Un manual práctico para desarrollar aplicaciones en Android alineadas a las guías de Material Design

compAs

Un manual práctico para desarrollar aplicaciones en Android alineadas a las guías de Material Design

Autores

César Andrés Alcívar Aray
Angel Marcel Plaza Vargas
Michelle Agustina Varas Chiquito
Michael Mario Melo Parrales

Un manual práctico para
desarrollar aplicaciones en Android
alineadas a las guías de Material Design

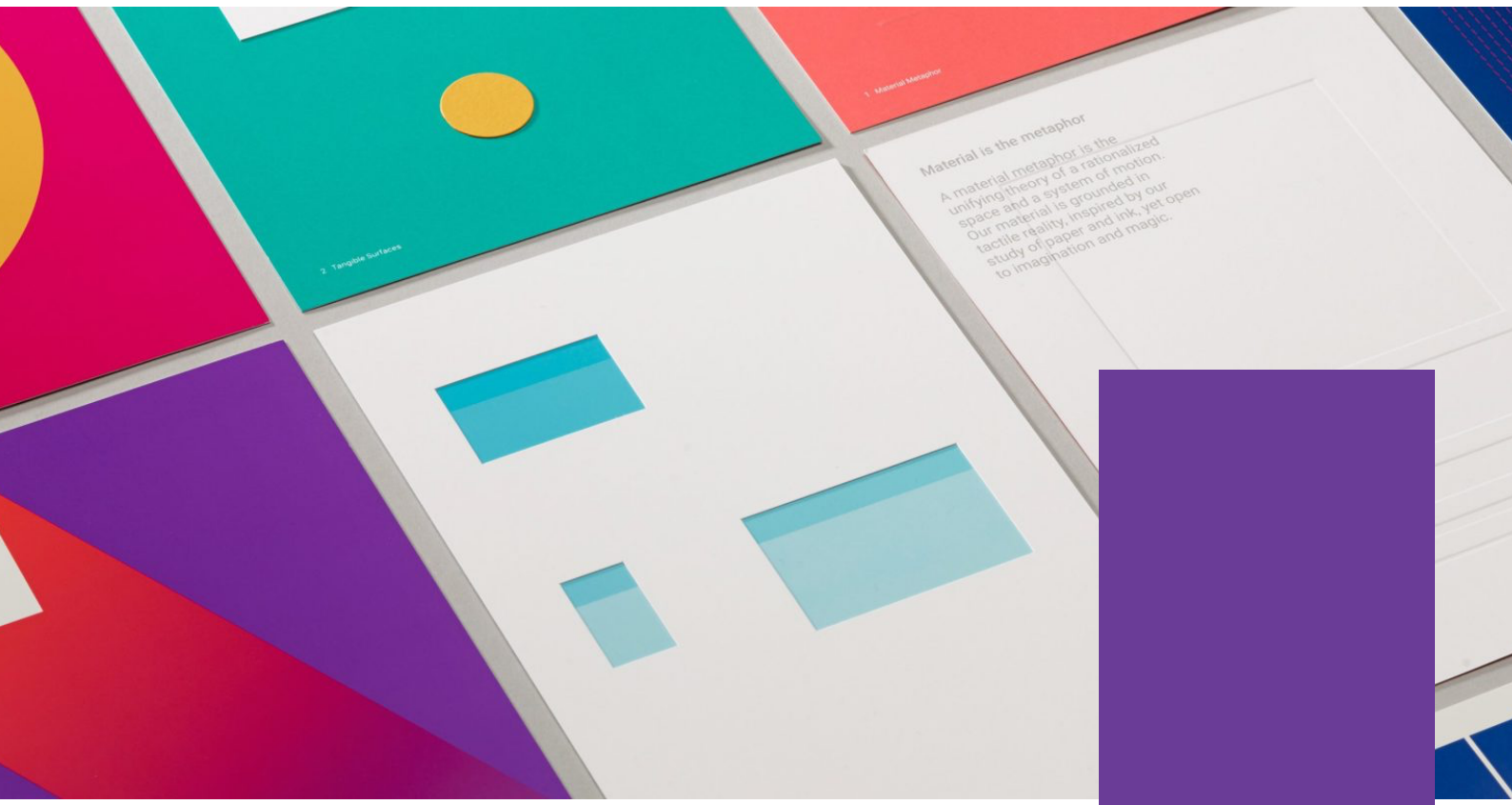
Autores

César Andrés Alcívar Aray Docente de la Universidad de Guayaquil
Angel Marcel Plaza Vargas Docente de la Universidad de Guayaquil
Michelle Agustina Varas Chiquito Docente de la Universidad de Guayaquil
Michael Mario Melo Parrales Universidad de Guayaquil

Este texto ha sido sometido a un proceso de evaluación por pares externos
con base en la normativa del editorial

Primera edición: Junio 2018
Diseño de portada y diagramación:
Grupo Compás
Equipo Editorial
ISBN: 978-9942-770-99-8

Tabla de contenido


2 Tangible Surfaces
1 Material Metaphor
Material is the metaphor
A material metaphor is the
unifying theory of a rationalized
space and a system of motion.
Our material is grounded in
tactile reality, inspired by our
study of paper and ink, yet open
to imagination and magic.

Agradecimiento

Me gustaría agradecer a todas las personas que me apoyaron y alentaron a través del proceso de escritura, pero en especial a mi esposa Lorena, por estar constantemente junto a mí y por su comprensión en innumerables noches y fines de semana trabajando en los capítulos de este libro. Mi amor infinito hacia ti.

Agradezco a mis padres por apoyarme en mis estudios. Si no fuera por ustedes, no tendría lo que tengo y no pudiese haber escrito este texto. Los quiero mucho.

A mi Hijo Galito, mi inspiración en todo momento, este libro también es tuyo.

César Alcívar Aray

Prefacio

Este libro es una guía práctica para aprender a desarrollar aplicaciones móviles en la plataforma Android alineadas a las guías de Material Design, el cual ayudará a los desarrolladores a conocer las buenas prácticas de diseño y cómo llevar al código fuente los comportamientos definidos en Material Design a sus aplicaciones móviles.

El libro está compuesto de seis capítulos. En el primer capítulo, exploramos los comportamientos de los componentes de interfaz de usuario a través de wireframes. Del capítulo dos al capítulo cinco, crearemos varios proyectos en Android Studio para desarrollar los comportamientos de los widgets Appbar, Navigation Drawer, Bottom Navigation, TextField, Snackbar y Floating Action Button definidos en la guía de Material Design. En el capítulo seis, analizaremos cómo las principales redes sociales han adaptado Material Design a sus aplicaciones móviles.

Qué cubre el libro

Capítulo 1, *Del Diseño del Código*. Se explicarán brevemente algunos conceptos de Material Design y lo que necesita conocer un desarrollador de software. Mostraremos los comportamientos, usos y estilos de los widgets que desarrollaremos a través de los capítulos de este libro.

Capítulo 2, *Primera Aplicación con Material Design*. Analizaremos los fundamentos de diseño que los desarrolladores necesitan conocer para avanzar sin dificultades a través de este libro. Se proporcionará información sobre algunas herramientas que están disponibles en Internet para definir colores e íconos. Finalmente, crearemos nuestro primer proyecto utilizando algunos componentes de Material Design.

Capítulo 3, *AppBar*. Veremos las métricas de Material Design para algunas listas de elementos. Se desarrollará el comportamiento del estándar Appbar, del Appbar con Tabs, del Appbar con espacio flexible con y sin imagen, y del Appbar con contenido sobrepuesto.

Capítulo 4, *Navigation Drawer Y Bottom Navigation*. Desarrollaremos los comportamientos del Navigation Drawer y del Bottom Navigation. Utilizaremos como base el proyecto `AppbarConTab` del capítulo 2 y cambiaremos el Appbar con pestañas por un Navigation Drawer y por un Bottom Navigation.

Capítulo 5, *TextFields, Snackbar, FloatingActionButton*. Veremos las métricas de los componentes TextField, `SnackBar` y `FloatingActionButton`. Desarrollaremos algunos comportamientos para validar información de entrada a través de TextFields. Aplicaremos el Snackbar para mostrar mensajes consecutivos sobre un proceso. Utilizaremos componentes de terceros para implementar comportamiento del `FloatingActionButton` que no están soportados por el Framework de Android.

Capítulo 6, *Material Desing en las principales aplicaciones móviles de redes sociales*. Analizaremos la navegabilidad y cómo se muestra la información en las aplicaciones móviles Whatsapp, Twitter, Instagram y Facebook en la plataforma Android y qué tan alineadas están a las guías de Material Design.

Para quién es este el libro

Si eres desarrollador de software y tienes dificultades para crear aplicaciones en la plataforma Android alineadas a las guías de Material Design, este libro es para ti.

Para sacarle provecho, necesitas estar familiarizado con el desarrollo de aplicaciones móviles en la plataforma Android, incluyendo navegabilidad en Android Studio, uso de Layouts, ciclo de vida de Activities y Fragments, y paso de valores entre Activities y Fragments.

Además, se necesita tener conocimientos básicos de Java, como clases y objetos, interfaces, listeners, paquetes, clases internas y clases genéricas.

En caso de que no hayas desarrollado alguna aplicación en Android y no conozcas Java, puedes comenzar con algún libro introductorio de Android y Java, para volver después a este libro.

Qué se necesita para este libro

Para desarrollar los ejemplos de este libro, necesitaremos tener instalados los siguientes programas:

- Android Studio Versión 3.1 o superior
- API 16 de Android o superior
- Java 1.8 o superior

Convenciones

En este libro, usted encontrará una serie de estilos de texto que se distinguen entre los diferentes tipos de información.

Las palabras dentro del texto que hacen referencia a objetos, clases, widgets, nombres de archivos y rutas son mostradas con el tipo de letra `Consolas`, como se visualiza a continuación: `objetos`, `clases`, `widgets`, `nombres de archivos` y `rutas`.

Cuando se escribe un nuevo bloque de código, tendrá la siguiente forma:

```
public class Ejemplo{
     private int id;
     private String nombre;
}
```

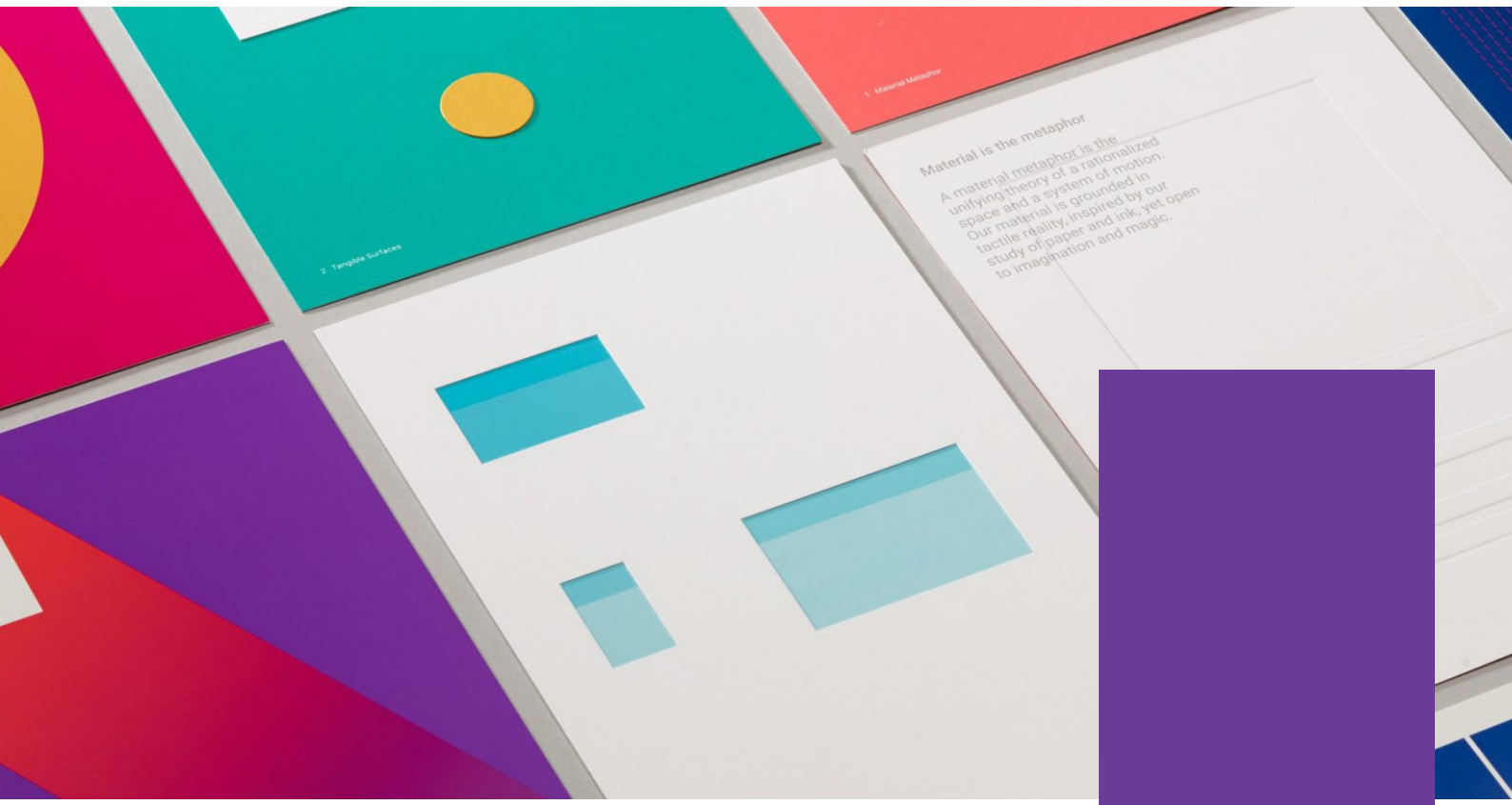
Cuando se actualice un bloque de código, este se mostrará en negritas, como se lo indica a continuación:

```
public class Ejemplo{
     private int id;
     private String nombre;
     private String apellidos
}
```

El tipo de escritura en los archivos layout, son similares a un bloque de código, pero cuando una sentencia aparezca en negritas, significa que más adelante se va a explicar su función.

Descargar el código fuente e imágenes

El código fuente de los proyectos realizados en este libro lo puedes descargar en GitHub en la siguiente dirección: https://github.com/cesarade/DelDisenoAlCodigo. Las imágenes las puedes descargar en la siguiente dirección: https://drive.google.com/drive/folders/0B0mMTdWK8rKKQkd3aFhKTFVPb3M.


2 Tangible Surfaces
1 Material Metaphor
Material is the metaphor
A material metaphor is the
unifying theory of a rationalized
space and a system of motion.
Our material is grounded in
tactile reality, inspired by our
study of paper and ink, yet open
to imagination and magic.

Capítulo 1. Del Diseño al Código

Cuando empiezas a desarrollar una aplicación móvil, deseas que las personas disfruten utilizarla. Si no eres diseñador, o no conoces sobre las buenas prácticas de diseño de interfaz de usuario, buscarás en Internet información que te ayude a desarrollar una interfaz amigable. En una de esas búsquedas, seguramente te encontrarás con lo que se conoce como Material Design.

Empiezas a investigar sobre Material Design y te agrada la idea de que tu aplicación implemente sus buenas prácticas, utilizando elementos visuales y comportamientos comunes con otras aplicaciones, lo que facilitará una rápida adaptación de los usuarios con tu aplicación. Sin embargo, el tiempo es más largo del que tenías previsto en desarrollar la interfaz de usuario debido a que te tropiezas con algunas cosas que no conocías. Por ejemplo, para implementar un App Bar necesitas modificar el tema de la aplicación, o cuando el elemento raíz es un `Coordinator Layout`, si no colocas el atributo `app: layout_behavior="@string/appbar_scrolling_view_behavior"` en el `RecyclerView`, el Appbar se sobrepondrá con el `RecyclerView`.

Material Design establece el qué, pero el cómo puede resultar abrumador para muchos desarrolladores, ya que la mayoría no lidia con elementos de diseño de las interfaces de usuario, tales como los colores, los tipos de letras, las posiciones, las elevaciones, las métricas, entre otros.

Material Desing abarca muchos aspectos sobre el diseño aplicaciones web, de escritorio y móvil. En este libro, codificaremos aplicaciones móviles alineadas a las directivas de Material Design. Nos enfocaremos en los comportamientos de componentes de navegación, de acentuación y listas para presentar información.

En este primer capítulo, se explicarán brevemente algunos conceptos de Material Design y lo que necesita conocer un desarrollador de software. Mostraremos los comportamientos, los usos y los estilos de los widgets que serán desarrollados a través de los capítulos de este libro. En este primer capítulo, se tratarán los siguientes tópicos:

- Material Design para desarrolladores
- Navegabilidad en Material Design
- Lista de contenido en Material Design
- Text Fields, SnackBars, Floating Action Buttons

Material Design para desarrolladores

Material Design es un lenguaje visual que provee una extensa guía de estilos que incorpora principios, elementos visuales, métricas y comportamientos similares para diseñar y desarrollar aplicaciones de escritorio, móviles y web; de tal forma que los usuarios puedan tener una experiencia unificada en todas las plataformas y tamaños de los dispositivos.

En Material Design, los materiales se comportan como papeles reales. Estos papeles se componen de capas, tienen movimiento, elevación y no se traspasan con otros. Para los desarrolladores, un material equivale a un control de interfaz de usuario o widget. Los widgets tienen las dimensiones x, y, z. El eje z proporciona efectos visuales de elevación y sombra, lo que da la impresión de sobreposición entre los widgets.

En Android 5.0 (Lollipop API 21) se introd (Pablo Perea, 2017) (Bill Phillips, 2015) (Ruiz, 2015)ujo Material Design, y a partir de esta versión se han ido incrementando nuevos widgets y comportamientos que se integran fácilmente a los patrones de diseño de Material Design. Appbar, Snackbar, Floating Action Button, EditText Floating Label, Navigation View, TabLayout, Bottom Navigation, entre otros, son algunos de los componentes que se han ido agregando para el desarrollo de aplicaciones en la plataforma Android. Para las versiones anteriores a Lollipop, Google creó bibliotecas de compatibilidad que permiten implementar los componentes de Material Design en versiones antiguas de Android.

