Apuntes

Descargar estos apuntes

Tema 6. Interfaz de Usuario II

Índice

Sliders

Permiten a los usuarios ver y seleccionar un valor (o rango) del intervalo mostrado a lo largo de una barra. Son ideales para ajustar configuraciones como el volumen y el brillo, o para aplicar filtros de imagen. Al interactuar con un control deslizante, los cambios deben reflejarse inmediatamente al usuario.

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"
6           android:valueFrom="0.0"
7           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"
16          android:stepSize="5"
17          android:value="30"/>
```

📌 Aclaraciones:

- **Línea 6** con el atributo `android:valueFrom` establecemos el valor mínimo del `slider`
- **Línea 7** con el atributo `android:valueTo` establecemos el valor máximo.
- **Línea 16** con el atributo `android:stepSize` establecemos el valor del incremento del `slider` discreto.
- **Línea 17** con el atributo `android:value` podemos establecer un valor inicial al `slider` ya sea éste continuo o discreto.

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
}
```

EjercicioPropuesto:

- Realizar un nuevo proyecto que permita modificar el color de fondo de nuestro `layout` a través de tres `slider` que permitan variar el **RGB** del color del mismo.

Chips

Uno de los componentes más atractivos de la librería `Material Design` es el `Chip`. Existen cuatro tipos: de entrada, de filtro, de elección y de acción.

Los `Chips` se utilizan habitualmente agrupados. Para poder hacerlo de forma eficiente se recomienda la utilización del componente `ChipGroup` que permite patrones de comportamiento sobre la vista.

Los cambios de un chip se pueden observar así:

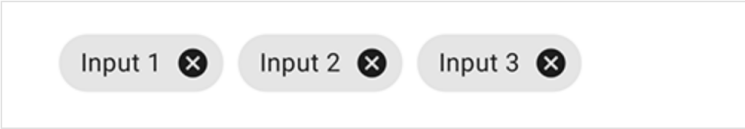
```
chip.setOnClickListener {
   // Responds to chip click
}

chip.setOnCloseIconClickListener {
    // Responds to chip's close icon click if one is present
}

chip.setOnCheckedChangeListener { chip, isChecked ->
    // Responds to chip checked/unchecked
}
```

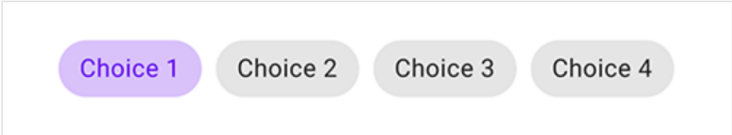
Input chips

Como caraterísticas generales pueden contener un icono de chip opcional, un icono de cierre opcional y opcionalmente se pueden marcar.



Choice Chip

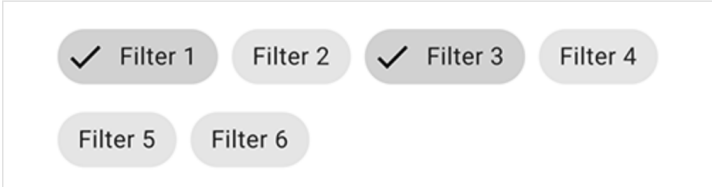
Permiten la selección de un único elemento de las opciones de `Chip` existentes. Cuando seleccionamos un `Chip` automáticamente se desmarca el que estuviera seleccionado.



Filter Chips

Los chips de filtro utilizan etiquetas o palabras descriptivas para filtrar el contenido.

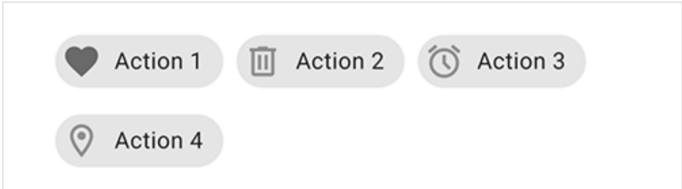
Los chips de filtro delinean y muestran claramente las opciones en un área compacta. Son una buena alternativa para alternar botones o casillas de verificación.



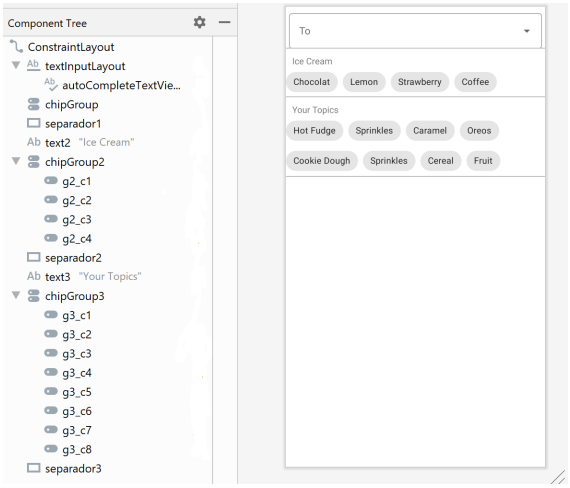
Action Chips

Los chips de acción ofrecen acciones relacionadas con el contenido principal. Deben aparecer de forma dinámica y contextual en una interfaz de usuario.

