Programación Multimedia y Dispositivos Móviles

UD 3. Diseño de aplicaciones móviles

Javier Carrasco

Curso 2024 / 2025

Diseño de aplicaciones móviles

3. Diseño de aplicaciones móviles

Este capítulo es un inciso en el camino del desarrollo de aplicaciones móviles, aunque el tema tratado está muy ligado, concretamente afronta el tema del diseño, del aspecto de las aplicaciones, pero únicamente mediante ligeras nociones en lo que a ello respecta, ya que no es el objetivo principal de este libro. También se verán otro tipo de recursos que ayudarán en el diseño, pero más desde un punto de vista organizativo.

3.1. Material Design

Material Design[1] fue presentado durante la *Google I/O 2014*. Según Google, *Material Design* es un **concepto**, unas pautas a seguir enfocadas al diseño utilizado, pero no solo en **Android**[2], también se pueden aplicar a la web y otras plataformas.

Puedes encontrarás una gran cantidad de recursos gratuitos en la web para poder aplicarlos en los proyectos desarrollados.

- *Material Theme Builder* (https://m3.material.io/theme-builder)
- *Material UI* (https://materialui.co/)
- *Material Design Palette* (https://www.materialpalette.com/)
- *Google Fonts Icons* (https://fonts.google.com/icons)
- *Icons8* (https://icons8.com/)
- *Iconmonstr* (https://iconmonstr.com/)

Básicamente siguen las "normas" establecidas por Google en el campo de *Material Design*. Estos recursos permitirán automatizar ciertas tareas de diseño para aplicar a los proyectos. Usando ***Material Theme Builder*** (*Custom*) para modificar la aplicación *HelloWorld!!*, la idea es añadir nuevos colores al recurso `colors.xml` y se actualizarían los temas.

El primer paso será acceder al recurso web para buscar una combinación de colores que se adapte a las necesidades.

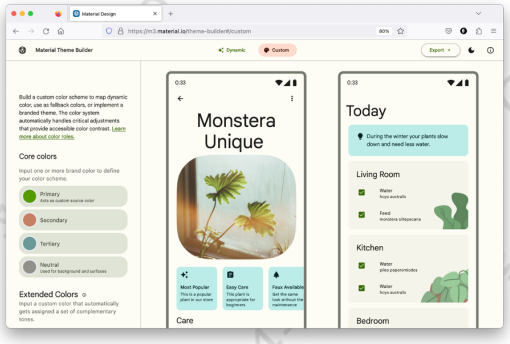


Figura 1

1 Material Design (https://material.io/)
2 Material Design for Android (https://developer.android.com/guide/topics/ui/look-and-feel)

El siguiente paso será hacer la selección de colores, si te fijas, verás que al ajustar el color primario a tu gusto, el resto se adaptará, aunque también puedes modificarlos manualmente. El recurso te irá mostrando como queda la combinación elegida en diferentes vistas de la aplicación.

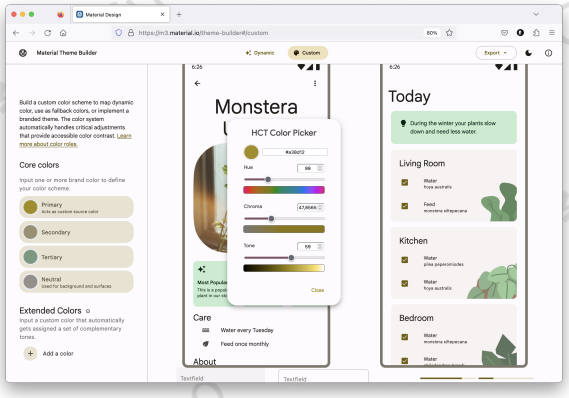


Figura 2

Una vez obtengas la paleta que te guste, descárgala utilizando la opción *Export*, que encontrarás en la esquina superior derecha, seleccionando la opción *Android Views (XML)*. El resultado de la descarga será un fichero llamado `material-theme.zip`. Este archivo contendrá la estructura que puedes ver en la figura adyacente. Puedes sobrescribir los ficheros directamente o copiar el contenido. Esta segunda opción puede evitarte problemas en el *manifest* con el nombre del tema.

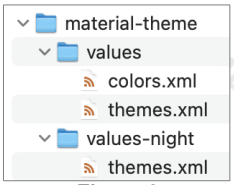
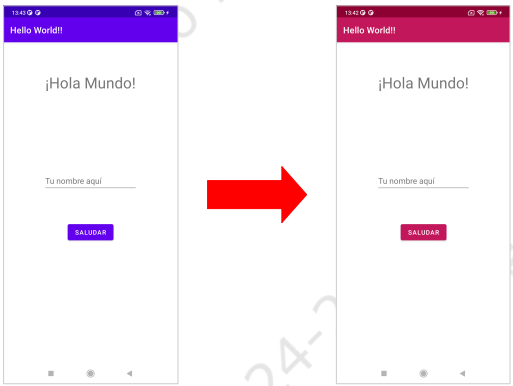


Figura 3

Si te fijas en el código, se han adaptado todos los colores salvo el que pertenece a la propiedad `statusBarColor`, que ya coge su valor desde el atributo existente `colorPrimaryVariant`. El resultado puede ser algo parecido, si has coincidido en los colores, a lo que se muestra en la siguiente imagen. Recuerda cambiar el color asignado al botón en el código.