Material Design abarca muchos conceptos que los diseñadores de interfaces de usuario seguramente apreciarán, pero este libro se enfocará en los desarrolladores, para que estos puedan llevar los comportamientos de los elementos visuales de Material Design a sus aplicaciones Android. Seguramente, nos quedaremos cortos sobre Material Design. Si deseas conocer más del tema, por favor visita **`https://material.io/`**.

Navegabilidad en Material Design

El usuario debe ser capaz de entender y utilizar cualquier aplicación inmediatamente sin necesidad de un manual o tutorial. El tipo de navegación que se utilice en las aplicaciones es esencial para que esta sea intuitiva y predecible.

Para elegir el tipo de navegación de una aplicación, se debe considerar el tipo de usuario que la utilizará, las rutas más frecuentes donde navegarán los usuarios y las acciones que se realizarán dentro de la aplicación.

Para establecer algún tipo de navegabilidad en los dispositivos móviles, también se debe considerar el tamaño del mismo. El contenido se debe organizar por secciones de forma que los usuarios encuentren poco a poco información relevante. Si el usuario desea ver

más información, este podrá seguir navegando hacia información más detallada a través de controles de interfaz de usuario claramente cliqueables.

En Android, encontramos algunos widgets que permiten definir el tipo de navegabilidad de una aplicación. Material Design detalla los estilos y comportamientos para estos widgets, los cuales serán analizados con más detalle a continuación.

Back Button

Permite al usuario regresar a la pantalla anterior, de la mima forma que una ruta de navegación en una página web. Este botón generalmente está ubicado en la parte superior izquierda del Appbar.

Figura 1.1 Wireframe del BackButton

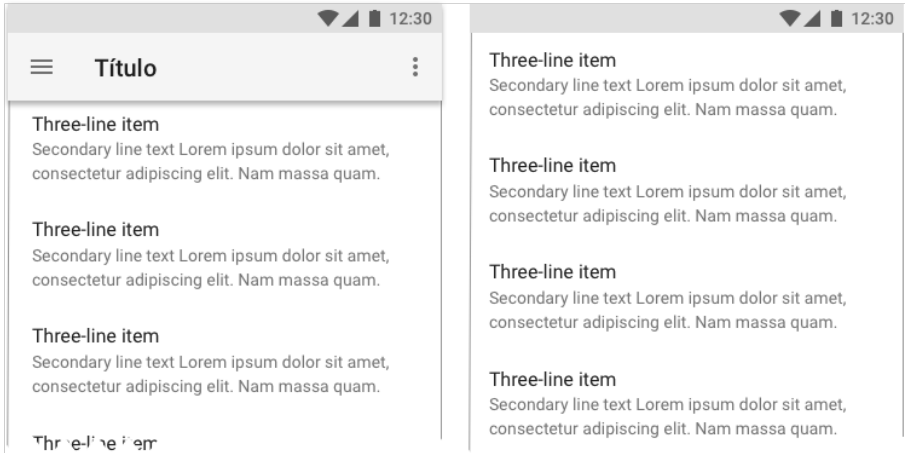
App Bar

El Appbar es una barra de herramientas que se ubica en la parte superior de la aplicación. Proporciona una estructura visual y elementos conocidos por los usuarios para navegar entre las actividades o fragmentos y proporciona acceso a las principales acciones de la aplicación.

Material Design provee guías para establecer comportamientos del Appbar en relación al contenido de la aplicación. El comportamiento estándar del Appbar tiene dos opciones de scrolling, las cuales se definen a continuación:

1. El Appbar se mantiene fija cuando el usuario desliza el contenido de la lista hacia abajo.
2. El Appbar va desapareciendo de la pantalla cuando el usuario desliza el contenido de la lista hacia arriba.

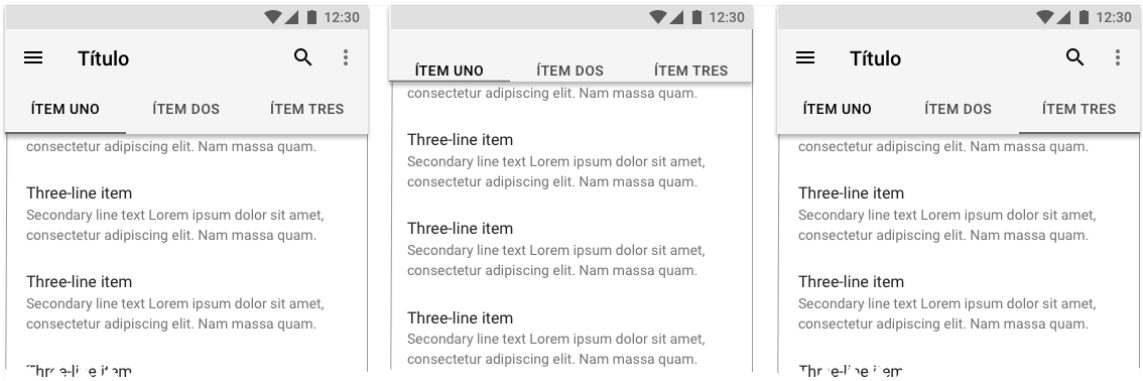
Figura 1.2 Wireframe del comportamiento estándar del App Bar



El comportamiento de Appbar con *tabs* (pestañas) proporciona navegación para diferentes secciones de contenido, permitiendo pasar de una página a otra cuando se presiona un tab. Este tipo de navegación se usa mucho en aplicaciones móviles y web, siendo muy común en las aplicaciones Android. Puede existir junto con un menú superior. Este comportamiento tiene tres opciones de scrolling:

1. El Appbar se mantiene fija, mostrando las pestañas y el Appbar cuando el usuario desliza el contenido de la lista hacia abajo.
2. El Appbar va desapareciendo de la pantalla cuando el usuario desliza el contenido de la lista hacia arriba y se mantienen las pestañas de navegación.
3. Cuando se presiona alguna pestaña del Appbar, cambia el contenido o la página en relación a la pestaña que se seleccionó.

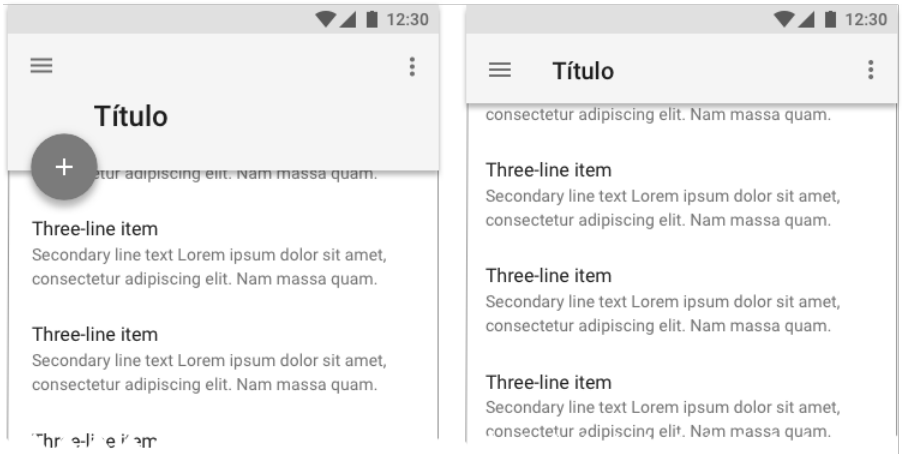
Figura 1.3 Wireframe del comportamiento del Appbar con Tabs



El comportamiento del Appbar con espacio flexible es muy parecido al comportamiento estándar. Este tiene un espacio definido por el desarrollador y un `FloatingActionButton` establecido en la esquina superior izquierda del Appbar. Este tiene dos vistas:

1. El Appbar se mantiene fijo, mostrando el espacio extra y el `Floating Action Button`, cuando el usuario desliza el contenido de la lista hacia abajo.
2. Cuando el usuario desliza el contenido de la lista hacia arriba, va desapareciendo el espacio extra y el `Floating Action Button`. El título de la Appbar se va ubicando en la parte superior y va disminuyendo su tamaño.

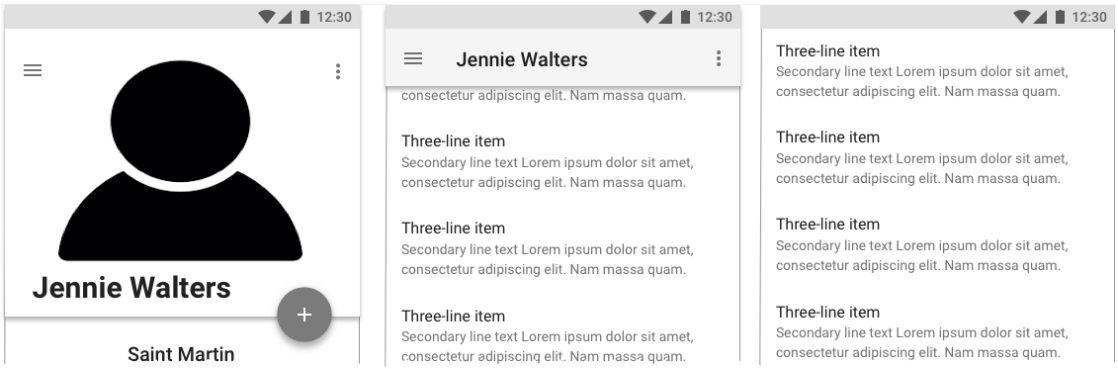
Figura 1.4 Wireframe del comportamiento del App Bar con espacio flexible



El comportamiento del Appbar con espacio flexible e imagen normalmente no se utiliza para establecer la navegabilidad en una aplicación móvil. Se utiliza para mostrar información más detallada de un ítem que pertenece a una lista de información. Este comportamiento tiene tres vistas:

1. El Appbar, la imagen y un `Floating Action Button` colocado en la esquina superior derecha del Appbar se mantienen fijos cuando el usuario desliza el contenido de la lista hacia abajo.
2. Cuando el usuario desliza por primera vez el contenido hacia arriba, la imagen va desapareciendo hasta que el Appbar llegue a su altura por defecto
3. En un segundo deslizamiento del contenido hacia arriba, el Appbar va desapareciendo hasta quedar visible solo el status bar.

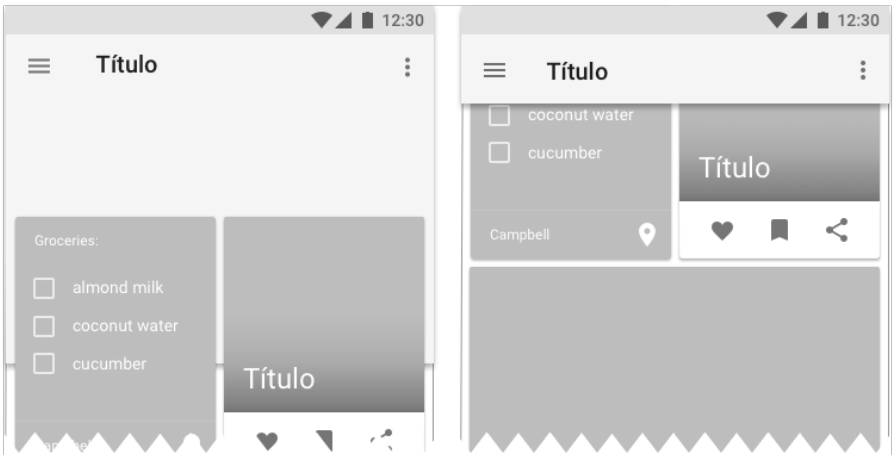
Figura 1.5 Wireframe del comportamiento estándar del Appbar con espacio flexible con imagen



El comportamiento del Appbar con espacio flexible y contenido sobrepuesto tiene dos vistas:

1. El Appbar y el contenido sobrepuesto se mantienen fijos cuando el usuario desliza la lista hacia abajo.
2. Cuando el usuario desliza el contenido de la lista hacia arriba, va desapareciendo el espacio extra y el contenido se va ubicando debajo del Appbar.

Figura 1.6 Wireframe del comportamiento del Appbar con espacio flexible y contenido sobrepuesto

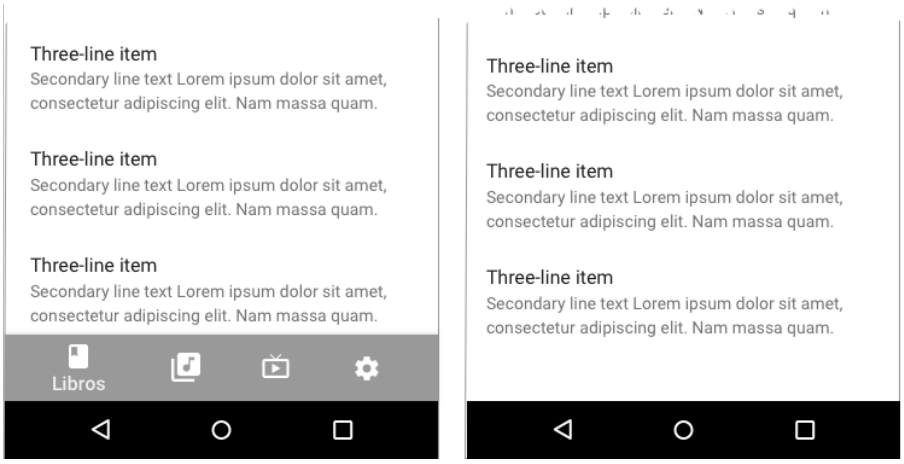


Bottom Navigation

El comportamiento del Appbar con espacio flexible e imagen normalmente no se utiliza para establecer la navegabilidad en una aplicación móvil. Se utiliza para mostrar información más detallada de un ítem que pertenece a una lista de información. Este comportamiento tiene tres vistas:

1. El *Bottom Navigation* se mantiene fijo cuando el usuario desliza el contenido de la lista hacia abajo.
2. El *Bottom Navigation* va desapareciendo de la pantalla cuando el usuario desliza el contenido de la lista hacia arriba.

Figura 1.7 Wireframe del Bottom Navigation



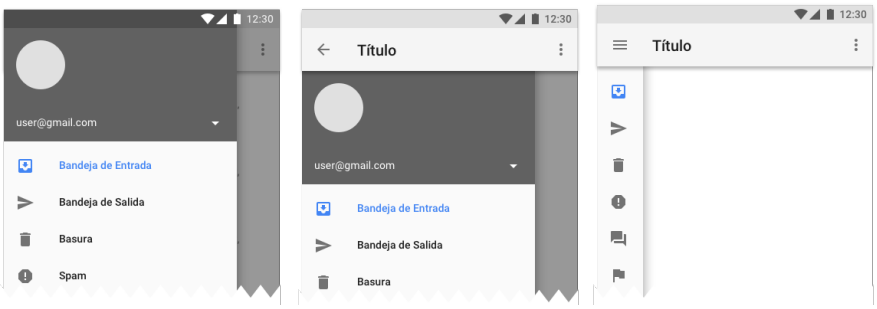
Navigation Drawer

Navigation Drawer es un menú lateral que sale del lado izquierdo o del lado derecho de la aplicación cuando se desliza el dedo por la pantalla en los bordes o hacemos clic en un botón superior que normalmente tiene un ícono con tres líneas horizontales en el *App Bar*.

Una de las ventajas de este tipo de navegación es que permite tener una cantidad indefinida de opciones. Este menú es muy popular para crear aplicaciones móviles no nativas, como las aplicaciones híbridas o aplicaciones web para dispositivos móviles.

Material Design establece algunos comportamientos para un *Navigation Drawer*, entre los que tenemos *Navigation Drawer* con Altura Completa, *Navigation Drawer* debajo del *App Bar* y *Navigation Drawer* variante mínima, como se muestra a continuación:

Figura 1.8 Wireframe de los comportamientos del Navigation Drawer



Independientemente del comportamiento, el *Navigation Drawer* tiene dos formas:

1. El *Navigation Drawer* permanece oculto.
2. Cuando el usuario hace clic en el botón que tiene el ícono con tres líneas horizontales, sale un menú del lado izquierdo mostrando las opciones de la aplicación.