Una alternativa a los chips de acción son los botones, que deben aparecer de manera persistente y consistente.



Veamos un diseño donde implementar todas estas vistas:



Colocaremos un control **`autoCompleteTextView`** que nos permitirá ir seleccionando nombres de una lista y los añadiremos como **`InputChip`** dentro de un **`chipGroup`** de forma dinámica(posteriormente modificaremos esta parte para incluir el chip seleccionado dentro del **`autoCompleteTextView`**).

También tendremos otro **`chipGroup`** para agrupar los **`chipChoice`** y otro **`chipGroup`** para los **`chipFilter`**. Cada uno de esas vistas van separadas por un delimitador **`view`**.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
    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.textfield.TextInputLayout
        android:id="@+id/textInputLayout"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox.ExposedDropdownMe
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:hint="To"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

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

    <com.google.android.material.chip.ChipGroup
        android:id="@+id/chipGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:singleLine="false"
        style="@style/Widget.MaterialComponents.ChipGroup"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/textInputLayout">
    </com.google.android.material.chip.ChipGroup>

    <View
        android:id="@+id/separador1"
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:background="@android:color/darker_gray"
        app:layout_constraintBottom_toBottomOf="@id/chipGroup"
        app:layout_constraintLeft_toLeftOf="parent" />

    <TextView
        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="Ice Cream"
        app:layout_constraintLeft_toLeftOf="parent"
```

```
        app:layout_constraintTop_toBottomOf="@id/separador1" />

    <com.google.android.material.chip.ChipGroup
        android:id="@+id/chipGroup2"
        app:singleSelection="true"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:singleLine="true"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text2">

        <com.google.android.material.chip.Chip
            android:id="@+id/g2_c1"
            style="@style/Widget.MaterialComponents.Chip.Choice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Chocolat" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g2_c2"
            style="@style/Widget.MaterialComponents.Chip.Choice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Lemon" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g2_c3"
            style="@style/Widget.MaterialComponents.Chip.Choice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Strawberry" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g2_c4"
            style="@style/Widget.MaterialComponents.Chip.Choice"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Coffee" />
    </com.google.android.material.chip.ChipGroup>

    <View
        android:id="@+id/separador2"
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:background="@android:color/darker_gray"
        app:layout_constraintBottom_toBottomOf="@id/chipGroup2"
        app:layout_constraintLeft_toLeftOf="parent" />

    <TextView
        android:id="@+id/text3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_margin="10dp"
        android:text="Your Topics"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/separador2" />

    <com.google.android.material.chip.ChipGroup
        android:id="@+id/chipGroup3"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text3">

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c1"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hot Fudge" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c2"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Sprinkles" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c3"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Caramel" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c4"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Oreos" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c5"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Cookie Dough" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c6"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:text="Sprinkles" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c7"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Cereal" />

        <com.google.android.material.chip.Chip
            android:id="@+id/g3_c8"
            style="@style/Widget.MaterialComponents.Chip.Filter"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Fruit" />
    </com.google.android.material.chip.ChipGroup>

    <View
        android:id="@+id/separador3"
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:background="@android:color/darker_gray"
        app:layout_constraintTop_toBottomOf="@id/chipGroup3"
        app:layout_constraintLeft_toLeftOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

EjercicioPropuesto:

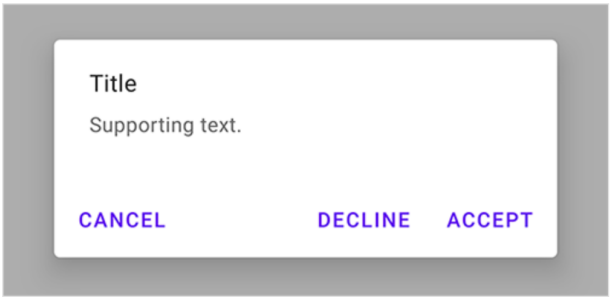
- Modificar el ejercicio anterior de forma que se incorporen los `chip` al campo `AutoCompleteTextView`. Hay que conseguir que el botón de eliminación del `chip` esté activo y utilizable.

