



Componentes de texto

Los view (o componentes o widgets) son los elementos que vamos a utilizar para crear el IU, los cuales pueden ser de muchos tipos:

Campos de texto: Los campos de texto permiten a los usuarios introducir texto en una interfaz de usuario. Suelen aparecer en formularios y cuadros de diálogo. En la paleta de diseño, podemos ver los siguientes campos de texto:

- **TextView:** El TextView en Android es un widget que muestra texto al usuario como su nombre lo sugiere. Claramente esto lo hace ser uno de los componentes más usados en interfaces de usuario para proyectar cabeceras, títulos, texto informativo, etiquetas y muchos otros.

Atributos de TextView:

- Usa el atributo **android:text** para asignar un literal String
- Cambia el color del texto a través del atributo **android:textColor**. Este recibe la referencia a un recurso de color o valores RGB.
- El atributo **android:textSize** determina el tamaño del texto y se recomienda asignarle valores en pixeles escalados o sp.
- Usa el atributo **android:textStyle** para asignar uno de los siguientes estilos o combinaciones de ellos: normal, bold y italic. El valor por defecto es normal y si deseas combinarlos usa el símbolo '|'.
- El atributo **android:typeface** acepta las siguientes constantes para especificar el estilo de fuente del TextView: normal, sans, serif y monospace.
- Usa el atributo **android:autoLink** si deseas habilitar la detección de patrones que coincidan con esquemas como: correos (email), [urls](#) (web), teléfonos (pone), entre otros. Por defecto, tiene el valor none (ninguno), se puede usar también el valor all para representar todos los patrones.
- También puedes usar **android:textColorLink** para modificar el color de creación del link.
- En el caso que desees convertir toda la entrada de texto a mayúsculas, aplica true al atributo **android:textAllCaps**
- El atributo **android:textAppearance** permite definir en conjunto el color de texto, tipo de letra, tamaño y estilo de texto. Normalmente usarás este atributo para asignar las apariencias prefabricadas en los estilos del sistema, por ejemplo: @style/TextAppearance.AppCompat.Headline

Ab TextView

Ab Plain Text

Ab Password

Ab Password (Numeric)

Ab E-mail

Ab Phone

Ab Postal Address

Ab Multiline Text

Ab Time

Ab Date

Ab Number

Ab Number (Signed)

Ab Number (Decimal)

Ab AutoCompleteText...

Ab MultiAutoComplete...

Ab CheckedTextView

Ab TextInputLayout

- **PlainText:** es la extensión de un TextView con la capacidad de editar su contenido para recibir texto por parte del usuario. En el fichero XML aparece como EditText. Visualmente tiene un texto auxiliar llamado **android:hint** que ayude al usuario a saber que dato se espera que se introduzca. Revisar que en **android:text** no aparezca ningún texto.



Para recuperar el valor de texto de un EditText usa el método `getText()`, el devuelve directamente un objeto `Editable`, y no `String`. La cual es una interfaz de texto dinámico y configurable. Sin embargo al usar `toString()` es posible obtener la cadena de texto plana.

El atributo **android:inputType** por defecto tiene valor "text". Este atributo determina el tipo de teclado virtual que aparecerá ante el usuario, por ejemplo, si tu campo de texto es para un número telefónico usa el valor pone: "phone".

A continuación, se muestra una tabla con los valores más frecuentes para `inputType`, que corresponden con algunos de los componentes que vemos en la paleta.

Constante	Descripción
<code>text</code>	Recibe texto plano simple
<code>textPersonName</code>	Texto correspondiente al nombre de una persona
<code>textPassword</code>	Protege los caracteres que se van escribiendo con puntos
<code>numberPassword</code>	Contraseña de solo números enmascarada con puntos
<code>textEmailAddress</code>	Texto que será usado en un campo para emails
<code>phone</code>	Texto asociado a un número de teléfono
<code>textPostalAddress</code>	Para ingresar textos asociados a una dirección postal
<code>textMultiLine</code>	Permite múltiples líneas en el campo de texto
<code>time</code>	Texto para determinar la hora
<code>date</code>	Texto para determinar la fecha
<code>number</code>	Texto con caracteres numéricos
<code>numberSigned</code>	Permite números con signo
<code>numberDecimal</code>	Para ingresar números decimales

Para forzar el número máximo de caracteres que recibirá el EditText usa el atributo **android:maxLength**.

