

Tema 5. Interfaz de Usuario I

[Descargar estos apuntes](#)

Índice

1. [ViewBinding](#)
2. [Botones](#)
 1. [Filled, elevated button](#)
 2. [Filled, unelevated button](#)
 3. [Outlined button](#)
 4. [Text button](#)
 5. [Icon button](#)
 6. [Manejar eventos botón](#)
 7. [Floating Action Button](#)
 8. [Toggle Button](#)
3. [ImageView](#)
4. [Text Fields](#)
 1. [TextView](#)
 2. [EditText](#)
 3. [Floating Labels](#)
 4. [AutoComplete TextView](#)
5. [CheckBox](#)
6. [RadioButton](#)
7. [Switches](#)
8. [Sliders](#)
9. [Chips](#)
10. [Toast](#)
11. [SnackBar](#)
 1. [SnackBar con acción](#)
 2. [Descartar SnackBar](#)

ViewBinding

Si ya eres conocedor de Android, conocerás findViewById, que permite vincular las vistas de la aplicación con las clases, y así poder acceder a determinadas propiedades de los widgets de las vistas.

Desde la versión 3.6 de Android Studio, es posible utilizar View Binding para enlazar los elementos de los layouts con las clases.

Si estamos trabajando con una versión de Android Studio superior a la 4.0 la configuración del archivo `build.gradle` a nivel de `Module:app` es la siguiente:

```
android {  
    ...  
    viewBinding {  
        enabled = true  
    }  
}
```

Una vez habilitada la vinculación de vista para un proyecto, por cada archivo xml se generará una clase de vinculación al mismo, que será utilizado para hacer referencias a las vistas del mismo.

si nuestro archivo se llama `activity_secundaria.xml` la clase de vinculación generada se llamará `ActivitySecundariaBinding`.

Para poder utilizar la vinculación de vistas hay que hacer lo siguiente en el método `onCreate()` de la actividad:

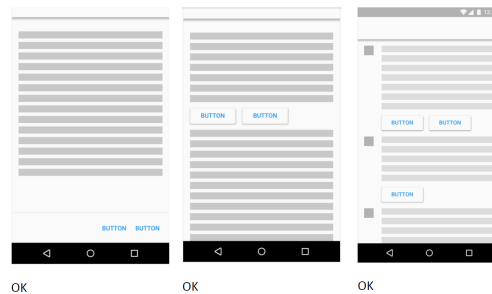
```
private lateinit var binding: ActivitySecundariaBinding  
  
override fun onCreate(savedInstanceState: Bundle) {  
    super.onCreate(savedInstanceState)  
    binding = ActivitySecundariaBinding.inflate(layoutInflater)  
    val view = binding.root  
    setContentView(view)  
}
```

A partir de ahora podemos hacer referencia a las vistas de nuestro xml. Lo veremos a continuación tal y como vayamos viendo distintos componentes.

Botones

Los botones forman una parte funcional muy importante en cualquier aplicación. Existen varios tipos estándar de botones: **Floating Action Button** (botón circular con una acción muy concreta en nuestra aplicación), **Filled, elevated button** (botón con relieve con efecto de pulsación y fondo color), **Filled, unelevated button** (botón con relieve sin efecto de pulsación y color fondo), **Outlined button** (botón fondo transparente y borde), **Text button** (tiene un fondo transparente con texto en color) y **Icon button** (incorpora un icono al botón, puede ser con texto y sin texto).

La elección de un botón u otro va a depender de la importancia que se le quiera dar al mismo y de la disposición del mismo.



[Aquí tenemos un enlace donde se encuentra información sobre este componente que ofrece Material Design](#)

Para definir un botón de un tipo u otro vamos a asignar un estilo al botón.

Vamos a crear un proyecto nuevo que llamaremos **EjemploBotones** y donde iremos incorporando los distintos componentes que veremos a continuación.

Filled, elevated button

Botón elevado con fondo y efecto de pulsación. Se utiliza para acciones finales del tipo **Guardar** o **Confirmar**. Si no se especifica ningún atributo de estilo para este elemento, este es el estilo que se utilizará por defecto.