Diálogos

Un cuadro de diálogo es un tipo de ventana modal que aparece por encima del contenido de la aplicación para proporcionar información crítica o solicitar una acción. Los cuadros de diálogo deshabilitan todas las funciones de la aplicación cuando aparecen y permanecen en la pantalla hasta que se confirman, se descartan o se toma una acción requerida.

Alert dialog

El siguiente ejemplo muestra un cuadro de diálogo de alerta.

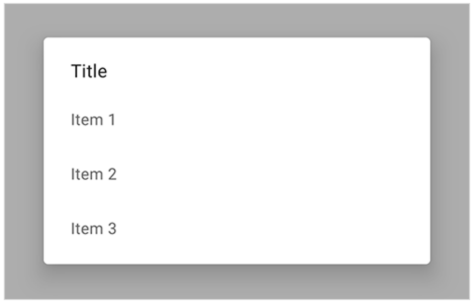


Para gestionar los eventos sobre el mismo:

```
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

Muestran los elementos de una lista, que son inmediatamente procesables cuando se seleccionan. No tienen botones de acción.



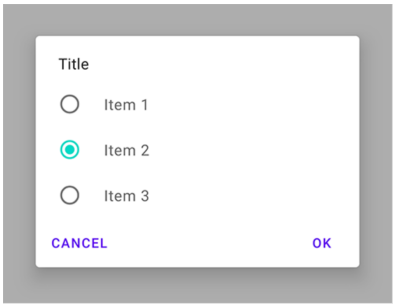
Para gestionar los eventos sobre el mismo:

```
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

Los diálogos de confirmación brindan a los usuarios la posibilidad de proporcionar una confirmación final de una elección antes de comprometerse con ella, para que tengan la oportunidad de cambiar de opinión si es necesario.



Para gestionar los eventos sobre el mismo:

```
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

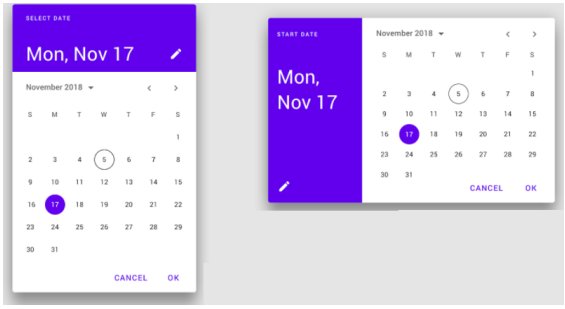
Los cuadros de diálogo de pantalla completa son los únicos cuadros de diálogo sobre los que pueden aparecer otros cuadros de diálogo.

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

> Ejercicio resuelto Dialogo_Personalizado

DataPicker

Permiten a los usuarios seleccionar una fecha o un rango de fechas. Los `dataPicker` se adaptan a la orientación del dispositivo:



Lo más frecuente es que estos controles no aparezcan directamente en la interfaz, sino que cuando pulsamos un campo de edición asociado a una fecha, nos aparezca un diálogo (o `fragmentDialog`) que incluye el `dataPicker`.

Crearemos el proyecto **EjemploDataPicker** y colocamos en la actividad principal un `TextInputEditText` en el escuchador correspondiente al `onClick` activaremos el `dataPicker` y sobre él visualizaremos la entrada realizada en el `dataPicker` cuando este se cierre.

Es importante que nuestro `TextInputEditText` tenga las siguientes propiedades `android:clickable="false"` y `android:focusable="false"` que evitan que nuestro campo de edición se pueda seleccionar o pulsar, evitando de esta manera que aparezca el teclado, solamente se mostrará el `dataPicker`.
Veamos el código:

```
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,"")
        }
    }
}
```

EjercicioPropuesto:

- Modificar el ejercicio anterior de forma que si el campo de edición tiene una fecha, la muestra al abrir el `dataPicker` .
- Añadir dos campos de fecha más 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

Los `Progress indicators` informan a los usuarios sobre el estado de un proceso en ejecución, como cargar una aplicación, enviar un formulario o guardar actualizaciones, utilizan animaciones para captar la atención. Nunca son simplemente decorativos.

Material Design ofrece dos tipos de indicadores de progreso visualmente distintos: indicadores de progreso lineales y circulares. Conviene utilizar siempre el mismo tipo para la misma acción.

Los indicadores de progreso pueden ser determinados (cuando se conoce o se puede calcular la finalización del proceso) o indeterminados (no se conoce el fin de la tarea o no es necesario indicar cuánto tiempo va a durar la misma).

A medida que se dispone de más información sobre un proceso, un `Progress indicator` puede pasar de un estado indeterminado a uno determinado.

```
<!-- 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.

> EjercicioResueltoProgress: tareas asíncronas