Desarrollo de aplicaciones móviles I

Tema Nº5:Componentes Básicos de Android.

Indicador de logro Nº5:Aplica los componentes básicos de la interfaz de usuario de una aplicación en Android a través de la implementación de una aplicación móvil que integren los elementos más comunes.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº5:**

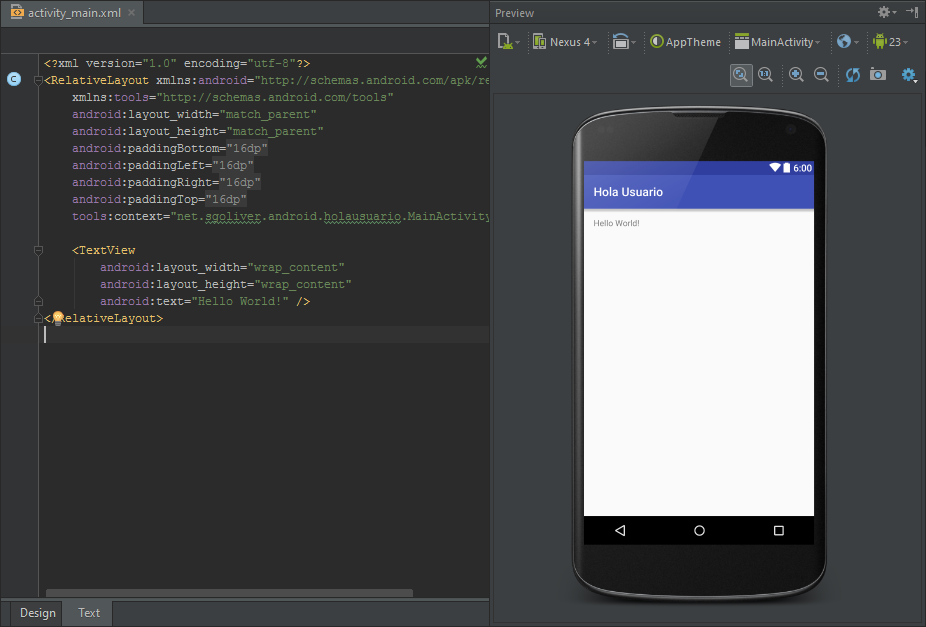
Creando nuestro entorno de desarrollo.