Como habrás observado, `themes.xml` tiene una variante *night*, para tema oscuro, deberás hacer lo mismo para que se adapte a esa posibilidad.

A continuación, se añadirá un icono a la aplicación, en el *Button* se incluirá un icono de los que se pueden encontrar en **Android Studio**. Para ello, abre el fichero `activity_main.xml` y selecciona el botón. Ya dentro, ve a la vista de código, y añade las siguientes líneas en la etiqueta del *Button* para asignarle un icono y seguir haciendo uso de *Material*.

```
<Button
    ...
    app:icon="@android:drawable/star_on"
    app:iconGravity="textStart"
    app:iconTintMode="multiply"
    ... />
```

En concreto, con la primera propiedad se indica el icono que se desea añadir, fíjate que justo a la izquierda de la línea aparecerá el icono seleccionado, si pulsas sobre él se abrirá un navegador de imágenes para buscar una imagen de manera gráfica. La segunda, `iconGravity`, indica la posición del icono dentro del botón, y la tercera propiedad, `iconTintMode`, permite definir la relación del icono con el color de fondo del botón. Prueba a modificar los valores de las dos últimas propiedades para ver el efecto.

Ahora la aplicación deberá lucir como se muestra en la siguiente imagen, puede variar según el icono que hayas escogido y el color de fondo del botón.

Ahora se personalizará el icono, esta vez se utilizarán los iconos de ***Google Fonts*** (https://fonts.google.com/icons). Se buscará uno que identifique la opción de saludar, una vez lo tengas, bastará con pinchar sobre el icono y en el desplegable que aparece a la derecha, seleccionar la opción para **Android** y descargar.

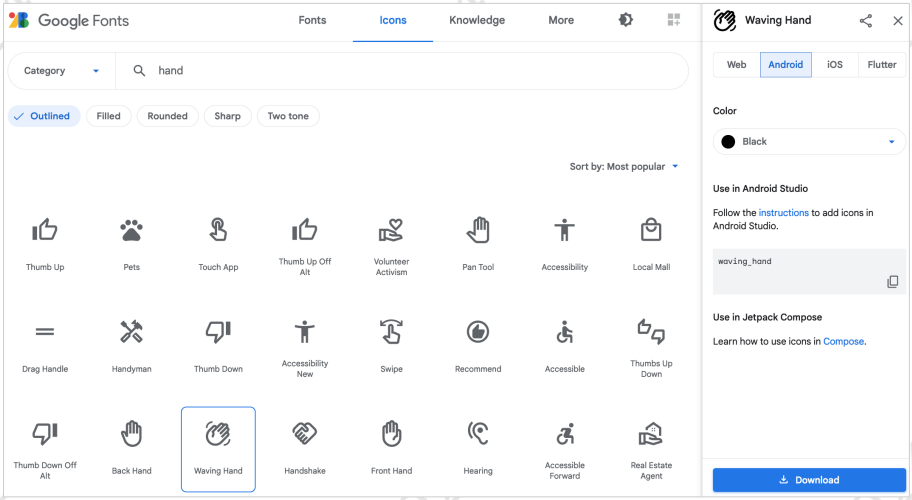


Figura 4

El fichero descargado contiene el icono en diferentes densidades estructurado en distintas carpetas. La forma de añadir todas las densidades es la siguiente, debes abrir la pestaña ***Resource Manager*** que encontrarás a la izquierda del IDE.

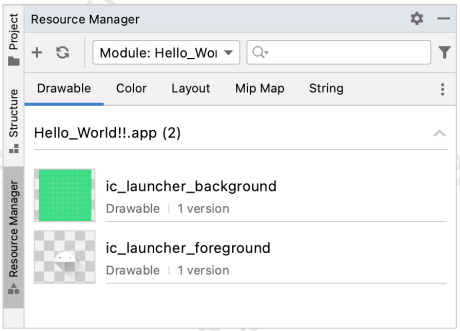


Figura 5

A continuación, utilizando el símbolo más "**+**" de la esquina superior izquierda, utiliza la opción ***Import Drawables*** para seleccionar la carpeta raíz que contiene el icono con todas sus densidades, esta acción abrirá una vista previa de la importación a realizar, donde aparecerá todo por defecto, si quieres eliminar algún icono de la importación, bastará con pinchar en "*Do not import*". Tras pulsar el botón *Next*, se mostrará el resumen de la importación y, al pulsar *Import*, se añadirán los iconos. Verás el cambio en el *Resource Manager* y en la estructura del proyecto.