Listas de contenido en Material Design

Una lista muestra un conjunto de datos homogéneos que deben ser fáciles de leer. Los datos pueden ser textos o imágenes o una combinación de ambas. Material Design sugiere algunos modelos y métricas para implementar listas de contenido. Encontraremos algunos modelos de listas en este

link `https://material.io/guidelines/components/lists.html`. Estos modelos pueden ser utilizados en cualquier contexto, sin olvidar que son diseños sugeridos, por lo que como desarrolladores podemos implementar estas listas en cualquier aplicación o modificar la apariencia de las mismas. A lo largo de este libro, utilizaremos dos modelos de listas, las cuales describimos a continuación:

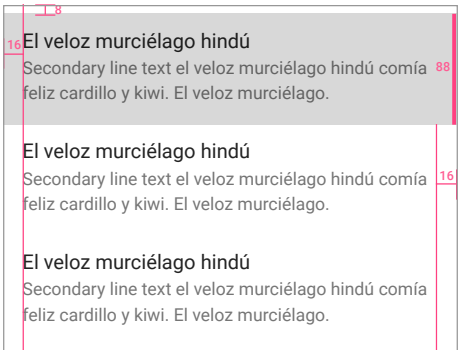
Lista de tres líneas sin imagen

La lista de tres líneas presenta las siguientes métricas:

- *Primary text Font*: Roboto Regular 16sp
- *Secundary text Font*: Roboto Regular 14sp
- *Text padding*, left: 16dp
- *Text padding*, right: 16dp
- *Text padding*, top: 16dp
- *Text padding*: bottom: 20dp

A continuación, se muestra gráficamentente esta lista:

Figura 1.9 Métricas para lista de tres líneas



A continuación, se detalla con más exactitud las métricas de esta lista:

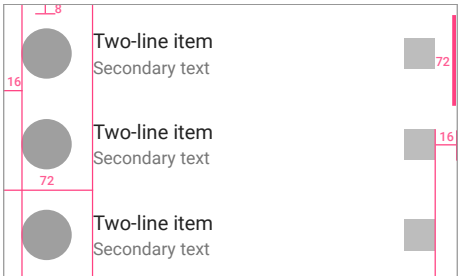
Lista de dos líneas con imagen

La lista de dos líneas con imagen presenta las siguientes métricas:

- Primary text Font: Roboto Regular 16sp
- Secundary text Font: Roboto Regular 14sp
- Tile height: 72dp
- Icon left padding: 16dp
- Text left padding: 72dp
- Text right padding: 16dp

A continuación, se muestra gráficamentente esta lista:

Figura 1.10 Métricas para lista de dos líneas con imágenes



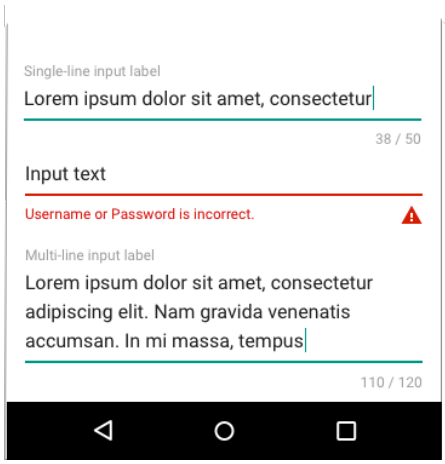
Text Fields, Snack Bars y Floating Action Buttons

Desde que Material Design fue introducido en Android, se han actualizado y agregado nuevos componentes de interfaz de usuario. En esta sección, veremos el uso, estilos y comportamiento de algunos de estos *widgets*.

Text Field

Los *Text Fields* permiten a los usuarios ingresar y seleccionar texto. Sirven para validar las entradas, ayudar a los usuarios a corregir errores, autocompletar palabras y brindar sugerencias.

Figura 1.11 Wireframe de Text Fields



Snack Bar

Este *widget* provee de mensajes breves de una operación a través de una hoja que aparece en la parte inferior de la pantalla. Contiene una sola línea de texto directamente relacionada con la operación que se realiza.

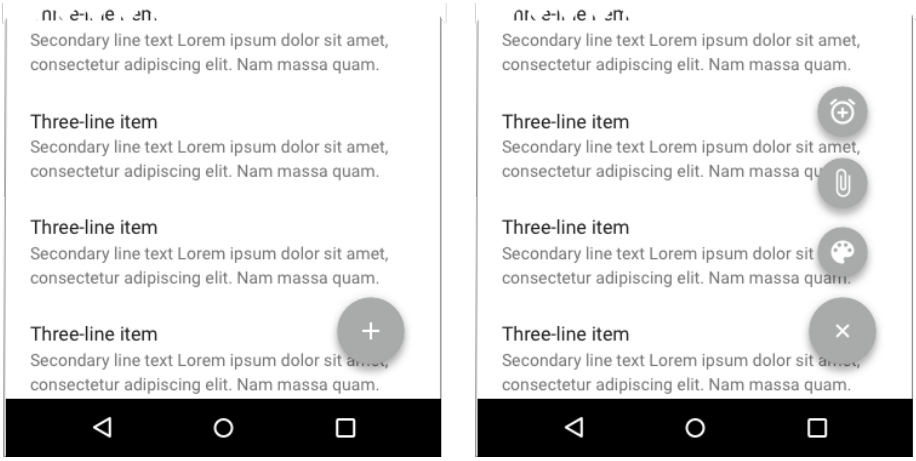
Figura 1.12 Wireframe de Snackbar

Floating Action Button

Floating Action Button (FAB) es un botón redondo que se eleva por encima del contenido de la aplicación. Viene en tamaño predeterminado y mini. FAB ajusta su posición en respuesta de otros elementos de la interfaz de usuario. Se utiliza para mostrar la acción principal de una aplicación o mostrar acciones relacionadas.

Material Design define algunos comportamientos para este *widget*; se puede transformar en una simple hoja de material o en un menú, entre otros. A continuación, se muestra cómo un FAB se puede utilizar como un menú de acciones de la aplicación:

Figura 1.13 Wireframe de Floating Action Button



Resumen

En este capítulo se explicaron brevemente algunos conceptos de Material Design. Se expusieron los comportamientos, usos y estilos de los widgets Appbar, Navigation Drawer, Bottom Navigation, Text Field, Floating Action Button y Bottom Sheet y cómo mostrar información en listas de contenido.

Capítulo 2. Primera aplicación con Material Design

En este capítulo, analizaremos los fundamentos de diseño que los desarrolladores necesitan conocer para avanzar sin dificultades a través de este libro. Se proporcionará información sobre algunas herramientas que están disponibles en Internet para definir colores e íconos. Finalmente, crearemos nuestro primer proyecto utilizando algunos componentes de Material Design.

Se tratarán los siguientes tópicos:

- Colores, estilos y temas
- Íconos
- Crear el primer proyecto con Material Design

Colores

Los colores en Android se gestionan como recursos en el archivo `res/values/colors.xml`. De esta manera, un recurso color en una localización específica puede ser utilizado por muchos widgets dentro de una aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Los colores `primaryColor`, `primaryColorDark` y `colorAccent` son creados por defecto en `res/values/colors.xml` cuando se crea un proyecto en Android Studio. Más adelante en este capítulo, veremos con más detalle estos colores y la importancia que tienen para las aplicaciones Android.

Estilos

Un estilo es un conjunto de atributos que pueden ser aplicados a uno o más widgets. Para crear estilos, vamos a `res/values/styles.xml` y creamos el estilo "miBoton".

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

```
<resources>
    <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
      …
    </style>
    <style name="miBoton">
        <item name="android:background">@color/colorAccent</item>
    </style>
</resources>
```

En el estilo `miBoton` se definió el atributo `android:background`. Cuando este estilo es aplicado a muchos widgets, cualquier cambio en los atributos del estilo actualizará la apariencia de los mismos. A continuación, se muestra cómo utilizar el estilo `miBoton` en un elemento `Button`:

```
<Button
    android:id="@+id/miBoton"
    style="@style/miBoton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ...
/>
```

Temas

Los temas van un paso más allá de los estilos. Todos los atributos definidos en un tema se aplican automáticamente a toda la aplicación. Los atributos del tema pueden hacer referencia a recursos concretos, como los colores, o almacenar referencias de estilos. Los temas son definidos en el archivo `res/values/styles.xml`.

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar
">
      …
        <item name="android:buttonStyle">@style/miBoton</item>
    </style>
    <style name="miBoton" parent="@android:style/Widget.Button">
        <item name="android:background">@color/colorAccent</item>
    </style>
</resources>
```

En el tema `AppTheme` existe una referencia del estilo `miBoton`. Si la aplicación utiliza a `AppTheme` como tema global, todos los botones de la aplicación tendrán el color de fondo de `colorAccent`.

El tema global para la aplicación se define en el atributo `android:theme`, en el archivo `AndroidManifest.xml`.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ec.edu.ug.holamaterialdesign">
    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
      …
        android:theme="@style/AppTheme">
      ...
    </application>
</manifest>
```

Temas, estilos y colores

Los atributos de un tema tienen un alcance más amplio que los atributos de un estilo. En el archivo `res/values/styles.xml` se establecen los atributos colorPrimary, colorPrimaryDark y colorAccent dentro del tema AppTheme.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

En el atributo `colorPrimary`, se define el color primario de cualquier aplicación. Este color es utilizado por defecto en algunos widgets, por ejemplo, cuando no se define un color en el atributo `android:background` de un Toolbar, este toma el color de `colorPrimary`.

`colorPrimaryDark` es utilizado por el `StatusBar`. Normalmente, este color es un poco más oscuro que `colorPrimary`. `StatusBar` es una característica agregada en `Lollipop`, por lo que en versiones anteriores, este será de color negro, independiente del color de `colorPrimaryDark`.

`colorAccent`, o color secundario, se sugiere que debería ser un contraste con `colorPrimary`. Este color es tomado por defecto por algunos widgets cuando no se establecen valores en sus atributos, entre los que tenemos la propiedad `android:background` del `Floating Action Button` o `android:backgroundTint` de un `EditText`.

Herramientas para definir colores

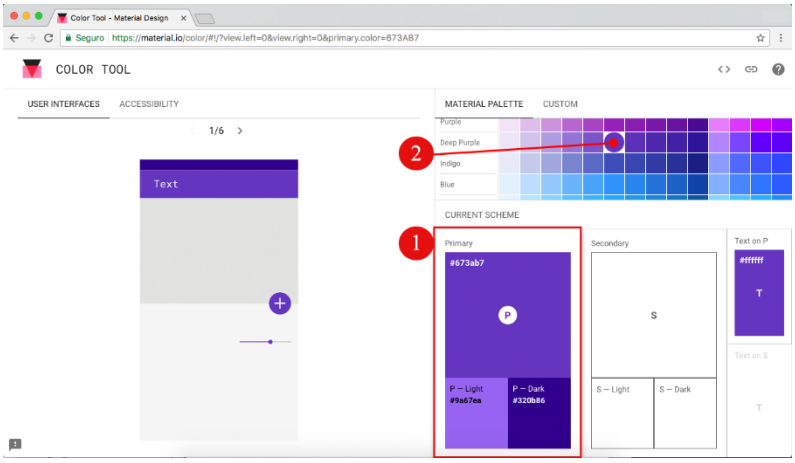
Material Design provee una paleta de colores que los desarrolladores de software pueden utilizar como guía para especificar los colores de una aplicación. Esta paleta está ubicada en https://material.io/guidelines/style/color.html.

Google sugiere que `primaryColor` sean los colores 500, `primaryColorDark` sean los colores 700 y `colorAccent` sean los colores que empiecen con A.

Figura 2.1 Color Palette Indigo

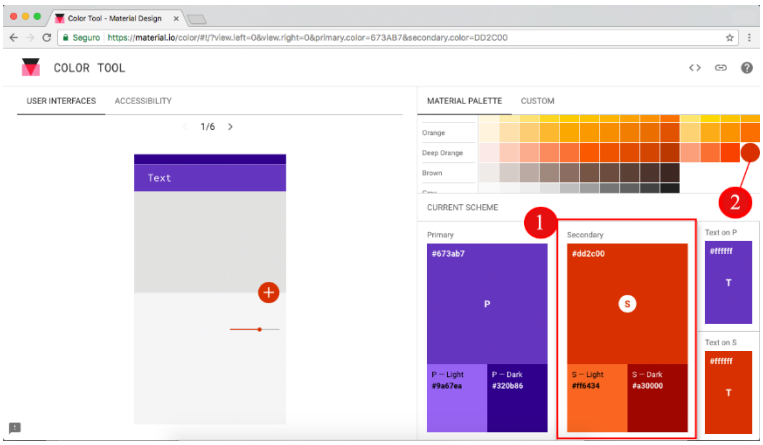
Existen algunas herramientas en Internet que ayudan a decidir qué colores utilizar en una aplicación. Una de estas es https://material.io/color/. Para empezar a utilizarla, hacemos clic en el cuadro `Primary`, luego elegimos `Deep Purple` con el valor de 500.

Figura 2.2 Selección de colorPrimary en Color Tool



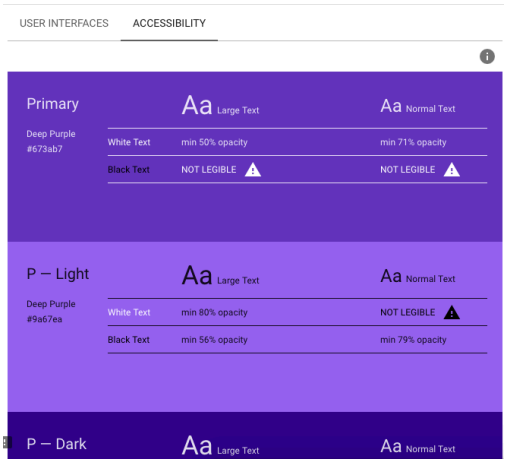
Hacemos clic en el cuadro `Secondary`; a continuación elegimos `Deep Orange` con el valor de 700.

Figura 2.3 Selección de colorAccent en Color Tool



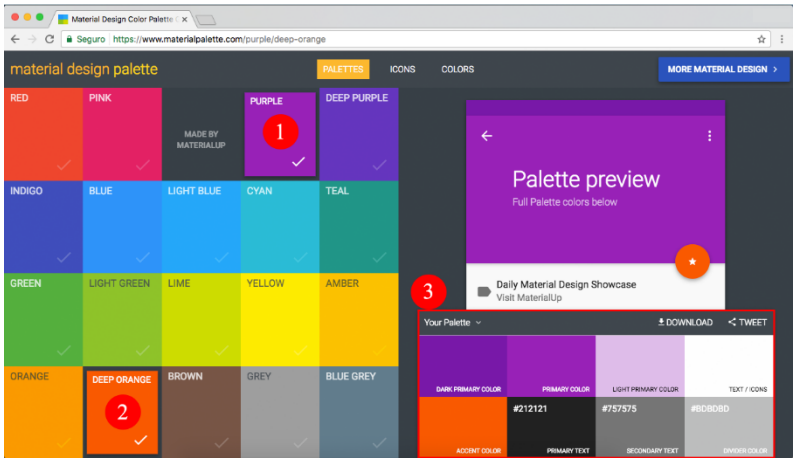
Luego navegamos a ACCESSIBILITY y vemos las métricas y colores sugeridos por esta herramienta que podemos utilizar en nuestra aplicación.

Figura 2.4 Métricas de los colores seleccionados en Color Tool



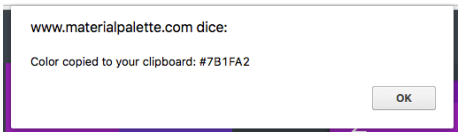
Otra herramienta que podemos utilizar es Material Palette ubicada en https://www.materialpalette.com/. Al dar el primer clic en cualquier color elegimos `colorPrimary`, en el segundo clic elegimos `colorAccent`. Como resultado, a la derecha de la paleta tenemos los colores sugeridos por esta herramienta.

Figura 2.5 Selección de colorPrimary y colorAccent en Material Palette



Cuando hacemos clic en los cuadros de los colores sugeridos, el color se copia en el clipboard del sistema operativo y este puede ser pegado en cualquier lugar.

Figura 2.6 Clic en el Dark Primary Color

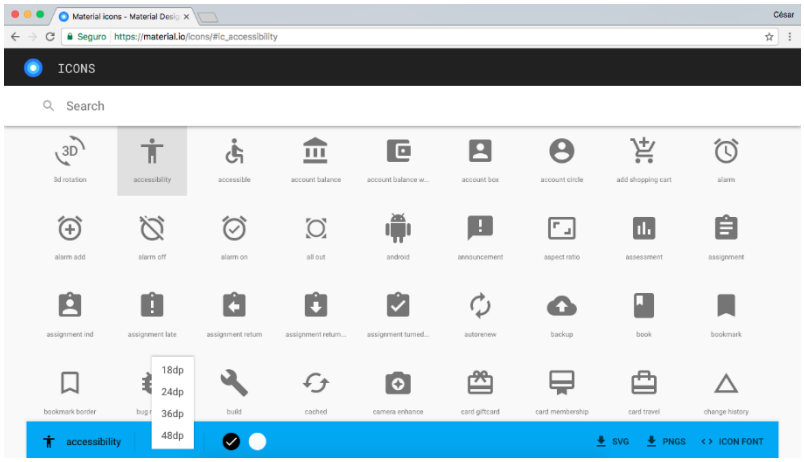


Íconos

El sistema operativo Android es ejecutado en una variedad muy amplia de dispositivos que ofrecen diferentes tamaños y densidades. Cuando se establece un ícono en una aplicación, el sistema operativo escalará el ícono de acuerdo al tamaño de la pantalla del dispositivo.

Si la pantalla tiene poca resolución y gran tamaño, la calidad del icono será bastante baja. Para evitar este problema, podemos tener la misma imagen con distintos tamaños, entonces el sistema operativo elegirá la mejor opción para el dispositivo. Material Design proporciona íconos que pueden ser descargados en https://material.io/icons/.

Figura 2.7 Íconos de Material Design



Estos íconos pueden ser utilizados en las plataformas de Android, ios y web. Cada ícono puede ser descargado en diferentes tamaños, en color blanco o negro. Se descarga un archivo comprimido y dentro de este se encuentran los directorios web, ios y Android.

Dentro del directorio Android existen imágenes con distintas densidades de pixeles. Normalmente, incluimos todos los archivos dentro del directorio Android al proyecto. Tenga en cuenta que al duplicar las imágenes varias veces, aumentará el tamaño de la aplicación.

Figura 2.8 Descarga del ícono accessibility de los iconos de Material Design

Los sufijos en los nombres de los directorios dentro de `Android` hacen referencia a la densidad de pixeles de los dispositivos.

Tabla 2.1 Densidad de pixeles de Íconos

mdpi	medium-density screens (160dpi)
hdpi	high-density screens (240dpi)
xhdpi	extra-high-density screens (320dpi)
xxhdpi	extra-extra-high-density screens (320dpi)

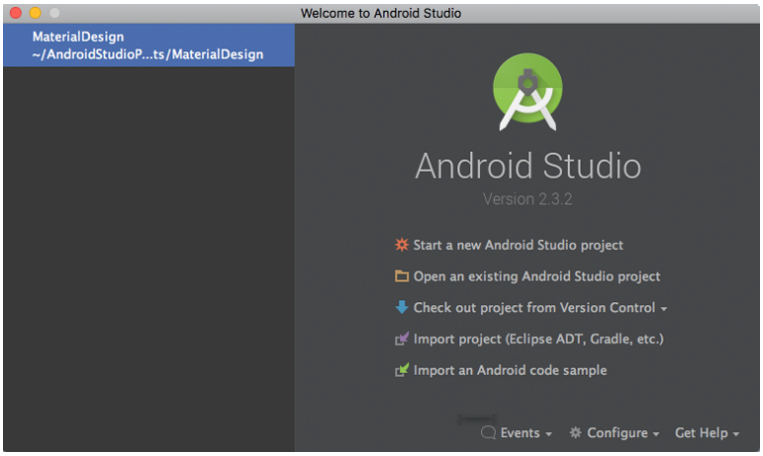
Creando el primer proyecto con Material Design

Para realizar nuestra primera aplicación, utilizaremos algunos componentes de Material Design. Agregaremos un `Toolbar` y un `FloatingActionButton` en la actividad principal de la aplicación.

