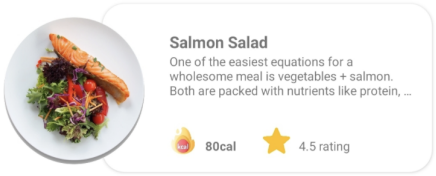
Aquí tenemos un enlace donde se encuentra información ofrecida sobre este tema

Otra lectura no oficial sobre el tema

Empecemos agregando la siguiente dependencia en nuestro `build.gradle(app)`:

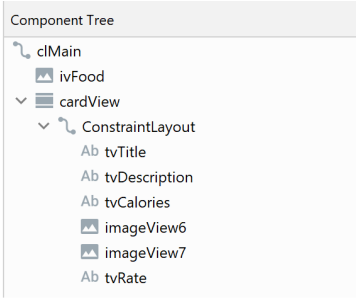
```
implementación 'androidx.constraintlayout:constraintlayout:2.0.0-beta1'
```

Vamos a crear una vista de diseño como la de la siguiente imagen:



Sobre ella definiremos dos transiciones, una sobre la imagen, de forma que cuando la pulsemos se mueva al centro y duplique su tamaño, y otra que haga que la descripción de la receta desaparezca.

El esquema de nuestra vista completa será el siguiente:



El archivo xml correspondiente sería el siguiente:

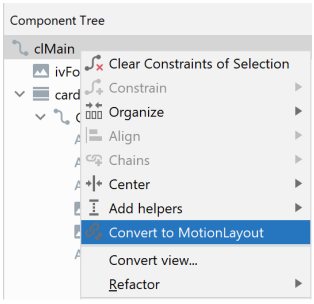
```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/clMain"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/ivFood"
        android:layout_width="150dp"
        android:layout_height="150dp"
        android:layout_marginTop="8dp"
        android:elevation="10dp"
        android:scaleType="fitXY"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/img_salmon_salad" />

    <androidx.cardview.widget.CardView
        android:id="@+id/cardView"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:layout_marginStart="100dp"
        android:layout_marginLeft="100dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        app:cardCornerRadius="20dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <TextView
                android:id="@+id/tvTitle"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_marginStart="60dp"
                android:layout_marginLeft="60dp"
                android:layout_marginTop="24dp"
                android:layout_marginEnd="8dp"
                android:layout_marginRight="8dp"
                tools:text="Salmon Salad"
                android:textSize="18sp"
```

```
            android:textStyle="bold"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
            android:id="@+id/tvDescription"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="60dp"
            android:layout_marginLeft="60dp"
            android:layout_marginEnd="16dp"
            android:layout_marginRight="8dp"
            android:ellipsize="end"
            android:maxLines="3"
            tools:text="One of the easiest equations for a wholesome meal is vegetables +
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/tvTitle" />

        <TextView
            android:id="@+id/tvCalories"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="8dp"
            android:layout_marginLeft="8dp"
            android:layout_marginBottom="16dp"
            tools:text="80cal"
            android:textStyle="bold"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toEndOf="@+id/imageView6" />

        <ImageView
            android:id="@+id/imageView6"
            android:layout_width="24dp"
            android:layout_height="24dp"
            android:layout_marginStart="60dp"
            android:layout_marginLeft="60dp"
            android:layout_marginBottom="16dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:srcCompat="@drawable/ic_calories" />

        <ImageView
            android:id="@+id/imageView7"
            android:layout_width="24dp"
            android:layout_height="24dp"
            android:layout_marginStart="24dp"
            android:layout_marginLeft="24dp"
            android:layout_marginBottom="16dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toEndOf="@+id/tvCalories"
```