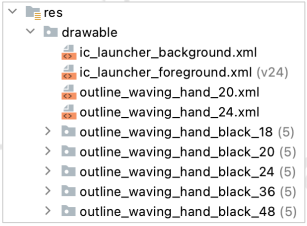


Figura 6

Para hacer uso del nuevo icono añadido, bastará con modificar la propiedad `app:icon` añadida al *Button* en el *layout*. Añade la que más se adapte a tu diseño.

```
1  app:icon="@drawable/outline_waving_hand_24"
```

Este mismo sistema que se acaba de ver para añadir imágenes, se puede utilizar para añadir una única imagen, por ejemplo, para crear el icono de la aplicación, la imagen que se verá desde el sistema operativo. Si añades una sola imagen, para el siguiente ejemplo se añadirá la imagen `outline_emoji_people_black_48dp.png` que puedes encontrar en *Google Fonts*, se optará por la de mayor densidad, para mantener la calidad de la imagen. Tras realizar la acción, en este caso, al no existir ficheros XML con información de la imagen, únicamente se añade el fichero PNG.

Para activar la imagen recién añadida como icono de la aplicación, bastará con modificar las siguientes dos propiedades de la etiqueta ***application*** del fichero `AndroidManifest.xml`. La propiedad `icon` establece el icono de la aplicación, la segunda, `roundIcon`, si se dispone de un icono diferente para máscaras circulares, que no sería el caso.


```
1  android:icon="@drawable/outline_emoji_people_black_48dp"
2  android:roundIcon="@drawable/outline_emoji_people_black_48dp"
```

3.1.1. Image Asset

Otra opción para añadir imágenes a un proyecto es el uso de ***Image Asset***. Para ello, se utilizará una imagen descargada en tu ordenador, por ejemplo, la imagen PNG `ic_pan_tool_black_48.png`. Como se comentó en el capítulo anterior, el nuevo icono irá ubicado en la carpeta `res/mipmap`, al seleccionar la carpeta, pulsa con el botón derecho y selecciona la opción ***New > Image Asset***, esta opción abrirá la siguiente ventana, ***Asset Studio***.

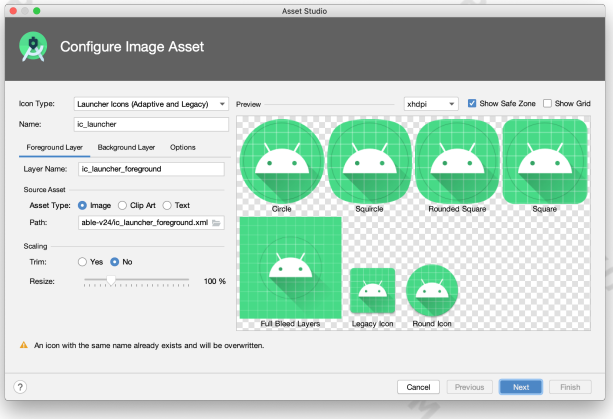


Figura 7

Ahora añade la imagen deseada mediante la opción *Path*, modifica los valores como se ve en la imagen, o prueba según tus preferencias en la sección *Foreground Layer*. Para conseguir un icono con fondo transparente, en la opción *Icon Type* deberás seleccionar *"Action Bar and Tab Icons"*.

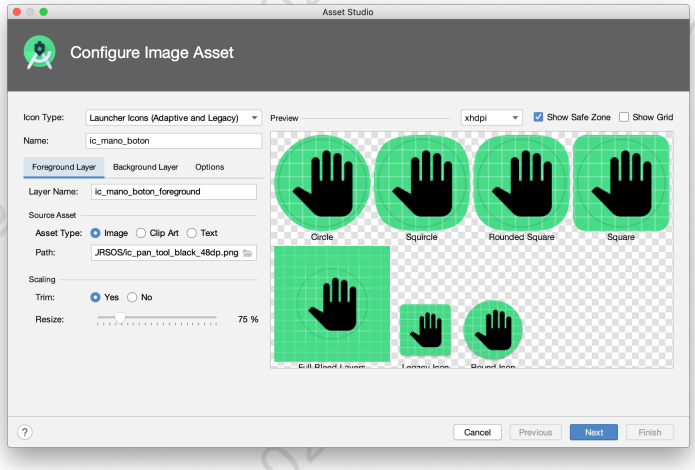


Figura 8

A continuación, en la pestaña *Background Layer,* puedes ajustar el fondo que desees para la imagen, utilizando un color sólido o una imagen.

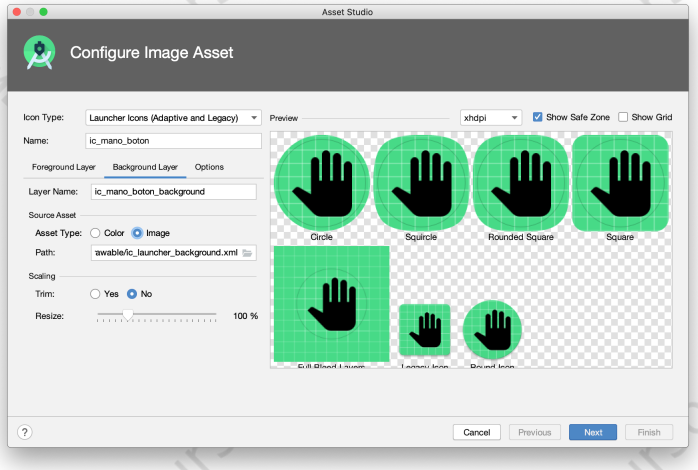


Figura 9

En la pestaña *Options* encontrarás opciones de creación de la imagen/icono, como por ejemplo, para APIs inferiores a la versión 25, para la propia 25 y el icono para *Google Play Store.* Pulsa el botón *Next* para terminar con el proceso.