Para esto, realizaremos los siguientes pasos:

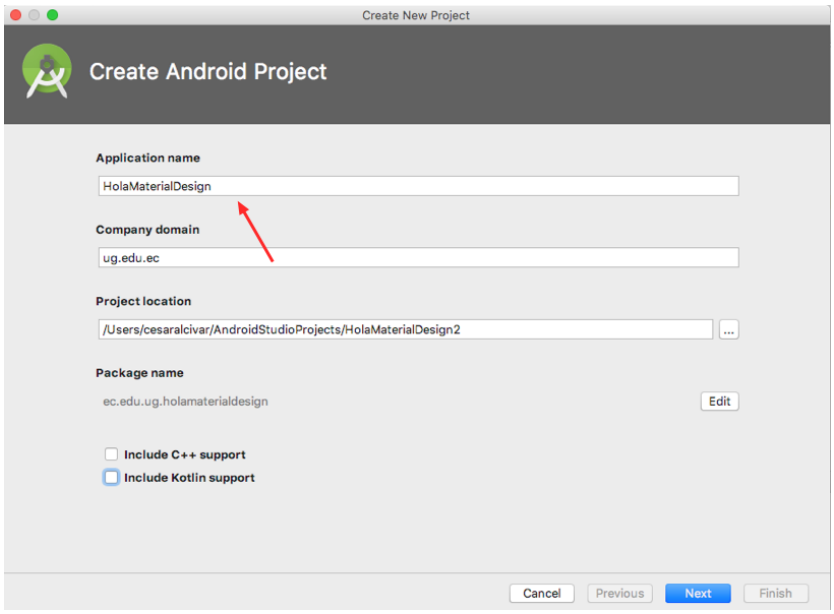
1. Abrimos Android Studio. Luego, clic en "Start a new Android Studio project".

Figura 2.9 Pantalla de Creación de HolaMaterialDesign



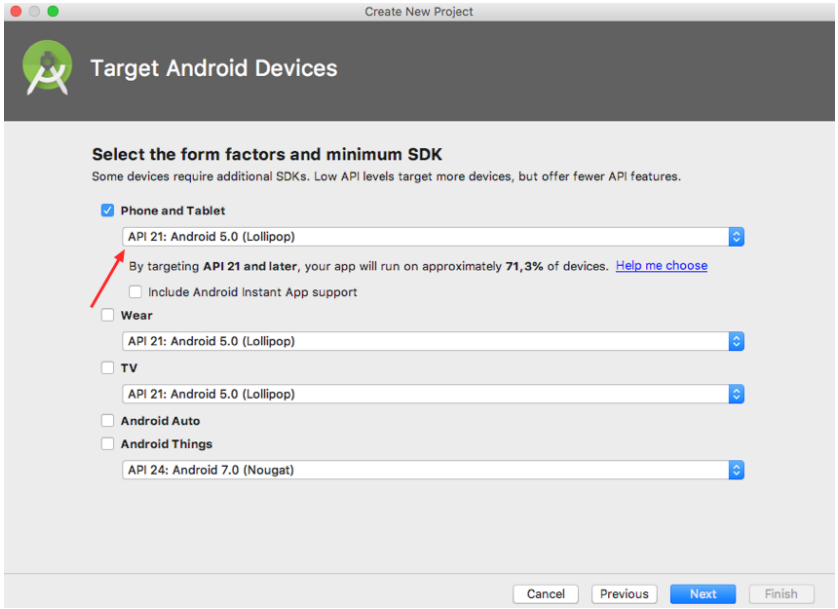
2. Escribimos el nombre de la aplicación "HolaMaterialDesign". Clic en "Next".

Figura 2.10 Pantalla para asignar el nombre al proyecto HolaMaterialDesign



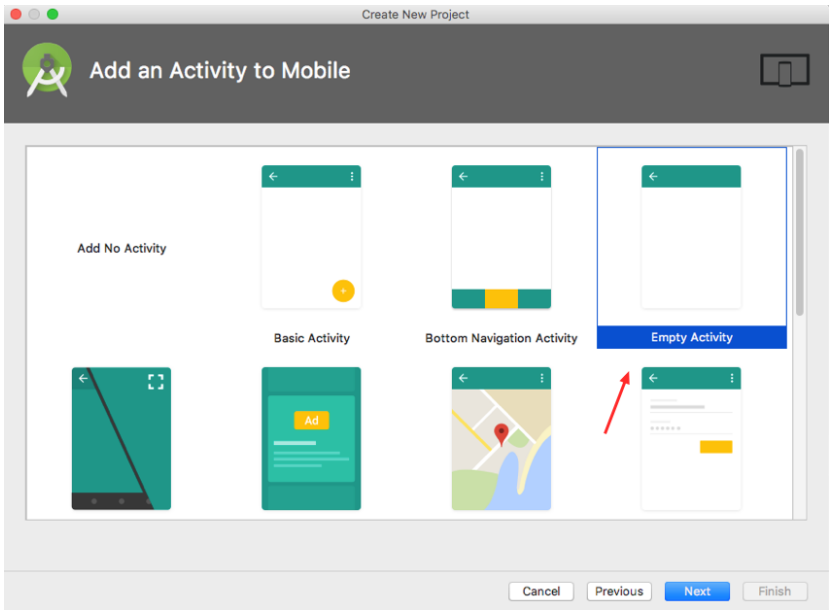
3. Seleccionamos el API 21 o Android 4.0 (Lollipop) para nuestra primera aplicación.

Figura 2.11 Pantalla para elección de la versión mínima de HolaMaterialDesign



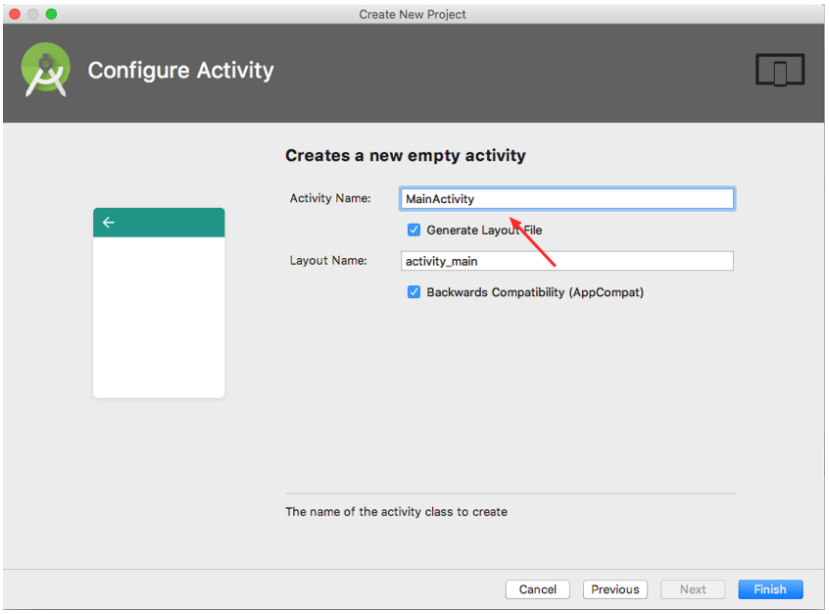
4. Se muestra las plantillas disponibles en Android Studio. Elegimos “Empty Activity”. Clic en “Next”.

Figura 2.12 Elección de la plantilla de HolaMaterialDesign



5. Establecemos el nombre de la actividad principal "MainActivity". Luego, clic en "Finish".

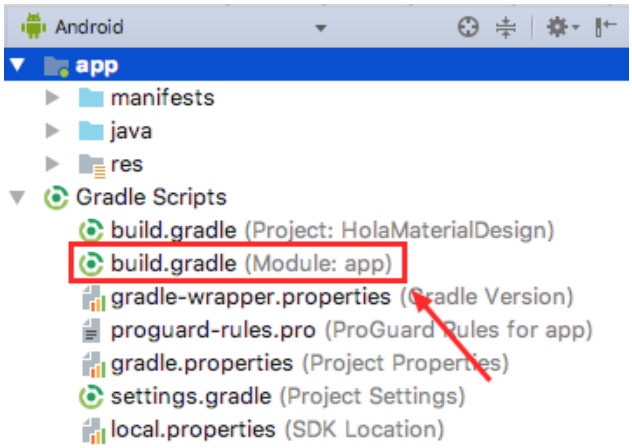
Figura 2.13 Elección del nombre de la Actividad principal en HolaMaterialDesign



Es buena práctica poner el sufijo `Activity` en el nombre de una actividad. Para el nombre del `layout` correspondiente a la actividad, se revierte el nombre: se lo pone en minúsculas y cada palabra es separada por un subguión.

6. Vamos al archivo `build.gradle`.

Figura 2.14 build.gradle de HolaMaterialDesign

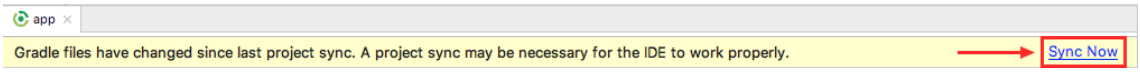


7. Antes de agregar un `Appbar` y un `FloatingActionButton` necesitamos añadir la librería Design Support Library en el archivo `build.gradle (Module: app),` como se muestra a continuación:

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support.constraint:constraint-
layout:1.0.2'
    implementation
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.1'
    androidTestImplementation
'com.android.support.test.espresso:espresso-core:3.0.1'
}
```

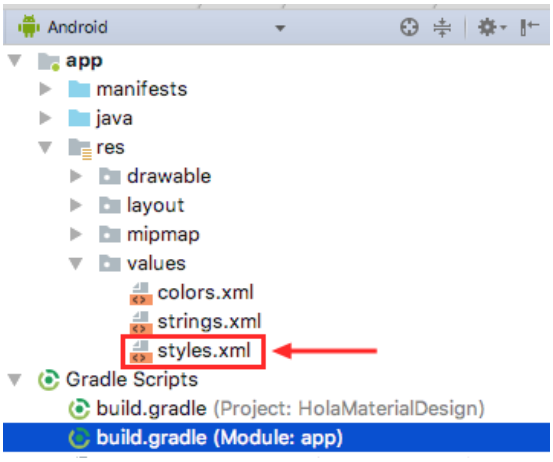
8. Damos clic en "Sync Now" para sincronizar el proyecto con las librerías recientemente agregadas al proyecto.

Figura 2.15 Sincronizar Design Support Library con el proyecto



9. Vamos al archivo styles.xml.

Figura 2.16 style.xml de HolaMaterialDesign



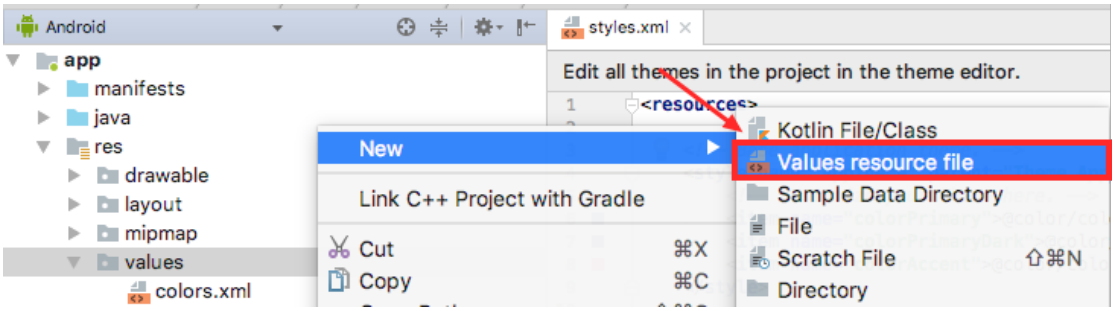
10. Cambiamos el tema de Theme.AppCompat.Light.DarkActionBar a Theme.AppCompat.Light.NoActionBar, como se muestra a continuación:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

El estilo `DarkActionBar` preestablece un `ActionBar` en todas las actividades dentro de un proyecto. Este elemento es poco flexible y no agrega todas las características de Material Design. Al reemplazarlo por el estilo `NoActionBar` ninguna actividad tendrá un `ActionBar` y en su lugar podemos reemplazarlo con un `AppBar`.

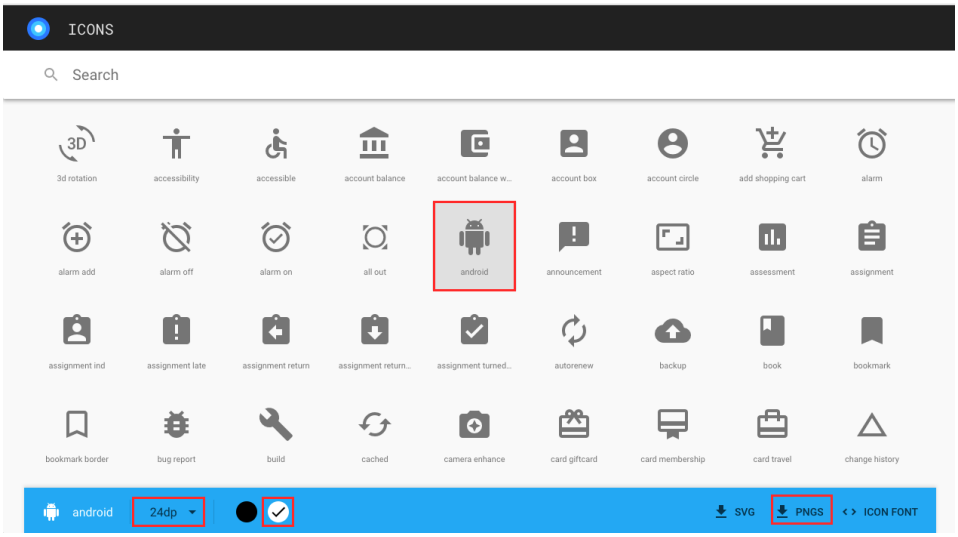
11. Damos clic secundario en values > New > Values resource file.

Figura 2.17 Creando el nuevo recurso para las dimensiones



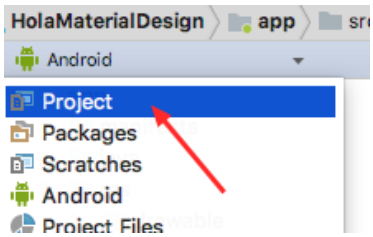
12. Ir a https://mater (Material Design) (Material Design, s.f.)ial.io/icons/ para descargar un ícono para el `FloatingActionButton`. Elija cualquier ícono de color blanco, 24dp y descárguelo como PNG.

Figura 2.18 Íconos para HolaMaterialDesign



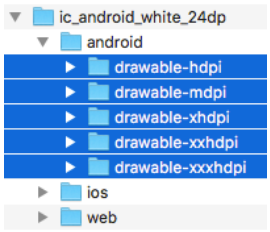
13. Cambiamos la vista de la aplicación a “Project”.

Figura 2.19 Cambio a la vista Project



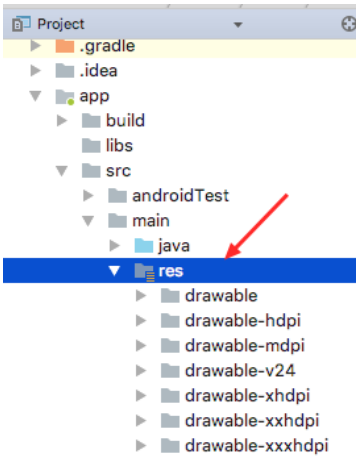
14. Copiamos las imágenes que se han descargado.

Figura 2.20 Íconos descargados para HolaMaterialDesign



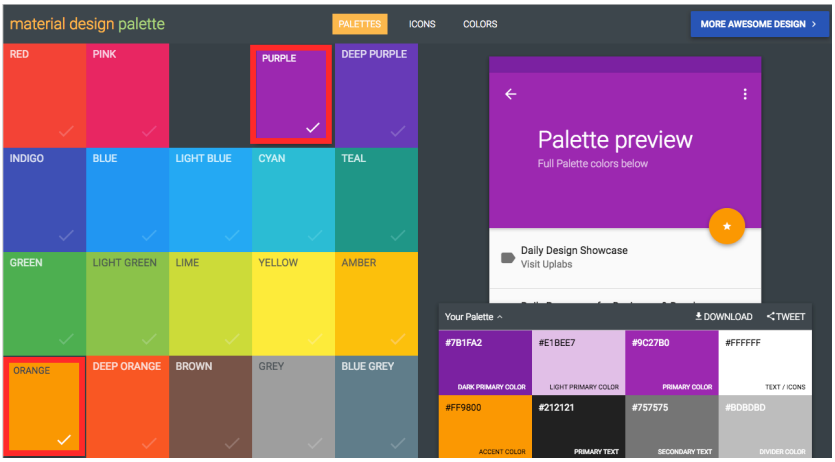
15. Y pegamos los íconos en la carpeta res.

Figura 2.21 Íconos copiados a HolaMaterialDesign



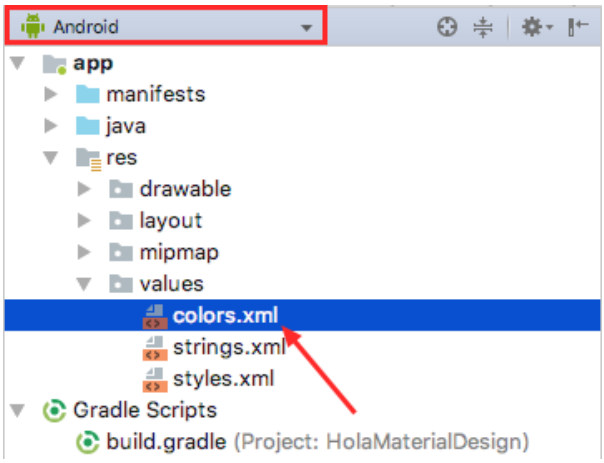
16. Ir a https://www.materialpalette.com/, hacemos el primer clic en PURPLE y el segundo en ORANGE.

Figura 2.22 Elección de colores para HolaMaterialDesign



17. Retornamos a la vista “Android”. Vamos al archivo “colors.xml”.

Figura 2.23 Archivo colors.xml de HolaMaterialDesign

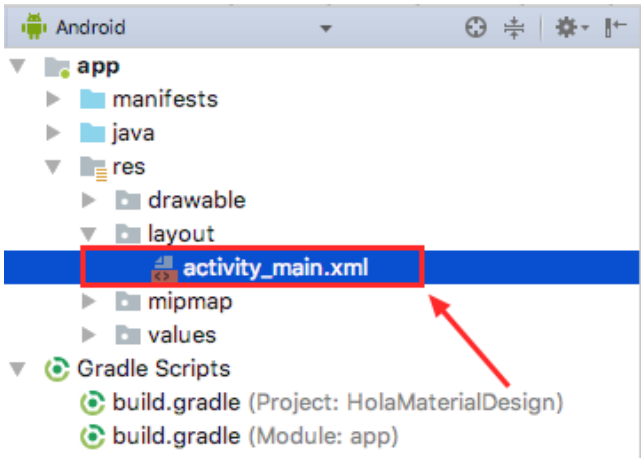


18. Copiamos los colores del sitio materialpallete en `res/values/colors.xml` como se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#9c27b0</color>
    <color name="colorPrimaryDark">#7b1fa2</color>
    <color name="colorAccent">#ff9800</color>
</resources>
```

Como mencionarnos anteriormente, `colorPrimary` establece el color por defecto de algunos componentes, entre ellos el `Appbar`. `colorPrimaryDark` establece el color del status bar, y los widgets que requieren acentuación tomarán el color de `colorAccent`. Vamos al archivo `activity_main.xml`.