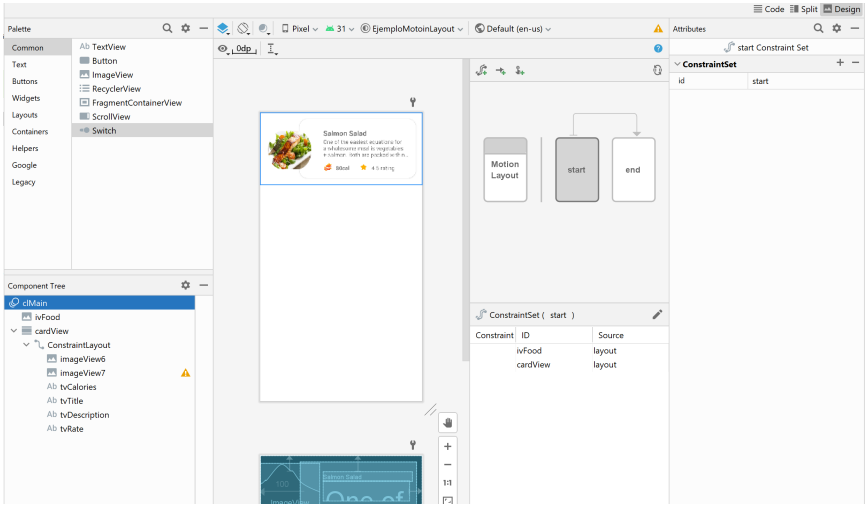
```
                app:srcCompat="@drawable/ic_star" />

            <TextView
                android:id="@+id/tvRate"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="8dp"
                android:layout_marginLeft="8dp"
                android:layout_marginBottom="16dp"
                tools:text="4.5 rating"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toEndOf="@+id/imageView7" />
        </androidx.constraintlayout.widget.ConstraintLayout>
    </androidx.cardview.widget.CardView>
</androidx.constraintlayout.widget.ConstraintLayout >
```

Vamos a utilizar la interfaz géfica para tansformar nuestro `ConstraintLayout` a `MotionLayout`

En nuestro arbol de diseño seleccionamos `clMain` y pulsamos el botón derecho:



En estos momentos el **IDE** se actualizará a la configuración `MotionLayout`

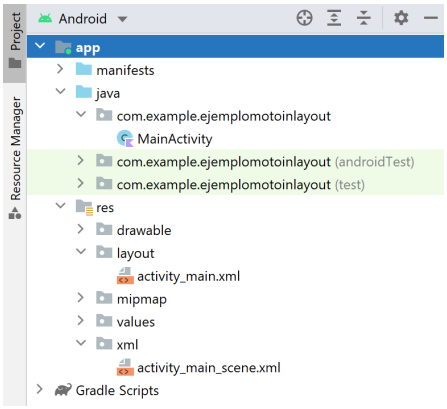


Las modificaciones introducidas en nuestro diseño son las siguientes:

- El diseño de nuestro **XML** pasa a ser `MotionLayout`

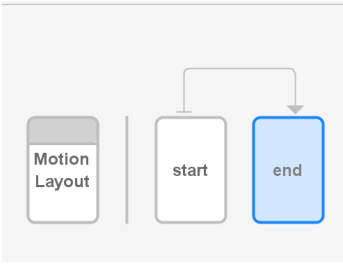
```
<androidx.constraintlayout.motion.widget.MotionLayout xmlns:android="http://schemas.a
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/clMain"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
7 app:layoutDescription="@xml/activity_main_scene">
```

- Además, se ha agregado una carpeta `xml` dentro de la carpeta `res` que contiene el archivo `activity_main_scene.xml`. Este archivo es referenciado en el `MotionLayout` como se ve en la línea 7 anterior.

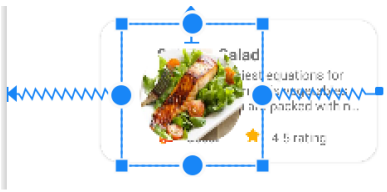


Vamos a establecer una animación sobre la `ImageView` **ivFood**.

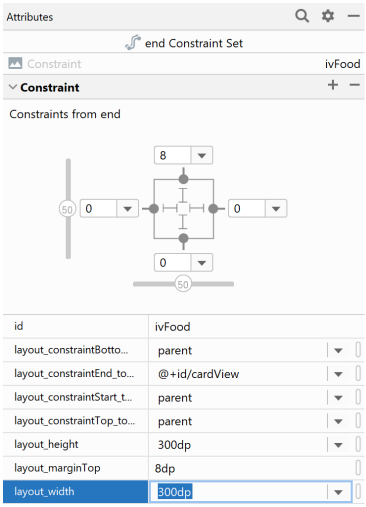
Seleccionamos en nuestro esquema el **ivFood** y en el editor de interfaz de movimiento seleccionamos `end`



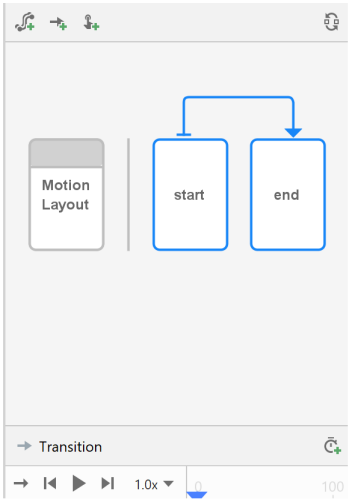
Seleccionamos la restricción final de la imagen y la arrastramos hasta la restricción principal:



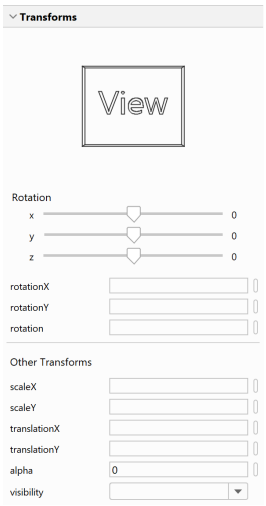
La imagen se colocará en el centro. Ahora en los atributos cambiamos el tamaño de `layout_height` y `layout_width` a 300dp para redimensionar la vista de la imagen y crear el efecto de redimensionamiento.