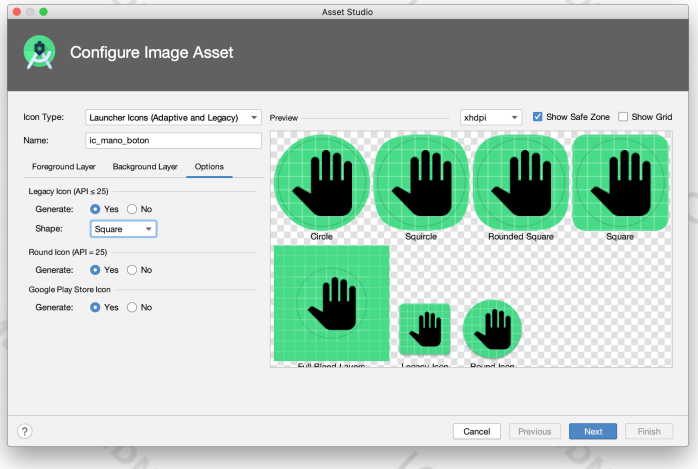


Figura 10

Se mostrará un resumen con todo lo que se creará a continuación, detalles del fichero de salida y las ubicaciones que ocuparán los distinto archivos creados.

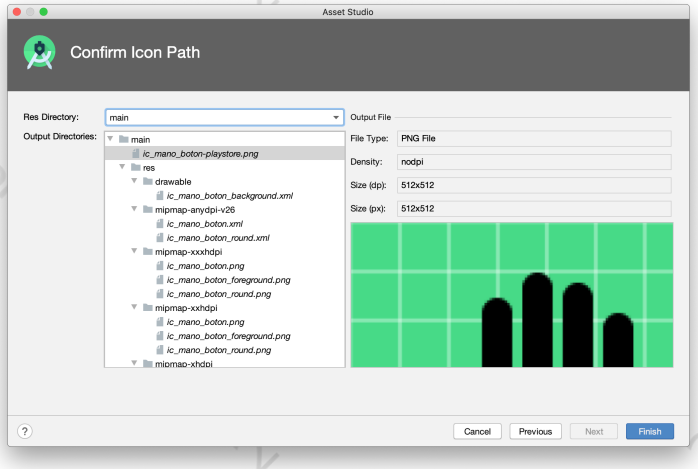


Figura 11

Podrás observar como cambia el directorio `res/mipmap`, apareciendo la nueva incorporación en sus diferentes densidades. Ahora, cambia el icono utilizado en el botón tal como has hecho en el apartado anterior.

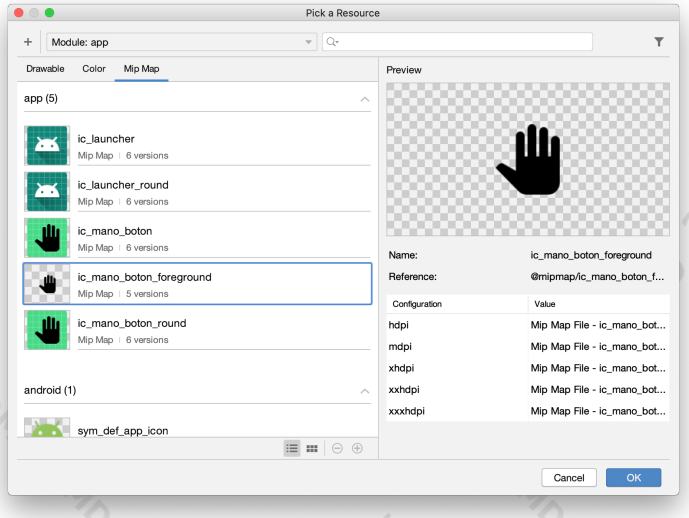


Figura 12

El resultado que obtendrás debería ser el que muestra la siguiente figura.

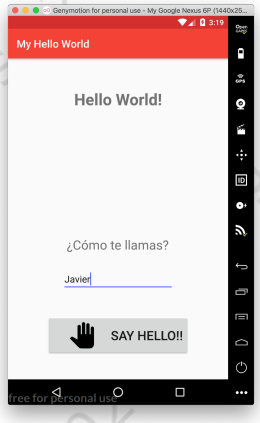


Figura 13

3.1.2. Vector Asset

Esta herramienta, a diferencia de *Image Asset*, permite añadir una imagen vectorial al proyecto, creando un fichero XML que adaptará, o re-dibujará, la imagen según se necesite. Para lanzar el asistente se utilizará la opción ***New > Vector Asset***, se abrirá de nuevo la ventana **Asset Studio**, pero con la siguiente configuración, será necesario disponer de un fichero en formato SVG, para el ejemplo se utiliza la imagen `ic_pan_tool_48px.svg`.

