Apunts

Tema 4. Material Design. Patrons de disseny bàsics.

Descarregar aquests apunts

Índex

Introducció

Material Design es va presentar durant la Google I/O 2014, i comprén unes guies o pautes que han de seguir les aplicacions de l'ecosistema de Google, ja siguen per a Android com a Web.
Les característiques que defineixen Material Design són: disseny net, colorit (a través de paletes de colors ja definides), amb efectes bidimensionals (a través d'efectes d'il·luminació i ombra) i l'ús d'animacions.

Patrons de disseny de la interfície d'Android

Tots els sistemes operatius proposen diferents maneres d'interactuar amb els elements en pantalla. Conéixer la diferència entre ells i utilitzar elements familiars per a l'usuari, assegura que se senta còmode i segur usant l'aplicació. Les característiques més comuns són: simplicitat (comptar amb pocs elements, però sobretot, que aquells presents en la interfície tinguen una funció ben definida que contribuïsca a complir l'objectiu de l'app i ajude a l'usuari), consistència (l'usuari espera que les aplicacions es comporten de la mateixa manera) i navegació intuïtiva.

Patrons MD

ActionBar y ToolBar

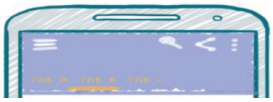
La ActionBar és un element visible en la part superior de la pantalla d'una aplicació Android. En ella es recull entre altres coses el nom de l'aplicació o de la activity en ús. A més pot incloure la icona de l'aplicació, menú overflow i botons de menú.

La ActionBar va ser llançada per Google en 2013 amb el llançament d'Android 3.0 (API 11). Totes les aplicacions que utilitzen el tema predeterminat proporcionat per Android (Theme.AppCompat.Light.DarkActionBar), contenen una ActionBar de manera predeterminada. No obstant això, els desenvolupadors poden personalitzar-ho de diverses formes segons les seues necessitats.

La ToolBar és un element de vista que s'integra en els dissenys XML d'una activity. Va ser introduït per l'equip de Google amb el llançament d'Android Lollipop (API 21). És molt més flexible i personalitzable, i a diferència e la ActionBar no està necessàriament fixada en la part superior de l'aplicació, és més podem col·locar més d'una.

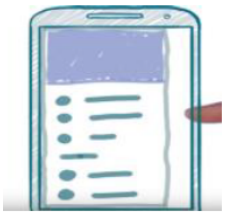
Tabs o pestanyes

Permeten la navegació entre diferents seccions de l'aplicació. S'agrupen amb AppBarLayout en la ToolBar. Es pot canviar de l'una a l'altra pestanya a través de gestos o prement sobre cadascun d'elles.



Navigation Drawer

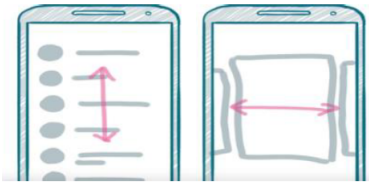
Menú que es llisca des de la part esquerra de la pantalla i conté el nivell més alt d'opcions de navegació.



Scrolling y Paging

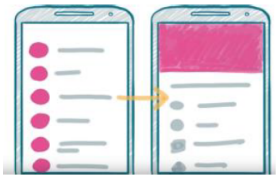
Un ScrollView permetrà visualitzar de manera còmoda la informació quan aquesta no cap totalment en la pantalla. Si la informació pot visualitzar-se de manera completa no apareix el scroll i sinó apareixerà de manera automàtica.

El ViewPager ens permetrà disposar de diverses pàgines que seran accessibles amb gestos d'esquerra i dreta o a dalt i a baix.



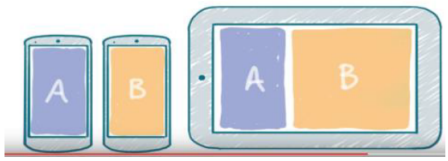
Llistes

És molt comú l'ús d'aquest patró de disseny en les aplicacions per a mòbils. Quan vam mostrar una llista d'elements i premem sobre un d'ells, apareix una segona finestra on es detalla la informació de l'element premut.



