Apuntes

Resumen. Interfaz de Usuario

Descargar estos apuntes

Índice

ViewBinding

La configuración del archivo `build.gradle` a nivel de `Module:app` es la siguiente:

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

> 📌 Hay que recordar sincronizar gradle.

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)
    }
```

Botones

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).

Buttons Material Design

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

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.

```
style="@style/Widget.MaterialComponents.Button"
```

Filled, unelevated button

```
style="@style/Widget.MaterialComponents.Button.UnelevatedButton"
```

Outlined button

```
<com.google.android.material.button.MaterialButton
 style="@style/Widget.MaterialComponents.Button.OutlinedButton"
 ....
/>
```

Text button

```
<com.google.android.material.button.MaterialButton
style = "@style/Widget.MaterialComponents.Button.TextButton"
...
/>
```

Icon button

```
 <com.google.android.material.button.MaterialButton
style = "@style/Widget.MaterialComponents.Button.Icon"
    app:icon = "@drawable/ic_email_black_24dp"
....
/>
```

Descargar iconos Google.

Manejar eventos botón

Para gestionar la pulsación realizada sobre un botón tenemos varias posibilidades: usar atributo `onClick` del botón en el **layout.xml**, un escuchador anónimo o implementar la interfaz `View.OnClickListener`.

```
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(){
            ...
        }

        binding.materialButtonOutlined.setOnClickListener(this)

    }

    override fun onClick(v: View?) {
        when (v) {
            binding.materialButtonOutlined-> ...
        }
    }

   }
```

```
       <com.google.android.material.button.MaterialButton
            android:id="@+id/material_button_raised"
            style="@style/Widget.MaterialComponents.Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/button_label_raised"
            android:layout_gravity="center"
            android:onClick="mifunción"/>
```

Toggle Button

`app:singleSelection` hacer que solamente haya uno seleccionado con la propiedad.

```
    <com.google.android.material.button.MaterialButtonToggleGroup
        android:id="@+id/toggleGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:checkedButton="@id/btnAndroid"
        app:singleSelection="true">
        <Button
            android:id="@+id/btnAndroid"
            style="@style/Widget.MaterialComponents.Button.OutlinedButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Android" />
        <Button
            android:id="@+id/btniOS"
            style="@style/Widget.MaterialComponents.Button.OutlinedButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="iOS" />
    </com.google.android.material.button.MaterialButtonToggleGroup>
```

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding

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

        binding.toggleGroup.addOnButtonCheckedListener { group, checkedId, isChecked ->
            if (isChecked) {
                if (checkedId == R.id.btnAndroid) {
                    ...
                }
            }
        }

        //listener del botón
        binding.btniOS.setOnClickListener {
            ...
        }
    }
}
```

Floating Action Button (FAB)

```
    <com.google.android.material.floatingactionbutton.FloatingActionButton
           android:id="@+id/fab"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:src="@drawable/ic_add"
           app:fabSize="normal"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

binding.addFab.shrink()
binding.addFab.extend()
binding.imageFab.show()
binding.imageFab.hide()
binding.imageText.visibility= View.VISIBLE
binding.imageText.visibility= View.GONE

Información sobre FAB y transiciones.

TextInputLayout etiquetas flotantes

`TextInputLayout` puede tener dos aspectos distintos que se definen con un estilo `FilledBox` y `OutlinedBox`.

Text Fields Material Design.