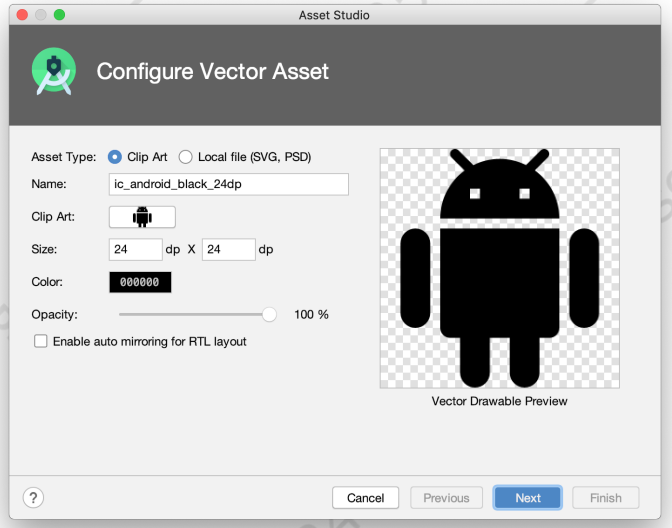


Figura 14

Como se utilizará una imagen personalizada, selecciona la opción ***Local file***. El resto de opciones son bastante intuitivas.

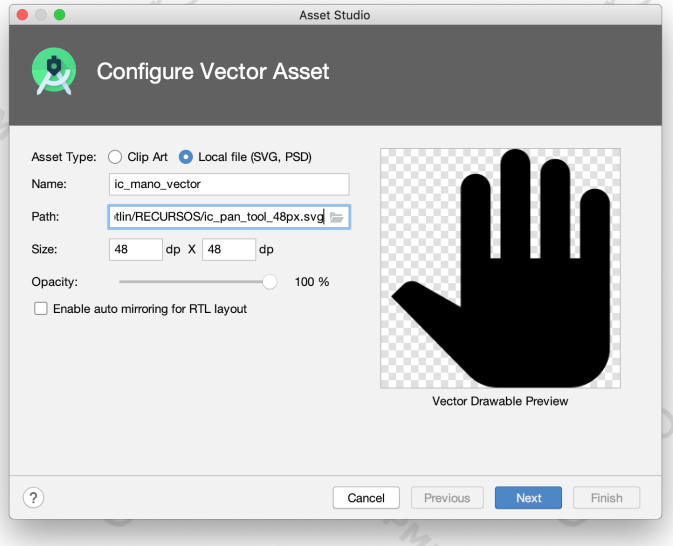


Figura 15

Al igual que en el paso anterior, se mostrará un resumen de la acción, aquí ya puedes pulsar la opción *Finish*.

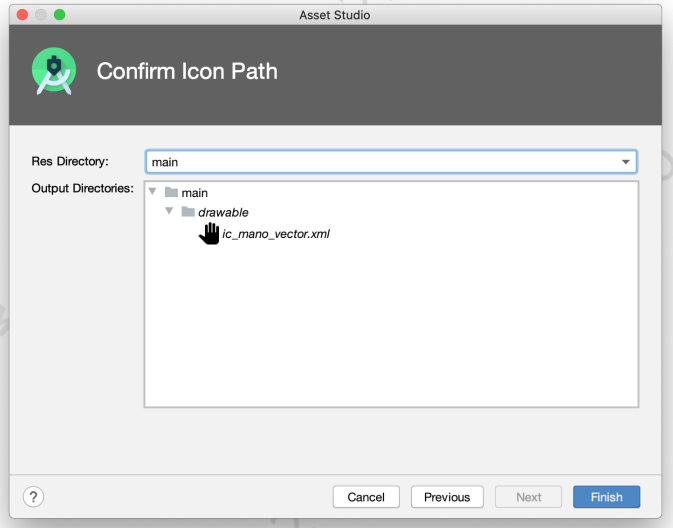


Figura 16

Podrás ver como cambia, en este caso, el directorio `res/drawable`.

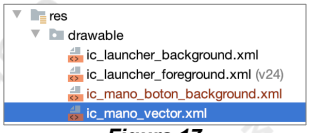


Figura 17

Ahora, ya puedes cambiar el icono utilizado en el botón como se ha hecho anteriormente. El resultado puede ser como se muestra a continuación.

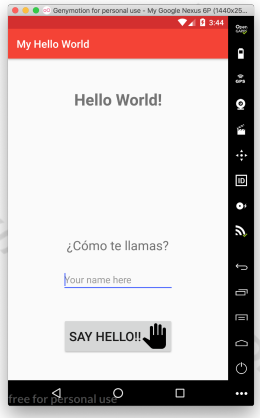


Figura 18

3.2. Layouts

Como se ha visto en el capítulo anterior, los *layouts*[3], o diseños, permiten definir la estructura visual de las *Activities*. Existen dos formas de crear los UI (*User Interface*).

- **Crear la UI con XML**. Android utiliza XML simple que coincide con las clases y subclases de vistas, como las usadas para *widgets* y diseños. Recuerda el ejemplo de *HelloWorld!!*, donde se trabaja con el fichero XML `activity_main.xml`.
- **Instanciar los elementos de diseño en tiempo de ejecución**. La aplicación puede crear objetos *View* y *ViewGroup*, así como manipular sus propiedades, desde el código fuente.

