**UI en Android: ¿Por qué? ¿Cómo?**

En reasumen:

**XML**(eXtensible Markup Language):  
es un lenguaje de etiquetas, es decir, cada paquete de información está delimitado por dos ***etiquetas/tags***  
como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación.

<H1>Mateo</H1> <--- HTML

<Nombre>Mateo</Nombre> <--- XML

<H1> y <Nombre> son etiquetas/tags. Ambas encierran el texto o paquete de información *Mateo*. <br>  
La etiqueta <H1> es de HTML, y se encarga de mostrar visualmente el texto *Mateo* en la página web en un tamaño  
determinado pero no dice nada del significado de *Mateo*: si es una ciudad o un nombre, por ejemplo. <br>  
En cambio la etiqueta <Nombre> es de XML y nos dice que *Mateo* es un nombre de persona,  
por lo tanto XML se preocupa del significado del texto que encierra y no de la apariencia de cómo se muestre el texto en la página web.  
Por eso se dice que XML es un lenguaje de etiquetas, que como hemos dicho anteriormente, separa el contenido de la presentación.<br><br>  
Los **tags** (*etiquetas*) pueden tener **atributos**, que son una manera de incorporar características o propiedades a las etiquetas de un documento.  
El atributo **consta de dos partes**: La **propiedad** del elemento y el **valor** de la propiedad,  
que siempre va entre comillas doble (“) o simple (‘).

<TAG atributo="valor">

Contenido

</TAG>

<TAG-2 atributo="valor"/>

Ejemplo practico:

<Datos-Nacimiento>

<Persona>

<Nombre>Mateo</Nombre>

<Fecha>15.10.2012</Fecha>

<Ciudad>Madrid</Ciudad>

<Peso>3.1Kg</Peso>

<Estatura>45cm</Estatura>

</Persona>

<Persona>

<Nombre>Maribel</Nombre>

<Fecha>11.09.2012</Fecha>

<Ciudad>Sevilla</Ciudad>

<Peso>3Kg</Peso>

<Estatura>40cm</Estatura>

</Persona>

</Datos-Nacimiento>

* **namespaces**: es una forma de dar un *contexto* (scope), a un tag.  
  Es definir donde un elemento tiene un valor definido.
* <NAMESPACE:TAG atr="val">
* Contenido
* </NAMESPACE:TAG>

En otras palabras, es un medio para organizar clases dentro de un entorno,  
agrupándolas de un modo más lógico y jerárquico. <br>  
Ejemplo:

<Profesor:Luis

trabajo:horario="9-11"

trabajo:email="luisprof@uni.com"

persona:id="@user9283">

</Profesor:Luis>

<br>  
En Android veremos mucho el ***namespace*** de Androdid en la siguiente forma:

<TextView android:text="Hola mundo" />

*¿Pero como saber de donde proviene un namespace?*<br>  
Con la palabra reservada xmlns y con un URI/dirección que harà a entender de donde  
un xml tiene su procedencia, de esa manera xml va a poder entender cuando un valor tiene  
o no tiene un contexto y cuando puede darle o asignarle un comportamiento.<br>  
Siguiendo el ejemplo anterior:

<... xmlns:android="http://schemas.android.com/apk/res/android"/>

android es un alias asignado por nosotros. Podemos ver esto como una forma de encapsular otra etiqueta dentro una clase *Android*.

# Enlazando nuestro layout con el código

**Estructura de res:**

* drawable: Representa gráficos (Todo aquello que pueda ser dibujado en una pantalla).
* layout: Representa todas las estructuras de pantallas que creemos.
* mipmap: Aquí guardaremos iconos.
* values: Aquí administraremos los recursos (Colores, cadenas, dimensiones o arreglos).

**Recursos más complejos:**

* font: Aquí guardaremos las fuentes de la aplicación.
* anim: Contendrá xml para animaciones.
* xml: Contendrá preferencias de usuario y datos más complejos.
* raw: Contendrá archivos como vídeos o audios.

En reasumen:  
Cuando compilas tu aplicación, cada archivo de diseño XML se compila en un recurso View.  
Debes cargar el recurso de diseño desde el código de tu aplicación, en la implementación de devolución de llamada Activity.onCreate().  
Para eso, llama a setContentView() pasando la referencia a tu recurso de diseño en forma de R.layout.layout\_file\_name.