Multipanell

És una concreció del patró anterior, de manera que en dispositius que disposen de pantalles grans o depenent de si la pantalla la col·loquem horitzontal, visualitzarem dos panells o més en la pantalla, obtenint una visualització optimitzada de la vista i del seu contingut.



Paleta de colors

El sistema de color de Material Design t'ajuda a aplicar color a la teua interfície d'usuari d'una manera simplificada. Els temes de color estan dissenyats per a ser harmoniosos, garantir text accessible i distingir elements i superfícies de la interfície d'usuari fàcilment.

Com vam veure en un tema anterior, els colors es gestionen des de l'arxiu `colors.xml` i el seu contingut en generar el projecte és similar a:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
</resources>
```

Ací tenim un enllaç on es troba informació sobre la gestió de colors que ofereix Material Design

En aquest enllaç tenim una eina on provar les diferents paletes i descarregar la configuració de colors triada.

Una vegada descarregat l'arxiu cal copiar-lo a la carpeta `res/values` substituint l'arxiu de configuració de colors predeterminat pel nou.

És necessari canviar de nom en els `name` als colors per a fer-los coincidir en el arxiu `res/values/styles.xml`

Layouts

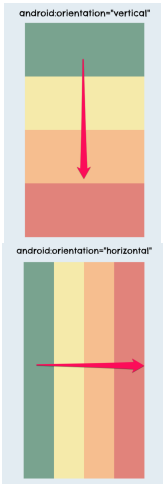
Els layouts són elements no visibles destinats a controlar la distribució, posició i dimensions dels controls que s'insereixen en el seu interior. Aquests elements estenen de la classe ViewGroup.

La definició de la interfície gràfica d'una aplicació es definirà a través d'arxius XML o a través de codi de manera programàtica.

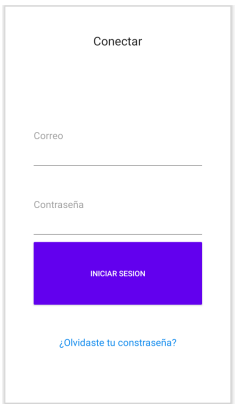
> 📌 Existeixen diferents tipus de layouts: FrameLayout, LinearLayout, RelativeLayout, GridLayout, ConstraintLayouy i CardView.

LinearLayout

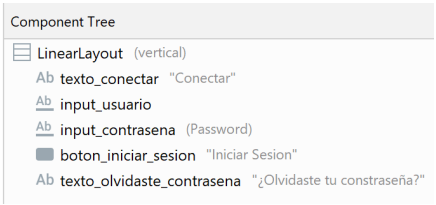
Un `LinearLayout` és un view group que distribueix els seus fills en una sola dimensió establida. És a dir, o tots organitzats en una sola columna (vertical) o en una sola fila (horitzontal). L'orientació pots triar-la a través de l'atribut `android:orientation`.



Vegem un exemple, dissenyarem la vista que apareix en la següent imatge:



Per a això creem un nou projecte, que cridarem **EjemploLinearLayout**. Els components que col·locarem són els següents:



L'arxiu XML corresponent al disseny seria:

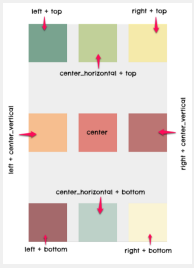
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="48dp">

    <TextView
        android:id="@+id/texto_conectar"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:text="Conectar"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/input_usuario"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:hint="Correo" />

    <EditText
        android:id="@+id/input_contrasena"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:ems="10"
        android:hint="Contraseña"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/boton_iniciar_sesion"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:text="Iniciar Sesion" />

    <TextView
        android:id="@+id/texto_olvidaste_contrasena"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center_horizontal"
        android:gravity="center_vertical"
        android:text="¿Olvidaste tu contraseña?"
```

```
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#0E8AEE" />
</LinearLayout>
```

📌 Aclariments:

- **Línies 3 i 4**: S'utilitzen les propietats `android:layout_width` i `android:layout_height` per a determinar les dimensions del layout o de qualsevol element de vista. Els valors que poden prendre són `match_parent` i `wrap_content` , és a dir s'ajusta a la grandària del contenidor ocupant tot el seu espai, o s'ajusta al contingut o grandària de la vista respectivament. També se li pot assignar un valor concret en unitats `dps`
- **Línia 5** amb la propietat `android:orientation` especifiquem com es distribuiran els elements dins del contenidor, horitzontalment o verticalment.
- **Línies 10-12** Per a crear un disseny lineal en el qual cada camp secundari use la mateixa quantitat d'espai en la pantalla, defineix el `android:layout_height` de cada vista en `0dp` (per a un disseny vertical) o el `android:layout_width` de cada vista en `0dp` (per a un disseny horitzontal). Després, fixa el `android:layout_weight` de cada vista en 1. Si dones valors diferents a 1 la grandària de layout es distribuirà seguint aquesta distribució.

- **Línia 13** amb la propietat `android:layout_gravity` alineem cada vista dins del layout.



- **Línia 23** amb la propietat `android:hint` visualitza un text en el `EditText` i quan punxem en ell desapareix.
- **Línia 31** amb la propietat `android:ems` s'usa per a determinar la grandària d'un caràcter segons la quantitat de punts que use la font del text.
- **Línia 33** amb la propietat `android:inputType` s'usa per a determinar el tipus de caràcters que es permeten com a entrada en el `EditText` . Aquest atribut determina també el tipus de teclat virtual que li apareixerà a l'usuari per a realitzar l'entrada de caràcters.
- **Línia 50** amb la propietat `android:gravity` s'usa per a determinar el tipus d'alineament del text dins d'una vista. Els valors són idèntics als de la propietat

```
      android:layout_gravity
```

CardView

Ací tenim un enllaç on es troba informació oferida sobre aquest element en Material Design.

Aquest nou component anomenat `CardView` és la implementació que ens proporciona Google de l'element visual en forma de targetes d'informació que tant utilitza en moltes de les seues aplicacions, entre elles Google Now.

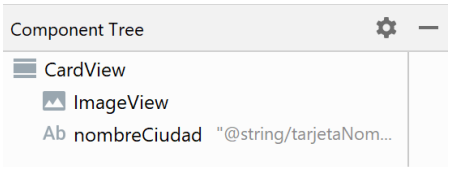
Està inclòs en la llibreria **material**, per la qual cosa cal agregar aquesta dependència al nostre projecte:

```
dependencies {
     implementation 'com.google.android.material:material:1.2.1'
}
```

Vegem un exemple, dissenyarem el cardView que apareix en la següent imatge:

Per a això creem un nou projecte, que cridarem **EjemploCardView**.

Els components que col·locarem són els següents:



L'arxiu XML corresponent al disseny seria:

```xml
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    card_view:cardCornerRadius="8dp"
    card_view:cardElevation="12dp">

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:src="@drawable/sanfrancisco"
        android:scaleType="centerCrop"
        android:contentDescription="@string/todo" />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/nombreCiudad"
        android:padding="10dp"
        android:text="@string/tarjetaNombreCiudad"
        android:layout_gravity="bottom"
        android:background="#8c000000"
        android:textColor="@color/white"
        android:textSize="30sp"
        android:textStyle="bold"/>
</com.google.android.material.card.MaterialCardView>
```

📌 Usa aquestes propietats per a personalitzar l'aparença del widget **`CardView`** :

- Per a definir el radi de la cantonada dels teus dissenys, usa l'atribut **`card_view:cardCornerRadius`** .
- Per a definir el color de fons d'una targeta, usa l'atribut **`card_view:cardBackgroundColor`** .
- Per a efecte d'elevació amb ombra usa l'atribut **`card_view:cardElevation`** .

✋ EjercicioPropuestoCardView

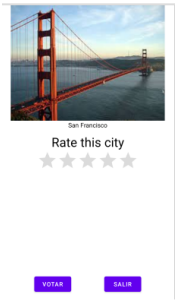
Constraint Layout

Per a poder comprendre com funciona aquest layout, primer hem de conéixer com s'integren les vistes o components en els layouts. Vegem la següent imatge:

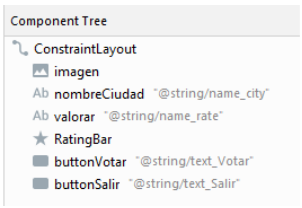
Cada lateral d'un component ve referenciat per un nom específic que permet fer referència a aquest. `Top` és la part superior, `Bottoom` és la part inferior, `Start` és la part esquerra i `End` és la part dreta.