Para esta segunda opción es necesario tener muy claro el espacio que se está usando y la estructura que se busca. Para iniciarse en este mundo, se centrará la atención en la primera opción. Para esta se dispone de la herramienta *layoutopt*, o *Layout Editor*, que ya has utilizado. Esta se lanza cada vez que se abre un fichero XML de *layout*.

3 *Layout* (https://developer.android.com/guide/topics/ui/declaring-layout)

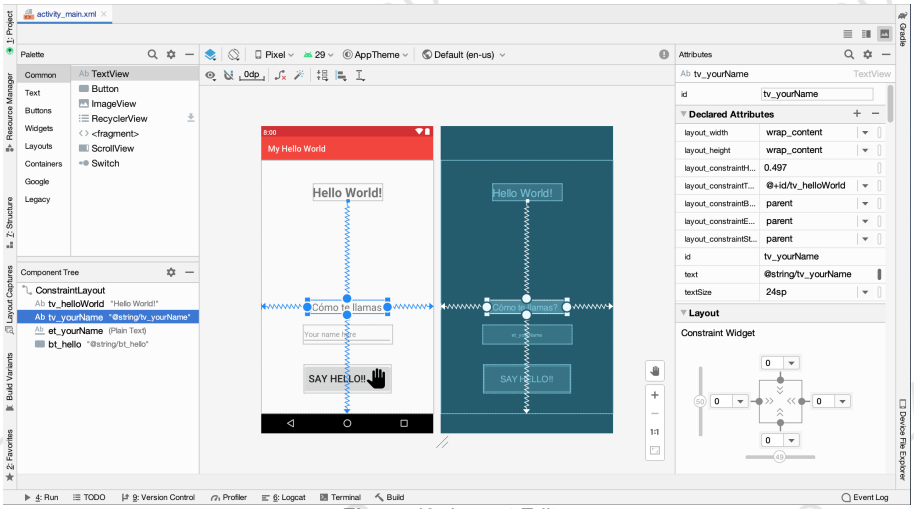


Figura 19: *Layout Editor*

La ventana se divide en 3 columnas. La columna de izquierda contiene la paleta, donde se encuentran todos los elementos que se pueden añadir a las vistas y, el árbol de componentes, que mostrará la jerarquía de los elementos añadidos. La columna de la derecha contendrá los atributos del componente seleccionado en ese momento en el árbol. La columna central muestra la parte de diseño en sí, a la que se podrá ir arrastrando los componente para ajustarlos. En esta misma columna, en la parte superior, se dispone de las opciones de visualización del área de trabajo.

Una opción que puede ser de ayuda es *Show Layout Decoration*, que permite ver todos los elementos “decorativos” que aparecerían en el dispositivo móvil, elementos del sistema operativo.

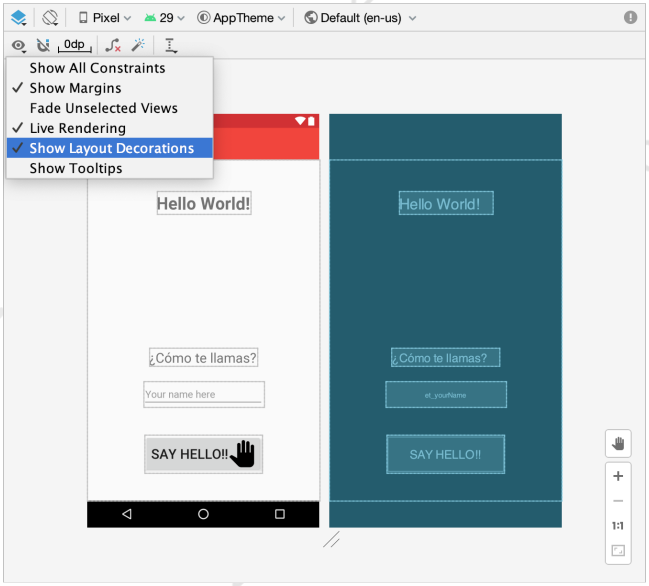


Figura 20

Para conocer más en profundidad *Layout Editor*[4] puedes visitar la documentación ofrecida por Google.

Android Studio ofrece una gestión gráfica de los recursos utilizados en el proyecto mediante ***Resource Manager***[5], que ya has visto. Esta vista permite una gestión más amigable, e incluso ofrece la posibilidad de *drag and drop* para recursos *drawables*.

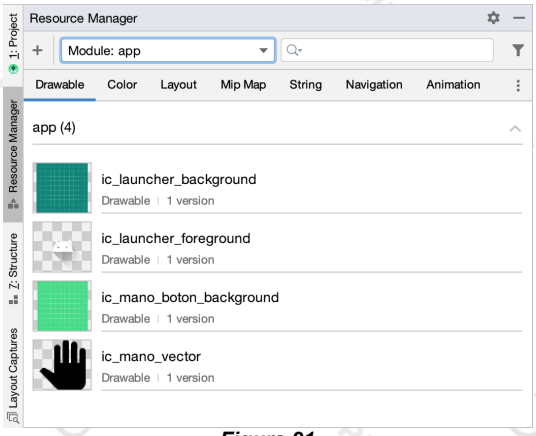


Figura 21

Dentro del diseño de UI existen elementos pueden ayudar a diseñar las vistas, estos son los ***layouts***. En función del tipo de vista que se esté diseñando, vendrá mejor un tipo u otro. Estos tipos son.