```
<com.google.android.material.textfield.TextInputLayout
    style="@style/Widget.MaterialComponents.TextInputLayout.FilledBox"
    android:id="@+id/text_input_layout_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Name"
    app:startIconDrawable="@drawable/account"
    app:helperText="Required"
    android:layout_weight="1"
    app:counterEnabled="true"
    app:counterMaxLength="20">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/nameText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:singleLine="true" />
</com.google.android.material.textfield.TextInputLayout>
```

AutoComplete TextView

La lista de sugerencias se obtiene de un adaptador de datos y aparece solo después de un número determinado de caracteres definido por el `completionThreshold`. Puede estar envuelto en un `TextInputLatout` con un estilo `ExposedDropdownMenu` que permite con la aparición de un icono flecha el desplegar la lista.

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/text_input_layout_pais"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.ExposedDropdownMenu"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Pais"
    app:boxCornerRadiusTopEnd="10dp"
    app:boxCornerRadiusTopStart="10dp">

    <com.google.android.material.textfield.MaterialAutoCompleteTextView
        android:id="@+id/paisText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textAutoComplete"
        android:singleLine="true"
        android:completionThreshold="2"/>
</com.google.android.material.textfield.TextInputLayout>
```

Para poder rellenar la lista de selección para el control `AutoCompleteTextView` necesitamos definir un archivo de recursos en `res/values/` que llamamos `country.xml` donde definiremos los valores de la lista:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="paises">
        <item></item>
        <item>España</item>
        <item>Islas Salomon</item>
    </string-array>
</resources>
```

Finalmente en el método `onCreate()` de la actividad:

```
val adapter = ArrayAdapter(this,
    android.R.layout.simple_list_item_1,
    resources.getStringArray(R.array.paises))
    binding.paisText.setAdapter(adapter)
```

Controles de selección

CheckBox

```
<CheckBox
android:id="@+id/check_box1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="true"
android:text="Microphone access"/>
```

```
binding.checkBox1.setOnCheckedChangeListener{
        buttonView, isChecked ->
        if (isChecked){

        }else{

        }
}
```

RadioButton

```
<RadioGroup
android:id="@+id/radioGroup"
android:checkedButton="@+id/radio_button_1"
android:layout_width="match_parent"
android:layout_height="wrap_content">
    <RadioButton
    android:id="@+id/radio1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="Allow notifications"/>
    <RadioButton
    android:id="@+id/radio2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checked="true"
    android:text="Turn off notifications"/>
</RadioGroup>
```

```
binding.radioGroup.setOnCheckedChangeListener {
    _ , checkedId ->
        when(checkedId){
            binding.radio1.id-> ...
            binding.radio2.id-> ...
    }
}
```

Switches

```
<com.google.android.material.switchmaterial.SwitchMaterial
android:id="@+id/switch1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:checked="true"
android:text="Cellular data" />
```

```
binding.switch1.setOnCheckedChangeListener {
    buttonView, isChecked->
        if (isChecked){

        }else{

        }
}
```

SnackBar

```
Snackbar.make(findViewById(R.id.constraintLayout),
"La acción seleccionada se ha realizado correctamente",
Snackbar.LENGTH_SHORT).show()
```

SnackBar con acción

```
Snackbar.make(
    findViewById(R.id.constraintLayout),
    "La acción seleccionada se ha realizado corrctamente",
    Snackbar.LENGTH_INDEFINITE)
    .setAction("ACEPTAR") {
    // Responds to click on the action
    }
    .show()
```

Descartar SnackBar

Para que la `snackBar` aparezca por debajo y provoque un desplazamiento hacia arriba, es necesario que el contenedor principal sea un `CoordinatorLayout`.

Además el hecho de que utilicemos este elemento nos va a proporcionar también una acción adicional sobre la `snackBar`, concretamente el poder descartarla con un gesto.

Sliders

Los controles deslizantes pueden usar iconos en ambos extremos de la barra para representar una escala numérica o relativa. El rango de valores o la naturaleza de los valores, como el cambio de volumen, se pueden indicar con iconos.

Pueden ser continuos (permiten seleccionar un valor aproximado subjetivo) o discretos (permiten seleccionar un valor exacto).