**Subtema 5.1:**

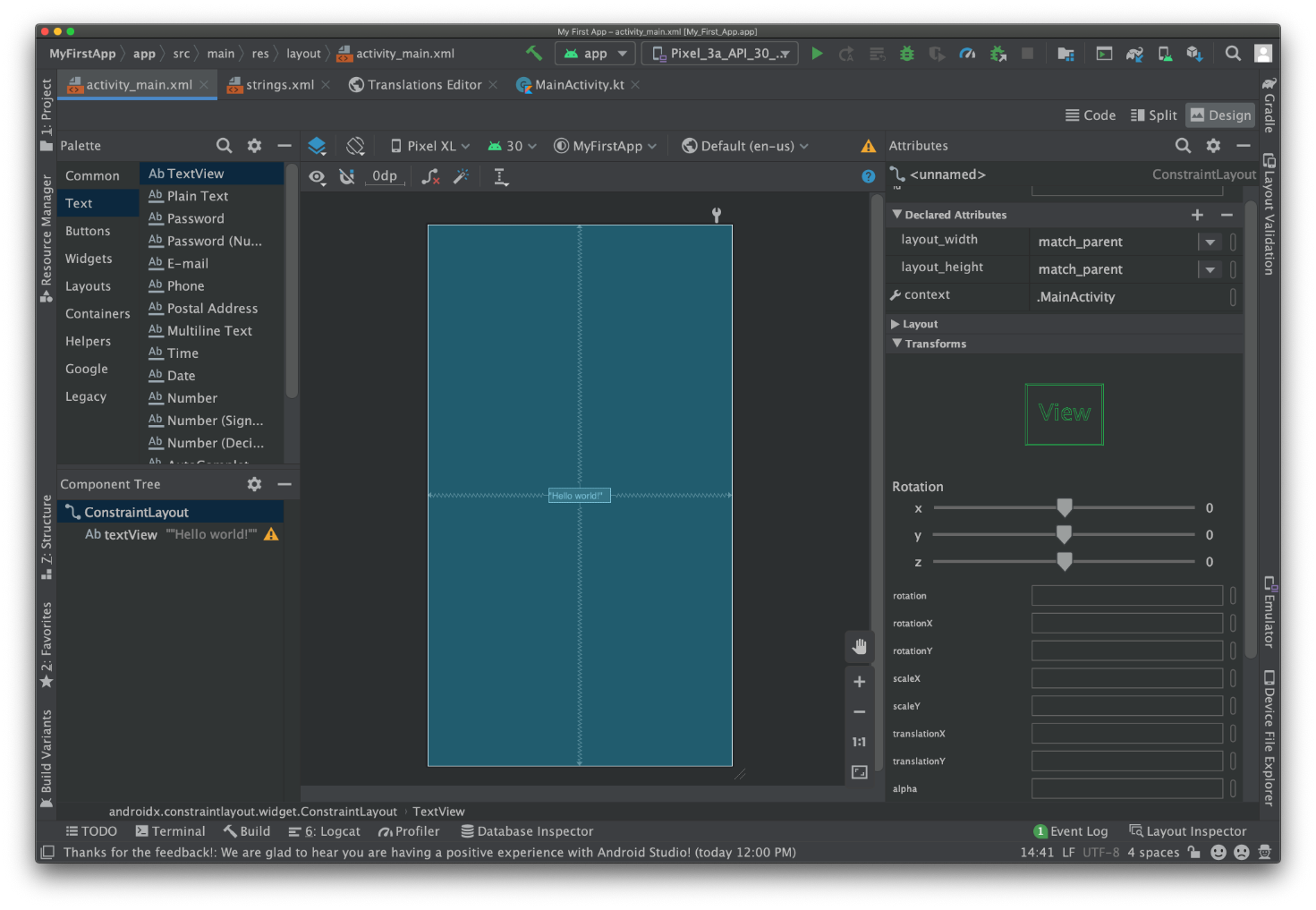
XML vs. Editor Gráfico

**Ejemplos:**

Editor XML.



Editor gráfico.



**Subtema 5.2:**

Diferencia entre LinearLayout y Relative Layout.

**Ejemplos:**

FrameLayout

Éste es el más simple de todos los layouts de Android. Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia). Por ello, suele utilizarse para mostrar un único control en su interior, a modo de contenedor (placeholder) sencillo para un sólo elemento sustituible, por ejemplo una imagen.

Los componentes incluidos en un FrameLayout podrán establecer sus propiedades android:layout\_width y android:layout\_height, que podrán tomar los valores «match\_parent» (para que el control hijo tome la dimensión de su layout contenedor) o «wrap\_content» (para que el control hijo tome la dimensión de su contenido).

Veamos un ejemplo donde colocamos un FrameLayout que ocupará toda la pantalla (estableciendo como «match\_parent» tanto su ancho como su alto) y en su interior un botón (Button) con el ancho ajustado a su contenedor («match\_parent«) y el alto a su contenido («wrap\_content«):

Texto

Descripción generada automáticamente

Con el código anterior conseguimos un layout tan sencillo como el siguiente:

Forma

Descripción generada automáticamente

LinearLayout

El siguiente tipo de layout en cuanto a nivel de complejidad es el LinearLayout. Este layout apila uno tras otro todos sus elementos hijos en sentido horizontal o vertical según se establezca su propiedad android:orientation.

Al igual que en un FrameLayout, los elementos contenidos en un LinearLayout pueden establecer sus propiedades android:layout\_width y android:layout\_height para determinar sus dimensiones dentro del layout.

Veamos un ejemplo donde colocamos dos botones dentro de un LinearLayout con orientación vertical:

Texto

Descripción generada automáticamente

El layout anterior quedaría de la siguiente forma:

Tabla

Descripción generada automáticamente con confianza baja

Pero en el caso de un LinearLayout, tendremos otro parámetro con el que jugar, la propiedad android:layout\_weight. Esta propiedad nos va a permitir dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellas.