- ***ConstraintLayout***, permite disponer los elementos libremente permitiendo establecer relaciones entre todos los elementos y la propia vista padre.
- ***Guideline* (horizontal)** y ***Guideline* (vertical)**, permite crear líneas que sirven de guía para la ubicación de los elementos.
- ***LinearLayout* (horizontal)**, coloca los elementos en una línea horizontal.
- ***LinearLayout* (vertical)**, coloca los elementos en una columna.
- ***FrameLayout***, permite el cambio dinámico de los elementos que contiene.
- ***TableLayout***, distribuye los elementos en forma de tabla.
- ***TableRow***, se utiliza como hijo de *TableLayout*, se comporta de manera lineal.
- ***Space***, permite crear espacios entre elementos de una vista.

4 *Layout Editor* (https://developer.android.com/studio/write/layout-editor)
5 *Resource Manager* (https://developer.android.com/studio/write/resource-manager)

3.3. Temas y estilos

Los temas y estilos son una serie de propiedades que se aplican al diseño de la aplicación. La forma más sencilla de aplicar un tema es editar el fichero `AndroidManifest.xml` y buscar la propiedad `android:theme` de *application*. En función de como se quiera aplicar, deberá colocarse en un sitio u otro.

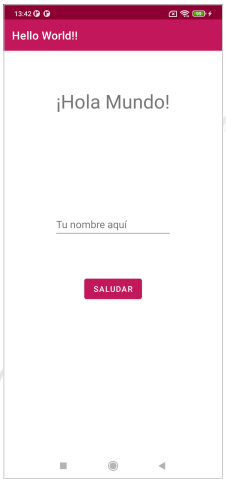


Figura 22

- Al aplicarse a la etiqueta `application`, la aplicación entera adopta el tema seleccionado.
- Si se aplica a la etiqueta `activity`, solo esa *Activity* aplicará el tema, por tanto, se pueden aplicar diferentes temas a diferentes *activities*.

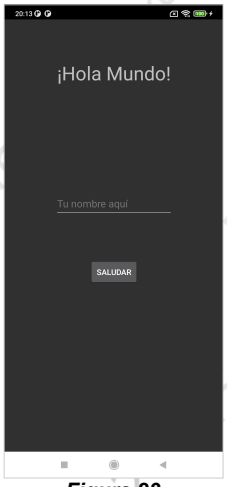


Figura 23

Por ejemplo, al crear el proyecto *HelloWorld!!*, por defecto, la propiedad `android:theme` está aplicada a la etiqueta `application`, asignando el tema a la aplicación completa. Te habrás fijado que su valor es `"@style/Theme.HelloWorld"`, este valor hace referencia al fichero `res/values/themes.xml` (incluida la versión *night*), donde se podrán crear y personalizar los estilos, además, podrá existir más de un estilo en el mismo fichero. Se debe entender que un tema, no es más que un estilo aplicado.

Si cambiases el valor establecido en la propiedad `android:theme` para la etiqueta `application` por `@style/Theme.AppCompat.NoActionBar`, al no estar definido en el fichero `themes.xml`, se aplica el tema por defecto en cuestión, cambiando la configuración de la aplicación como muestra la figura de fondo oscuro.

Si echas un vistazo al fichero `res/values/themes.xml`, verás que el tema por defecto hereda de una ya existente (*parent*). Puede variar según el sistema operativo que se esté utilizando, la versión de AS, o incluso la API mínima elegida.

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
    <style name="Theme.HelloWorld" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/primaryColor</item>
        <item name="colorPrimaryVariant">@color/primaryDarkColor</item>
        <item name="colorOnPrimary">@color/primaryTextColor</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/secondaryColor</item>
        <item name="colorSecondaryVariant">@color/secondaryDarkColor</item>
        <item name="colorOnSecondary">@color/secondaryTextColor</item>
        <!-- Status bar color. -->
```

```
        <item name="android:statusBarColor" tools:targetApi="l">
            ?attr/colorPrimaryVariant
        </item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

Se puede añadir un nuevo tema que herede de *Theme.HelloWorld* y adaptarlo según las necesidades, por ejemplo, el siguiente código irá dentro de la etiqueta *resources* del fichero `themes.xml`, lleva cuidado, si quieres que se adapte a los dos modos (claro y oscuro) deberá estar en ambos ficheros.

```
<style name="MyAppTheme" parent="Theme.HelloWorld">
    <item name="colorPrimary">#5D00FF</item>
    <item name="colorPrimaryDark">#350985</item>
    <item name="colorAccent">#ECA529</item>
    <item name="android:textColorPrimary">#212121</item>
    <item name="android:textColorSecondary">#E051F0</item>
</style>
```

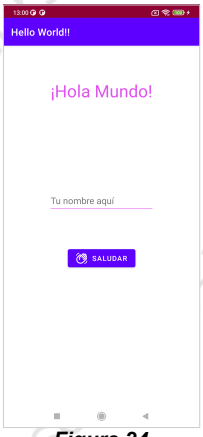


Figura 24

Ahora, si modificas el valor de la propiedad `android:theme` en la etiqueta `application` dentro del fichero `AndroidManifest.xml` con el nuevo estilo creado y lanzas nuevamente la aplicación, verás como los colores cambian aplicándose en cascada.