Per a poder afegir components hem de restringir la seua posició definint uns ancoratges basats en els laterals de cada component. Són necessaris almenys dos ancoratges.

Vegem un exemple pràctic. Anem a disenyar la següent interfície:



Creem un projecte amb el nom **ejemploConstraintLayout** i integrarem els següents components:



Com veiem inserim en el contenidor una `ImageView`, dues `TextView` una `RatingBar` i dues `Button`. No us preocupeu perquè més endavant explicarem amb més detall aquests components, ara només volem utilitzar-los per a explicar `ConstraintLayout`.

L'arxiu XML corresponent és el següent:

```
<?xml version="1.0" encoding="utf-8"? >
<androidx.constraintlayout.widget.ConstraintLayout
    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"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imagen"
        android:layout_width="match_parent"
        android:layout_height="720dp"
        android:layout_marginTop="0dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/sanfrancisco" />

    <TextView
        android:id="@+id/nombreCiudad"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/imagen"
        android:text="@string/name_city"
        android:textAlignment="center"
        android:textSize="15dp"
        android:textColor="@color/black"/>

    <TextView
        android:id="@+id/valorar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/nombreCiudad"
        android:text="@string/name_rate"
        android:textAlignment="center"
        android:textSize="30dp"
        android:textColor="@color/black"
        android:layout_marginTop="10dp"/>

    <RatingBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/valorar"
        app:layout_constraintEnd_toEndOf="parent"
        android:numStars="5"/>

    <Button
        android:id="@+id/buttonVotar"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/text_Votar"
54          app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
56          app:layout_constraintEnd_toStartOf="@id/buttonSalir"
            android:layout_marginBottom="45dp"/>

        <Button
            android:id="@+id/buttonSalir"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
64          android:text="@string/text_Salir"
            app:layout_constraintBottom_toBottomOf="parent"
66          app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toEndOf="@id/buttonVotar"
            android:layout_marginBottom="45dp"/>
    </androidx.constraintlayout.widget.ConstraintLayout>
```

📌 Aclariments:

- **ImageView** línies 10..16:
  Línia 10 amb la propietat **`android:id="@+id/…"`** donem un nom a la vista per a posteriorment poder fer referència a ella.
  Línia 11 amb la propietat **`android:width`** indiquem l'ample d'aquesta view, el valor **`match_parent`** ajusta l'ample a l'ample disponible en el contenidor que rep la view, el valor **`wrap_content`** ajusta l'ample a la grandària de la imatge.
  Línia 12 amb la propietat **`android:height`** es defineix l'altura de la view, la definim en aquest cas indicant l'altura en dp perquè puguen reescalar-se correctament en diferents dispositius.
  Línia 13 la propietat **`android:marginTop`** permet definir el marge superior respecte al contenidor principal, com volem que estiga el més amunt possible, li llevem el marge superior.
  Ara ve la part que més ens interessa, els ancoratges de la nostra imatge.
  Línia 14 amb la propietat **`constraintStart_toStartOf:`** especifiquem que el Start del nostre component estarà ancorat al començament del seu pare, amb això aconseguim mantindre-ho a l'esquerra.
  Línia 15 amb la propietat **`constraintTop_toTopOf:`** especifiquem que la part superior del nostre component estarà ancorada a la part superior del pare.
  Línia 16 finalment, necessitem carregar la imatge en la view, utilitzem per a això la propietat **`app:srcCompat:`** .
- **TextView** línies 22..27
  En la línia 22 i 23 es defineixen els ancoratges, com es pot apreciar, el text està d'esquerra a dreta així doncs, ancorarem el Start al Start del pare, igual que es va fer en

la imatge. La part superior del text, haurem d'ancorar-la a la part inferior de la imatge perquè ens quede just davall.
En la propietat `android:text:` referenciem un recurs string on tenim el text que volem que aparega en el **TextView**.
Amb `android:textAlignment:` centrem el text independentment del gran o xicoteta que siga la pantalla del terminal.

