

Resum per a l'ús propi de Juanjo

Índex

1. Temes i estils
2. Material Design
3. Components i Material Design 2

Tema 4. Material Design. Patrons de disseny bàsics.

Temes i estils

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.

Els temes estan definits a `res\values\themes\themes.xml`

Podríem canviar l'aparença dels nostres `TextView` afegint un estil creat per nosaltres. Per a això afegim al arxiu anomenat `styles` en la carpeta `values` amb el següent contingut:

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

 **Tip:** Un altra possibilitat es donar el estil al component i després **Boto dret -> Refactorizar** i ens oferirà generar el estil al mateix fitxer que hem indicat.

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.

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

Si desitges aplicar aquest estil a tots els edit, pots declarar-lo en el ítem `textViewStyle` del teu tema a `values\themes\themes.xml` de la següent manera:

```
<style name="Theme.MiTema" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
  <item name="textViewStyle">@style/miTextViewStyle</item>
  ...
</style>
```

Material Design


Defineix diferents components basics per a tindre un 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. També en dona unes normes d'ús dels diferents components per homogeneïtzar les interfícies de les aplicacions.

Components i Material Design 2

- **La AppBar:** és un element de vista que s'integra en els dissenys XML d'una activity i pot estar amunt o baix del quadre que defineix la nostra activity.

 **Nota:** Ha tingut altres noms i funcionalitat al llarg de las diferents versions del API.

- **Tabs o pestanyes:** Permeten la navegació entre diferents seccions de l'aplicació. S'agrupen amb `AppBarLayout` en la AppBar. Es pot canviar de l'una a l'altra pestanya a través de gestos o prement sobre cadascun d'elles.

 **Avis:** Hem d'anar amb compte que el geste de lliscar no este present en altres components dins de la pestanya.

- **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. D'altra banda, 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.
 - **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.

- **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.

Es gestionen des de l'arxiu de recursos `colors.xml`

🔑 [Configurador on-line de paletes bàsiques.](#)

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

```
<resources>
  <!-- Colors primaris -->
  <color name="primaryColor">#3f51b5</color>
  <color name="primaryLightColor">#757de8</color>
  <color name="primaryDarkColor">#002984</color>
  <!-- Colors secundaris -->
  <color name="secondaryColor">#b2dfdb</color>
  <color name="secondaryLightColor">#e5ffff</color>
  <color name="secondaryDarkColor">#82ada9</color>
  <!-- Colors para el text o la font -->
  <color name="primaryTextColor">#ffffff</color>
  <color name="secondaryTextColor">#000000</color>
</resources>
```

Ens generarà ...

1. **Colors primaris:** El més comú i el que tindran els components principals.
2. **Colors secundaris:** Es opcional i s'hauria d'usar sos per accentuar element de selecció a la app com:
 - Controls de selecció, lliscadors e interruptors
 - Text seleccionat destacat.
 - Barres de progress
 - Enllaços

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

- **Layouts:** Tenim diferents tipus però és molt important veure quí està dins de quí i per això deuríem tindre en compte el **Component Tree** i veure la jerarquia del components.

1. **LinearLayout:** Components u al costat d'altre amb orientació horitzontal o vertical.

Podem destacar les propietats:

- `android:layout_width` i `android:layout_height` valor fixes amb `dps` o millor `match_parent` (s'ajusta al pare) o `wrap_content` (s'ajusta al contingut)
- `android:layout_weight` **fracció de pes** dins del layout del contingut i direcció establerta.

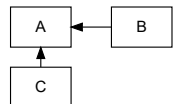
🔑 **Nota:** Si dividim imaginàriament el continent en **9** parts y el contingut té `android:layout_weight = 3` vol dir que el ocupa **3/9** del continent. Així doncs, la suma de tots els `layout_weight` hauria de ser **9**.

Si no hem assignat `layout_weight` el valor serà el establert en `layout_height` i `layout_width` en `dp` o `em`.

- `android:layout_gravity` valor a aplicar al **contingut** amb 9 posicions d'alineació dins del continent.
- `android:gravity` s'usa per a determinar el tipus d'alineament del **text** dins d'una continent.

2. **ConstraintLayout:** Ens permet definir posicions relatives de forma flexible a més de facilitar efectes d'animació. La idea és afegir components i restringir la seua posició definint uns ancoratges basats en els laterals de cada component.

- En la figura, **B** està restringit per a estar sempre a la dreta de **A**, i **C** està restringit davall de **A**. No obstant això, aquestes restriccions no impliquen alineació, per la qual cosa **B** encara pot moure's cap amunt i cap avall.



- Podem ancorar-nos a la **línia base d'un text que estiga dins d'una vista** amb la línia base de text dins d'una altra vista.