```
    <!-- Continue slider -->
    <com.google.android.material.slider.Slider
        android:id="@+id/continueSlider"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:valueFrom="0.0"
        android:valueTo="100.0"/>

    <!-- Discrete slider -->
    <com.google.android.material.slider.Slider
        android:id="@+id/discrereeSlider"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:valueFrom="0.0"
        android:valueTo="100.0"
        android:stepSize="5"
        android:value="30"/>
```

Podemos atender los cambios sobre el `slider` con los escuchadores:

```
slider.addOnSliderTouchListener(object : Slider.OnSliderTouchListener {
    override fun onStartTrackingTouch(slider: Slider) {
        // Responds to when slider's touch event is being started
    }

    override fun onStopTrackingTouch(slider: Slider) {
        // Responds to when slider's touch event is being stopped
    }
})

slider.addOnChangeListener { slider, value, fromUser ->
    // Responds to when slider's value is changed
}
```

Diálogos

Alert dialog

```
MaterialAlertDialogBuilder(context)
        .setTitle(resources.getString(R.string.title))
        .setMessage(resources.getString(R.string.supporting_text))
        .setNeutralButton(resources.getString(R.string.cancel)) { dialog, which ->
            // Respond to neutral button press
        }
        .setNegativeButton(resources.getString(R.string.decline)) { dialog, which ->
            // Respond to negative button press
        }
        .setPositiveButton(resources.getString(R.string.accept)) { dialog, which ->
            // Respond to positive button press
        }
        .show()kotlin
```

Simple dialog

```
val items = arrayOf("Item 1", "Item 2", "Item 3")

MaterialAlertDialogBuilder(context)
        .setTitle(resources.getString(R.string.title))
        .setItems(items) { dialog, which ->
             // Respond to item chosen
         }
        .show()
```

Confirmation dialog

```
val singleItems = arrayOf("Item 1", "Item 2", "Item 3")
val checkedItem = 1

MaterialAlertDialogBuilder(context)
        .setTitle(resources.getString(R.string.title))
        .setNeutralButton(resources.getString(R.string.cancel)) {
            dialog, which ->
                // Respond to neutral button press
        }
        .setPositiveButton(resources.getString(R.string.ok)) {
            dialog, which ->
                // Respond to positive button press
        }
        // Single-choice items (initialized with checked item)
        .setSingleChoiceItems(singleItems, checkedItem) {
            dialog, which ->
                // Respond to item chosen
         }
        .show()
```

Es posible también seleccionar más un elemento de los presentados en el diálogo. Para implementar este tipo de diálogo:

```
val multiItems = arrayOf("Item 1", "Item 2", "Item 3")
val checkedItems = booleanArrayOf(true, false, false, false)

MaterialAlertDialogBuilder(context)
        ...
        //Multi-choice items (initialized with checked items)
        .setMultiChoiceItems(multiItems, checkedItems) {
            dialog, which, checked ->
                // Respond to item chosen
        }
        .show()
```

Full-screen dialog

Los cuadros de diálogo de pantalla completa son los únicos cuadros de diálogo sobre los que pueden aparecer otros cuadros de diálogo.

No existe una implementación de `Material Design` específica de un diálogo de pantalla completa. Podemos implementarlo usando un `DialogFragment`.

DataPicker

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)

        binding.dateText.setOnClickListener {
            val datePicker = MaterialDatePicker.Builder.datePicker()
                .setTitleText("Fecha de nacimiento")
                .setSelection(MaterialDatePicker.todayInUtcMilliseconds())
                .build()

            datePicker.addOnPositiveButtonClickListener {
                binding.dateText.setText(datePicker.headerText.toString())

            }
            datePicker.show(supportFragmentManager,"")
        }
    }
}
```

Se puede añadir dos campos de fecha al diseño con esquema **Fecha de Entrada** y **Fecha de salida** similar a los utilizados en una reserva de hotel, usando un `MaterialDatePicker.Builder.dateRangePicker` para su implementación

Progress indicators

```
<!-- Linear progress indicator -->
<com.google.android.material.progressindicator.LinearProgressIndicator
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
<!-- Circular progress indicator -->
<com.google.android.material.progressindicator.CircularProgressIndicator
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

A cualquiera de los dos tipos se le puede decir que es indeterminado con la propiedad `android:indeterminate="true"`, por defecto es determinado.