Si queremos ver como se visualiza esta transición, seleccionamos la misma en la vista y pulsamos el botón de reproducción.



Vamos a proceder ahora a animar el `CardView` para que desaparezca después de la transición de la imagen. Este efecto se consigue asignando el atributo `alpha` a un valor 0 en la categoría `Transforms` del `CardView` (es necesario tener seleccionado en la interfaz el elemento `End`).



Volvemos a pulsar el botón reproducir y vemos el efecto completo sobre la vista.

Por ahora el archivo `activity_main_scene` contiene lo siguiente:

```xml
<?xml version="1.0" encoding="utf-8"?>
<MotionScene
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:motion="http://schemas.android.com/apk/res-auto">

    <Transition
        motion:constraintSetEnd="@+id/end"
        motion:constraintSetStart="@id/start"
        motion:duration="1000">
    </Transition>

    <ConstraintSet android:id="@+id/start">
    </ConstraintSet>

    <ConstraintSet android:id="@+id/end">
        <Constraint
            android:layout_height="300dp"
            motion:layout_constraintStart_toStartOf="parent"
            android:layout_marginTop="8dp"
            motion:layout_constraintTop_toTopOf="parent"
            motion:layout_constraintBottom_toBottomOf="parent"
            motion:layout_constraintEnd_toEndOf="@+id/cardView"
            android:layout_width="300dp"
            android:id="@+id/ivFood" />
        <Constraint
            android:id="@+id/cardView"
            motion:layout_constraintEnd_toEndOf="parent"
            android:layout_width="match_parent"
            android:layout_height="150dp"
            motion:layout_constraintBottom_toBottomOf="parent"
            android:layout_marginBottom="16dp"
            android:layout_marginEnd="16dp"
            android:layout_marginStart="100dp"
            motion:layout_constraintTop_toTopOf="parent"
            android:layout_marginLeft="100dp"
            android:layout_marginRight="16dp"
            motion:layout_constraintStart_toStartOf="parent"
            android:layout_marginTop="16dp"
            android:alpha="0" />
    </ConstraintSet>
</MotionScene>
```

Solamente nos falta programar esta acción cuando se pulsa sobre la imagen, de forma que la transición se iniciará con el **`onClick`** sobre la misma. Para ello añadimos a nuestro archivo **`activity_main_scene`** dentro de la etiqueta **`transition`** el evento del **`onClick`**:

```xml
<OnClick motion:targetId="@id/ivFood"/>
```

Temas y estilos

El diseño visual de una aplicación Android es representado a través de reglas contenidas en Estilos y Temas. Estas herramientas permiten que los programadores y diseñadores generen una interfaz más amigable y personalizada de sus apps, para establecer una identidad que impacte al usuario final.

La forma más sencilla de aplicar un tema es editar el fichero `AndroidManifest.xml` y buscar la propiedad `android:theme` y asignar un estilo concreto. Si esta propiedad se aplica a la etiqueta `application` la aplicación entera adopta el tema seleccionado. Si se aplica a la etiqueta `activity` únicamente esa actividad aplicará el tema. De esto se deduce que se pueden aplicar diferentes temas a cada actividad.

Aquí tenemos un enlace donde se encuentra información ofrecida sobre temas y estilos.

Vamos a utilizar el proyecto **EjemploLinearLayout** para trabajar con estilos y temas.

Editaremos nuestro archivo `themes.xml`:

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.EjemploLinearLayout"
    parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">
              ?attr/colorPrimaryVariant
        </item>
        <!-- Customize your theme here. -->
    </style>

    <style name="Theme.EjemploLinearLayoutPropio"
            parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">#F44336</item>
        <item name="colorPrimaryVariant">#D32F2F</item>
        <item name="colorOnPrimary">@color/black</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">#F06292</item>
        <item name="colorSecondaryVariant">#F48FB1</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">
          ?attr/colorPrimaryVariant
        </item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

> 📌 **Líneas 17 a 29** definimos un nuevo tema modificando los colores y dándole el nombre `Theme.EjemploLinearLayoutPropio` y que hereda con el atributo `parent` del tema por defecto.

