Tema 2. Diseño de aplicaciones móviles

2.1 Vistas y diseño de aplicaciones Android

2.1.1 Material Design

El material design es un estilo de diseño desarrollado por Google que se presentó al público en 2014, a la par de la versión *Android Lollipop*. Este se enfoca en los **aspectos visuales** que corresponden al sistema operativo Android, aunque también es una normativa que se aplica en diferentes páginas web y plataformas.

En otras palabras, este es *un lenguaje de diseño con el que Google entiende la interacción entre una persona y un dispositivo*. Con este estilo de diseño, Google nos presenta unas normativas o guías que nos indican cómo podemos diseñar una app o web para su sistema operativo, tanto para una versión de un dispositivo móvil como de escritorio.

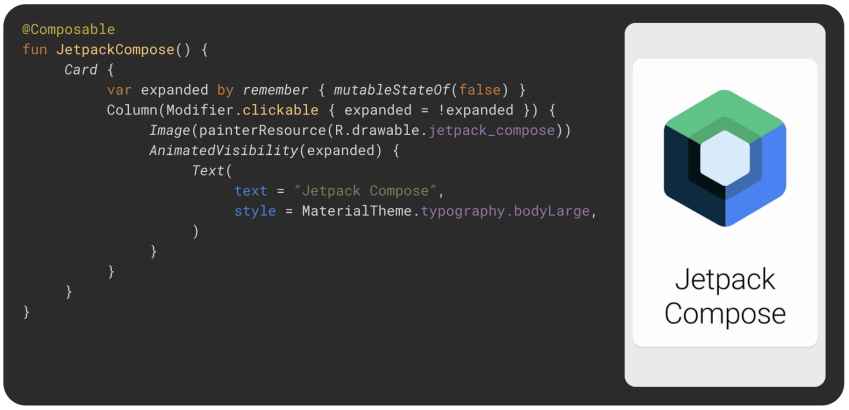
Podemos decir que este sistema ha llegado a facilitar tanto el diseño web como el diseño de apps. En su web Material.io es posible encontrar toda la información sobre este lenguaje visual; además, podemos acceder a diferentes elementos o recursos que nos sirven para ensayar diferentes diseños.

Podemos decir que **el propósito** de material design es *unificar el diseño de aplicaciones web y móviles para que todo tenga una coherencia estética y funcional*, logrando transmitir una *experiencia de usuario similar en todos los dispositivos* o plataformas manejadas o incentivadas por Google y que este sistema de diseño tenga congruencia con su identidad corporativa.

2.1.2 Jetpack Compose

Jetpack Compose es el kit de herramientas moderno de Android para compilar *IU nativas*. Simplifica y acelera el desarrollo de la IU en Android. Haz que tu app cobre vida rápidamente con menos código, herramientas potentes y APIs intuitivas de Kotlin.

Puedes empezar a trabajar con Jetpack Compose con el siguiente curso.



2.2 Interfaces de usuario: clases asociadas

La interfaz de una aplicación Android es aquello que aparece en la pantalla, que el usuario puede ver, y por tanto, interactuar con él.

Android nos ofrece un gran número de componentes implementados y preparados para su uso en la interfaz (todos heredan de la clase *View*).

La librería *android.view* nos proporciona los elementos de la interfaz para construir las vistas, que se suele hacer usando un fichero XML, ubicado en la carpeta res/layout.

La clase *View* como clase principal en la jerarquía de vistas tiene una serie de atributos que serán heredados por todos los elementos que de ella provienen.

ID: cualquier objeto de tipo View debe tener un identificador asociado que lo identifique de forma única.

```
<Button
    android:id="@+id/boton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginBottom="8dp"
    android:text="Boton1"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/boton2"
    app:layout_constraintHorizontal_bias="0.101"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.499"
    android:onClick="boton1Click"/>
```

Height (layout_height) y Width (layout_width): se utilizan para definir la altura y la anchura del objeto tanto de un layout como en una vista o View. Los valores podemos asignarlos directamente usando las dimensiones que queremos definir, aunque no suele ser la práctica habitual. Es más recomendable utilizar los valores *WRAP_CONTENT* y *MATCH_PARENT*.