Figura 2.24 Archivo activity_main.xml de HolaMaterialDesign



19. En `activity_main.xml`, agregamos los siguientes elementos:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            app:title="Título del Appbar"
            app:titleTextColor="@android:color/white"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />

    </android.support.design.widget.AppBarLayout>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/FAB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|bottom"
        android:layout_margin="16dp"
        android:src="@drawable/ic_android_white_24dp"/>

</android.support.design.widget.CoordinatorLayout>
```

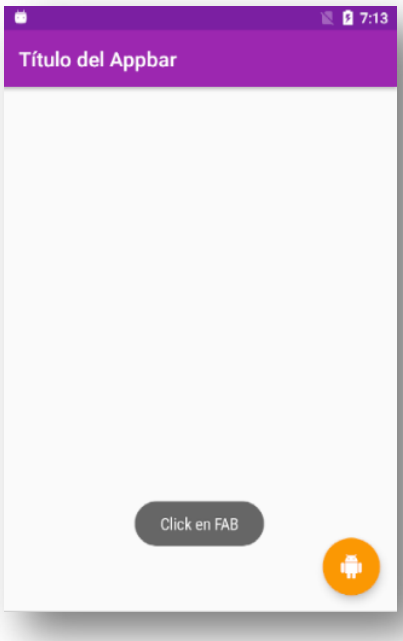
20. En la clase MainActivity establecemos el evento Clic del `FloatingActionButton`, como se muestra a continuación.

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        FloatingActionButton fab = findViewById(R.id.FAB);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this,"Click en
FAB",Toast.LENGTH_SHORT).show();
            }
        });

    }
}
```

21. Para finalizar, ejecutamos la aplicación y el resultado se muestra a continuación:

Figura 2.25 Ejecución final de HolaMaterialDesign



Resumen

En este capítulo se explicaron conceptos básicos de Material Design. Vimos los fundamentos de diseño que necesitan conocer los desarrolladores de software para

avanzar sin dificultades a través de este libro. Se proporcionó información de algunas herramientas que están disponibles en Internet para definir colores e íconos. Finalmente, creamos nuestro primer proyecto utilizando componentes de Material Design.

Capítulo 3. App Bar

El Appbar es un tipo especial de barra de herramientas que se ubica en la parte superior de la aplicación. Proporciona una estructura visual y elementos conocidos por los usuarios, mejorando la experiencia de uso. Entre sus funciones está la de proporcionar navegación entre las actividades o fragmentos, o acceso a las principales acciones de la aplicación. En este capítulo, veremos las métricas del Appbar, y cómo llevar los comportamientos de este elemento al código fuente.

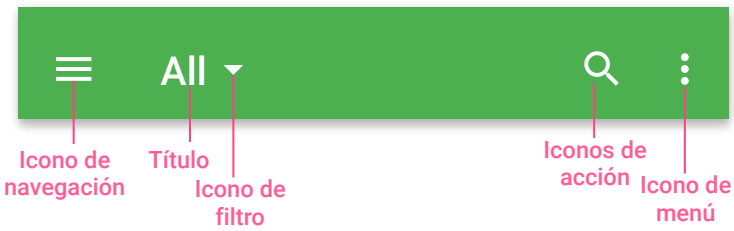
Se cubrirán los siguientes tópicos:

- Guías para el Appbar
- Appbar estándar
- Appbar con espacio flexible
- Appbar con espacio flexible e imagen
- Appbar con espacio flexible y contenido sobrepuesto

Diseño estándar App Bar

Las guías de Material Design para el Appbar sugieren que en su lado izquierdo puede tener un botón de navegación, una flecha hacia atrás, u omitir cualquier ícono cuando no se requiera. Tendrá un título que puede mostrar el nombre de la aplicación, descripción de una página, o un filtro que se haya realizado en la aplicación y puede tener un color distinto de los íconos si se necesitare un mayor realce visual. En el lado derecho se mostrarán los íconos que reflejan las principales acciones de la aplicación. El ícono menú contiene acciones secundarias como ayuda, configuración, o retroalimentación, entre otras. Todos los íconos del Appbar deberían tener el mismo color.

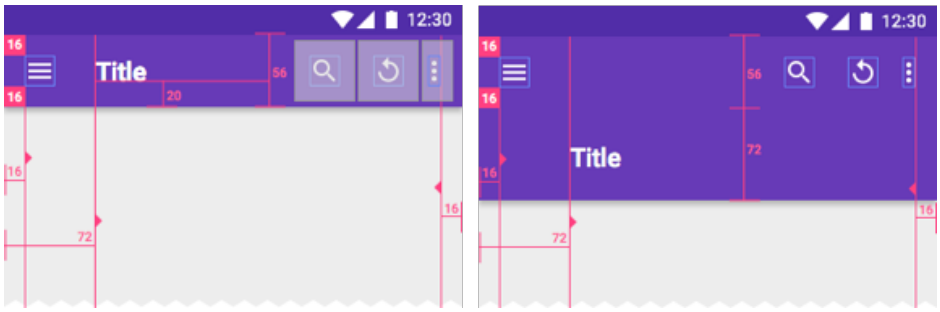
Figura 3.1 Elementos del Appbar



Las alturas por defecto de un Appbar son: 48dp en Landscape y 56dp en Portrait. Además, existen otras medidas que se muestran a continuación:

- *Appbar height*: 56dp
- *Appbar left and right padding*: 16dp
- *Appbar icon top, bottom, left padding*: 16dp
- *Appbar title left padding*: 72dp
- *Appbar title bottom padding*: 20dp

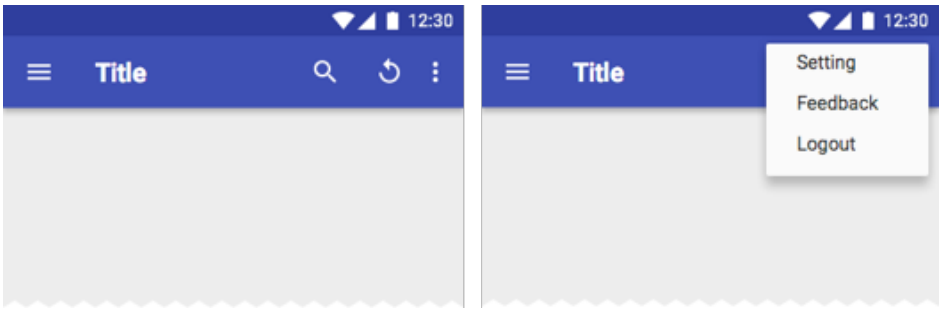
Figura 3.2 Métricas por defecto de una App Bar



Menús de un App Bar

El menú es un componente común de la interfaz de usuario. Cuando se toca el ícono de menú de la Appbar, aparece una vista con las acciones secundarias de la aplicación.

Figura 3.3 Menú de una Barra de App



Barra de Estados (Status Bar)

En Android, la barra de estados contiene íconos de notificación e íconos del sistema. La altura es de 24dp.

Figura 3.4 Barra de Estado

Estándar del App Bar

Cuando implementamos un Appbar en Android, este está compuesto por un `AppbarLayout` y un `Toolbar`. Cuando se implementa el comportamiento estándar de un

Appbar, este se contrae y se expande cuando se desplaza el contenido de la pantalla en dichas direcciones.

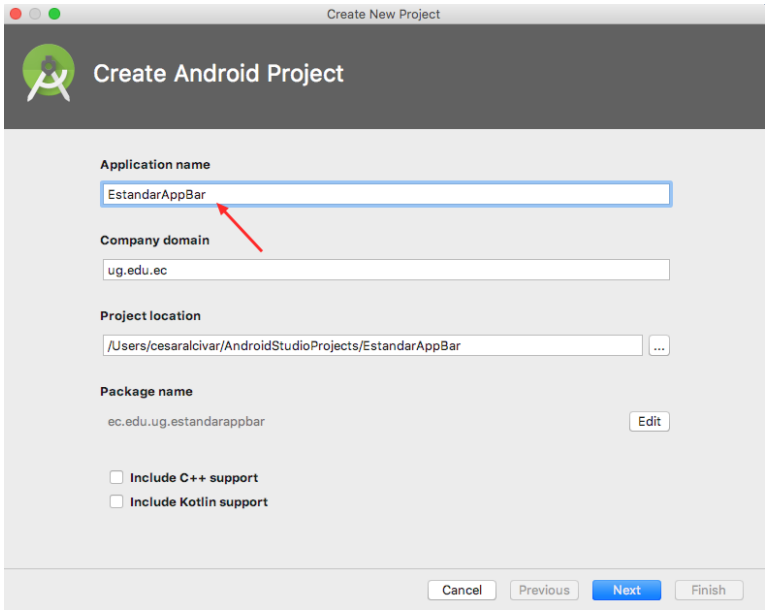
Para desarrollar el comportamiento estándar del App Bar, implementaremos una lista de noticias alineada a las métricas de diseño de Material Design.

Creación y configuración del Proyecto estándar App Bar

Para desarrollar el comportamiento estándar del Appbar, realizaremos los siguientes pasos:

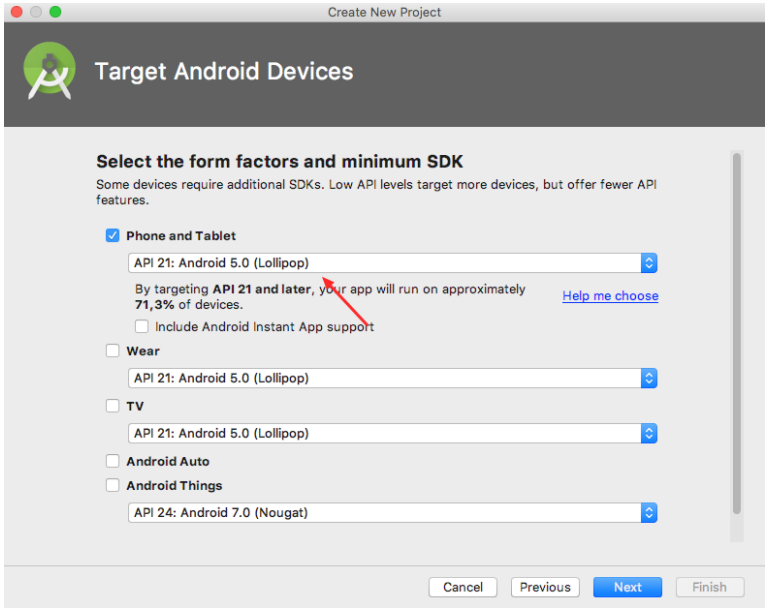
1. Abrimos Android Studio. Establecemos el nombre del proyecto "EstandarAppBar". Clic en "Next".

Figura 3.5 Estableciendo el nombre EstándarAppBar al proyecto



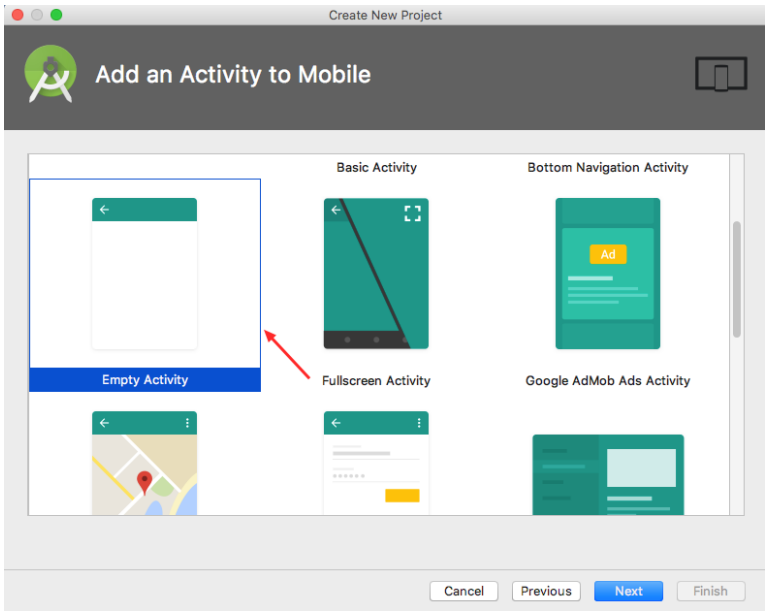
2. Seleccionamos el SDK mínimo, el API 21: Android 5.0. Clic en "Next".

Figura 3.6 SDK mínimo para EstandarAppBar



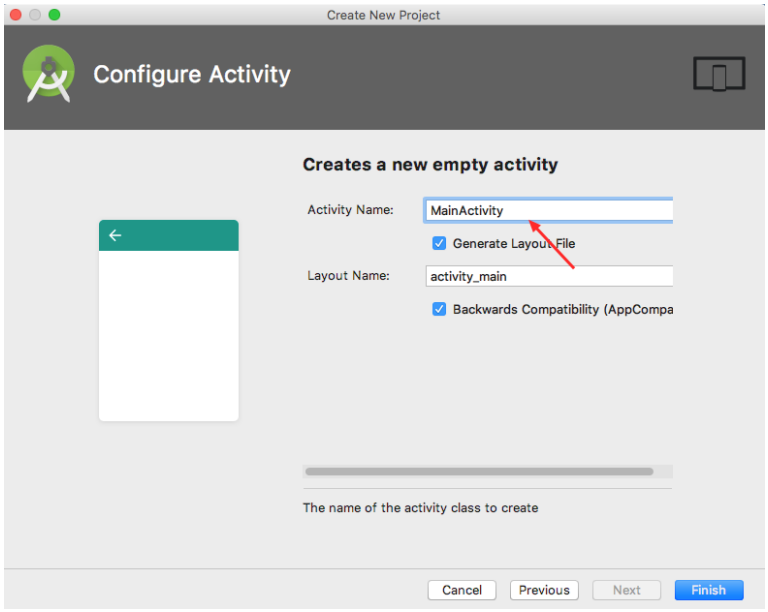
3. Elegimos la plantilla del proyecto. Para nuestro caso `Empty Activity`. Clic en "Next".

Figura 3.7 Seleccionamos la plantilla Empty Activity



4. Creamos la `Activity` y el layout principal del proyecto. Clic en "Finish".

Figura 3.8 Asignamos la Activity principal del proyecto



5. Vamos a `res/values/styles.xml`. Cambiamos el tema de `DarkActionBar` a `NoActionBar`, como se muestra a continuación:

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
</resources>
```

6. En `res/values/colors.xml`, actualizamos los colores `colorPrimary` y `colorPrimaryDark` y agregamos `textColorPrimary`, `textColorSecundary` y `colorDivider`.