Por ejemplo, mirando el diseño XML *activity\_main.xml* que nos crea Android Studio al crear un *empty project*,  
para poder cargarlo en nuestra actividad lo hariamos de la siguiente manera:

**override** **fun** **onCreate**(savedInstanceState: **Bundle?**) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

}

La clase **R** es una clase que **tiene referencia directa a los recursos dentro del package *res***.  
Por ejemplo podriamos acceder al valor *app\_name* dentro del recurso *strings.xml* asi:

println(R.string.app\_name)

# ViewGroup y View: Diferencias básicas

En reasumen:

* **View**: es un elemento que se va a mostrar por pantalla así como tal.
  + TextView
  + ImageView
  + EditText
* **ViewGroup**: es un elemento que sirve para agrupar elementos. Nota: cuando tenemos un grupo de elementos dentro de un ViewGroup, los cambios que hagamos sobre este afectaran los elementos dentro de el.  
  Ejemplo:
* <LinearLayout android:gravity="start"
* <TextView />
* <ImageView />
* <EditText />

</LinearLayout>

En reasumen:  
android:layout\_width,android:layout\_height: nos permiten especificar el ancho y la altura con medidas exactas,  
aunque probablemente no quieras hacerlo con mucha frecuencia.

Generalmente, usarás una de estas constantes para establecer el ancho o la altura:

* **wrap\_content**: indica a tu vista que modifique su tamaño conforme a los requisitos de este contenido.
* **match\_parent** indica a tu vista que se agrande tanto como lo permita su grupo de vistas principal.

En general, no se recomienda especificar el ancho ni la altura de un diseño con unidades absolutas como píxeles. En cambio, el uso de medidas relativas como unidades de píxeles independientes de densidad (**dp**), **wrap\_content**  
o **match\_parent** es un mejor enfoque, ya que ayuda a garantizar que tu aplicación se muestre correctamente en dispositivos con pantallas de diferentes tamaños.

# Otros atributos y el namespace tools

<h1>En reasumen:</h1>

## android:id

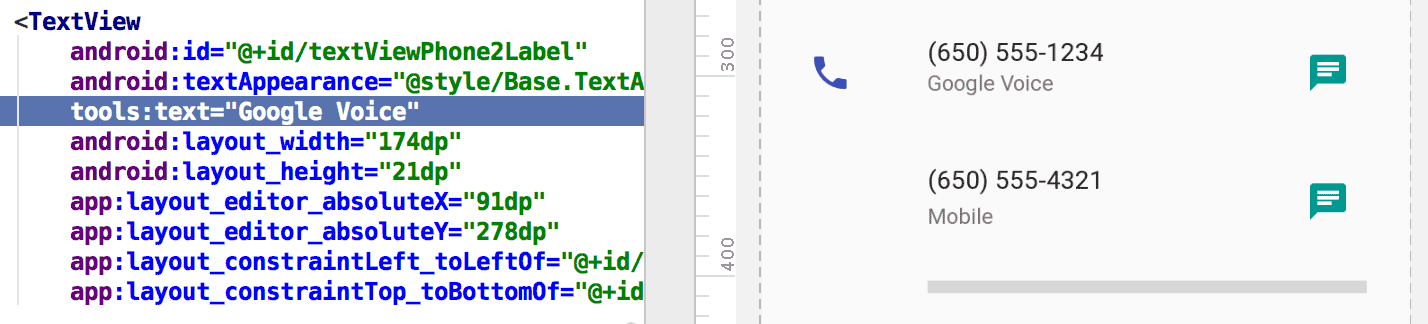
Cualquier objeto **View** puede tener un **ID** entero asociado para identificarse de forma única dentro del árbol.  
Cuando se compila la aplicación, se hace referencia a este ID como un número entero, pero el ID se asigna normalmente en el archivo XML de diseño como una string del atributo id.  
Este es un atributo XML común para todos los objetos View (definido por la clase View) y lo utilizarás muy a menudo.  
La sintaxis de un ID dentro de una etiqueta XML es la siguiente: android:id="@+id/my\_button"

