Interfaz de usuario en Android: Controles básicos (II)

Después de haber hablado en el artículo anterior de los controles de tipo botón, en esta nueva entrega nos vamos a centrar en otros tres componentes básicos imprescindibles en nuestras aplicaciones: las imágenes (`ImageView`), las etiquetas (`TextView`) y por último los cuadros de texto (`EditText`).

Control ImageView [API]

El control **`ImageView`** permite mostrar imágenes en la aplicación. La propiedad más interesante es **`android:src`**, que permite indicar la imagen a mostrar. Nuevamente, lo normal será indicar como origen de la imagen el identificador de un recurso de nuestra carpeta `/res/drawable`, por ejemplo `android:src="@drawable/unaimagen"`. Además de esta propiedad, existen algunas otras útiles en algunas ocasiones como las destinadas a establecer el tamaño máximo que puede ocupar la imagen, `android:maxWidth` y `android:maxHeight`, o para indicar cómo debe adaptarse la imagen al tamaño del control, **`android:scaleType`** (`CENTER`, `CENTER_CROP`, `CENTER_INSIDE`, …). Además, como ya comentamos para el caso de los controles `ImageButton`, al tratarse de un control de tipo imagen deberíamos establecer siempre la propiedad `android:contentDescription` para ofrecer una breve descripción textual de la imagen, algo que hará nuestra aplicación mucho más accesible.

```
<ImageView android:id="@+id/ImgFoto"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_launcher"
android:contentDescription="@string/imagen_ejemplo"
/>
```

Si en vez de establecer la imagen a mostrar en el propio layout XML de la actividad quisiéramos establecerla mediante código utilizaríamos el método `setImageResorce(…)`, pasándole el ID del recurso a utilizar como contenido de la imagen.

```
1    ImageView img= (ImageView)findViewById(R.id.ImgFoto);
2    img.setImageResource(R.drawable.ic_launcher);
```

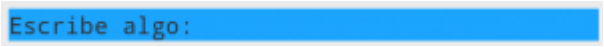
En cuanto a posibles eventos, al igual que comentamos para los controles de tipo botón en el apartado anterior, para los componentes `ImageView` también podríamos implementar su evento `onClick`, de forma idéntica a la que ya vimos, aunque en estos casos suele ser menos frecuente la necesidad de capturar este evento.

Control TextView [API]

El control `TextView` es otro de los clásicos en la programación de GUIs, las etiquetas de texto, y se utiliza para mostrar un determinado texto al usuario. Al igual que en el caso de los botones, el texto del control se establece mediante la propiedad `android:text`. A parte de esta propiedad, la naturaleza del control hace que las más interesantes sean las que establecen el formato del texto mostrado, que al igual que en el caso de los botones son las siguientes: `android:background` (color de fondo), `android:textColor` (color del texto),`android:textSize` (tamaño de la fuente) y `android:typeface` (estilo del texto: negrita, cursiva, …).

```
<TextView
android:id="@+id/LblEtiqueta"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/escribe_algo"
android:background="#ff1ca5ff"
android:typeface="monospace"/>
```

La etiqueta tal cual se ha definido en el código anterior tendría el siguiente aspecto:



De igual forma, también podemos manipular estas propiedades desde nuestro código. Como ejemplo, en el siguiente fragmento recuperamos el texto de una etiqueta con `getText()`, y posteriormente le concatenamos unos números, actualizamos su contenido mediante `setText()` y le cambiamos su color de fondo con `setBackgroundColor()`.

```
1    final TextView lblEtiqueta = (TextView)findViewById(R.id.LblEtiqueta);
2    String texto = lblEtiqueta.getText().toString();
3    texto += "123";
4    lblEtiqueta.setText(texto);
5    lblEtiqueta.setBackgroundColor(Color.BLUE);
```

Control EditText [API]