Reduce la capacidad del campo de texto a una sola línea con el atributo **android:singleLine**. De lo contrario el EditText aceptará múltiples líneas en su contenido y el teclado virtual usará como tecla de acción el salto de línea  en vez de la confirmación . El valor por defecto para `singleLine` es false.

Al igual que todos los componentes, un campo de texto también puede cambiar a estado deshabilitado con el atributo **android:enabled** y el valor false. Si deseas hacerlo programáticamente usa `setEnabled()` con el valor false.

Para filtrar los caracteres que podrán ser aceptados en un EditText es posible usar el atributo **android:digits** con una lista de elementos permitidos, por ejemplo, si solo queremos que se admitan los dígitos del 0 al 9 en un campo de texto, para conseguirlo pones la lista "0123456789".

Ejemplo: Crear campo de texto para el número de tarjeta de crédito del cliente. Se deben permitir como máximo 19 caracteres (3 espacios entre bloques de 4 caracteres) y solo dígitos del 0 al 9. En este caso, no podemos utilizar un EditText con `inputType="number"` ya que también debe admitirse el espacio en blanco, por lo que usamos `android:digits` en un campo EditText con `inputType="text"`.

- **AutoCompleteTextView:** hereda directamente de EditText. Se utiliza para mostrar sugerencias de autocompletado al usuario mientras teclea letras en un campo de texto. El ítem que se seleccione del menú desplegado será asignado como valor al campo.

En la siguiente tabla, aparecen los atributos más comunes:

Atributo	Descripción
<code>android:completionHint</code>	Determina el texto de indicación para el menú de sugerencias
<code>android:completionHintView</code>	Determina el view usado para representar al texto de indicación
<code>android:completionThreshold</code>	Define el número de caracteres que se deben escribir para desplegar las sugerencias
<code>android:dropDownAnchor</code>	Id del view que servirá como punto de aparición del menú de sugerencias
<code>android:dropDownHeight</code>	Altura general del menú de sugerencias. Puedes usar <code>wrap_content</code> para ajustar al contenido o <code>match_parent</code> para permitir un despliegue expandido en el padre
<code>android:dropDownHorizontalOffset</code>	Cantidad de píxeles que el menú estará separado horizontalmente
<code>android:dropDownSelector</code>	Selector para especificar el color de las sugerencias según el estado
<code>android:dropDownVerticalOffset</code>	Cantidad de píxeles que el menú estará separado verticalmente
<code>android:dropDownWidth</code>	Ancho general del menú de sugerencias
<code>android:popupBackground</code>	Establece el fondo del menú

Se debe indicar el número de caracteres que se tienen que escribir para que aparezcan las sugerencias usando el atributo **android:completionThreshold**.

Es necesario un adaptador para indicar los ítems que aparecerán en el menú emergente de sugerencias. La forma más fácil para poblar las sugerencias es usar un array de strings desde los recursos. Es decir, creamos un elemento `<string-array>` dentro de `strings.xml` e incluimos los ítems que serán mostrados.

```
<resources>
    <string name="app_name">Ejemplo Autocompletado</string>

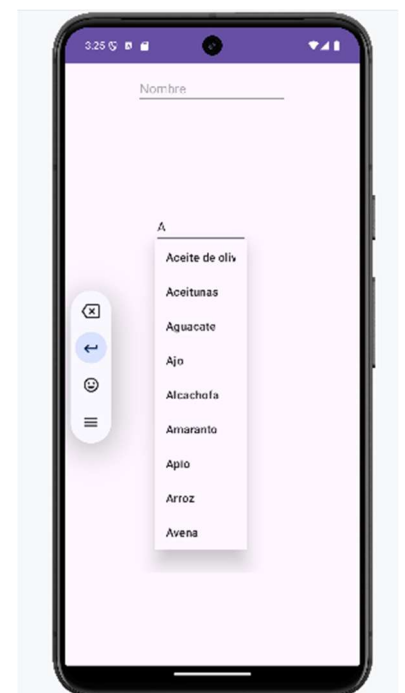
    <!-- Fuente de datos -->
    <string-array name="products">
        <item>Aceite de oliva</item>
        <item>Aceitunas</item>
        <item>Aguacate</item>
        <item>Ajo</item>
        <item>Alcachofa</item>
        <item>Amaranto</item>
        <item>Apio</item>
        <item>Arroz</item>
        <item>Avena</item>
    </string-array>
</resources>
```