Hay que ir ahora al archivo `AndroidManifest.xml` y modificarlo, asignando este nuevo tema a la aplicación (línea 11):

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ejemplolinearlayout">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.EjemploLinearLayoutPropio">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Aunque se definen ambos con la etiqueta `styles` hay que tener en cuenta que son dos cosas distintas. Los temas se aplican a una jerarquía de vistas, mientras que los estilos se definen sobre una vista concreta.

Podríamos cambiar la apariencia de nuestros `EditText` añadiendo un estilo creado por nosotros. Para ello creamos un archivo llamado `styles` en la carpeta `values`:

```xml
<resources>
    <style name="textViewStyle">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#FFAB91</item>
    </style>
</resources>
```

Este estilo puede ser asignado a todos nuestros `TextViews` o solamente a los que nosotros queramos. Se lo asignaremos al que tiene identificador `ìnput_usuario` en el archivo xml de la actividad principal.

```xml
<EditText
        android:id="@+id/input_usuario"
        style="@style/textViewStyle"
        android:hint="Correo"/>
```

Añadir ToolBar a nuestra aplicación

`Toolbar` es el componente que usaremos como reemplazo del `Action Bar`. Este componente nos permitirá:

- Reemplazar el `ActionBar` por una vista personalizada, en la que podremos añadir imágenes, textos, botones, y asociar eventos, como lo hacemos con cualquier otro layout.
- Cambiar la ubicación del típico `ActionBar`. Por ejemplo, podemos usar el `Toolbar` en la parte inferior.

Es usual que nuestro proyecto haga uso del `ActionBar` por defecto. Como nuestra intención es usar el `Toolbar`, entonces debemos deshabilitar el `ActionBar`.

Creamos un nuevo proyecto que llamaremos **EjemploToolBar**. Vamos a nuestro archivo **themes.xml** y asegurarnos de asignar las siguientes propiedades:

```
    <item name="windowNoTitle">true</item>
    <item name="windowActionBar">false</item>
```

Vamos a utilizar para los ejemplos un `CoordinatorLayout`, básicamente nos permitía gestionar interacciones entre sus elementos hijos.

Dentro de este elemento incluiremos la `ToolBar`. Podemos ver en esta imagen los componentes utilizados y cómo quedará la vista:

Veamos como queda el xml de la activity_main:

```
    <?xml version="1.0" encoding="utf-8"?>
    <androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.androi
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/topAppBar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:title="@string/app_name"
14          style="@style/Widget.MaterialComponents.Toolbar.Primary" />
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

En la línea 14 especificamos el estilo de nuestra `ToolBar`

Aquí tenemos un enlace donde se encuentra información ofrecida sobre este elemento en Material Design.

Aplicar Scrolling a la ToolBar

En muchas aplicaciones cuando hacemos un scroll sobre la vista principal interesa ocultar la ToolBar, a este efecto se le llama **scrolling**.

> 📌 Para ejemplificar este efecto seguimos en el proyecto **EjemploToolBar**, copiamos el archivo **activity_main.xml** y lo renombramos como **activity_main_scrolling.xml** (a partir de ahora iremos creando archivos xml nuevos para cada uno de los efectos que desarrollemos sobre la `ToolBar`)

Veamos como quedaría este archivo y posteriormente pasaremos a las explicaciones:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.androi
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:fitsSystemWindows = "true">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/app_bar_layout"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:gravity="center"
        android:theme="@style/Widget.MaterialComponents.Toolbar.Primary">

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/topAppBar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:title="@string/app_name"
            style="@style/Widget.MaterialComponents.Toolbar.Primary"
            app:layout_scrollFlags="scroll|enterAlways"/>
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <TextView
            android:text="@string/hello_world"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            android:textSize="20dp"
            android:fitsSystemWindows = "true" />
    </androidx.core.widget.NestedScrollView>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Para poder aplicar este efecto tenemos que tener en cuenta los siguiente:

- **Línea2:** El `CoordinatorLayout` debe envolver directamente a los objetos que desees relacionar mediante una acción.
- **Línea8:** `android:fitsSystemWindows` este atributo a `true` permite ajustar la ventana de la activity por debajo de la ToolBar.
- **Línea 10:** `AppBarLayout` siempre debe envolver a la `Toolbar` y los demás componentes pertenecientes a ella (como pestañas, imágenes, etc).

- **Línea 23:** La forma en que se afectan los hijos de `AppBar` se determina en `app:layout_scrollFlags`.
- **Línea 26 y 29:** Hacer desaparecer la `AppBar` por scrolling requiere que haya un objeto con scroll marcado con el atributo `app:layout_behavior="@string/appbar_scrolling_view_behavior"`. Este debe declararse por debajo de la `AppBar`. En nuestro ejemplo será un `NestedScrollView` que además incluye un `TextView`

> ✋Los valores que puede tomar `app:layout_scrollFlags` y que controlan el tipo de comportamiento con que desaparece la `Toolbar` son:
>
> - `scroll` : indica que un view desaparecerá al desplazar el contenido.
> - `enterAlways` :vuelve visible al view ante cualquier signo de scrolling.
> - `enterAlwaysCollapsed` : vuelve visible el view solo si se mantiene el scroll en la parte superior del contenido.
> - `exitUntilCollapsed` : desaparece el view hasta que sus dimensiones superen la altura mínima.

Aplicar Collapsing a la ToolBar

Para controlar las reacciones de expansión y contracción de los elementos de vista que se encuentran dentro de un `AppBarLayout` (una imagen, pestañas o cualquier otro elemento) es necesario utilizar un layout especial que envuelva la `ToolBar`, este layout se llama `CollapsingToolbarLayout`.

Añadamos este efecto a nuestra `ToolBar`: (activity_maincollapsing.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.androi
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity"
    android:fitsSystemWindows = "true">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/app_bar_layout"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:gravity="center"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <com.google.android.material.appbar.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">

            <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                app:srcCompat="@drawable/image"
                app:layout_collapseMode="parallax" />

            <com.google.android.material.appbar.MaterialToolbar
                android:id="@+id/topAppBar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:title="@string/app_name" />
        </com.google.android.material.appbar.CollapsingToolbarLayout>>
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <TextView
            android:text="@string/hello_world"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            android:textSize="22dp" />
    </androidx.core.widget.NestedScrollView>
```