- El següent **TextView** ha d'anar ancorat a la seua esquerra amb el pare i estar per davall al text que acabem de col·locar prèviament, línies 33 i 34.
  L'única propietat que no hem vist anteriorment és `android:marginTop` línia 39 i s'utilitza per a deixar un marge superior respecte a altres components, en aquest cas, 10dp.
- El següent component és una **RatingBar** ha d'anar ancorat a la seua esquerra amb el pare i estar per davall al **TextView** que té com aneu **valorar** línies 44..46.
  Amb la propietat `android:numStars:` en la línia 47 triem el nombre d'estreles que desitgem que apareguen en la **RatingBar**.
  ✋ Amb la propietat `android:stepSize: 0.5` ens permetria seleccionar mitja estrela(provar vosaltres)
- Vegem el codi per a incorporar els dos **Button** a la vista.
  Buscarem que tots dos botons estiguen a la mateixa altura i ho aconseguirem donant un marge inferior. A més, els col·locarem ancorats entre ells quant a l'eix horitzontal es refereix i ancorats alhora al pare, en aquest cas cadascun en el lateral que corresponga i respecte a la part inferior per a poder ajustar-lo més còmodament.

Temes i estils

El disseny visual d'una aplicació Android és representat a través de regles contingudes en Estils i Temes. Aquestes eines permeten que els programadors i dissenyadors generen una interfície més amigable i personalitzada de les seues apps, per a establir una identitat que impacte a l'usuari final.

La forma més senzilla d'aplicar un tema és editar el fitxer `AndroidManifest.xml` i buscar la propietat `android:theme` i assignar un estil concret. Si aquesta propietat s'aplica a l'etiqueta `application` l'aplicació sencera adopta el tema seleccionat. Si s'aplica a l'etiqueta `activity` únicament aqueixa activitat aplicarà el tema. D'això es dedueix que es poden aplicar diferents temes a cada activitat.

Ací tenim un enllaç on es troba informació oferida sobre temes i estils.

Utilitzarem el projecte **EjemploLinearLayout** per a treballar amb estils i temes.

Editarem el nostre arxiu `themes.xml`:

```xml
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.EjemploLinearLayout"
           parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">
            ?attr/colorPrimaryVariant
        </item>
        <!-- Customize your theme here. -->
    </style>

    <style name="Theme.EjemploLinearLayoutPropio"
           parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">#F44336</item>
        <item name="colorPrimaryVariant">#D32F2F</item>
        <item name="colorOnPrimary">@color/black</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">#F06292</item>
        <item name="colorSecondaryVariant">#F48FB1</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">
            ?attr/colorPrimaryVariant
        </item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

📌 **Línies 17 a 29** definim un nou tema modificant els colors i donant-li el nom `Theme.EjemploLinearLayoutPropio` i que hereta amb l'atribut `parent` del tema per defecte.

Cal anar ara a l'arxiu `AndroidManifest.xml` i modificar-ho, assignant aquest nou tema a l'aplicació (línia 11):

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ejemplolinearlayout">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.EjemploLinearLayoutPropio">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Encara que es defineixen tots dos amb l'etiqueta `styles` cal tindre en compte que són dues coses diferents. Els temes s'apliquen a una jerarquia de vistes, mentres que els estils es defineixen sobre una vista concreta.

Podríem canviar l'aparença dels nostres `EditText` afegint un estil creat per nosaltres. Per a això creem un arxiu anomenat `styles` en la carpeta `values`:

```xml
<resources>
    <style name="textViewStyle">
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#FFAB91</item>
    </style>
</resources>
```

Aquest estil pot ser assignat a tots nostres `TextViews` o solament als quals nosaltres vulguem. Li ho assignarem al que té identificador `input_usuario` en l'arxiu xml de l'activitat principal.

