



# Programación multimedia y dispositivos móviles

UT-2. Material Design, animaciones y Gifs

<https://m3.material.io/>



# Material Design

[Material design](#) es un estilo de diseño desarrollado por Google que se presentó al público en 2014, a la par de la versión Android Lollipop. Este se enfoca en los aspectos visuales que corresponden al sistema operativo Android, aunque también es una normativa que se aplica en diferentes páginas web y plataformas.

**“Material Design es un concepto, una filosofía, unas pautas enfocadas al diseño utilizado en Android, pero también en la web y en cualquier plataforma.”**

Material Design es una guía completa para el diseño visual, interactivo y de movimiento en plataformas y dispositivos. A fin de usar material design en tus apps para Android, sigue las pautas definidas en su especificación y usa los nuevos componentes y estilos disponibles en la biblioteca de compatibilidad de material design.

<https://developer.android.com/guide/topics/ui/look-and-feel?hl=es-419>



# Material Design

Es un diseño donde la **profundidad, las superficies, los bordes, las sombras y los colores** juegan un papel principal.

Precisamente este diseño basado en objetos es una manera de intentar aproximarse a la realidad, algo que en un mundo donde todo es táctil y virtual es difícil. Material Design quiere guiarse por las leyes de la física, donde las animaciones sean lógicas, los objetos se superpongan pero no puedan atravesarse el uno al otro y demás.

¿Cómo se traslada esto a Android? Pues básicamente delimitando claramente el tipo de menús, los botones y los tipos de imágenes a elegir.



# Material Design - Actualizaciones

Después de la renovación de 2018, Google comenzó a rediseñar la mayoría de sus aplicaciones en una versión personalizada y adaptada de Material Design llamada Google Material Theme, también denominada "[Material Design 2](#)" que enfatiza mucho los espacios en blanco, las esquinas redondeadas, iconos coloridos, barras de navegación inferiores y utiliza una versión condensada de tamaño especial de la fuente Product Sans patentada de Google llamada Google Sans.

En Google I/O en mayo de 2021, Google anunció un nuevo concepto en Android 12 conocido como "Material You" (también conocido como "[Material Design 3](#)"), que enfatiza una mayor animación, botones más grandes y la capacidad de una interfaz de usuario personalizada. temas que se generarán a partir del fondo de pantalla del usuario.



# Material Design - Sistema de colores (v1 y v2)

En la documentación de Material Design encontramos varias páginas que son muy útiles para entender mejor cómo personalizar el diseño de nuestras apps en Android.

En primer lugar vamos al [sistema de colores](#). Cuando vimos temas y estilos, definimos un nuevo tema en el que dimos valor a 3 colores:

- ❏ **colorPrimary**. Es el color principal de la aplicación, y se utilizará entre otras cosas como color de fondo de la action bar.
- ❏ **colorOnPrimary**. Color utilizado para el texto y los iconos que se muestran encima del color primario.
- ❏ **colorPrimaryVariant**. Se utiliza para distinguir dos elementos de la aplicación mediante el color principal, como la barra superior de la aplicación y la barra del sistema.



## Material Design - Sistema de colores (v1 y v2)

En la página del [sistema de colores](#) podemos ver que se pueden definir hasta 9 colores así como explica el uso de estos colores en las diferentes partes de una aplicación.

De la misma forma da [pautas para definir los temas personalizados](#) y proporciona una herramienta, la paleta de colores, dónde podemos por un lado en función del color principal que queramos tener en la app, ver cuales deben ser sus variantes.



# Material Design - Nuevos diseños

De forma general los proyectos que creamos se apoyan en temas de Material Design 3 (más redondeados, más sombras, más profundidad) y a los tipos de letra.

Adicionalmente, a nivel de controles, MD3 nos ofrece algunos estilos que podemos implementar fácilmente en nuestros layouts si lo deseamos.