Esto es más difícil de explicar que de comprender con un ejemplo. Si incluimos en un LinearLayout vertical dos botones (Button) y a uno de ellos le establecemos un layout\_weight="1" y al otro un layout\_weight="2" conseguiremos como efecto que toda la superficie del layout quede ocupada por los dos botones y que además el segundo sea el doble (relación entre sus propiedades weight) de alto que el primero.

Texto

Descripción generada automáticamente

Con el código anterior conseguiríamos un layout como el siguiente:

Gráfico, Gráfico de rectángulos

Descripción generada automáticamente

Así pues, a pesar de la simplicidad aparente de este layout resulta ser lo suficiente versátil como para sernos de utilidad en muchas ocasiones.

TableLayout

Un TableLayout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla.

La estructura de la tabla se define de forma similar a como se hace en HTML, es decir, indicando las filas que compondrán la tabla (objetos TableRow), y dentro de cada fila las columnas necesarias, con la salvedad de que no existe ningún objeto especial para definir una columna (algo así como un TableColumn) sino que directamente insertaremos los controles necesarios dentro del TableRow y cada componente insertado (que puede ser un control sencillo o incluso otro ViewGroup) corresponderá a una columna de la tabla. De esta forma, el número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados, y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.

Por norma general, el ancho de cada columna se corresponderá con el ancho del mayor componente de dicha columna, pero existen una serie de propiedades que nos ayudarán a modificar este comportamiento:

* android:stretchColumns. Indicará las columnas que pueden expandir para absorber el espacio libre dejado por las demás columnas a la derecha de la pantalla.
* android:shrinkColumns. Indicará las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la derecha de la pantalla.
* android:collapseColumns. Indicará las columnas de la tabla que se quieren ocultar completamente.

Todas estas propiedades del TableLayout pueden recibir una lista de índices de columnas separados por comas (ejemplo: android:stretchColumns="1,2,3") o un asterisco para indicar que debe aplicar a todas las columnas (ejemplo: android:stretchColumns="\*").

Otra característica importante es la posibilidad de que una celda determinada pueda ocupar el espacio de varias columnas de la tabla (análogo al atributo colspan de HTML). Esto se indicará mediante la propiedad android:layout\_span del componente concreto que deberá tomar dicho espacio.

Veamos un ejemplo con varios de estos elementos:

Texto

Descripción generada automáticamente

El layout resultante del código anterior sería el siguiente:

Texto

Descripción generada automáticamente con confianza baja

GridLayout

Sus características son similares al TableLayout, ya que se utiliza igualmente para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas. La diferencia entre ellos estriba en la forma que tiene el GridLayout de colocar y distribuir sus elementos hijos en el espacio disponible. En este caso, a diferencia del TableLayout indicaremos el número de filas y columnas como propiedades del layout, mediante android:rowCount y android:columnCount. Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, como hacíamos con el elemento TableRow del TableLayout, sino que los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad android:orientation) hasta completar el número de filas o columnas indicadas en los atributos anteriores. Adicionalmente, igual que en el caso anterior, también tendremos disponibles las propiedades android:layout\_rowSpan y android:layout\_columnSpan para conseguir que una celda ocupe el lugar de varias filas o columnas. Por su parte, para definir cómo ocupar el espacio horizontal disponible ocupado por las columnas debemos jugar con la propiedad layout\_columnWeight de cada elemento hijo.

Existe también una forma de indicar de forma explícita la fila y columna que debe ocupar un determinado elemento hijo contenido en el GridLayout, y se consigue utilizando los atributos android:layout\_row y android:layout\_column. De cualquier forma, salvo para configuraciones complejas del grid no suele ser necesario utilizar estas propiedades.