## Atributos específicos y atributos compartidos

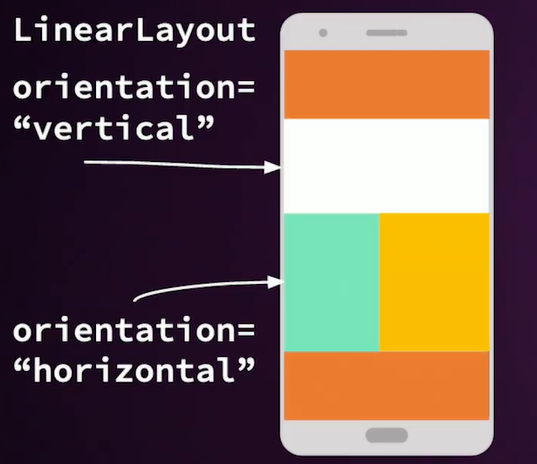
Existen atributos compartidos y otros específicos para cada elemento.  
Por ejemplo, el atributo android:background, que nos permite establecer un color, es un atributo que existe para todos los elementos de android; el atributo android:hint="Enter password" es un atributo especifico de la View EditText

## Namespace tools

Si en algún momento no quieres utilizar un atributo especifico y quieres solamente utilizarlo en tiempo de diseño para poder ver como se vería, existe un namespace denominado tools.  
Es un namespace que nos permite ver en tiempo de diseño como se verá nuestra aplicación sin generar una versione final de la aplicación con esos valores.

Por ejemplo, si se establece el valor del atributo android:text durante el tiempo de ejecución o si quieres ver el diseño con un valor diferente del valor predeterminado, puedes agregar tools:text para especificar texto solo para la vista previa de diseño.  


# LinearLayout: Organizacion lineal



**LinearLayout**  
Alinea los elementos de manera secuencial según su orientación, no se recomienda utilizar este tipo de layout si se va a trabajar con elementos muy grandes; también añadir que dentro de un LinearLayout pueden convivir otros LinearLayout añadidos. Existen 2 posibles orientaciones:  
**OrientationVertical:** Los elementos se alinean de arriba hacia abajo.  
**OrientationHorizontal:** Los elementos se alinean de derecha a izquierda

**Drawable-mdpi:** Recursos para pantallas de densidad media (mdpi) (~ 160dpi). 48x48 (línea base 1.0x) para densidad media (mdpi)  
**Drawable-hdpi:** Recursos para pantallas de alta densidad (hdpi) (~ 240 ppp). 72x72 (1.5x) para alta densidad (hdpi)  
**Drawable-xhdpi:** Recursos para pantallas de densidad extra alta (xhdpi) (~ 320dpi). 96x96 (2.0x) para densidad extra alta (xhdpi)  
**Drawable-xxhdpi:** Recursos para pantallas de extra-extra-alta-densidad (xxhdpi) (~ 480dpi). 144x144 (3.0x) for extra-extra-high-density (xxhdpi)  
**Drawable-xxxhdpi:** Recursos para usos extra-extra-extra-alta-densidad (xxxhdpi) (~ 640dpi). 192x192 (4.0x) para densidad extra-extra-alta-alta (xxxhdpi)

**File->New->VectorAset:** En esta ruta encontraremos iconos, formas y figuras propias de Android para el diseño de interfaces. Los íconos se definen como archivos xml que a través de vactores dibujan imágenes. Es importante cambiarles el nombre, y si accedemos a ellos podemos cambiar ciertas características de la figura, como el color y tamaño. Siempre se guardan dentro del directorio drawable.  
android:fillColor="@color/icon\_color"

**<ImageView/>**  
Con esta etiqueta declaramos imágenes dentro del código xml. Eventualmente para declarar la ruta en la que se encuentra la imagen nos apoyamos de Android:src=”ruta del archivo”.  
Dentro de esta etiqueta también debemos de especificar el layout\_width y el layout\_height de la imagen.

<ImageView

android:layout\_gravity="center"

android:layout\_width="160dp"

android:layout\_height="120dp"

android:src="@drawable/platzi" />

**Density Point:** Densidad de puntos(dp), esta es la unidad de medida para el tamaño de las imágenes. Android lo trabaja de una manera porcentual, de forma que es la máquina virtual de Android la que decide el tamaño de dicha imagen según los tamaños de pantalla de cada dispositivo.

**Layout\_gravity**  
Este elemento afecta solamente a los elementos en el que se declara. Se utiliza para dar una posición a los elementos dentro de la pantalla.  
android:layout\_gravity=“center”

**layout\_margin**  
Se utiliza para dar un contorno al elemento en el que se declara, por ejemplo se pueden declarar dentro de 2 EditText para separarlos uno del otro.  
android:layout\_margin=“12dp”

**drawablePadding**  
Se utiliza para separar íconos y texto que yacen dentro de un mismo view. Por ejemplo separar un ícono y el texto dentro de un EditText. Su unidad de medida es dp.  
android:drawablePadding="12dp"

**drawableStart**  
Esta línea de código funciona para declarar íconos dentro de un EditText. Recordar que para que esto sea posible, el ícono ya debe de estar dentro del directorio drawable.  
android:drawableStart="@drawable/vector\_person"

**<Space/>**  
Se utiliza para separar elementos mediante “espacios”, de forma que ocupan un lugar de la pantalla pero sin ser visibles en pantalla.

<Space

android:layout\_width="match\_parent"

android:layout\_height="0dp"

android:layout\_weight="1"/>

En este ejemplo de Space, se declararon los siguientes atributos:  
**Layout\_width=”match\_parent”** ◊ Esto hace que el espacio tome todo el ancho en el espacio que se le asignó  
**Layout\_height=”0dp”** ◊ Esto inhabilita el height/altura, para dar continuidad al atributo weight  
**Layout\_weight=”1”** ◊ Hace que el elemento que le sigue(puede ser un botón) se vaya hasta el fondo de la pantalla, de esta forma el espacio ocupa casi todo el espacio disponible en pantalla y así separando los EditText y el botón.

**<Button/>**  
Esta etiqueta se utiliza para crear botones, dentro del mismo se pueden declarar los siguientes atributos:

android:layout\_marginStart="12dp"

android:layout\_marginEnd="12dp"

android:background="@color/colorPrimary"

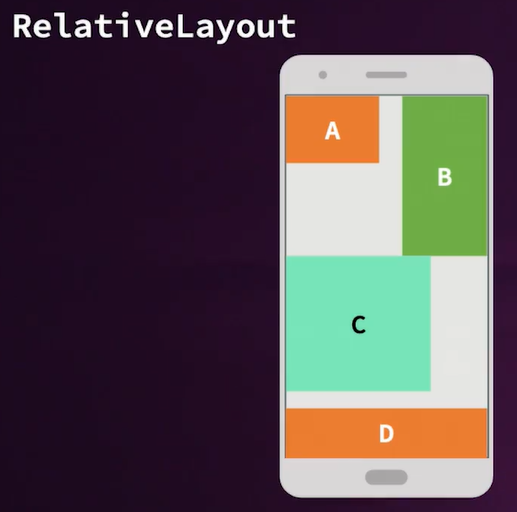
android:text="@string/button\_login"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

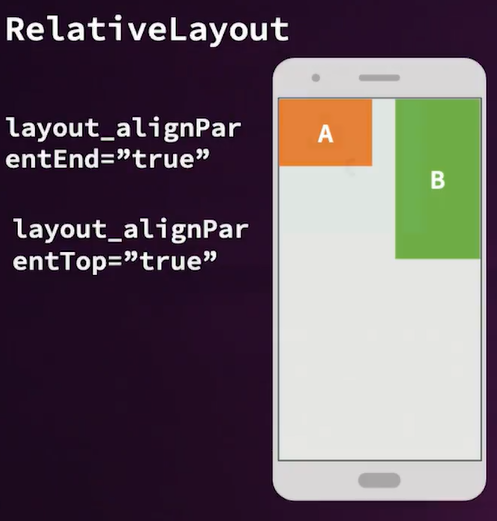
**layout\_marginStart=”12dp”** ◊ se utilizó para dar un espacio entre el borde izquierdo de la pantalla y el botón.  
**layout\_marginEnd=”12dp”** ◊ se utilizó para dar un espacio entre el borde derecho de la pantalla y el botón.  
**Background** ◊ color de fondo, traído desde el archivo colors.xml  
**Text** ◊ texto dentro del botón, importado del archivo Strings.xml  
Width y height ◊ tamaño del botón