```
<resources>
    <color name="colorPrimary">#FF9800</color>
    <color name="colorPrimaryDark">#F57C00</color>
    <color name="textColorPrimary">#212121</color>
    <color name="textColorSecundary">#757575</color>
    <color name="colorDivider">#BDBDBD</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

7. Agregamos los siguientes recursos `string` en `res/values/strings.xml`.

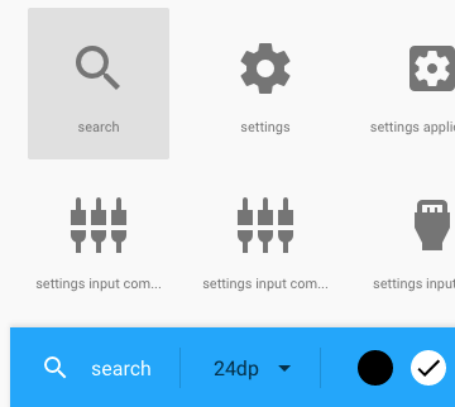
```
<string name="noticas">Noticias</string>
<string name="buscar">Buscar</string>
```

8. En `Gradle Scripts/build.gradle (Module:app)`, agregamos la librería de compatibilidad Design.

```
…
implementation 'com.android.support:appcompat-v7:27.1.1'
implementation 'com.android.support.constraint:constraint-
layout:1.1.0'
implementation 'com.android.support:design:27.1.1'
testImplementation 'junit:junit:4.12'
testCompile 'junit:junit:4.12'
…
```

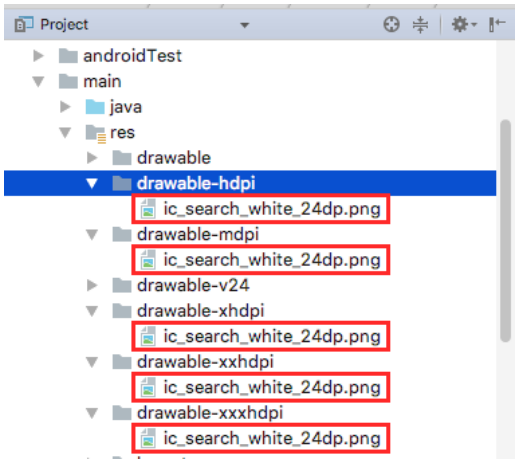
9. Vamos a https://material.io/icons/#ic_search y descargamos el ícono search de 24dp color blanco.

Figura 3.9 Ícono search 24dp, color blanco



10. Cambiamos a la vista "Project" y pegamos las carpetas de íconos dentro de `res/`.

Figura 3.10 Pegamos el ícono search en el proyecto



Layouts del Estándar Appbar

En esta sección crearemos los widgets necesarios para establecer el comportamiento estándar del Appbar. En `activity_main` agregamos los siguientes elementos:

```
<android.support.design.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">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            app:title="@string/noticas"
            android:layout_width="match_parent"
            android:background="@color/colorPrimaryNoticia"
            android:layout_height="?attr/actionBarSize"
            android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBa
r"
            app:layout_scrollFlags="scroll|enterAlways" />

    </android.support.design.widget.AppBarLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"/>

</android.support.design.widget.CoordinatorLayout>
```

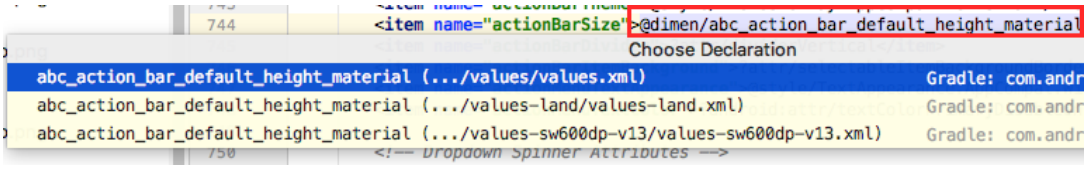
Asignamos recursos `id` para los widgets `Toolbar` y `RecyclerView`. Esto nos permitirá inflar estos objetos en la clase `MainActivity`.

Toolbar

Observe que el atributo `android:layout_height` del `Toolbar` tiene un valor que empieza con "`?attr…`". Estos valores son tomados del tema principal de la aplicación, en nuestro

caso, el tema es `"Theme.AppCompat.Light.NoActionBar"`. Si navegamos a través de este tema (Ctrl+clic en el tema), hasta el atributo `actionBarSize`, notamos que `@dimen/abc_action_bar_default_height_material` hace referencia a dos valores, en `Portrait` a `values.xml` y `Landscape` a `values-land.xml`.

Figura 3.11 Valores de actionBarSize



Si navegamos hacia los valores, observamos que el valor `Portrait` es 56dp y el valor `Landscape` es 48dp, justamente los valores que recomienda Material Design para la altura del Appbar. Notemos que algunos valores por defecto de Material Design ya están implementados en Android.

Existen otros atributos importantes en el `Toolbar` que nos permiten entender su forma y su comportamiento. El atributo `app:layout_scrollFlags="scroll|enterAlways"` hace que este se vaya ocultando o apareciendo cada vez que se deslice el contenido de la pantalla.

El atributo `android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"` establecerá el color blanco en el título del `Toolbar`, en la flecha de navegación y en el texto de búsqueda del `SearchView` que utilizaremos más adelante.

RecyclerView

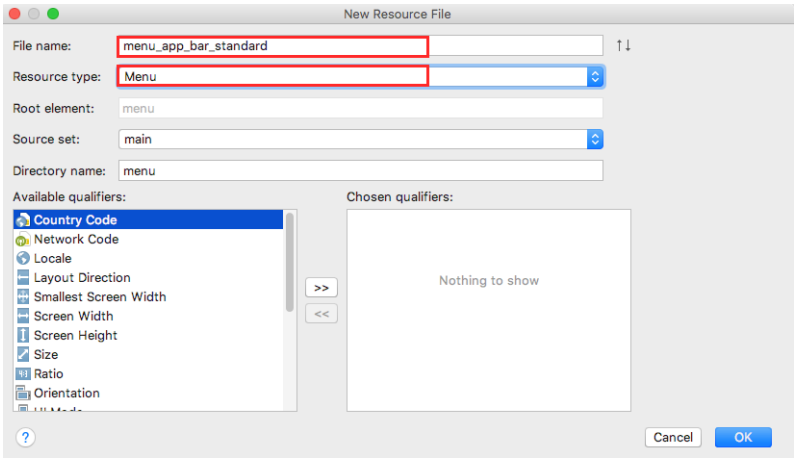
Cuando el contenedor padre es un `CoordinateLayout`, y se establece el atributo `app:layout_scrollFlags` en el Toolbar, el `RecyclerView` necesita tener el atributo `app:layout_behavior="@string/appbar_scrolling_view_behavior"` para que no se superponga con el `Toolbar`. Con estos ajustes, obtendremos el comportamiento estándar del Appbar.

Menú con SearchView

La aplicación va a tener la opción de filtrar las noticias por su contenido. La mejor forma de realizar esta acción es a través de un `SearchView` en el Appbar.

Para crear un `SearchView`, damos clic secundario en la carpeta `res`, luego en el menú emergente navegamos hasta "New>Android Resource File". En "Directory name", elegimos "menu". Establecemos el nombre de "menu_app_bar_standard".

Figura 3.12 Creando el menú menu_app_bar_standard



Una vez creado el menú, agregamos los siguientes elementos:

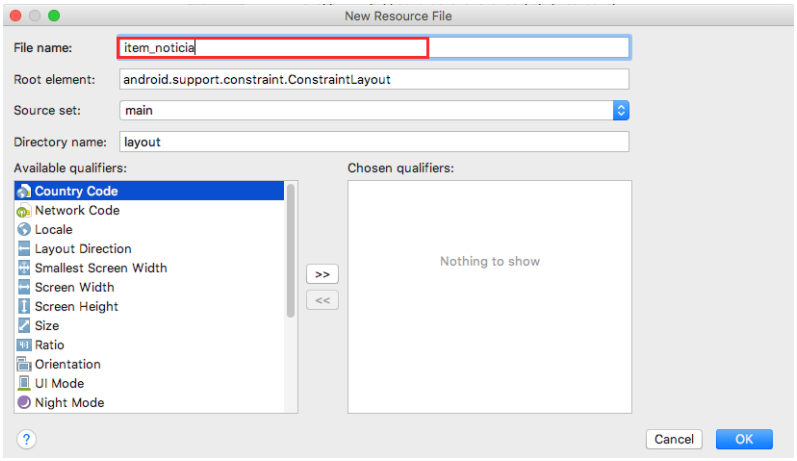
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_search"
        android:icon="@drawable/ic_search"
        android:title="@string/buscar"
        app:actionViewClass="android.support.v7.widget.SearchView"
        app:showAsAction="ifRoom|collapseActionView"/>

</menu>
```

El atributo `app:actionViewClass="android.support.v7.widget.SearchView"` hace que el `MenuItem` aparezca como un botón de búsqueda en el Appbar. Cuando el usuario presiona ese botón, se mostrará una caja de texto que servirá para escribir el contenido que se desea filtrar.

Ítem de la Lista

Para crear el ítem o fila de la lista de noticias, damos clic secundario en "res/layout", luego navegamos hasta "New>Android Resource File" y creamos un layout llamado "item_noticia".

Figura 3.13 Creando el ítem para la lista de noticias



Una vez creado el layout, agregamos los siguientes elementos:

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

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="16dp"
        android:paddingBottom="20dp"
        android:paddingLeft="16dp"
        android:paddingRight="16dp"
        android:orientation="vertical">
        <TextView
            android:id="@+id/txt_titulo"
            android:maxLines="1"
            android:textColor="@color/textColorPrimary"
            android:textSize="16dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <TextView
            android:id="@+id/txt_contenido"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:maxLines="2"
            android:textColor="@color/textColorSecundary"
            android:textSize="14dp" />

    </LinearLayout>
    <View
        android:background="@color/colorDivider"
        android:layout_width="match_parent"
```

```
        android:layout_height="1dp" />

</LinearLayout>
```

Cada fila de la lista de noticias utilizará el layout `item_noticia`. Para cada ítem de la lista, se establecieron las métricas de Material Design de la sección Métricas de Material Design para una lista de tres líneas. El `paddingTop`, `paddingLeft` y `paddingRight` de 16dp. El `paddingBottom` se establece en 20dp. El `txt_titulo` es el `Primary Text`, con un tamaño de 16sp. `txt_contenido` es el `Secondary Text`, con un tamaño de 14sp.

Las Clases de Dominio

La noticia tendrá un título y un contenido. Dentro del proyecto creamos una carpeta llamada `dominio` y allí agregamos una clase llamada `Noticia` con los atributos `titulo` y `contenido` con sus respectivos métodos `getters` y `setters`.

```
package deldisenoalcodigo.estandarappbar.dominio;

public class Noticia {
    private String titulo;
    private String contenido;

    public Noticia(String titulo, String contenido) {
        this.titulo = titulo;
        this.contenido = contenido;
    }

    // getters setters…
}
```

La clase `Noticia` tiene un constructor que recibe dos parámetros de entrada de tipo `String`. El primero es el `titulo` y el segundo es el `contenido`.

Clase Singleton

Para proveer de noticias a la aplicación utilizaremos el patrón `singleton`. Una clase `singleton` permite crear una sola instancia de sí misma. Esta instancia existirá mientras la aplicación se esté ejecutando y mantendrá disponibles una lista de objetos `Noticia` a través de cualquier cambio del ciclo de vida de una `Activity` o `Fragment`.

Para crear un `singleton`, la clase debe tener un constructor privado y un método `get()`. Si el objeto existe, entonces el método retorna a la instancia. Si la instancia no existe, entonces en el método `get()` se llamará al constructor para crearla.

Crearemos una clase singleton en la carpeta `dominios` llamada `NoticiaSingleton`. Agregamos el siguiente código:

```
package deldisenoalcodigo.estandarappbar.dominio;

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

public class NoticiaSingleton {

    private static NoticiaSingleton noticiaSingleton;
    private List<Noticia> noticias = new ArrayList<>();

    private NoticiaSingleton(){
        noticias = new ArrayList<>();
        noticias.add(new Noticia("Estos son los mejores juegos para
Android", "En la Google Play Store existen unos cuantos juegos
excelentes para la venta. Echa un vistazo a la lista de los
mejores"));
        noticias.add(new Noticia("¿Quieres editar tweets? Pronto nueva
habilidad en Fenix", "La posibilidad de editar tweets puede que nunca
llegue oficialmente a Twitter, pero el cliente de Fenix para Twitter
está trabajando actualmente en modificar los tweets."));
        noticias.add(new Noticia("Pronto podrá compartir cualquier
tipo de archivo en WhatsApp", "Los usuarios de WhatsApp ahora pueden
compartir cualquier tipo de archivo que quieran incluyendo APKs."));
        noticias.add(new Noticia("Estos son los 56 nuevos emoji para
Android", "Los emojis nuevos incluyen un vampiro, un zombi,
dinosaurios, y un manojo de otros animales, criaturas y alimentos."));
        noticias.add(new Noticia("La aplicación móvil de YouTube
pronto adaptará los videos para llenar la pantalla", "YouTube hizo
algunos anuncios bastante grandes hoy, incluyendo la palabra que su
aplicación móvil pronto permitirá a los usuarios."));
    }
    public static NoticiaSingleton get(){
        if(noticiaSingleton==null)
            noticiaSingleton = new NoticiaSingleton();
        return noticiaSingleton;
    }
    public List<Noticia> getNoticias(){
        return noticias;
    }
}
```

Actividades

Para presentar elementos en una lista, podemos utilizar un `ListView` o un `RecyclerView`. La diferencia entre una y otra es que el `ListView` mantiene todos los elementos en memoria y el `RecyclerView` mantiene en memoria solo los elementos que se muestran en la pantalla.

En el método `onCreate de MainActivity`, inflamos el `RecyclerView` de `activity_main`. Luego, le asignamos el `LayoutManager LinearLayoutManager`. Un `LayoutManager` es el responsable de la forma en que se posicionan los elementos dentro del `RecyclerView`. `LinearLayout` hace que los elementos se muestren de forma lineal, uno debajo del otro.

```
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    RecyclerView recyclerView = (RecyclerView)
findViewById(R.id.recycler_view);
    recyclerView.setLayoutManager(new
LinearLayoutManager(MainActivity.this));

   }
…
}
```

Para utilizar el recurso menú `menu_app_bar_standard` en el Appbar, debemos hacer que el `toolbar` de `activity_main` se comporte como un `Action Bar`. Para hacer esto, inflamos el `toolbar` y pasamos su referencia al método `setSupportActionBar`.

```
class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
    …
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
}
…
}
```

RecyclerView, Adapter y ViewHolder

Un `RecyclerView` requiere de una subclase `Adapter` y una subclase `ViewHolder`. Un `ViewHolder` representa un elemento en la lista y el `Adapter` crea los elementos `ViewHolder` para el RecyclerView. Empezaremos definiendo el `ViewHolder` como una clase interna de `MainActivity`.

```
public class NoticiasActivity extends AppCompatActivity {
    …
    private class NoticiaHolder extends RecyclerView.ViewHolder{
```

```
        private TextView txtTitulo;
        private TextView txtContenido;

        public NoticiaHolder(View itemView) {
            super(itemView);
            txtTitulo = (TextView)
itemView.findViewById(R.id.txt_titulo);
            txtContenido = (TextView)
itemView.findViewById(R.id.txt_contenido);

        }
        private void enlazar(Noticia noticia){
            txtTitulo.setText(noticia.getTitulo());
            txtContenido.setText(noticia.getContenido());
        }
    }
}
```

El constructor de la clase `NoticiaHolder` recibe del `Adapter` una referencia de inflada de `item_noticia`. Allí, se utilizan los dos atributos de clases: `txtTitulo` y `txtContenido`.

El `RecyclerView` se comunicará con el `Adapter` cuando este necesite crear un `ViewHolder` o conectarse con un objeto `Noticia`. Una vez definido el `ViewHolder`, creamos el `Adapter`.

```
public class MainActivity extends AppCompatActivity {
    …
    private class NoticiaAdapter extends
RecyclerView.Adapter<NoticiaHolder>{

        private List<Noticia> noticias;

        public NoticiaAdapter(List<Noticia> noticias){
            this.noticias = noticias;
        }

        public void setNoticias(List<Noticia> noticias){
            this.noticias = noticias;
            this.notifyDataSetChanged();
        }

        @Override
        public NoticiaHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
            LayoutInflater
layoutInflater=LayoutInflater.from(MainActivity.this);
            View
view=layoutInflater.inflate(R.layout.item_noticia,parent,false);
            return new NoticiaHolder(view);
        }
```

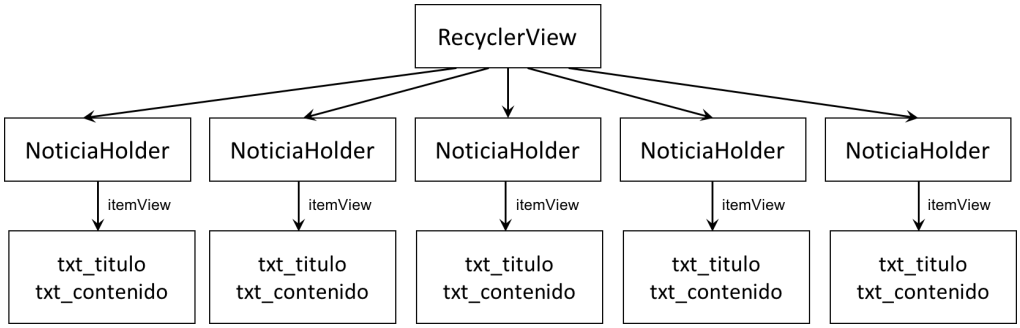
```
        @Override
        public void onBindViewHolder(NoticiaHolder holder, int
position) {
            Noticia noticia = noticias.get(position);
            holder.enlazar(noticia);
        }

        @Override
        public int getItemCount() {
            return noticias.size();
        }
    }
}
```

La clase `NoticiaAdapter` tiene un constructor que recibe una lista de noticias. Luego, se iguala la variable de instancia `noticias`, de la misma clase. Esta variable tendrá todas las noticias que se mostrarán en el `RecyclerView`.

`onCreateViewHolder` es llamado por el `RecyclerView` cuando este necesita mostrar un nuevo ítem en la pantalla. Se infla el layout `item_noticia` en la variable `view`. A continuación, esta variable es pasada al `ViewHolder` para instanciar a `txt_titulo` y `txt_contenido`.