El control `EditText` es el componente de edición de texto que proporciona la plataforma Android. Permite la introducción y edición de texto por parte del usuario, por lo que en tiempo de diseño la propiedad más interesante a establecer, además de su posición/tamaño y formato, es el texto a mostrar, atributo `android:text`. Por supuesto si no queremos que el cuadro de texto aparezca inicializado con ningún texto, no es necesario incluir esta propiedad en el layout XML. Lo que sí deberemos establecer será la propiedad `android:inputType`. Esta propiedad indica el tipo de contenido que se va a introducir en el cuadro de texto, como por ejemplo una dirección de correo electrónico (`textEmailAddress`), un número genérico (`number`), un número de teléfono (`phone`), una dirección web (`textUri`), o un texto genérico (`text`). El valor que establezcamos para esta propiedad tendrá además efecto en el tipo de teclado que mostrará Android para editar dicho campo. Así, por ejemplo, si hemos indicado "`text`" mostrará el teclado

completo alfanumérico, si hemos indicado "`phone`" mostrará el teclado numérico del teléfono, etc.

```
<EditText
android:id="@+id/TxtBasico"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:inputType="text" />
```

Al igual que ocurría con los botones, donde podíamos indicar una imagen que acompañara al texto del mismo, con los controles de texto podemos hacer lo mismo. Las propiedades **`drawableLeft`** o **`drawableRight`** nos permite especificar una imagen, a izquierda o derecha, que permanecerá fija en el cuadro de texto.

Otra opción adicional será indicar un texto de ayuda o descripción (***hint***), que aparecerá en el cuadro de texto mientras el usuario no haya escrito nada (en cuanto se escribe algo este texto desaparece). Para esto utilizaremos las propiedades **`android:hint`** para indicar el texto y **`android:textColorHint`** para indicar su color.

Veamos un ejemplo utilizando las propiedades anteriores:

```
1
2
3
4
5
6
7
    <EditText android:id="@+id/TxtImagenHint"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:drawableLeft="@drawable/ic_usuario"
        android:hint="@string/usuario"
        android:textColorHint="#CFCFCF"
        android:inputType="text" />
```

Y su aspecto sería el siguiente:

Para recuperar y establecer el desde nuestro código podemos utilizar los métodos `getText()` y`setText(`*`nuevoTexto`*`)` respectivamente:

```
1    EditText txtTexto = (EditText)findViewById(R.id.TxtBasico);
2    String texto = txtTexto.getText().toString();
3    txtTexto.setText("Hola mundo!");
```

Un detalle que puede haber pasado desapercibido. ¿Os habéis fijado en que hemos tenido que hacer un `toString()` sobre el resultado de `getText()`? La explicación para esto es que el método `getText()` no devuelve directamente una cadena de caracteres (`String`) sino un objeto de tipo `Editable`, que a su vez implementa la interfaz `Spannable`. Y esto nos lleva a una característica interesante del control `EditText`, y es que no sólo nos permite editar *texto plano* sino también *texto enriquecido* o con formato.

Puede mejorar la experiencia del usuario mostrando un teclado específico en función del tipo de campo. Esto es posible gracias al atributo **android:inputType.**

Este atributo puede tomar los siguientes valores:

- text (valor por defecto): teclado normal.
- textCapCharacters: teclado todo en mayúsculas.
- textCapWords: primera letra automáticamente en mayúsculas.
- textAutoCorrect: activa la corrección automática.
- textMultiLine: texto en varias líneas.
- textNoSuggestions: sin sugerencias de corrección.
- textUri: permite introducir una URL web.
- textEmailAddress: dirección de correo electrónico.
- textEmailSubject: asunto de correo electrónico.
- textShortMessage: activa el acceso directo a smiley en el teclado.
- textPersonName: permite introducir el nombre de una persona (muestra speech to text en la parte inferior izquierda).
- textPostalAddress: permite introducir una dirección postal (muestra speech to text en la parte inferior izquierda del teclado).
- textPassword: entrada de una contraseña.

- textVisiblePassword: entrada de una contraseña visible.
- number/numberSigned/numberDecimal/phone/datetime/date/time: teclado numérico.

Esta lista no es exhaustiva.

Etiquetas Flotantes (*Floating Labels*)

Como contenido extra de este capítulo vamos a hablar de un nuevo control publicado recientemente por Google dentro de la nueva librería de diseño (*Design Support Library*) que nos ayuda a implementar uno de los componentes relacionados con los cuadros de texto que se mencionan en las especificaciones de *Material Design*. Se trata de las etiquetas flotantes, que no es más que un *hint* (más arriba vimos lo que era esto) que, en vez de desaparecer, se desplaza automáticamente a la parte superior del cuadro de texto cuando el usuario pulsa sobre él.

El componente en cuestión se llama **`TextInputLayout`**, y es muy sencillo de utilizar. Lo primero que haremos será añadir la librería de diseño a nuestro proyecto (en el artículo anterior sobre botones explicamos con más detalle cómo hacerlo) añadiendo su referencia al fichero *build.gradle*:

```
1    dependencies {
2        //...
3        compile 'com.android.support:design:22.2.0'
4    }
```

Tras esto, simplemente tendremos que añadir un nuevo `EditText` con *hint* a nuestra interfaz, pero esta vez dentro de un contenedor de tipo `TextInputLayout`:

```
<android.support.design.widget.TextInputLayout
android:id="@+id/TiLayout"
android:layout_width="match_parent"
android:layout_height="wrap_content" >

<EditText android:id="@+id/TxtInput"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:hint="Escribe un número par" />

</android.support.design.widget.TextInputLayout>
```

Como podéis ver no hacemos nada especial, ni asignamos ninguna propiedad nueva, simplemente incluimos el cuadro de texto dentro del nuevo componente `TextInputLayout`.

Si ejecutamos en este momento la aplicación podremos ver cómo aparece la etiqueta flotante con tan sólo pulsar sobre el cuadro de texto.

Con esto ya tendríamos la funcionalidad básica de las etiquetas flotantes, pero el control ofrece también la posibilidad de mostrar errores (muy útiles por ejemplo para mostrar errores de validación) bajo el cuadro de texto. Para ello podemos utilizar los métodos **`setErrorEnabled(true)` y `setError()`.** El primero de ellos reservará espacio debajo del cuadro de texto para mostrar los errores. El segundo nos servirá para indicar el texto del error o para eliminarlo (pasando `null` como parámetro). A modo de ejemplo, he añadido un nuevo botón a la aplicación (`btnComprobar`) donde comprobaré si lo introducido por el usuario es o no un número par, mostrando el error correspondiente cuando aplique:

```
1     txtInputLayout = (TextInputLayout)findViewById(R.id.TiLayout);
2     txtInputLayout.setErrorEnabled(true);
3
4     txtInput = (EditText)findViewById(R.id.TxtInput);
5
6     btnComprobar = (Button)findViewById(R.id.BtnInputLayout);
7     btnComprobar.setOnClickListener(new View.OnClickListener() {
8         @Override
9         public void onClick(View view) {
10            String num = txtInput.getText().toString();
11
12            if(num.isEmpty() || Integer.parseInt(num)%2 != 0)
                  txtInputLayout.setError("Error: No es un número par!");
              else
```

```
13                  txtInputLayout.setError(null);
14              }
15      });
16
17
```

Si volvemos a ejecutar ahora la aplicación de ejemplo e introducimos un número impar, veremos cómo el mensaje de error aparece correctamente bajo el cuadro de texto:

Puedes consultar y/o descargar el código completo de los ejemplos desarrollados en este artículo accediendo a la página del curso en GitHub.

Enlaces de interés:

- Campos de texto en Material Design
- Campos de texto en Guía de diseño Android
- Campos de texto en Guía de desarrollo Android
- Presentación “Advanced Android TextView” (Chiu-Ki Chan)
- Artículo “Spans, a Powerful Concept” (Flavien Laurent)
- Implementación de *Floating Label*: post y código (Chris Banes)
- Librería Calligraphy (fuentes personalizadas)