# RelativeLayout: organizando con referencias

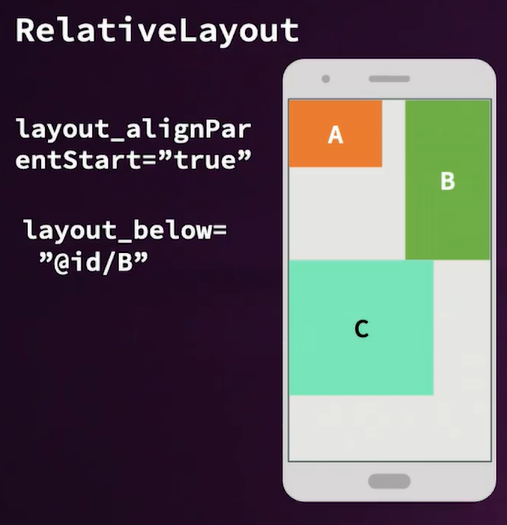


Posicionar el elemento “A” en la parte superior izquierda 

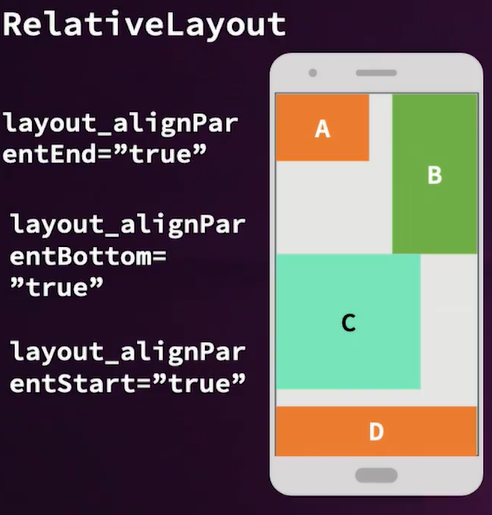
Posicionar el elemento “b”



Posicionar el elemento “c”



Posicionar el elemento “d”



Documentación y Referencia de RelativeLayout

<https://developer.android.com/guide/topics/ui/layout/relative>

<https://developer.android.com/reference/kotlin/android/widget/RelativeLayout?hl=en>

# RelativeLayout: organizando con referencias

# https://i.imgur.com/WbcTCZU.png

# FrameLayout: Uso práctico

El framelayout normalmente se utiliza para tener solo un elemento el cual podras sustituir como una imagen, si pones mas vistas estan se iran superponiendo como una pila de capas,donde la mas reciente agregada se vera primero , asi que algunos lo usan para sus placeholder. otros para mostrar un elemento encima de otro, por las capas. pero no es usado para diseñar toda una interfaz, como el login, imaginate poner margin o gravity a todo.

Un dato: Si es cierto q los dp es una unidad de medida que usa android para lo que es el espacio en pantalla. Fue creada para estandarizar una forma de medida, ya que Android tiene una gran cantidad de pantallas con densidades de pixeles muy variadas.  
Bueno pero **android teniendo esta medida de DPs, divide cualquier pantalla en una cuadricula (como un cuaderno cuadriculado) donde cada cuadradito es de 8dp x 8dp,** por esa razon es bueno intentar diseñar teniendo esa regla para las medidas de los views, paddings, margenes, imagenes,iconos,etc. : multiplos de 8 (8dp.16dp, 24 dp), por ejemplo si vas a poner altura a un appBar, no pongas 60, ponle 64, o si pones una altura para esa imageview y quieres poner 20 no pongas 20, ponle 16 o 24, osea siempre intenta llevar tus medidas a multiplos de 8 , es una recomendacion que hacen en la documentacion oficial. esto explica algo que veran constantemente en Android Studio, cuando el IDE autogenera alguna medida normalmente pone 8dp, 16dp, 24dp, etc . por ejemplo cuando estas usando los constraint layout y quieres poner margenes ,android studio te sugiere 8,16,24,etc…esto es por\*\* la famosa regla de 8\*\* anteriormente mencionada.

# Layouts externos: ConstraintLayout

Es importante que todos los elemetos dentro de un constraintLayout deben tener al menos una alineación horizontal y una alineación vertical; sino AS no va a permitirnos compilar nuestro proyecto.

Para estructurar el XML de manera ordenada aplicaremos el comando: **CRTL+ALT+L**

El ConstraintLayout es que se encarga de que los elementos estén estirados con respectos a otros.  
**Ejemplo:** Si se hiciera invisible el elemento usuario el elemento password heredaría la configuración previa de usuario.

# ¿Qué es un estilo?

un buen post para diferenciar un estilo y un tema.  
<https://medium.com/androiddevelopers/android-styling-themes-vs-styles-ebe05f917578>

Crear un estilo

*<!-- Estilo hijo-->* <**style name="AppTheme.EditTextStyle" parent="AppTheme"**>  
 <**item name="android:textSize"**>48sp</**item**>  
 <**item name="android:drawablePadding"**>12dp</**item**>  
 </**style**>

Agregar el estilo a un elemento

**style="@style/AppTheme.EditTextStyle"**

Puedes definir:

<style name="AppTheme.MyView">

Esto ya dice MyView va a heredar de AppTheme

En lugar de:

<style name="AppTheme.MyView" parent="AppTheme">

Pues el . es una forma de definir la herencia, si defines el atributo parent le dices a android que ignore esto y herede de ese estilo especifico.

# ¿Qué es un tema?

La diferencia entre en tema y un estilo es que los temas se aplican a todos los componentes del tema padre y los estilos se aplican a componentes específicos.

Los temas y los estilos se crean de una manera similar.

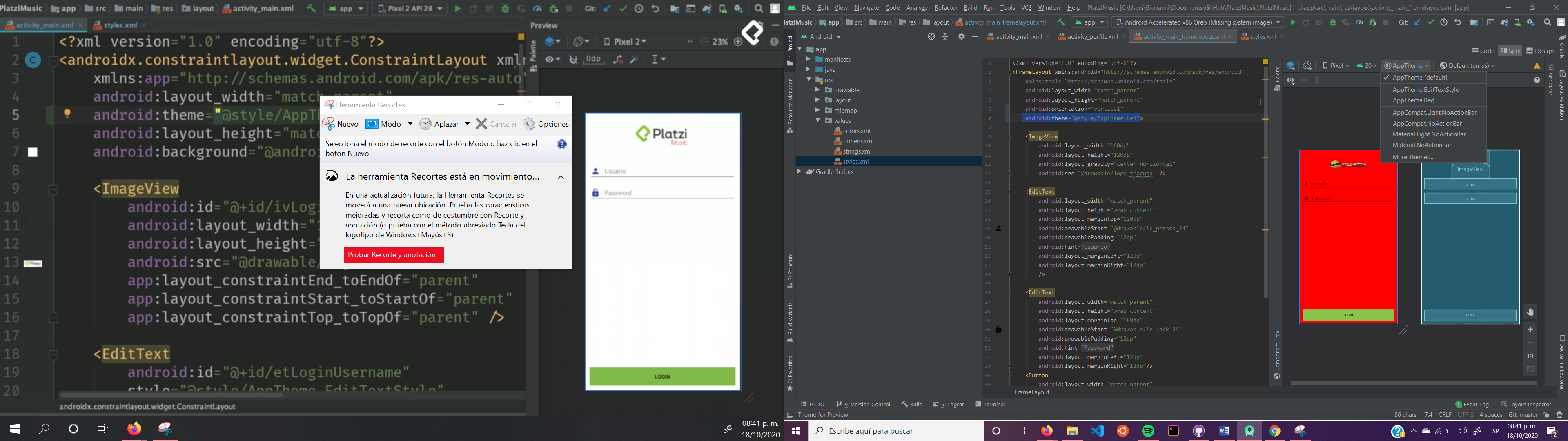
Creación de un tema

<!-- Tema -->  
<style name="AppTheme.Red" parent="AppTheme">  
 <item name="android:background">#FFFF0000</item>  
</style>

Llamar el tema

android:theme="@style/AppTheme.Red"

Si no podemos visualizar el tema que aplicamos en el layuot padre, debemos seleccionarlo en este apartado



Documentación oficial de temas y estilos.

<https://developer.android.com/guide/topics/ui/look-and-feel/themes>

Tener estilos es una manera correca de compartir atributos para nuestros widgets, compartir atributos que representan como se ve la aplicación. Es una buena practica si en algún momento hay que rediseñar.

**Tema:** Es un estilo aplicado globalmente, quiere decir que cuando hagamos alguna modificación todos los elementos van a representar ese cambio.