Recordemos que el `RecyclerView` crea sus ítems mientras van apareciendo en la pantalla. La variable `position` del método `onBindViewHolder` contiene la posición del ítem que debe mostrarse en el `RecyclerView`. Es por eso que se obtiene un objeto noticia en esa posición y después es enviada al `ViewHolder` para llenar los `TextView` con el contenido de sus variables de instancia. A continuación, se muestra una vista de alto nivel de la relación entre `RecyclerView` y el `ViewHolder` de noticias:

Figura 3.14 RecyclerView de noticias

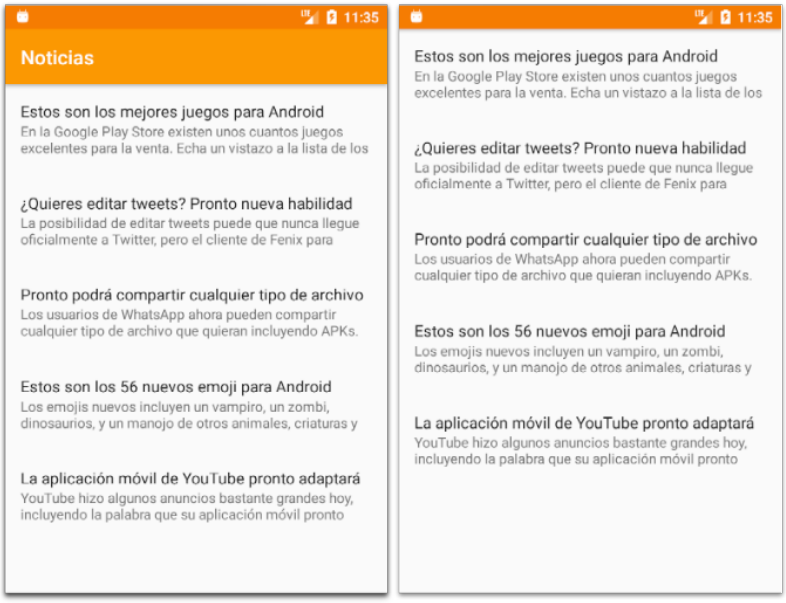


Para conectar el `RecyclerView` con el `Adapter`, creamos una variable de tipo `NoticiaAdapter` llamada `noticiaAdapter`, y por su constructor le enviamos una lista de noticias. Para finalizar, en el método `onCreate` establecemos el `Adapter` en el objeto `recyclerView`.

```
public class MainActivity extends AppCompatActivity {

    NoticiaAdapter noticiaAdapter = new
NoticiaAdapter(NoticiaSingleton.get().getNoticias());

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_noticias);
        …
        recyclerView.setAdapter(noticiaAdapter);
        …
        }
    }
    …
}
```

Ejecutamos la aplicación, observamos el comportamiento estándar del Appbar con un `RecyclerView`.

Figura 3.15 Estándar App Bar



Filtro de noticias

Antes de finalizar, agregaremos la funcionalidad de filtrar las noticias por su contenido. Sobrescribiremos el método `onCreateOptionsMenu`, el cual nos permite establecer un menú en el Appbar de la actividad y escribir el código para filtrar las noticias.

```
public class MainActivity extends AppCompatActivity {
    …
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_app_bar_standard,menu);

        MenuItem menuItem = menu.findItem(R.id.action_search);
        final SearchView searchView = (SearchView)
menuItem.getActionView();

        searchView.setOnQueryTextListener(new
SearchView.OnQueryTextListener() {
            @Override
            public boolean onQueryTextSubmit(final String query) {
                return false;
            }

            @Override
            public boolean onQueryTextChange(String newText) {
                List<Noticia> result = new ArrayList<>();
                for(Noticia
noticia:NoticiaSingleton.get().getNoticias()){
                    if(noticia.getContenido().toLowerCase().contains(n
```

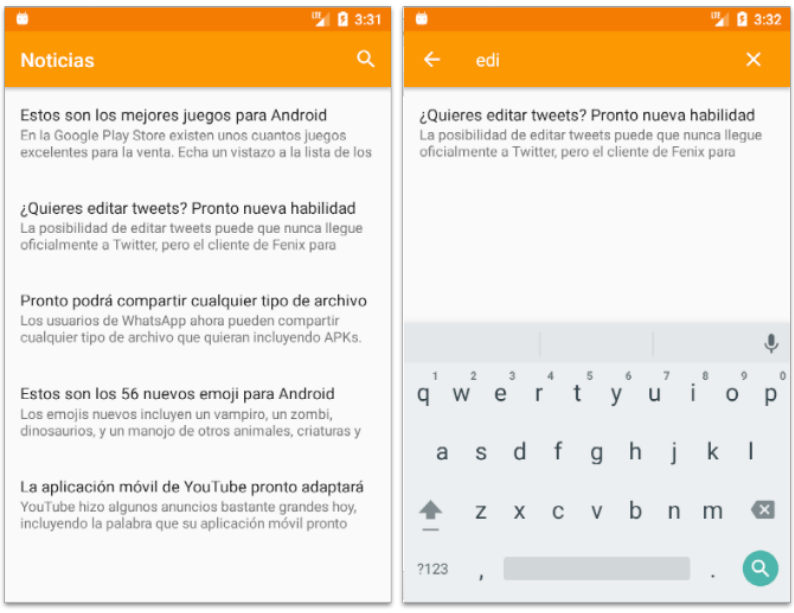
```
ewText.toLowerCase()))
                        result.add(noticia);
                }

                noticiaAdapter.setNoticias(result);
                return true;
            }
        });

        return super.onCreateOptionsMenu(menu);
    }
    …
}
```

Inflamos `menu_app_bar_standard`. Obtenemos el `MenuItem action_search` y realizamos un casting a `SearchView`. Luego, le asignamos el evento `setOnQueryTextListener`.

Dentro de `setOnQueryTextListener`, el método `onQueryTextChange`, se dispara cada vez que se escribe en el texto del `SearchView`. `result` almacena objetos noticia que coincidan con el texto del filtro y su variable contenido.

Una vez terminado este comportamiento, ejecutamos la aplicación. Se mostrará el siguiente resultado:

Figura 3.16 Resultado final del estándar App Bar



App Bar con Tabs

Cuando se implementa un App Bar con Tabs, se muestran pestañas o tabs de forma horizontal en la parte superior de la aplicación. Cada vez que hacemos clic en alguna opción, se cambia el contenido de la aplicación.

En esta sección, desarrollaremos una aplicación que muestre una lista de mascotas separadas por especies.

Creación y configuración del Proyecto App Bar con tabs

Para desarrollar el App Bar con Tabs, realizaremos los siguientes pasos:

1. Abrimos Android Studio. Creamos el proyecto "AppBarConTabs".
2. Seleccionamos el SDK mínimo, el API 21: Android 5.0.
3. Elegimos la plantilla del proyecto. Para nuestro caso `Empty Activity`.
4. Creamos la `Activity` y el layout principal del proyecto con el nombre `MainActivity`.
5. En `res/values/styles.xml`, cambiamos el tema `DarkActionBar` a `NoActionBar`.
6. En `res/values/colors.xml`, actualizamos los colores `colorPrimary` y `colorPrimaryDark`.

```
<resources>
    <color name="colorPrimary">#9c27b0</color>
    <color name="colorPrimaryDark">#7b1fa2</color>
    <color name="colorAccent">#ffc107</color>
</resources>
```

7. Agregamos los siguientes recursos String en `res/values/strings.xml`.

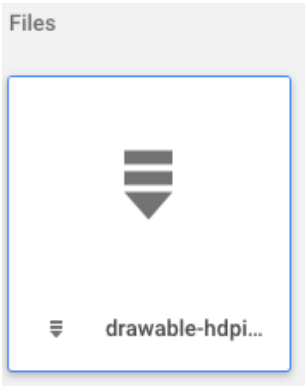
```
<string name="mascotas">Mascotas</string>
<string name="noticas">Noticias</string>
<string name="gatos">GATOS</string>
<string name="perros">PERROS</string>
<string name="caballos">CABALLOS</string>
<string name="peces">PECES</string>
```

8. En `Gradle Scripts/build.gradle (Module:app)`, agregamos la librería de compatibilidad Design.

```
…
implementation 'com.android.support:appcompat-v7:27.1.1'
implementation 'com.android.support.constraint:constraint-
layout:1.1.0'
implementation 'com.android.support:design:27.1.1'
testCompile 'junit:junit:4.12'
…
```

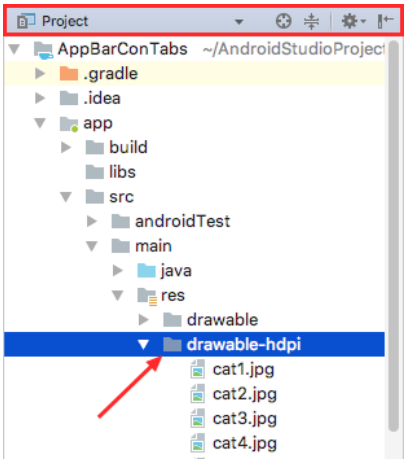
9. En https://drive.google.com/drive/folders/0B0mMTdWK8rKKdmVjd0JUVWhJcDg descargamos drawable-hdpi.zip, y luego lo descomprimimos.

Figura 3.17 Carpeta comprimida de las imágenes de mascota



10. Copiamos la carpeta `drawable-hdpi` en `res/`.

Figura 3.18 Imágenes copiadas en el proyecto



Diseño de los Layouts

Empezaremos con el layout `activity_main` agregando los siguientes elementos:

```
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.AppBarLayout
        android:layout_height="wrap_content"
        android:layout_width="match_parent">

        <android.support.v7.widget.Toolbar
            app:title="@string/mascotas"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:layout_scrollFlags="scroll|enterAlways"

android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"/>

        <android.support.design.widget.TabLayout
            android:id="@+id/tab_layout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
         android:theme="@style/ThemeOverlay.AppCompat.Dark"/>

    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
        android:id="@+id/view_pager"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</android.support.design.widget.CoordinatorLayout>
```

Para navegar por los tipos de mascotas, utilizaremos un `TabLayout`. Este elemento proporciona una navegación horizontal a través de Tabs.

El elemento `ViewPager` solo está disponible en la librería de soporte; no es una clase estándar del SDK de Android.

`ViewPager` permite al usuario desplazarse entre páginas de datos de derecha a izquierda. Normalmente, estas páginas son `Fragment`, lo cual es una manera conveniente de proveer contenido a cada página.

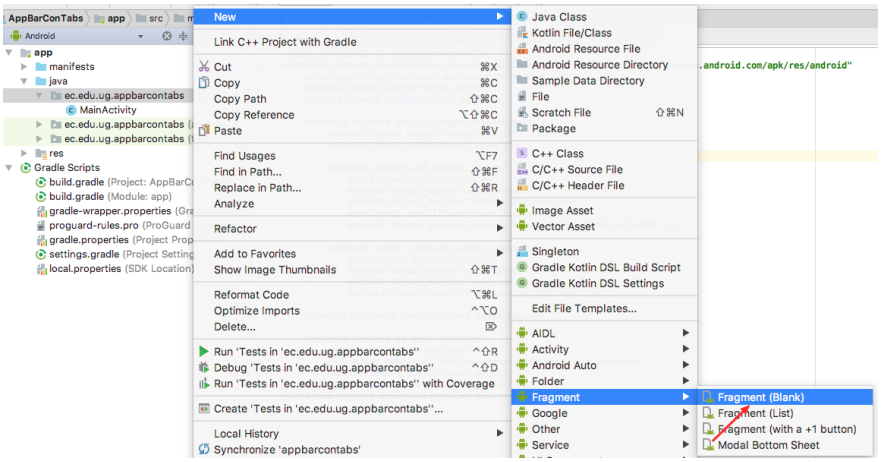
Fragment

Un `Fragment` representa un comportamiento o una parte de una interfaz de usuario en una `Activity`. Una `Activity` puede contener más de un `Fragment`, y este puede ocupar la pantalla entera o solo una parte de la pantalla.

Para que un Fragment pueda mostrase en la pantalla, tiene que inflar un layout. De esta manera, un mismo `Fragment` puede ser utilizado en diferentes `Activities` de la aplicación, contribuyendo a la reutilización de componentes de interfaz de usuario.

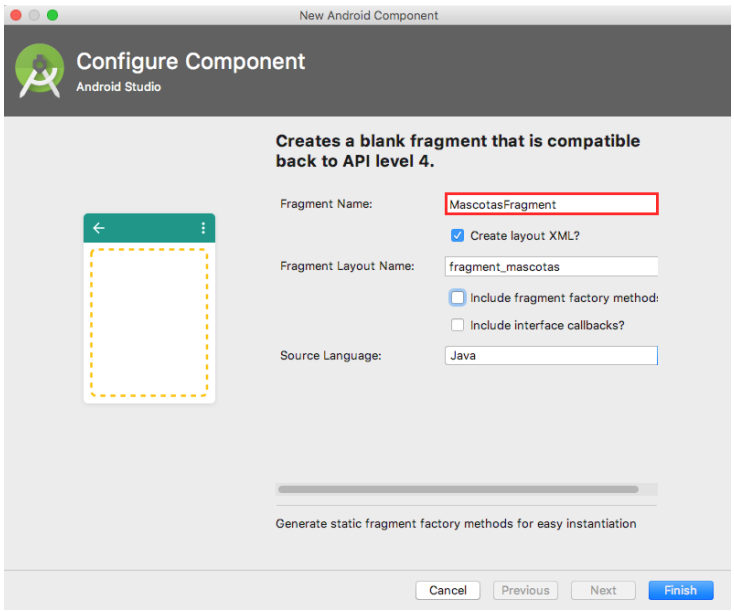
Empezaremos agregando un Fragment (Blank), como se muestra a continuación:

Figura 3.19 Agregando Fragment para Mascotas



Escribimos `MascotasFragment` en "`Fragment Name`" y seleccionamos "Create layout XML".

Figura 3.20 Estableciendo el Nombre al Fragment



Hemos creado una clase llamada `MascotasFragment` y un archivo layout llamado `fragment_mascotas`. Vamos a `res/layout/fragment_mascotas`, y agregamos el siguiente elemento:

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/recycler_view"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Utilizaremos `fragment_mascotas` para cargar las imágenes de mascotas en el RecyclerView.

Creamos otro layout en `app/res/layout` llamado `item_mascota`, y agregamos el siguiente elemento:

```
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/img_mascota"
    android:layout_margin="1dp"
    android:layout_width="match_parent"
    android:layout_height="150dp"
    android:layout_gravity="center"
    android:scaleType="centerCrop"/>
```

El layout `item_mascota` será el ítem que se muestre en el `RecyclerView` a través de un `ViewHolder`. Este mostrará las imágenes de las mascotas en el `widget ImageView`.

Las Clases de Modelo

Cada mascota tendrá una imagen y un tipo. Crearemos una carpeta llamada `dominios`. Allí agregamos una clase llamada `Mascota` con los atributos `imagen` y `tipoMascota` con sus respectivos métodos getters y setters.

```
package deldisenoalcodigo.appbarcontabs.dominio;

public class Mascota {
    private int imagen;
    private int tipoMascota;

    public Mascota(int imagen,int tipoMascota){
        this.imagen = imagen;
        this.tipoMascota = tipoMascota;
    }
    // getters setters…
}
```

En Android, las imágenes se gestionan como recursos en la clase `R`. Cada recurso tiene asignado un valor entero único que sirve para identificarlo a través de la aplicación.

La clase `Mascota` tiene un constructor que recibe dos parámetros de entrada de tipo `int`. El primero es `imagen` y el segundo es `tipoMascota`. La propiedad imagen almacenará el valor de un recurso imagen.

Clase Singleton

Para proveer de mascotas a la aplicación, utilizaremos el patrón de diseño `singleton`. Agregue una clase dentro de la carpeta `dominios` llamada `MascotaSingleton`. Establezca el siguiente código:

```
package deldisenoalcodigo.appbarcontabs.dominio;

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

import ec.edu.ug.appbarcontabs.R;


public class MascotaSingleton {

    private static MascotaSingleton mascotaSingleton;

    private List<Mascota> mascotas = new ArrayList<>();

    private MascotaSingleton(){
        mascotas.add(new Mascota(R.drawable.cat1, 0));
        mascotas.add(new Mascota(R.drawable.cat2, 0));
        mascotas.add(new Mascota(R.drawable.cat3, 0));
        mascotas.add(new Mascota(R.drawable.cat4, 0));
        mascotas.add(new Mascota(R.drawable.cat5, 0));
        mascotas.add(new Mascota(R.drawable.cat6, 0));
        mascotas.add(new Mascota(R.drawable.cat7, 0));
        mascotas.add(new Mascota(R.drawable.cat8, 0));
        mascotas.add(new Mascota(R.drawable.dog1, 1));
        mascotas.add(new Mascota(R.drawable.dog2, 1));
        mascotas.add(new Mascota(R.drawable.dog3, 1));
        mascotas.add(new Mascota(R.drawable.dog4, 1));
        mascotas.add(new Mascota(R.drawable.dog5, 1));
        mascotas.add(new Mascota(R.drawable.dog6, 1));
        mascotas.add(new Mascota(R.drawable.dog7, 1));
        mascotas.add(new Mascota(R.drawable.dog8, 1));
        mascotas.add(new Mascota(R.drawable.dog9, 1));
        mascotas.add(new Mascota(R.drawable.dog10, 1));
        mascotas.add(new Mascota(R.drawable.hor1, 2));
        mascotas.add(new Mascota(R.drawable.hor2, 2));
        mascotas.add(new Mascota(R.drawable.hor3, 2));
        mascotas.add(new Mascota(R.drawable.hor4, 2));
        mascotas.add(new Mascota(R.drawable.fis1, 3));
        mascotas.add(new Mascota(R.drawable.fis2, 3));
        mascotas.add(new Mascota(R.drawable.fis3, 3));
        mascotas.add(new Mascota(R.drawable.fis4, 3));
        mascotas.add(new Mascota(R.drawable.fis5, 3));
        mascotas.add(new Mascota(R.drawable.fis6, 3));
    }
```

```
    public static MascotaSingleton get(){
        if(mascotaSingleton==null)
            mascotaSingleton = new MascotaSingleton();
        return mascotaSingleton;
    }

    public List<Mascota> getMascotas(int tipoMascota){
        List<Mascota> result =new ArrayList<>();
        for(Mascota mascota:mascotas){
            if(mascota.getTipoMascota()==tipoMascota)
                result.add(mascota);
        }
        return result;
    }
}
```

En el constructor de la clase `MascotaSingleton`, agregamos objetos `Mascota` en la lista `mascotas`. Luego, en los constructores de `Mascota` asignamos el valor del recurso imagen en `imagen` y el valor de `tipoMascota`.