🔑 **Nota:** Per a crear una restricció de línia base, faça clic amb el **botó dret** en la vista de text que desitja restringir i després faça clic a **Mostrar línia base**. Després faça clic en la línia de base del text i arrossegament la línia a una altra línia de base.

- Podem afegir **guies horitzontals i verticals** dins del `ConstraintLayout` per ancorar o alinear els nostres components.
- Podem afegir **barreres horitzontals i verticals** dins del `ConstraintLayout` per subdividir els components que mai travessaran la barrera per dalt (horitzontal) o a l'esquerra (vertical). Per això, els components deuen estar subordinat a su barrera dins del '*Component Tree*' i la barrera s'ajustarà automàticament damunt del component més a dalt (horitzontal) o a l'esquerra del component més a l'esquerra (vertical)
- Podem seleccionar varis components al '*Component Tree*' i definir una cadena horitzontal o vertical i fer distribucions:
 1. Escapats proporcionalment.
 2. Junts o empaquetats.
 3. Estesos al exterior (Com escapats però el dels costat estan pegats al contenidor)
 4. Ocupant tota la amplària però ponderats am un pes. Per això, deurem per exemple en una cadena horitzontal definir el `layout-with = 0dp` del component i donar-li el pes mitjançant `layout_constraintHorizontal_weight = 1` (anàlogament en vertical).

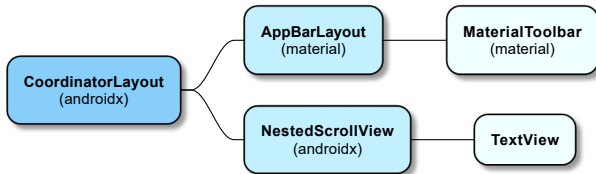
3. **MotionLayout:** <https://codelabs.developers.google.com/codelabs/motion-layout#0>

- **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.

Per ha fer-ho, anem al nostre arxiu **themes.xml** i assegurar-nos d'assignar les següents propietats:

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

Usant codi), utilitzarem per als exemples un **CoordinatorLayout**, bàsicament ens permet gestionar interaccions entre els seus elements fills. Dins d'aquest element inclourem la **MaterialToolbar** hauríem d'estirar d'un tirador una volta afegida.



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

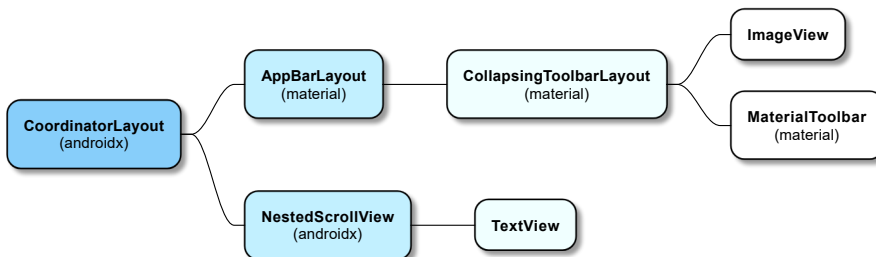
- El **CoordinatorLayout** ha d'embolicar directament als objectes que desitges relacionar mitjançant una acció per exemple lliscar.
- **android:fitsSystemWindows** aquest atribut a **true** permet ajustar la finestra de la activity per davall de la Toolbar.
- **AppBarLayout** sempre ha d'embolicar a la **MaterialToolbar** i els altres components pertanyents a ella (com a pestanyes, imatges, etc).
- Dins del **AppBarLayout** inclourem la **MaterialToolbar** (de material design) a lo millor hem d'estirar d'un tirador una volta afegida.
- Baix de **AppBarLayout** afegim un **NestedScrollView**

La forma en què s'afecten els fills de **AppBarLayout** es determina en **app:layout_scrollFlags="scroll|enterAlways"**.

Fer desaparèixer la **AppBarLayout** per scrolling requereix que hi haja un objecte amb scroll marcat amb l'atribut

app:layout_behavior="@string/appbar_scrolling_view_behavior".

El component **CollapsingToolbarLayout** ens permetrà controlar les reaccions d'expansió i contracció dels elements de vista que es troben dins d'un **AppBarLayout**



- Al **CollapsingToolbarLayout** afegirem 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 contraiga fins que arribi a la grandària de la **Toolbar**.
- Si incloem una imatge coma al exemple, per afegir un efecte de collapsing. En la propietat **app:layout_collapsedMode="parallax"** donem l'efecte d'animació que volem.

Nota: Un altre avantatge que podem aprofitar en disposar del control **MaterialToolbar** 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. Per exemple, dins d'un **CardView** com a barra de títol.