```xml
<EditText
        android:id="@+id/input_usuario"
        style="@style/textViewStyle"
        android:hint="Correo"/>
```

Afegir ToolBar a la nostra aplicació

`Toolbar` és el component que usarem com a reemplaçament del `Action Bar`. Aquest component ens permetrà:

- Reemplaçar el `ActionBar` per una vista personalitzada, en la qual podrem afegir imatges, textos, botons, i associar esdeveniments, com ho fem amb qualsevol altre layout.
- Canviar la ubicació del típic `ActionBar`. Per exemple, podem usar el `Toolbar` en la part inferior.

És usual que el nostre projecte faça ús del `ActionBar` per defecte. Com la nostra intenció és usar el `Toolbar`, llavors hem de deshabilitar el `ActionBar`.

Creem un nou projecte que anomenarem **EjemploToolBar**. Anem al nostre arxiu **themes.xml** i assegurar-nos d'assignar les següents propietats:

```
    <item name="windowNoTitle">true</item>
    <item name="windowActionBar">false</item>
```

Utilitzarem per als exemples un `CoordinatorLayout`, bàsicament ens permet gestionar interaccions entre els seus elements fills.

Dins d'aquest element inclourem la `ToolBar`. Podem veure en aquesta imatge els components utilitzats i com quedarà la vista:

Vegem com queda el xml de la activity_main:

```
    <?xml version="1.0" encoding="utf-8"?>
    <androidx.coordinatorlayout.widget.CoordinatorLayout
        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"
        tools:context=".MainActivity">

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/topAppBar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:title="@string/app_name"
            style="@style/Widget.MaterialComponents.Toolbar.Primary" />
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

En la línia 14 especifiquem l'estil de nostra `ToolBar`

Ací tenim un enllaç on es troba informació oferida sobre aquest element en Material Design.

Aplicar Scrolling a la ToolBar

En moltes aplicacions quan fem un scroll sobre la vista principal interessa ocultar la ToolBar, a aquest efecte se'n diu **scrolling**.

> 📌 Per a exemplificar aquest efecte seguim en el projecte **EjemploToolBar**, copiem l'arxiu **activity_main.xml** i ho canviem de nom com **activity_main_scrolling.xml** (a partir d'ara anirem creant arxius xml nous per a cadascun dels efectes que desenvolupem sobre la `ToolBar`)

Vegem com quedaria aquest arxiu i posteriorment pasarem a les explicacions:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    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"
    tools:context=".MainActivity"
    android:fitsSystemWindows = "true">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/app_bar_layout"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:gravity="center"
        android:theme="@style/Widget.MaterialComponents.Toolbar.Primary">

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/topAppBar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:title="@string/app_name"
            style="@style/Widget.MaterialComponents.Toolbar.Primary"
            app:layout_scrollFlags="scroll|enterAlways"/>
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <TextView
            android:text="@string/hello_world"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            android:textSize="20dp"
            android:fitsSystemWindows = "true" />
    </androidx.core.widget.NestedScrollView>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Per a poder aplicar aquest efecte hem de tindre en compte els següent:

- **Línia2:** El `CoordinatorLayout` ha d'embolicar directament als objectes que desitges relacionar mitjançant una acció.
- **Línia8:** `android:fitsSystemWindows` aquest atribut a `true` permet ajustar la finestra de la activity per davall de la ToolBar.

- **Línia 10:** `AppBarLayout` sempre ha d'embolicar a la `Toolbar` i els altres components pertanyents a ella (com a pestanyes, imatges, etc).
- **Línia 23:** La forma en què s'afecten els fills de `AppBar` es determina en `app:layout_scrollFlags`.
- **Línia 26 i 29:** Fer desaparéixer la `AppBar` per scrolling requereix que hi haja un objecte amb scroll marcat amb l'atribut `app:layout_behavior="@string/appbar_scrolling_view_behavior"`. Aquest ha de declarar-se per davall de la `AppBar`. En el nostre exemple serà un `NestedScrollView` que a més inclou un `TextView`

> ✋Els valors que pot prendre `app:layout_scrollFlags` i que controlen el tipus de comportament amb què desapareix la `Toolbar` són:
>
> - `scroll` : indica que un view desapareixerà en desplaçar el contingut.
> - `enterAlways` :torna visible al view davant qualsevol signe de scrolling.
> - `enterAlwaysCollapsed` : torna visible el view només si es manté el scroll en la part superior del contingut.
> - `exitUntilCollapsed` : desapareix el *view fins que les seues dimensions superen l'altura mínima.