Con todo esto en cuenta, para conseguir una distribución equivalente a la del ejemplo anterior del TableLayout, necesitaríamos escribir un código como el siguiente:

Texto

Descripción generada automáticamente

RelativeLayout

El último tipo de layout que vamos a ver es el RelativeLayout. Este layout permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout. De esta forma, al incluir un nuevo elemento X podremos indicar por ejemplo que debe colocarse debajo del elemento Y y alineado a la derecha del layout padre. Veamos esto en el ejemplo siguiente:

Texto

Descripción generada automáticamente

En el ejemplo, el botón (Button) se colocará debajo del cuadro de texto (EditText) dado que se ha establecido la propiedad android:layout\_below="@id/txtNombre" y alineado a la derecha del layout padre al establecer android:layout\_alignParentRight="true", Quedaría algo así:

Forma, Rectángulo

Descripción generada automáticamente

Al igual que estas tres propiedades, en un RelativeLayout tendremos un sinfín de propiedades para colocar cada control justo donde queramos. Veamos las principales [creo que sus propios nombres explican perfectamente la función de cada una]:

Posición relativa a otro control:

* android:layout\_above
* android:layout\_below
* android:layout\_toLeftOf
* android:layout\_toRightOf
* android:layout\_alignLeft
* android:layout\_alignRight
* android:layout\_alignTop
* android:layout\_alignBottom
* android:layout\_alignBaseline

Posición relativa al layout padre:

* android:layout\_alignParentLeft
* android:layout\_alignParentRight
* android:layout\_alignParentTop
* android:layout\_alignParentBottom
* android:layout\_centerHorizontal
* android:layout\_centerVertical
* android:layout\_centerInParent

Por último indicar que cualquiera de los tipos de layout anteriores poseen otras propiedades comunes como por ejemplo los márgenes exteriores (margin) e interiores (padding) que pueden establecerse mediante los siguientes atributos:

Opciones de margen exterior:

* android:layout\_margin
* android:layout\_marginBottom
* android:layout\_marginTop
* android:layout\_marginLeft
* android:layout\_marginRight

Opciones de margen interior:

* android:padding
* android:paddingBottom
* android:paddingTop
* android:paddingLeft
* android:paddingRight

**ConstraintLayout**

Actualmente es el layout recomendado por Google para definir cualquier interfaz de cierta complejidad. Aunque con los tipos de layout descritos hasta este momento en esta entrada podríamos definir casi cualquier interfaz, habitualmente anidando unos dentro de otros, el nuevo ConstraintLayout tiene la ventaja de ser capaz de conseguir los mismos resultados sin necesidad de anidar diferentes layouts, es decir, con un árbol o jerarquía de componentes mucho más «plana», y por tanto más eficiente.

La ventaja que ofrece este Layout a la hora de diseñarlo es que puedes poner una relación de cada elemento con otros elementos o con el mismo Layout haciéndolo totalmente responsive, teniendo la seguridad de que el control o elemento se mantendrá en su posición relativa dentro de la pantalla sea cual sea su tamaño.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

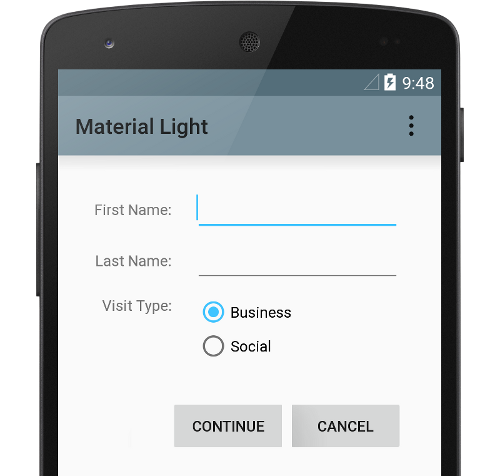
**Subtema 5.3:**

Widgets Básicos en Android.

**Ejemplos:**

Entre los principales widgets tenemos:

* TextView: Etiquetas de texto.
* EditText: Caja de texto.
* RadioButton: Selección de opción.
* Button: Botón para una acción.



**Actividad:**

1. CUESTIONARIO TÉCNICO

* ¿Cuál es la finalidad de utilizar layouts en nuestros diseños de interfaces de usuario?
* Detalle la diferencia entre in TableLayout y un GridLayout
* Indique la utilidad de las propiedades android:layout\_row y android:layout\_column al trabajar con un Grid\_Layout
* Indique cuándo se utiliza la propiedad match\_parent al trabajar con Layouts
* ¿Por qué Google recomienda el uso de Constrainr Layout al diseñar las interfaces de usuario?

1. CONCLUSIONES DE LA EXPERIENCIA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_