Como siempre, es la propia documentación de Google la que nos da luz:

<https://github.com/material-components/material-components-android/blob/master/docs/getting-started.md>

<https://github.com/material-components/material-components-android/blob/master/docs/theming/Color.md>

<https://material.io/blog/material-theme-builder>



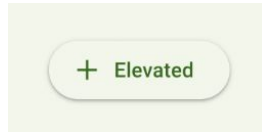
## Material Design - Nuevos diseños

Con el cambio de tema en nuestra aplicación por uno de los nuevos temas, ya podemos comprobar que cambian estilos a los controles que tengamos.

Pero además, y siempre a partir de tener un tema de Material Design 3, la propia documentación de Google te ofrece la posibilidad de definir mejor y personalizar más todos los [controles o componentes de la aplicación](#).

La guía nos indica qué diseños podemos conseguir. En cada apartado, pulsando en el apartado de MC Android nos llevará a GitHub nos indicará cómo personalizar los controles, de qué estilo heredar, qué propiedades podemos configurar....

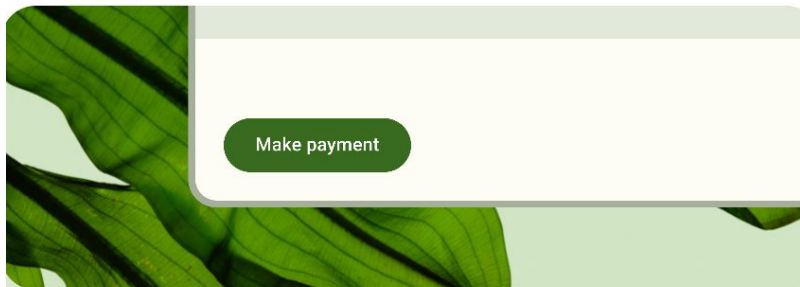
Vamos a empezar configurando en primer lugar un botón elevado.



Para ello, accedemos a la parte de componentes de “Common Buttons”, Resources y MC Android

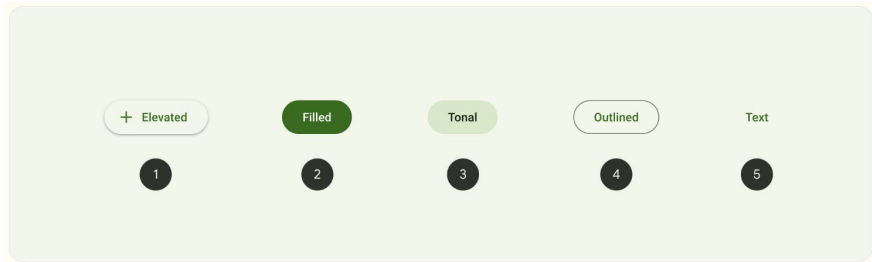


# Material Design - Nuevos diseños - Botones



## Common Buttons

Buttons help people initiate actions, from sending an email, to sharing a document, to liking a post.



## Resources

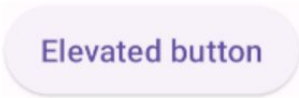
Type	Link	Status
Design	<a href="#">Design kit</a> (Figma)	Available
	<a href="#">M2 guideline – Buttons</a>	Available
	<a href="#">MDC- Android</a>	Available



# Material Design - Nuevos diseños - Botones

En un proyecto nuevo, o en uno que ya tengamos, debemos cambiar el manifiesto para heredar de un tema de Material Design 3. Yo he heredado de “Theme.Material3.DayNight”.

En el proyecto ponemos un botón normal con el texto que queramos. Dentro de la página de GitHub, vemos que para “elear” un botón simplemente debe heredar el estilo.