Se pueden hacer muchas más cosas, este es un campo muy extenso y no es posible abarcar todo lo que conlleva, pero aquí tienes algunas nociones básicas para comenzar. Por ejemplo, es posible aplicar diferentes estilos a distintos componentes de una aplicación, incluso a diferentes *activities* dentro de la misma aplicación.

A continuación, puedes ver el código para aplicar un estilo en concreto a una vista, en este caso al *TextView* que contiene la frase *"¡Hola Mundo!"* de la aplicación *HelloWorld!!*, mediante la propiedad *textAppearance*, que inicialmente se ha establecido con el estilo predeterminado *Display1*.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hola_Mundo"
    android:textAppearance="@style/TextAppearance.AppCompat.Display1"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.10" />
```

Se creará un nuevo estilo en el fichero `themes.xml` con las propiedades de diseño deseadas para el *TextView*.

```xml
<style name="MyCodeFont" parent="@style/TextAppearance.AppCompat.Display1">
    <item name="android:textColor">#E051F0</item>
    <item name="android:layout_width">wrap_content</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textStyle">bold</item>
    <item name="android:textSize">30sp</item>
</style>
```

Fíjate que se debe indicar la clase de la que hereda con la propiedad `parent`, ya que se parte del tema que ofrece Android, fíjate que es el estilo original que se puso al *widget* en la propiedad `textAppearance`. Date cuenta que únicamente se añaden *items* de tipo *android*. Ahora, en el código del *TextView* en `activity_main.xml` modifica la propiedad correspondiente para asignar el nuevo estilo.

```
...
android:textAppearance="@style/MyCodeFont"
...
```

También puedes sustituir esta propiedad por la propiedad `style`.

```
...
style="@style/MyCodeFont"
...
```

3.4. Otros recursos

En este punto se destacarán algunos de los recursos que se pueden utilizar dentro de la aplicación. Estos recursos contendrán ese contenido estático que se utilizará en el código que se vaya generando, como imágenes (que ya has visto), cadenas de texto para la interfaz, colores, dimensiones, etc.

Para poder acceder a todos estos recursos se hará uso del identificador (id) que se le asigne, al cual se podrá acceder mediante la clase `R` del proyecto.

3.4.1. Recurso Strings

El recurso *strings*[6] ya lo conoces, se encuentra en `/res/values/strings.xml`, y es el encargado de recoger todas las cadenas de texto que quieras utilizar en la aplicación.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="string_name">text_string</string>
</resources>
```

Este recurso permite eliminar los *warnings hardcoded text* producidos por insertar texto estático en el código o en los *layouts*. También se utiliza para añadir las traducciones de la aplicación, creándose un fichero *strings.xml* por cada idioma traducido.

6 Recurso *strings* (https://developer.android.com/guide/topics/resources/string-resource)

Dentro de este recurso puedes crear tres tipos de *strings*, el *string* básico que contiene una única cadena de texto (código que puedes ver arriba). Otro tipo son los *arrays* de *strings*, este recurso permite tener un *array* con diferentes cadenas accesibles por la posición desde el código.

```
<string-array name="planets_array">
    <item>Mercury</item>
    <item>Venus</item>
    <item>Earth</item>
    <item>Mars</item>
</string-array>
```

Para acceder a un *array* de *strings* desde el código se utilizará la siguiente instrucción.

```
val array: Array<String> = resources.getStringArray(R.array.planets_array)
```

Y por último, un *string* de cantidades (plurales), este tipo cadena permite añadir ajustar la concordancia de los textos.

```
<plurals name="numberOfSongsAvailable">
    <item quantity="one">%d song found.</item>
    <item quantity="other">%d songs found.</item>
</plurals>
```

En este caso, el recurso *plural* permite elegir para ajustarse a la concordancia, una canción encontrada o veinte canciones encontradas. Para hacer uso de este recurso se utilizará el siguiente código.

```
val count = getNumberOfSongsAvailable()
val songsFound = resources.getQuantityString(R.plurals.numberOfSongsAvailable, count, count)
```

La variable *count* contendrá el número de canciones devuelto, según el valor obtenido, el método *getQuantityString()* hará la selección que corresponda. Fíjate que *count* se pasa dos veces como parámetro, la primera vez es para seleccionar el plural adecuado, el segundo, será el valor a mostrar en la posición *%d*.

3.4.2. Recurso Dimensión

Este recurso permite establecer medidas para hacer uso de ellas, ya sea desde los *layouts* o desde el código. Ayuda en el diseño principalmente, permitiendo unificar medidas y hacer modificaciones rápidas. Este recurso lo encontrás en `/res/values/dimens.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="altura_edits">55dp</dimen>
</resources>
```

Para aplicarlo en un *layout* se indicará como `@dimen/altura_edits`. Para obtener un valor de dimensión desde código deberás utilizar el método `getDimension(R.dimen.altura_edits)`.