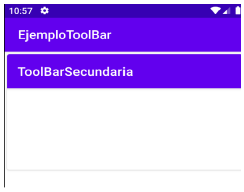
```
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

En nuestro ejemplo añadimos una imagen a la `AppBarLayout` además de la `ToolBar` de forma que aplicaremos un efecto de collapsing a la imagen hasta que desaparezca y solo se vea la `ToolBar`.

> 📌 Aclaraciones:
>
> - **Línea 17** con el elemento `CollapsingToolbarLayout` se nos permitirá realizar este efecto . En la **línea 22** se especifica la propiedad `app:layout_collapsedMode="scroll|exitUntilCollapsed"` , el primer flag indica que todas las vistas que van a hacer un offScreen se mantendrán en la parte superior de la pantalla. Y el segundo flag especifica que la `AppBar` se contraiga hasta que llegue al tamaño de la `ToolBar`
> - **Línea 24** se incluye la imagen que hará el efecto de collapsing. En la **línea 27** con la propiedad `app:layout_collapsedMode="parallax"` damos el efecto de animación que queremos.

Añadir ToolBar a otras partes de la interfaz

Otra ventaja que podemos aprovechar al disponer del control `ToolBar` como un componente independiente, es que podemos utilizarlos en otros lugares de nuestra interfaz, y no siempre como barra de acciones en la parte superior, y evidentemente, tener más de una `ToolBar`.
Podríamos colocar el componente `ToolBar` dentro de una tarjeta, tal y como se ve en la siguiente imagen:



Añadamos este efecto a nuestra `ToolBar`: (activity_toolbarcardview.xml):

```
    <androidx.core.widget.NestedScrollView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical" >

                <androidx.cardview.widget.CardView
                    xmlns:app="http://schemas.android.com/apk/res-auto"
                    android:id="@+id/card_view"
                    android:layout_gravity="center"
                    android:layout_width="match_parent"
                    android:layout_height="200dp"
                    app:cardUseCompatPadding="true"
                    app:cardCornerRadius="4dp">

20                  <com.google.android.material.appbar.MaterialToolbar
                        android:id="@+id/toolbarCard"
                        android:layout_width="match_parent"
                        android:layout_height="?attr/actionBarSize"
                        app:title="@string/app_name_toolbar"
25                      style="@style/Widget.MaterialComponents.Toolbar.Primary"/>
                </androidx.cardview.widget.CardView>
            </LinearLayout>
        </androidx.core.widget.NestedScrollView>
```

📌 Únicamente hemos modificado el **`androidx.core.widget.NestedScrollView`** añadiendo los elementos que se ven. El resto del diseño es idéntico al anterior.