Elevated button

In the layout:

```
<Button
    style="@style/Widget.Material3.Button.ElevatedButton"
    android:id="@+id/elevatedButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Elevated button"
/>
```

# Material Design - Nuevos diseños - Botones

Para añadir un icono al botón elevado, en la misma página vemos que el estilo del botón debe ser otro. Previamente debemos [haber añadido como recurso en nuestra aplicación](#) el icono a añadir. Yo he añadido el que hay en recursos que lo he descargado, nuevamente, de [Material Design](#) (Material Design proporciona además una guía de como diseñar iconos)



+ Elevated button

In the layout:

```
<Button
    style="@style/Widget.Material3.Button.ElevatedButton.Icon"
    ...
    app:icon="@drawable/ic_add_24dp"
/>
```



♥ Elevated button



## Material Design - Nuevos diseños - Botones

Hay que tener en cuenta que la parte programática de estos botones no ha cambiado nada. Son botones y para capturar sus eventos y darle lógica a nuestra aplicación, debemos tratarlos como cualquier otro botón. Esto es, si queremos codificar el evento click podemos hacer un método y llamarlo desde la propiedad de “onclick”.

Si queremos codificar cualquier otro evento, tendremos que instanciar un objeto de tipo Button, castear el botón para el que queramos capturar el evento (findViewById) y poner el escuchador del evento que queramos capturar.



# Material Design

Material Design te orienta para el diseño de tu app y te facilita desde el uso de sus propios estilos hasta el desarrollo de estilos propios, con imágenes / iconos propios...

Seguro que es mucho más potente de lo que ahora somos capaces de hacer, pero al menos en este punto nos facilita diseños modernos con muy poco esfuerzo.



## Animaciones y GIFs

El uso de imágenes animadas y/o GIFs no es inmediato en Android. Puedes “pegar” un GIF a un ImageView, pero este GIF no se verá animado, sino como una imagen estática.

Android Studio ofrece “de caja” la posibilidad de hacer animaciones y hay librerías de terceros que te permiten incrustar GIFs y que estos se animen. Vamos a ver las 2 opciones.



# GIFs

Ya he comentado, que el uso directo de GIFs no es posible. Pero hay librerías de terceros que te permiten añadir objetos similares al ImageView pero que si tratan los GIFs.

Hay varias opciones. En los apuntes dejo una que he probado y funciona y vamos a utilizar otra.

Vamos a utilizar una librería de Google llamada GLIDE

<https://github.com/bumptech/glide/blob/master/README.md>

Esta librería la utilizaremos más adelante para “pintar” en nuestra aplicación imágenes que no tenemos en el proyecto sino que se accede a ellas vía web.

# GIFs

Hacemos un proyecto con un ImageView.

En nuestro proyecto añadimos como recurso la imagen de los dados rodando que está en la carpeta de recursos: dado-imagen-animada-0092. La asignamos al ImageView y ejecutamos.

La imagen se vé, pero no en movimiento:







## GIFs

Vamos a incorporar a nuestro proyecto la librería de Glide. En el GitHub teníamos la última versión de esta librería:

Vamos a nuestro fichero `build.gradle` y en la parte de dependencias añadimos la siguiente línea:

```
implementation ("com.github.bumptech.glide:glide:4.16.0")
```

A la derecha nos aparecerá una bombilla para sincronizar estas librerías. Le damos.



# GIFs

La propia documentación de GitHub nos indica como utilizar esta librería. Es bastante sencillo:

Casteamos en  
ImageView

```
MyImage = findViewById(R.id.iv_gif)  
Glide.with( activity: this).load(R.drawable.dado_imagen_animada_0092).into(MyImage);
```

Usamos la clase Glide para indicar la actividad, la ruta de la imagen a cargar (en este caso está en el proyecto pero podría ser una url, y le indicamos el imagenview donde queremos ver el Gif.



# GIFs

También hay librerías de terceros que te permiten añadir objetos similares al ImageView pero que si tratan los GIFs. Yo he probado esta pero seguro que hay más

Vamos a nuestro fichero build.gradle y en la parte de dependencias añadimos la siguiente línea:

```
implementation ("pl.droidsonroids.gif:android-gif-drawable:1.2.28" )
```

A la derecha nos aparecerá una bombilla para sincronizar estas librerías. Le damos.