Aplicar Collapsing a la ToolBar

Per a controlar les reaccions d'expansió i contracció dels elements de vista que es troben dins d'un `AppBarLayout` (una imatge, pestanyes o qualsevol altre element) és necessari utilitzar un layout especial que embolique la `ToolBar`, aquest layout es diu `CollapsingToolbarLayout`.

Afegim aquest efecte a nostra `ToolBar`: (activity_maincollapsing.xml):

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity"
    android:fitsSystemWindows = "true">

    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/app_bar_layout"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:gravity="center"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <com.google.android.material.appbar.CollapsingToolbarLayout
            android:id="@+id/collapsing_toolbar"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed">

            <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:scaleType="centerCrop"
                app:srcCompat="@drawable/image"
                app:layout_collapseMode="parallax" />

            <com.google.android.material.appbar.MaterialToolbar
                android:id="@+id/topAppBar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:title="@string/app_name" />
        </com.google.android.material.appbar.CollapsingToolbarLayout>>
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.core.widget.NestedScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

        <TextView
            android:text="@string/hello_world"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textStyle="bold"
            android:textSize="22dp" />
    </androidx.core.widget.NestedScrollView>
```

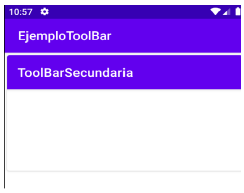
```
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

En el nostre exemple afegim una imatge a la `AppBarLayout` a més de la `ToolBar` de manera que aplicarem un efecte de collapsing a la imatge fins que desaparega i només es veja la `ToolBar`.

> 📌 Aclariments:
>
> - **Línia 17** amb l'element `CollapsingToolbarLayout` se'ns permetrà realitzar aquest efecte . En la **línia 22** s'especifica la propietat `app:layout_collapsedMode="scroll|exitUntilCollapsed"` , el primer flag indica que totes les vistes que faran un offScreen es mantindran en la part superior de la pantalla. I el segon flag especifica que la `AppBar` es contraga fins que arribe a la grandària de la `ToolBar`
> - **Línia 24** s'inclou la imatge que farà l'efecte de collapsing. En la **línia 27** amb la propietat `app:layout_collapsedMode="parallax"` donem l'efecte d'animació que volem.

Afegir ToolBar a altres parts de la interfície

Un altre avantatge que podem aprofitar en disposar del control `ToolBar` com un component independent, és que podem utilitzar-los en altres llocs de la nostra interfície, i no sempre com a barra d'accions en la part superior, i evidentment, tindre més d'una `ToolBar`.
Podríem col·locar el component `ToolBar` dins d'una targeta, tal com es veu en la següent imatge:



Afegim aquest efecte a nostra `ToolBar`: (activity_toolbarcardview.xml):

```
    <androidx.core.widget.NestedScrollView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical" >

                <androidx.cardview.widget.CardView
                    xmlns:app="http://schemas.android.com/apk/res-auto"
                    android:id="@+id/card_view"
                    android:layout_gravity="center"
                    android:layout_width="match_parent"
                    android:layout_height="200dp"
                    app:cardUseCompatPadding="true"
                    app:cardCornerRadius="4dp">

20                  <com.google.android.material.appbar.MaterialToolbar
                        android:id="@+id/toolbarCard"
                        android:layout_width="match_parent"
                        android:layout_height="?attr/actionBarSize"
                        app:title="@string/app_name_toolbar"
25                      style="@style/Widget.MaterialComponents.Toolbar.Primary"/>
                </androidx.cardview.widget.CardView>
            </LinearLayout>
        </androidx.core.widget.NestedScrollView>
```

📌 Únicamente hem modificat el **`androidx.core.giny.NestedScrollView`** afegint els elements que es veuen. La resta del disseny és idèntic a l'anterior.