**Archivo Manifest:** Es la sección donde definimos las pantallas que tenemos con respecto al código. (También es afectado por los temas)

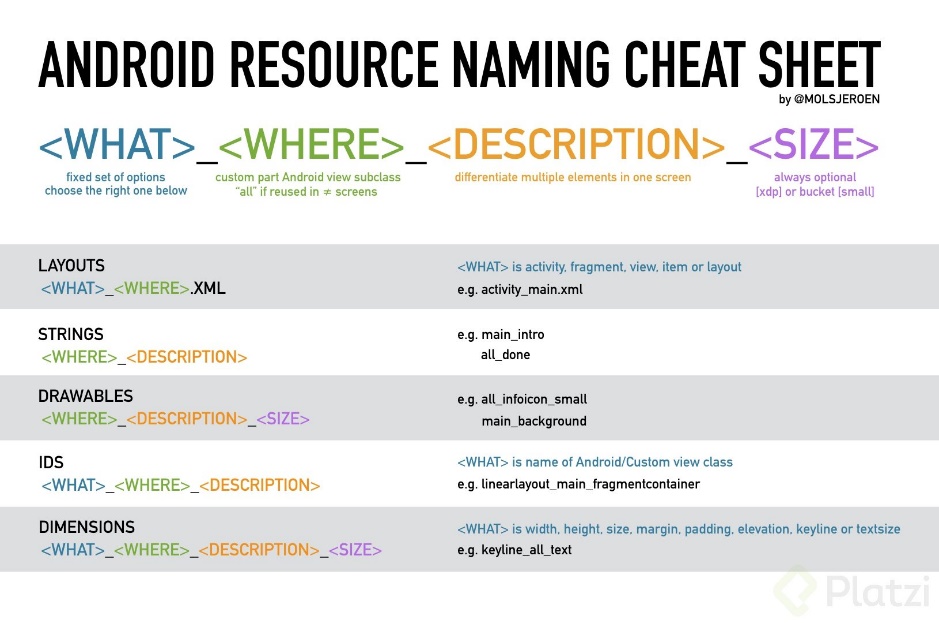
Un **Style** aplica sus reglas de manera independiente.  
Un **Theme** aplica sus reglas de manera global

Los **estilos** es una manera apropiada de compartir atributos para los elementos de manera individual.  
Los **temas** sirven para los mismo solo que aplicado globalmente, es decir, va a aplicar para viewgroups, layouts o para toda la aplicación. Cuando hagamos una modificación en un tema todos los elementos involucrados van a ver reflejado este cambio.

# Distribuyendo nuestros valores

Primero se resuelve el elemento, después el estilo y al final se tomará el tema

**IMPORTANTE**  
Si Android encuentra un **tema** aplicara este. Pero, si un elemento tiene un **estilo** aplicara el estilo. No obstante, si un elemento tiene un **atributo único interno** va a referenciar a este.  
**Ejemplo:** Esto quiere decir que si nosotros seteamos un fondo en el tema pero un elemento como un texto tiene su propio fondo, va a predominar el fondo del elemento, no del tema.



# Agregando widgets externos

Les dejo el link al repo de [Circle View Image](https://github.com/hdodenhof/CircleImageView)

Me pareció muy interesante el tema de los widgets, aquí les dejare un link donde explican como crear un widget desde la app : <https://developer.android.com/guide/topics/appwidgets?hl=es-419>

Agregaremos una **Librería** que nos permitirá colocar una foto de forma circular, para esto copiamos la siguiente librería:

implementation ‘de.hdodenhof:circleimageview:3.1.0’ [Enlace del Github](https://github.com/hdodenhof/CircleImageView)

La pegamos en los archivos de Gradle > build.gradle(Module:app)  
y la pegaremos dentro de **dependencies**; posteriormente Android pedirá que se sincronice el proyecto (Barra azul superior)  
[Lo que hace android es descargar toda la librería]  
\*\*\*Luego regresaremos a Usage y copiaremos el código y lo pegaremos en el activity.

ConstraitLayout a diferencia de RelativeLayout **no va a estirar las vistas**

Buenas tardes!! en el video menciona que con ConstraintLayout no es posible estirar elementos, pero he visto que si es posible añadiendo 0dp en width y utilizando constraintStart y constraintEnd, y 0dp en height y usando constraintTop y Bottom…

Saludos!!

# Reutilizando elementos

La diferencia entre Include y Merge es que **Include** va a halar todos los elementos que ya tengas en el Layout.

Merge: simplemente los va a agregar como si fueran elementos presentes en tu Layout.

Para centrar el logo, simplemente vamos a merge\_logo y ahi agregamos los constraint que centran el view.

Lo que sucede con merge es que android hace de cuenta como si ese ImageView fuera sido declarado dentro de nuestro propio layout donde lo agregamos, el no se da cuenta que viene de otra parte.

Entonces en merge\_logo podriamos definir como quiero que se vea mi elemento en los otros layout donde lo voy a invocar.