# GIFs

Luego en mi activity\_main.xml, ponemos la parte de código XML y añadimos un objeto GifImageView

```
<pl.droidsonroids.gif.GifImageView  
  
    android:layout_width="233dp"  
  
    android:layout_height="125dp"  
  
    android:layout_marginStart="68dp"  
  
    android:layout_marginTop="208dp"  
  
    android:src="@drawable/dado_imagen_animada_0092"  
  
    app:layout_constraintStart_toStartOf="parent"  
  
    app:layout_constraintTop_toTopOf="parent" />
```



# Animaciones

Luego, en el main\_activity de nuestra actividad instanciamos clases para el ImageView y el una clase “nueva”: AnimationDrawable

```
ImageView myiV1;
```

```
AnimationDrawable miTragaperras;
```

Ahora “castearemos” y le asignaremos como background, el xml con la secuencia de imágenes. También a la clase que va a gestionar la animación le asignamos el control del background del ImageView.

```
myiV1 = (ImageView) findViewById(R.id.iV1);
```

```
myiV1.setBackgroundResource(R.drawable.tragaperras);
```

```
miTragaperras = (AnimationDrawable) myiV1.getBackground();
```

# Animaciones

Para incrustar las animaciones que trae de caja Android, la filosofía es:

- Tengo un imageView
- Tengo un conjunto de imágenes estáticas
- Tengo un XML que indica el orden y el tiempo en el que estas imágenes pasan. También si la animación pasa una vez o pasa en un bucle infinito (hasta que la paremos).
- Hay una clase java del SDK de Android que gestiona la animación.

Vamos a hacerlo. Necesitamos un proyecto, con un imageView y un botón.

En este proyecto, vamos a añadir como recursos gráficos los jpg que tenemos dentro de la carpeta de recursos del aula virtual, dentro del rar de “Animación”.



Parar

# Animaciones

También vamos a añadir dentro de res/drawables (copy&paste) el fichero tragaperras.xml que está la carpeta de recursos. En este fichero tenemos las imágenes que vamos a animar, en que orden y el tiempo (en milisegundos). También si la animación se ejecuta una sola vez o se ejecuta de forma indefinida

```
<?xml version="1.0" encoding="UTF-8" ?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
    <item android:drawable="@drawable/bar_done" android:duration="100" />
    <item android:drawable="@drawable/cherry_done" android:duration="100" />
    <item android:drawable="@drawable/lemon_done" android:duration="100" />
    <item android:drawable="@drawable/orange_done" android:duration="100" />
    <item android:drawable="@drawable/sevent_done" android:duration="100" />
    <item android:drawable="@drawable/triple_done" android:duration="100" />
    <item android:drawable="@drawable/waternelon_done" android:duration="100" />
</animation-list>
```



# Animaciones

Definimos un objeto de tipo `AnimationDrawable`

```
lateinit var miTragaperras: AnimationDrawable
```

Asignamos al `imageView` el xml de tragaperras con la secuencia de las imágenes a mostrar

```
MyImage.setBackgroundResource(R.drawable.tragaperras)
```

Y asignamos al objeto de tipo `AnimationDrawable` el background del `imageView`

```
miTragaperras = MyImage.background as AnimationDrawable
```





# Animaciones

A partir de aquí, la animación podremos arrancarla y pararla con los métodos `start` y `stop` de la clase de animación:

```
miTragaperras.start(); / miTragaperras.stop();
```

Podemos poner el `start` en click del botón y el `stop` en el `longclick` (por ejemplo) para comprobar que efectivamente funcionan las animaciones.



# Animaciones

Para acabar, si tienes un GIF y quieres utilizar una animación, hay diversas aplicaciones que te separan en frames los GIFs.

<https://alejandroacedo.wordpress.com/2016/06/07/tutorial-7-anadir-gif-en-android-studio/>

<https://developer.android.com/guide/topics/graphics/drawable-animation?hl=es-419#java>