- Desde la versión *API 8* el valor **MATCH_PARENT** ha sustituido al valor **FILL_PARENT**, aunque éste puede seguir usándose sin problemas.

- Con el valor **WRAP_CONTENT**, el objeto sólo cogerá el espacio necesario para representarse con respecto al objeto contenedor o padre y con el valor **MATCH_PARENT** cogerá el espacio entero del padre.

Padding (padding) y Margins (layout_margin): ambos se utilizan para crear espacio alrededor de los objetos. Esto le da una mejor apariencia a nuestra interfaz de usuario.

- Con el atributo **margin** establecemos un espacio fuera del objeto con respecto a los otros objetos de alrededor.

- Con el atributo **padding** establece un espacio dentro del objeto, por ejemplo, en el caso de tener un Button, la distancia desde el texto del botón a las líneas del mismo.

Gravity: este atributo se utiliza para alinear los objetos. Por defecto se alinean a la izquierda, si deseamos poner alguna otra alineación debemos utilizarlo.

2.3 Layouts

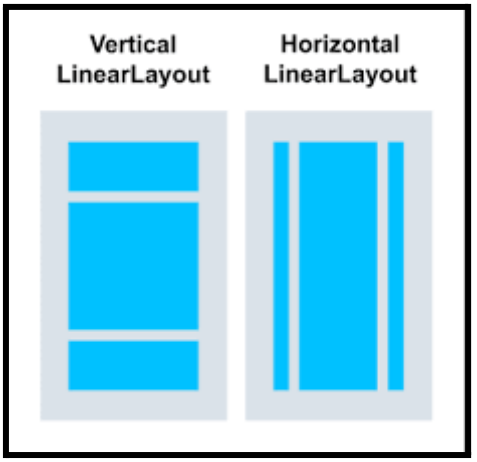
Los layouts son elementos no visuales destinados a controlar la distribución, la posición y las dimensiones de los controles que se insertan en su interior.

Dentro de cada uno pueden ubicarse todos los elementos que sean necesarios en la interfaz de la actividad, incluidos otros layouts.

Existen gran variedad de layouts, pero solo nos vamos a centrar en los más interesantes.

2.3.1 LinearLayout

Este layout apila uno tras otro todos sus elementos hijos, de forma horizontal o vertical, según se establezca su atributo android:orientation.



A continuación se muestra un ejemplo de fichero XML que define una interfaz que usa un *LinearLayout* para distribuir los diferentes elementos. Es interesante fijarse en la propiedad android:layout_weight.


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_weight="2"
        android:background="#FF0000"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:layout_weight="1"
        android:background="#00FF00"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:layout_weight="1"
        android:background="#0000FF"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</LinearLayout>
```

Notas:

- Puedes definir el color de fondo de los objetos de una vista usando la propiedad android:background.

- No olvides colocar el espacio de nombres (xmlns) como atributo en la etiqueta LinearLayout.

*** Crea una app que contenga un LinearLayout y la interfaz sea como en la siguiente imagen.*

2.3.2 FrameLayout

Es el más simple de los Layout. Actúa como un contenedor básico, donde los componentes introducidos se colocan unos encima de otros, sin posibilidad de repartirlos por el contenedor.

Su uso está recomendado para incluir un único componente que pueda aparecer o desaparecer en función de las necesidades de la app, normalmente una imagen.

Son usados especialmente para cargar un fragment (se verá en temas posteriores).

2.3.3 ConstraintLayout

Es el más recomendado actualmente, por las facilidades que incorpora a la hora de crear el diseño de nuestra aplicación y porque se adapta mejor a las resoluciones de los dispositivos.

En el siguiente enlace se encuentra la documentación sobre el ConstraintLayout en https://developer.android.com/reference/android/support/constraint/ConstraintLayout

También tenéis el siguiente enlace sobre cómo crear una interfaz de usuario responsive: https://developer.android.com/training/constraint-layout

La explicación de cómo funciona este tipo de Layout es muy compleja de explicar con texto, por lo que se recomienda visualizar el siguiente vídeo: https://www.youtube.com/watch?v=S7TwRIKqCu4