Vamos a utilizar como contenedor principal un **LinearLayout** con orientación **vertical**.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <com.google.android.material.button.MaterialButton
        android:id="@+id/material_button_salir"
        style="@style/Widget.MaterialComponents.Button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_label_salir"
        android:layout_gravity="center"/>
</LinearLayout>

```



El **colorPrimary** de su tema proporciona el fondo predeterminado para el botón.

Filled, unelevated button

```

<com.google.android.material.button.MaterialButton
    android:id="@+id/material_button_flat"
    style="@style/Widget.MaterialComponents.Button.UnelevatedButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_label_flat"
    android:layout_gravity="center"/>

```

Outlined button

```

<com.google.android.material.button.MaterialButton
    android:id="@+id/material_button_outlined"
    style="@style/Widget.MaterialComponents.Button.OutlinedButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_label_outlined"
    android:layout_gravity="center"/>

```

Text button

```

<com.google.android.material.button.MaterialButton
    android:id="@+id/material_button_text"
3 style = "@style/Widget.MaterialComponents.Button.TextButton"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "@string/button_label_text"
    android:layout_gravity="center"/>

```

Icon button

```

<com.google.android.material.button.MaterialButton
    android:id="@+id/material_button_icon"
3 style = "@style/Widget.MaterialComponents.Button.Icon"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
6 app:icon = "@drawable/ic_email_black_24dp"
    android:layout_gravity="center"/>

```

✧ Con el atributo **app:icon** (línea 6) indicamos el recurso que hay que cargar en el botón. Hay una serie de propiedades que podemos usar cuando usamos **IconButton**. El **app:iconSize="15dp"** para ancho y alto del icono. Podemos agregar el espacio entre el icono y el texto con **app:iconPadding="10dp"**. También hay una opción para cambiar el color del icono con **iconTint** atributo. Para cambiar la posición del icono, utilice el atributo **iconGravity**.

✧ [Aquí tenemos un enlace donde descargar iconos que ofrece Google](#)

Para incorporar texto al botón con icono:

```

<com.google.android.material.button.MaterialButton
    android:id="@+id/material_button_icon_text"
    style = "@style/Widget.MaterialComponents.Button.TextButton.Icon"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    app:icon = "@drawable/ic_email_black_24dp"
    android:text = "@string/button_label_icon"
    android:layout_gravity="center"
9 app:iconGravity="end"/>

```

✧ Con el atributo **app:iconGravity** (línea 9) indicamos la posición del icono respecto al texto.

Manejar eventos botón

Para gestionar la pulsación realizada sobre un botón tenemos varias posibilidades: usar atributo `onClick` del botón, un escuchador anónimo o implementar la interfaz `View.OnClickListener`. Lo más eficiente es implementar la interfaz, aunque encontraremos mucha documentación que implementa el listener anónimo.

Veamos el código de nuestra actividad:

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.materialButtonFlat.setOnClickListener(){
            Toast.makeText(applicationContext,"Pulsaste el botón FLAT",Toast.LENGTH_LONG)
        }

        binding.materialButtonOutlined.setOnClickListener(this)
        binding.materialButtonText.setOnClickListener(this)
        binding.materialButtonIcon.setOnClickListener(this)
        binding.materialButtonIconText.setOnClickListener(this)
    }

    override fun onClick(v: View?) {
        when (v) {
            binding.materialButtonOutlined-> visualizarToast("OUTLINED")
            binding.materialButtonText-> visualizarToast("TEXT")
            binding.materialButtonIcon-> visualizarToast("ICON")
            binding.materialButtonIconText-> visualizarToast("ICON CON TEXT")
            else -> {
                visualizarToast("OTRA COSA MARIPOSA")
            }
        }
    }
}

fun visualizarToast(view: View) {
    Toast.makeText(this,"Pulsaste el botón RAISED",Toast.LENGTH_LONG).show()
}

fun visualizarToast(s: String) {
    Toast.makeText(this,"Pulsaste el botón "+ s,Toast.LENGTH_LONG).show()
}
```

 **Manejar eventos botón:**

- **android:onClick** . Este atributo incorporado en la definición de un botón permite asignar un método público que se ejecutará cuando este se pulse (línea 32). Vamos a aplicarlo en nuestro primer botón **raised_button** y visualizaremos un **toast** (línea 33) indicando la pulsación del mismo:
- **listener anónimo** . Programamos el escuchador anónimo directamente sobre la vista e implementamos el método **setOnClickListener()** (línea 10). Incluimos este código en el **onCreate()** de la actividad:
- implementando interfaz **setOnClickListener** en la actividad (línea 1), registrar las vistas que van a ser objeto de implementación a través de la interfaz (líneas 14-17) y gestionar la pulsación (líneas 20-30 para el resto de botones). La línea 36 es una función para visualizar un **string** en un **toast** , de forma que para cada botón podamos sacar un texto personalizado del botón pulsado.

Floating Action Button

Toggle Button

ImageView

Text Fields

TextView

EditText

Floating Labels

Filled text fields

Outlines text fields

AutoComplete TextView

CheckBox

RadioButton

Switches

Sliders

Chips

Toast

SnackBar

SnackBar con acción

Descartar SnackBar



✎ Aclaraciones:

- **ImageView** líneas 10..16:




```
<?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

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

..

✎ EjercicioPropuestoCardView

 Aclaraciones:

