Diseño Android: Toolbar, pestañas y ViewPager2 con AndroidX y Material Components

07/12/2020

Uno de los diseños de interfaz más difundidos en Android son las pestañas (tabs) integradas en la barra de acciones. Podemos encontrar una infinidad de ejemplos de su implementación, y yo mismo suelo utilizarlo en mis apps, como en Pantanos de España

Al principio los widgets necesarios para implementar este patrón de diseño estaban disponibles en la hoy en día obsoleta Librería de compatibilidad. En este tutorial vamos a utilizar las versiones equivalentes disponibles en la librería Material Components for Android con el objetivo de crear una interfaz de pestañas estilo Material Design con una Toolbar\ActionBar que se oculte automáticamente al hacerse scroll en el contenido de la pantalla -fragments deslizables gestionados por un paginador de tipo ViewPager2-, por ejemplo desplazando un RecyclerView.

Veamos cómo construir nuestra interfaz de pestañas paso a paso pues son varios los componentes gráficos que hay que integrar y configurar. Nuestro proyecto de ejemplo, para Android Api 30 y compatible hasta Android 5.0, tendrá una única dependencia.

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

La aplicación consta de una Activity. Su layout se estructura en torno a dos grandes elementos:

- Un AppBarLayout que contiene tanto la Toolbarque hará de ActionBar de la app como el componente TabLayout responsable de mostrar las pestañas. AppBarLayout es un tipo de LinearLayout utilizado para dar un aspecto y comportamiento unificado a los widgets empleados para diseñar una AppBar.
- El *ViewPager* mostrará en pantalla la interfaz gráfica correspondiente a la pestaña seleccionada gracias a la integración que ofrece con TabLayout. También permite cambiar de página simplemente deslizándolas, de tal modo que estos cambios se van reflejando en el TabLayout.

En la actualidad hay dos componentes "paginadores" disponibles en las librerías de AndroidX: el ViewPager"de toda la vida", y una nueva versión denominada ViewPager2 introducida en 2019 y que actualmente es la que Google recomienda utilizar. ViewPager2 está implementada internamente con RecyclerView lo que le permite ofrecer numerosas mejoras frente a su predecesora, como la posibilidad de utilizar una orientación en vertical, animaciones en los cambios de páginas y el uso de "features" propias de RecyclerView tales como ItemDecoratoro DiffUtil, entre otros.

Con una configuración mínima, el layout completo queda tal que así.

/res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar"

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            />

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/Widget.MaterialComponents.TabLayout.PrimarySurface"
            />
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        />

</LinearLayout>
```

En nuestro ejemplo vamos a tener dos Fragments.

- TabNameFragment: simplemente mostrará en el centro de la pantalla el nombre de la pestaña seleccionada y que le pasaremos como un parámetro al típico "constructor estático" de los fragments.

```java
package com.danielme.android.tabs.fragments;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.annotation.StringRes;
import androidx.fragment.app.Fragment;

import com.danielme.tabs.R;

public class TabNameFragment extends Fragment {

  private static final String ARG_TAB_NAME = "ARG_TAB_NAME";

  public static TabNameFragment newInstance(@StringRes int tabName) {
    TabNameFragment frg = new TabNameFragment();

    Bundle args = new Bundle();
    args.putInt(ARG_TAB_NAME, tabName);
    frg.setArguments(args);

    return frg;
  }

  @Override
  public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup cor
          Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_tab, container, false)

    String title = getString(getArguments().getInt(ARG_TAB_NAME));
    ((TextView) layout.findViewById(R.id.textView)).setText(title);

    return layout;
  }

}
```

- RecyclerViewFragment. Tal y como el nombre sugiere, muestra un RecyclerView muy sencillo pues las filas del listado constan de un TextView.

```java
package com.danielme.android.tabs.fragments;

import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.DividerItemDecoration;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.danielme.tabs.R;

import java.util.ArrayList;
import java.util.List;

public class RecyclerViewFragment extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup cor
          Bundle savedInstanceState) {
    View layout = inflater.inflate(R.layout.fragment_recycler_view, contair

    RecyclerView recyclerView = layout.findViewById(R.id.recycler_view);
    recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));

    List<String> items = new ArrayList<>();
    for (int i = 0; i < 25; i++) {
      items.add("item " + i);
    }

    recyclerView.setAdapter(new SimpleTextRecyclerViewAdapter(getContext(),

    DividerItemDecoration dividerItemDecoration = new DividerItemDecoratior
            DividerItemDecoration.VERTICAL);
    recyclerView.addItemDecoration(dividerItemDecoration);

    return layout;
  }

}
```

Utilizando estos dos fragments, implementamos el Adapter del ViewPager especializando FragmentStateAdapter. Este adapter será responsable de informar del número de páginas, que serán cuatro en el ejemplo, entendiendo por página una interfaz gráfica encapsulada en un fragment que se mostrará dentro del mismo ViewPager, y devolver la página (fragment) que se debe mostrar. Los datos de cada Tab se han encapsulado en un enumerado.

```java
package com.danielme.android.tabs;

import androidx.annotation.DrawableRes;
import androidx.annotation.NonNull;
import androidx.annotation.StringRes;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.lifecycle.Lifecycle;
```

```java
import androidx.viewpager2.adapter.FragmentStateAdapter;

import com.danielme.tabs.R;

import java.util.HashMap;
import java.util.Map;

class ViewPagerAdapter extends FragmentStateAdapter {

  enum Tab {

    HOME(0, R.string.tab_home, R.drawable.baseline_home_white_24),
    FAV(1, R.string.tab_fav, R.drawable.baseline_favorite_white_24),
    MUSIC(2, R.string.tab_music, R.drawable.baseline_audiotrack_white_24),
    MOVIES(3, R.string.tab_movies, R.drawable.baseline_movie_white_24);
    final int title;
    final int icon;
    final int position;

    Tab(int position, @StringRes int title, @DrawableRes int icon) {
      this.position = position;
      this.title = title;
      this.icon = icon;
    }

    private static final Map<Integer,Tab> map;
    static {
      map = new HashMap<>();
      for (Tab t : Tab.values()) {
        map.put(t.position, t);
      }
    }

    static Tab byPosition(int position) {
      return map.get(position);
    }
  }

  public ViewPagerAdapter(@NonNull FragmentManager fragmentManager, @NonNul
    super(fragmentManager, lifecycle);
  }

  @NonNull
  @Override
  public Fragment createFragment(int position) {
    if (position == Tab.HOME.position)
      return TabNameFragment.newInstance(Tab.HOME.title);
    else if (position == Tab.FAV.position)
      return TabNameFragment.newInstance(Tab.FAV.title);
    else if (position == Tab.MUSIC.position)
      return TabNameFragment.newInstance(Tab.MUSIC.title);
    else if (position == Tab.MOVIES.position)
       return new RecyclerViewFragment();
    else
      throw new IllegalArgumentException("unknown position " + position);
  }

  @Override
  public int getItemCount() {
    return Tab.values().length;
```

```
    }
}
```

Ahora toca configurar en la Activity tanto el ViewPager como el TabLayout e integrarlos. Al ViewPager hay que proporcionarle el Adapter anterior y vamos a aplicar el siguiente drawable como separador de las páginas utilizando un ItemDecoration como si de un RecyclerView se tratase.

res/drawable/divider.xml

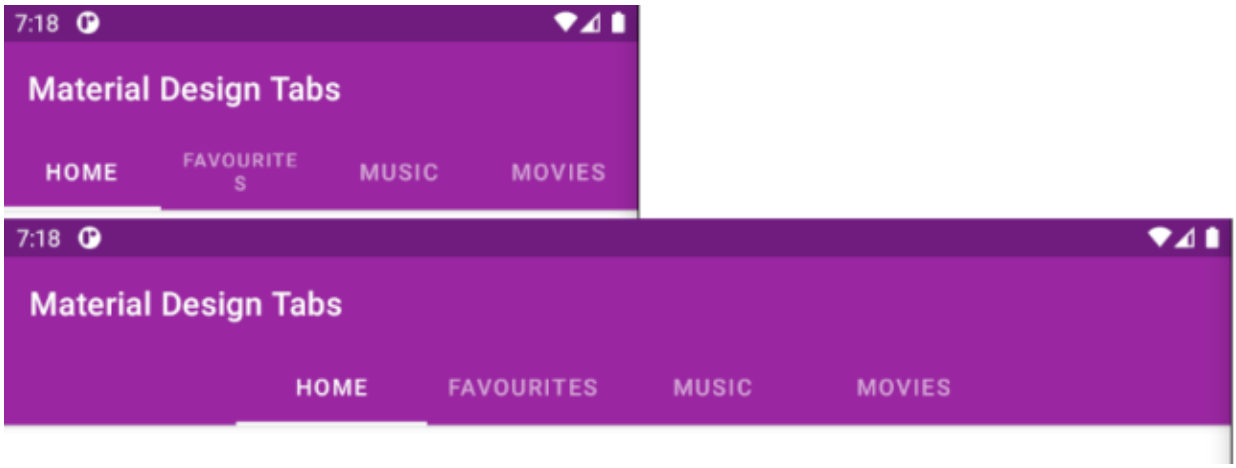
```xml
<?xml version="1.0" encoding="utf-8"?>

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

    <size
        android:width="@dimen/divider"
        />
    <solid android:color="@color/greyBackground" />

</shape>
```

```
private void setupViewPager() {
  viewPager = findViewById(R.id.viewpager);
  DividerItemDecoration dividerItemDecoration = new DividerItemDecoration(th
          DividerItemDecoration.HORIZONTAL);
  dividerItemDecoration.setDrawable(getResources().getDrawable(R.drawable.di
  viewPager.addItemDecoration(dividerItemDecoration);
  viewPager.setAdapter(new ViewPagerAdapter(getSupportFragmentManager(), get
}
```

La integración entre TabLayout y ViewPager2 es realizada por la clase TabLayoutMediator. Su cometido consiste en sincronizar los cambios en ambos elementos.

```
tabLayout = findViewById(R.id.tabs);
new TabLayoutMediator(tabLayout, viewPager,
        (tab, position) -> {
          tab.setText(ViewPagerAdapter.Tab.byPosition(position).title);
        })
        .attach();
```

El resultado es el siguiente.



Se navega entre las distintas páginas de forma secuencial deslizándolas, o bien pulsando en la pestaña. En ambos casos, la página\pestaña mostrada queda debidamente indicada en el TabLayout con un color de icono y texto distinto.

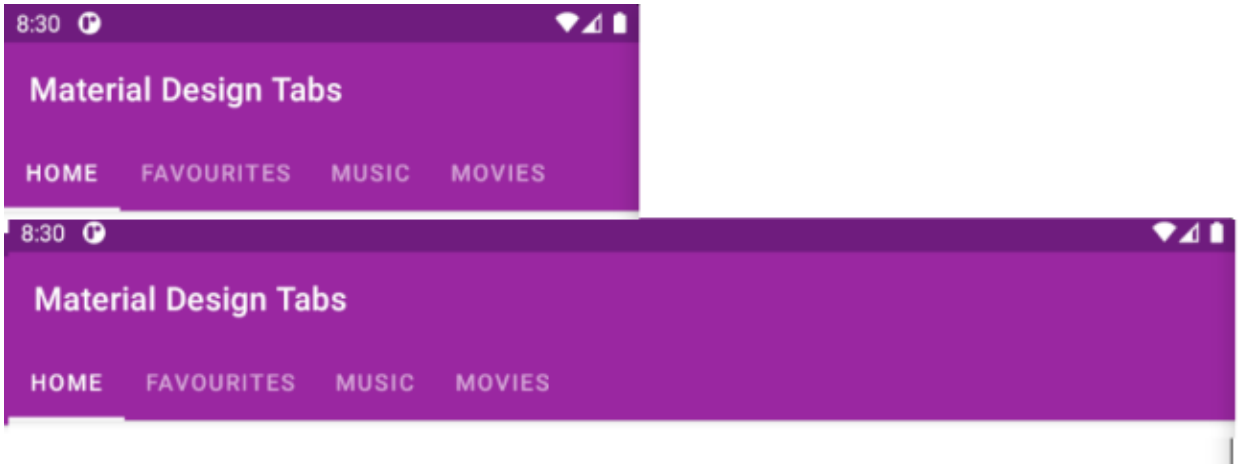
Con la configuración mínima que hemos hecho y los estilos seleccionados ya casi lo tenemos excepto por dos detalles: en vertical el texto tiene un molesto salto de línea y en horizontal las pestañas salen centradas

Podemos jugar con las siguientes propiedades

- app:tabMode="fixed" muestra todas las pestañas a la vez. Los textos se acortan si no caben.
- app:tabMode="scrollable" a cada pestaña se le asigna todo el espacio que necesite para mostrar el texto completo, de tal modo que si no caben todas en la pantalla se habilita un scroll horizontal.
- app:tabGravity="center" centra las pestañas en el layout, independientemente del tabMode
- app:tabGravity="fill" en modo fixed, las pestañas ocupan todo el espacio disponible

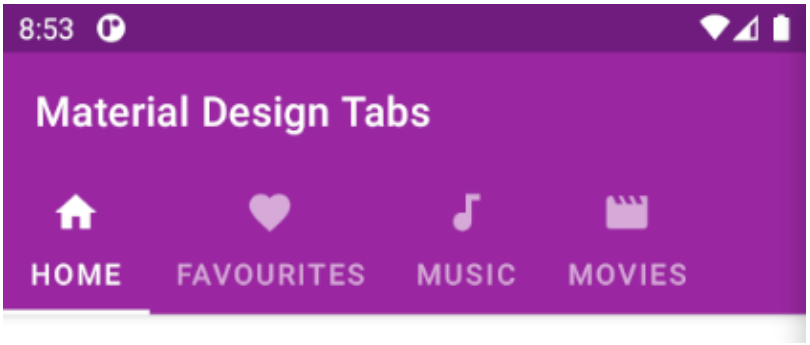
Esto es lo que dice la documentación oficial. Sin embargo, la única manera que he encontrado para mostrar las pestañas a la derecha en modo apaisado es la combinación scrollable-fill.

```
<com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:tabMode="scrollable"
            app:tabGravity="fill"
            style="@style/Widget.MaterialComponents.TabLayout.PrimarySurface'
            />
```



Si queremos mostrar iconos en las pestañas lo haremos en el momento en el que se configuran los títulos en el TabLayoutMediator.

```
new TabLayoutMediator(tabLayout, viewPager,
      (tab, position) -> {
         tab.setText(ViewPagerAdapter.Tab.byPosition(position).title);
         tab.setIcon(ViewPagerAdapter.Tab.byPosition(position).icon);
       })
      .attach();
```



Un par de tips rápidos.

- Desplazar programáticamente hasta una página. Indicar si se desea la animación habitual o que aparezca la página inmediatamente.

```
1 viewPager.setCurrentItem(3, false);
```

- Deshabilitar el swipe (desplazar) para cambiar de página. De este modo sólo se podrá navegar por el ViewPager con las pestañas

```
1 viewPager.setUserInputEnabled(false);
```

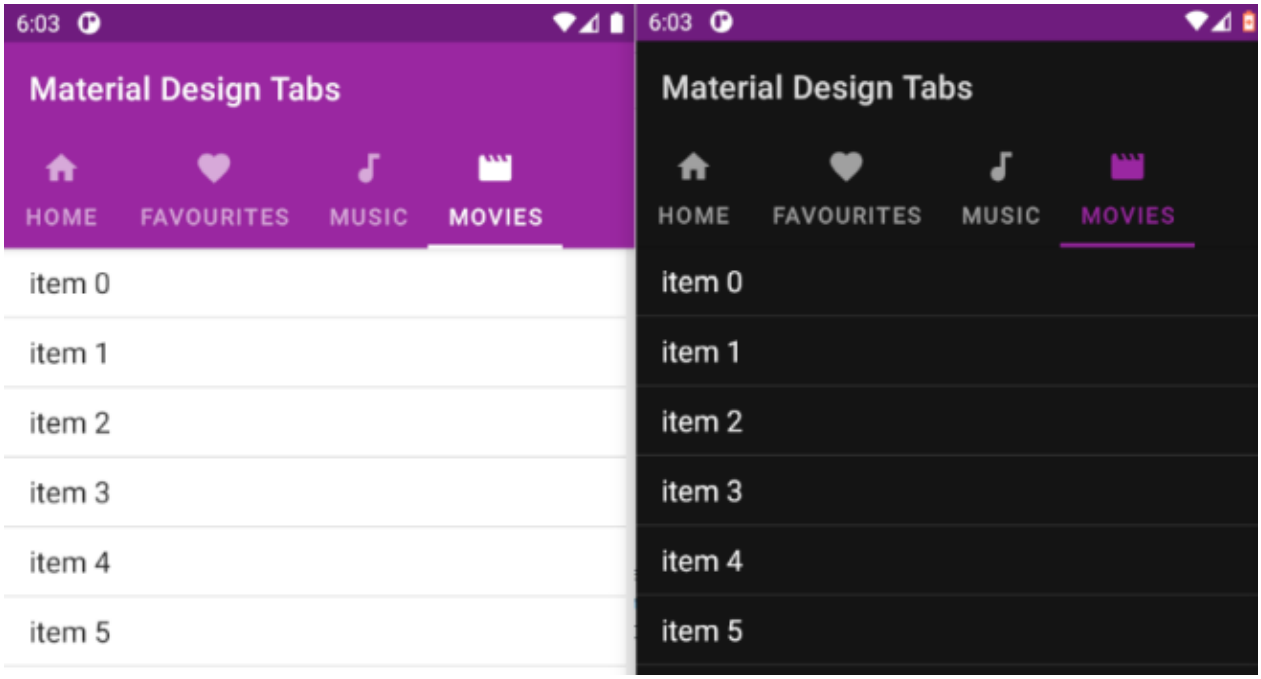
Colores

Este es el tema general de la app con soporte para modo oscuro.

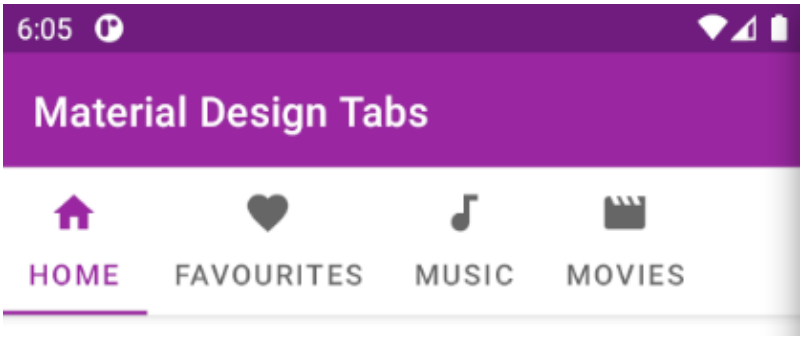
/res/values/themes.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <style name="AppTheme" parent="Theme.MaterialComponents.DayNight.NoActi
5          <item name="colorPrimary">@color/primary</item>
6          <item name="android:statusBarColor">@color/primaryDark</item>
7          <item name="toolbarStyle">@style/Widget.MaterialComponents.Toolbar.
8      </style>
9
10 </resources>
```

Los colores utilizados en el TabLayout son los adecuados para el colorPrimary si le aplicamos como estilo el tema *@style/Widget.MaterialComponents.TabLayout.Colored* o bien *@style/Widget.MaterialComponents.TabLayout.PrimarySurface* el cual aplica colorPrimary para el tema claro y colorSurface para modo oscuro (a la toolbar se le ha aplicado el tema equivalente). Para los elementos de la tab (indicador, texto e icono), se aplica el color definido en la propiedad colorOnPrimary del tema principal en modo claro y *colorOnSurface* en modo oscuro. En este último caso, el elemento seleccionado aparece con colorPrimary.



Si no se especifica ningún tema para el TabLayout, lo veríamos así.



Las siguientes propiedades son las más interesantes a mi parecer para personalizar el estilo del TabLayout si fuera necesario de forma más detallada que simplemente aplicando la configuración de colores del tema general de la aplicación.

- tabIndicatorColor, tabIndicator: configuran el color o un drawable respectivamente para darle estilo a la línea que aparece debajo del tab seleccionado y que se mueve de forma sincronizada con el movimiento de las páginas.
- tabIconTint, tabTextColor: aplica un color o drawable tanto al icono y como a su etiqueta respectivamente. Debemos aplicar estilos distintos en función de si la pestaña está seleccionada o no, así que hay que utilizar un drawable como el siguiente.

/res/drawable/tab_icon_selector.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <selector xmlns:android="http://schemas.android.com/apk/res/android">
3      <item android:color="@color/tab_color_selected" android:state_selected='
4      <item android:color="@color/tab_color_unselected"/>
5  </selector>
```

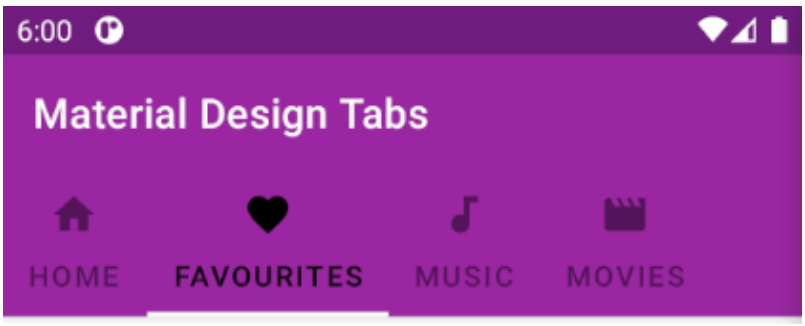
Lo aplicamos al tab.

```
1  app:tabIconTint="@drawable/tab_icon_selector"
2  app:tabTextColor="@drawable/tab_icon_selector"
```

Al elegir los colores, el "truco" está en utilizar exactamente el mismo para los dos estados, pero con una transparencia\opacidad alpha para el estado no seleccionado de tal modo que el icono y el texto aparezcan "apagados" en comparación con la pestaña activa. La trasparencia se aplica directamente en la definición del color añadiendo a la izquierda su valor (0-255) con dos dígitos hexadecimales.

/res/values/colors.xml

```
1  <item name="tab_color_selected" type="color">#000000</item>
2  <item name="tab_color_unselected" type="color">#70000000</item>
```

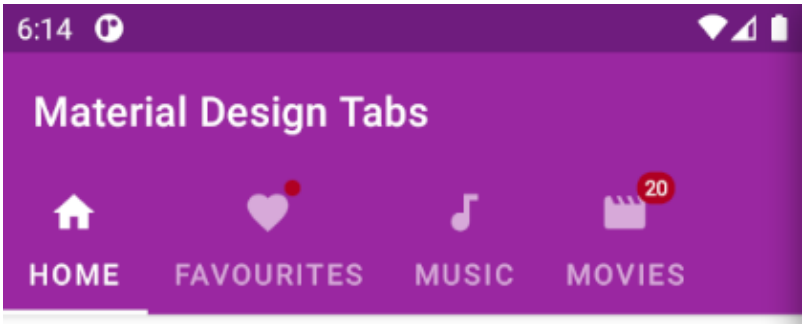


Indicadores (badges)

Material Design contempla la posibilidad de utilizar indicadores dinámicos en las pestañas para llamar la atención del usuario, por ejemplo para indicar que en una hipotética página con notificaciones tiene entradas pendientes de leer. Podemos encontrar un ejemplo en WhatsApp.

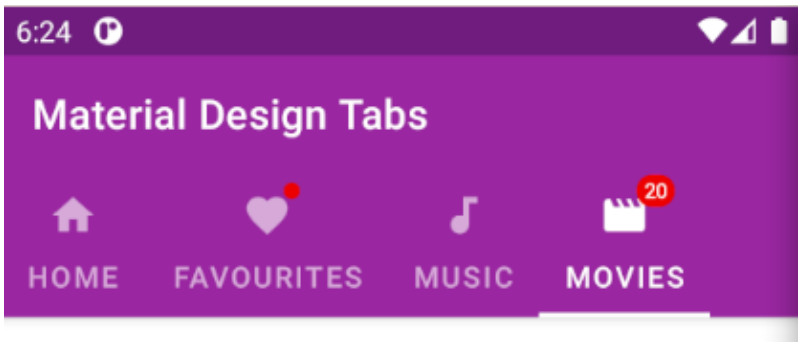
El componente TabLayout permite configurar estos indicadores fácilmente gracias a la integración que ofrece con el elemento BadgeDrawable.

```
1  tabLayout.getTabAt(ViewPagerAdapter.Tab.FAV.position)
2              .getOrCreateBadge()
3              .setVisible(true);
4  tabLayout.getTabAt(ViewPagerAdapter.Tab.MOVIES.position)
5              .getOrCreateBadge()
6              .setNumber(20);
```



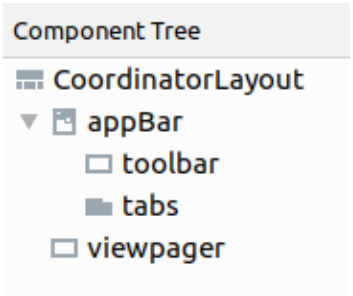
El color rojo no ofrece suficiente contraste con nuestro primary color pero podemos cambiarlo por uno más claro y llamativo.

```
1  tabLayout.getTabAt(pos)
2           .getOrCreateBadge()
3           .setBackgroundColor(getResources().getColor(R.color.tab_badge));
```



Ocultación automática

Al deslizarse en sentido vertical el contenido de la pantalla, el AppBarLayout con las pestañas puede permanecer fijo o bien desplazarse junto con el contenido de tal modo que quede completamente oculto o mostrando solo las pestañas (esto es lo más habitual), pero nunca ocultando las pestañas y dejando visible la ActionBar. Este comportamiento permite aprovechar mejor la pantalla para mostrar el contenido que está visualizando el usuario. Lo podemos configurar utilizando CoordinatorLayoutcomo vista "padre" de nuestros componentes.



```xml
<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBa

        <com.google.android.material.appbar.MaterialToolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:layout_scrollFlags="scroll|enterAlways|snap"
            />

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            style="@style/Widget.MaterialComponents.TabLayout.PrimarySurfac
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:tabGravity="fill"
            app:tabMode="scrollable" />

    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"/>

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

En el video se aprecia claramente cómo AppBarLayout se desplaza dejando oculta la toolbar, pero como la única pestaña con contenido scrollable es la última, si cambiamos de página con la toolbar oculta no habrá manera de volver a mostrarla a menos que volvamos a la última pestaña. Para evitar este problema tenemos que asegurar que el AppBarLayout se muestra en su totalidad cuando se seleccione una página de contenido no "scrolable" (precisamente es lo que hace WhatsApp), y este evento lo trataremos añadiendo un OnPageChangeCallback al ViewPager.

```
  viewPager.registerOnPageChangeCallback(new ViewPager2.OnPageChangeCallbac
    @Override
    public void onPageSelected(int position) {
      super.onPageSelected(position);
      if (position != ViewPagerAdapter.Tab.MOVIES.position) {
        appBarLayout.setExpanded(true);
      }
    }
  });
}
```