Luego relacionamos a la instancia del `AutoCompleteTextView` con una instancia de un `ArrayAdapter`. Este adaptador es construido a partir de la obtención del array de strings anterior. Y finalmente usamos `setAdapter()` para la asignación.

```
class MainActivity : AppCompatActivity() {

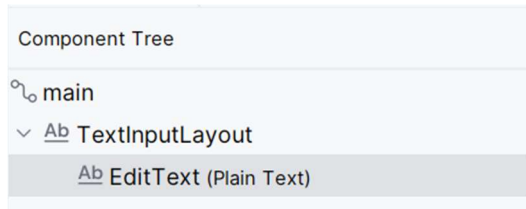
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val products = resources.getStringArray(R.array.products)
        val adapter: ArrayAdapter<String> = ArrayAdapter<String>(
            context: this,
            android.R.layout.simple_list_item_1, products
        )
        binding.listView1.setAdapter(adapter)
    }
}
```



- **TextInputLayout:** Se utiliza para envolver un edit text al cual le proveerá de una etiqueta. Con este componente cuando el usuario establece el foco en un campo de este tipo, su *hint* debe flotar hacia la parte superior, proporcionando espacio para el texto. Esto asegura que el usuario nunca pierda el contexto del contenido que está introduciendo.

En la vista de diseño, arrastraremos el edit text dentro del TextInputLayout:



Tal que en la vista XML, veremos que queda de esta manera:

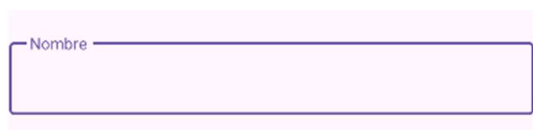
```
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="1dp"
    android:layout_marginTop="1dp"
    android:layout_marginEnd="1dp"
    android:layout_marginBottom="1dp"
    android:hint="Nombre"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <EditText
        android:id="@+id/EditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="1dp"
        android:layout_marginTop="5dp"
        android:layout_marginEnd="0dp"
        android:layout_marginRight="352dp"
        android:layout_marginBottom="1dp"
        android:ems="10"
        android:inputType="text"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    </com.google.android.material.textfield.TextInputLayout>
```

Tal que, al ejecutarlo, de primeras lo veremos así:



Al empezar a escribir, la sugerencia (hint) vemos que se ha desplazado hacia arriba:




El `TextInputLayout` tiene la capacidad para mostrar errores asociados al contenido digitado por usuario. Estos pueden ser debido a un formato incorrecto, una cantidad mínima de caracteres no satisfecha, algún carácter indebido, etc.

El método `setError()` se usa para mostrar los errores del `EditText`. También puedes pasar `null` como parámetro para limpiar los errores.

Con la clase **Patterns**, podemos validar si un número de teléfono es válido, o una dirección de email (`Patterns.EMAIL_ADDRESS`) o una URL (`Patterns.WEB_URL`). Si se quiere comprobar si una dirección IP es válida, se puede usar: `InetAddresses.isNumericAddress(String)`

```
private fun esTelefonoValido(telefono: String): Boolean {
    var esValido: Boolean
    if (Patterns.PHONE.matcher(telefono).matches()) {
        binding.telefono.setError(null)
        esValido = true
    }
    else {
        binding.telefono.setError("Teléfono inválido")
        esValido = false
    }
    return esValido
}
```

Un ejemplo de cómo aparecería el mensaje de error sería como se muestra a continuación:



The image shows a text input field with the label "Teléfono" and the text "aaaaaa" entered. A red exclamation mark icon is visible in the top right corner of the input field. Below the input field, a black error message box displays the text "Teléfono inválido".