Para la variable `tipoMascota` establecemos 0 para los gatos, 1 para los perros, 2 para los caballos y 3 para los peces. De esta manera, el método `getMascotas` retornará una lista de mascota filtradas por su tipo.

Fragments

Se va a utilizar un solo `Fragment` para mostrar las imágenes de las mascotas. Si se selecciona el Tab de gatos, se mostrarán imágenes de gatos, o si se selecciona el Tab de peces, se mostrarán imágenes de peces; todo esto en el mismo `Fragment`.

Para mostrar las imágenes de las mascotas en el `Fragment`, debemos pasar el tipo de mascota como argumento, y obtener una lista filtrada del método `getMascotas` de la clase `MascotaSingleton`. Para pasar argumentos entre `Activities` y `Fragments` utilizamos objetos `Bundle`.

Un `Bundle` almacena valores clave-valor. Para establecer un `Bundle` en un `Fragment`, utilizamos el método de instancia `Fragment.setArgument(Bundle)`. Para poder crear argumentos a un `Fragment`, es necesario realizarlo antes de que se llame al método `onCreate` y después de que se cree la instancia del `Fragment`.

Los programadores de Android siguen la convención de agregar un método estático llamado `newInstance()` en todas las clases `Fragment`. Este método crea una instancia del `Fragment` y establece argumentos a través de objetos `Bundles`.

Cuando se aloja un `Fragment` en una `Activity`, la `Activity` llama al método `newInstance()` en lugar del constructor del `Fragment`. En la `Activity` se puede enviar

cualquier parámetro requerido en `newInstance(…)` que el `Fragment` necesita para crear argumentos.

En MascotasFragment, escribimos un método llamado `newInstance`, que acepte un entero, cree un `Bundle`, cree una instancia de `Fragment` y establezca el argumento en el `Fragment`.

```
public class MascotasFragment extends Fragment {


    private static final String ARG_TIPO_MASCOTAS = "tipoMascota";
    private int tipoMascota;

    public static MascotasFragment newInstance(int tipoGeneroParam) {
        MascotasFragment fragment = new MascotasFragment();
        Bundle args = new Bundle();
        args.putInt(ARG_TIPO_MASCOTAS, tipoGeneroParam);
        fragment.setArguments(args);
        return fragment;
    }
}
```

Dentro del método estático `newInstance`, se instancia un objeto `MascotasFragment` llamado `fragment`. Se crea un objeto `Bundle` llamado `args`. En `args` agregamos en el valor de `tipoMascota` y la clave es el valor de `ARG_TIPO_MASCOTA`.

Cuando un Fragment necesita acceder a sus argumentos, este llama al método `getArguments()` y entonces al método `get+tipoDeDato()`. Escribimos código para recuperar al argumento `tipoMascopa` en onCreate().

```
public class MascotasFragment extends Fragment {
    …
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getArguments() != null)
            tipoMascota = getArguments().getInt(ARG_TIPO_MASCOTAS);
    }
}
```

Observe que se está guardando el argumento en la variable global de la clase `tipoMascota`. Esta variable servirá para filtrar el tipo de mascota.

Los `Fragments` inflan su layout en el método `onCreateView` y retorna un `view` inflado a la `Activity` que lo aloja. Aquí también se inflan los widgets del layout. Agregue el siguiente código en el método `onCreateView`.

```
public class MascotasFragment extends Fragment {
    …
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                             Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_mascotas,
container, false);
        MascotaAdapter mascotaAdapter = new
MascotaAdapter(MascotaSingleton.get().getMascotas(tipoMascota));
        RecyclerView recyclerView = (RecyclerView)
view.findViewById(R.id.recycler_view);
        recyclerView.setLayoutManager(new
StaggeredGridLayoutManager(2,StaggeredGridLayoutManager.VERTICAL));
        recyclerView.setAdapter(mascotaAdapter);
        return view;
    }
}
```

En `onCreateView` se crea una instancia de `MascotaAdapter`. Se envía una lista filtrada por el tipo de mascota. Se instancia un `RecyclerView`, se agrega el `LayoutManager` `StaggeredGridLayoutManager` y se le establece el `Adapter`.

Cada ítem en la lista de mascotas tendrá una sola imagen. Creamos una subclase interna `ViewHolder` llamada `MascotaHolder`. Esta clase tendrá una variable de `ImageView` que servirá para mostrar la imagen de la mascota.

```
public class MascotasFragment extends Fragment {
    …
    private class MascotaHolder extends RecyclerView.ViewHolder{

        private Mascota mascota;
        private ImageView imgMascota;

        public MascotaHolder(View itemView) {
            super(itemView);
            imgMascota = (ImageView)
itemView.findViewById(R.id.img_mascota);
        }

        private void bind(Mascota mascota){
            this.mascota = mascota;
imgMascota.setImageDrawable(getResources().getDrawable(mascota.getImag
en()));
        }
    }
}
```

Luego, creamos otra clase interna llamada `MascotaAdapter` que hereda de `RecyclerView.Adapter`. Como se muestra a continuación:

```
public class MascotasFragment extends Fragment {
    …
    private class MascotaAdapter extends
RecyclerView.Adapter<MascotaHolder>{

        private List<Mascota> mascotas;

        public MascotaAdapter(List<Mascota> mascotas){
            this.mascotas = mascotas;
        }

        @Override
        public MascotaHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
            LayoutInflater
layoutInflater=LayoutInflater.from(getContext());
            View
view=layoutInflater.inflate(R.layout.item_mascota,parent,false);
            return new MascotaHolder(view);
        }

        @Override
        public void onBindViewHolder(MascotaHolder holder, int
position) {
            Mascota mascota = mascotas.get(position);
            holder.bind(mascota);
        }

        @Override
        public int getItemCount() {
            return mascotas.size();
        }
    }

}
```

En el método `onCreateViewHolder` inflamos el layout `item_mascota` en la variable `view`. Después, enviamos `view` a una instancia de `MascotaHolder`. `item_mascota` solo existe un elemento `ImageView` que es instanciada en `MascotaHolder`.

ViewPager y PagerAdapter

`ViewPager` obtiene su contenido de un `PagerAdapter`. Las clases `FragmentPagerAdapter` y `FragmentStatePagerAdapter` son subclases de `PagerAdapter`.

`FragmentPagerAdapter` mantiene cada `Fragment` en memoria, haciendo el cambio más ligero y rápido cuando la cargan los Tabs. Sin embargo, esto puede ser costoso si tienes

muchos `Fragments`. `FragmentStatePagerAdapter` crea y destruye los Fragment grabando su estado.

Para esta aplicación tenemos cuatro Tabs. Dentro de `MainActivity`, creamos una clase interna llamada `MascotaPagerAdapter` que sea una subclase de `FragmentPagerAdapter.`

```
package ec.edu.ug.appbarcontabs;

import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.support.v4.app.Fragment;

public class MainActivity extends AppCompatActivity {
     …
     private class MascotaPagerAdapter extends FragmentPagerAdapter {


        public MascotaPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public Fragment getItem(int position) {
            return MascotasFragment.newInstance(position);
        }

        @Override
        public int getCount() {
            return 4;
        }

        @Override
        public CharSequence getPageTitle(int position) {
            switch (position){
                case 0:
                    return getString(R.string.gatos);
                case 1:
                    return getString(R.string.perros);
                case 2:
                    return getString(R.string.caballos);
                case 3:
                    return getString(R.string.peces);
                default:
                    return null;
            }
        }
    }
}
```

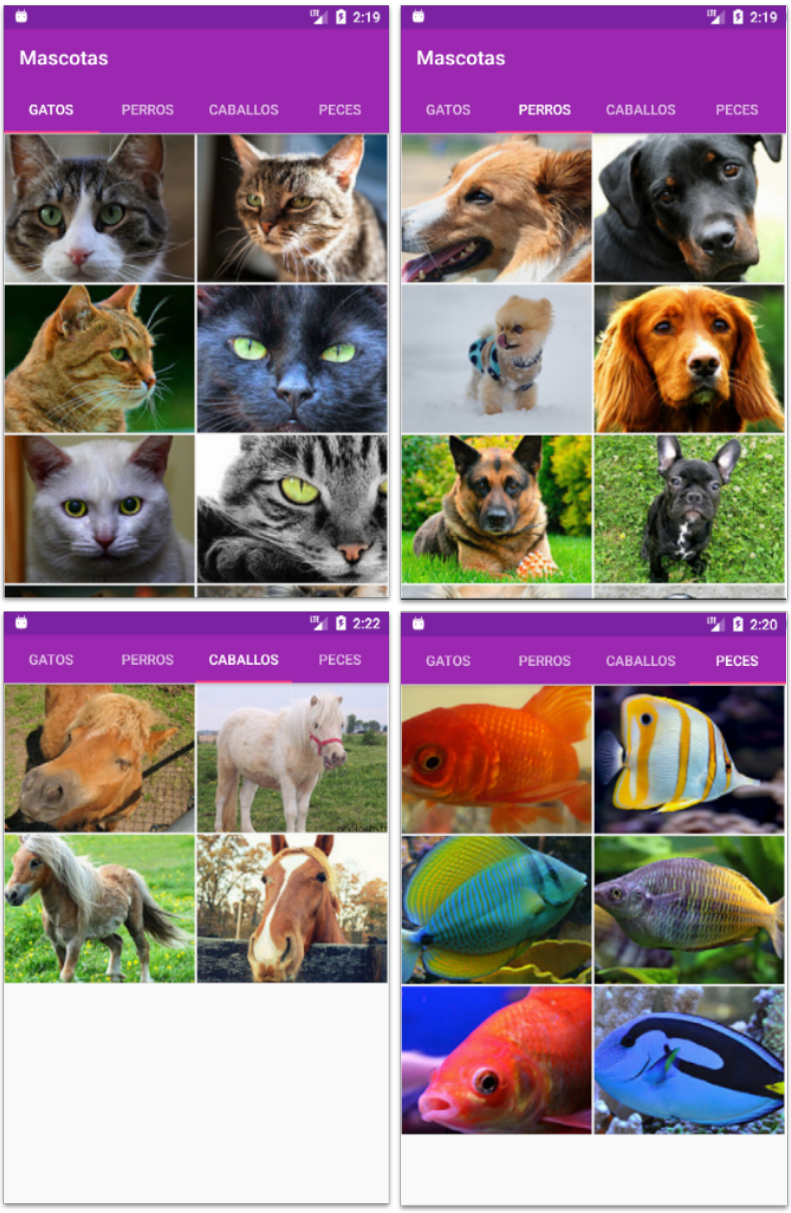
El método `getCount` retorna el total de Tabs que se van a visualizar en la pantalla. El método `getItem` retorna el `Fragment` de acuerdo a la posición que se le asigne. Para este caso, solo tenemos un `Fragment`, al cual enviamos el tipo de mascota o la posición. El método `getPageTitle` retorna el nombre del Tab a la posición determinada.

En el método `onCreate`, inflamos el `ViewPager` y lo establecemos en el `tabLayout`, como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ViewPager viewPager = (ViewPager)
findViewById(R.id.view_pager);
        PagerAdapter pagerAdapter = new
MascotaPagerAdapter(getSupportFragmentManager());
        viewPager.setAdapter(pagerAdapter);

        TabLayout tabLayout = (TabLayout)
findViewById(R.id.tab_layout);
        tabLayout.setupWithViewPager(viewPager);
    }
…
}
```

Una vez terminado este comportamiento, ejecutamos la aplicación. Se mostrará el siguiente resultado:

Figura 3.21 Resultado final del App Bar con Tabs



App Bar con espacio flexible

Cuando se implementa un Appbar con espacio flexible, este se contrae y se expande al desplazar el contenido de la pantalla en dichas direcciones. El título del Appbar aumenta

o disminuye de tamaño. Cuando el Appbar esté parcialmente expandido o contraído, este se coloca en el borde más cercano.

Este comportamiento tiene un botón flotante ubicado en la parte inferior izquierda del Appbar. Cuando el Appbar se contrae, este botón desaparece; y aparece cuando se expande el Appbar.

Para implementar el Appbar con espacio flexible, desarrollaremos una aplicación que muestre directorios y archivos en una lista de dos líneas con imágenes.

Creación y configuración del Proyecto App Bar con espacio flexible

Para desarrollar el Appbar con espacio flexible, realizaremos los siguientes pasos:

1. Establecemos el nombre del proyecto `AppBarConEspacioFlexible`.
2. Seleccionamos el SDK mínimo, el API 21: Android 5.0.
3. Elegimos la plantilla del proyecto. Para nuestro caso `Empty Activity`.
4. Creamos la `Activity` y el layout del proyecto con el nombre `MainActivity`.
5. En `res/values/styles.xml`, cambiamos el tema `DarkActionBar` a `NoActionBar`.
6. En `res/values/colors.xml`, actualizamos los colores `colorPrimary`, `colorPrimaryDark`, colorAccent, y agregamos `textColorPrimary`, `textColorSecundary` y `colorDivider`.

```
<resources>
    <color name="colorPrimary">#e91e63</color>
    <color name="colorPrimaryDark">#b0003a</color>
    <color name="colorAccent">#039be5</color>
    <color name="textColorPrimary">#212121</color>
    <color name="textColorSecundary">#757575</color>
    <color name="colorDivider">#bdbdbd</color>
</resources>
```

7. Agregamos los siguientes recursos String en res/values/string.xml

```
<string name="titulo">Mis documentos</string>
<string name="carpetas">Carpetas</string>
<string name="archivos">Archivos</string>
```

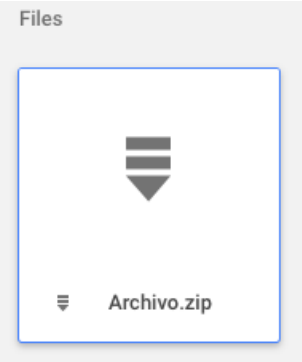
8. En `Gradle Scripts/build.gradle (Module:app)`, agregamos la librería de compatibilidad Design y sincronizamos.

…

```
implementation 'com.android.support:appcompat-v7:27.1.1'
implementation 'com.android.support.constraint:constraint-
layout:1.1.0'
implementation 'com.android.support:design:27.1.1'

...
```

9. En https://drive.google.com/drive/folders/0B0mMTdWK8rKKd1V4VzhsNmEwcDA descargamos Archivo.zip y luego lo descomprimimos.

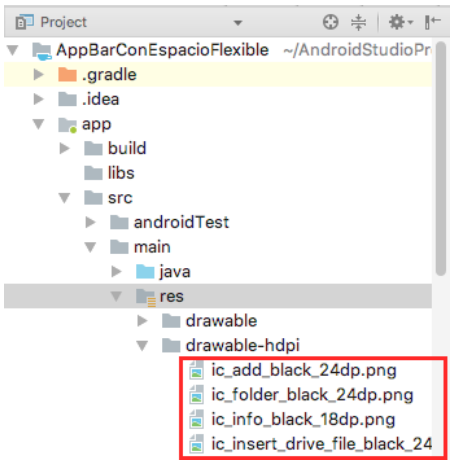
Figura 3.22 Imágenes del proyecto



Los íconos fueron descargados en https://material.io/icons/.

10. Copiamos las carpeta `drawable-hdpi`, `drawable-mdpi`, `drawable-xhdpi`, `drawable-xxhdpi` y `drawable-xxxhdpi` en `res/`.

Figura 3.23 Imágenes copiadas en el proyecto



Diseño de los Layouts

Empezaremos con el layout `activity_main` agregando los siguientes elementos:

```
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/app_bar_space"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fitsSystemWindows="true"
        android:theme="@style/ThemeOverlay.AppCompat.Light">

        <android.support.design.widget.CollapsingToolbarLayout
            app:title="@string/titulo"
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:fitsSystemWindows="true"
            app:contentScrim="@color/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed|snap"
            app:expandedTitleMarginStart="72dp">

            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />
        </android.support.design.widget.CollapsingToolbarLayout>

    </android.support.design.widget.AppBarLayout>

</android.support.design.widget.CoordinatorLayout>
```

CollapsingToolbarLayout

El elemento `CollapsingToolbarLayout` crea animaciones de desplazamiento del Appbar con un mínimo esfuerzo. Fue introducido en Lollipop y está diseñado para ser usado dentro del AppBarLayout.

El atributo `android:layout_height="150dp"` indica la altura máxima que el Appbar se va a expandir.

En `app:contentScrim` se establece el color del Appbar cuando se contrae. El color original retorna cuando se expande el Appbar. Para este caso, mantendremos el color de `colorPrimary` cuando se expanda y contraiga el Appbar.

Se establecieron los valores `"scroll|exitUntilCollapsed|snap"` en `app:layout_scrollFlags`. El valor `scroll` indica que el CollapsingToolbarLayout se mostrará y se ocultará cuando se desplace el contenido de la pantalla. El valor

`exitUntilCollapsed` permite que solo el `CollapsingToolbarLayout` se expanda y contraiga. El valor `snap` hace que `CollapsingToolbarLayout` se desplace hasta el borde más cercano cuando esté parcialmente expandido o contraído.

Toolbar

El atributo app:layout_collapseMode="pin" hace que el Toolbar se mantenga fijo cuando se desplace el `CollapsingToolbarLayout`.

NestedScrollView

El elemento `NestedScrollView` es como un `ScrollView`, con la diferencia de que este soporta scrolling anidado. Regresamos a `activity_main` y después del `AppBarLayout` agregamos un `NestedScrollView` con los siguientes elementos anidados:

```
…
</android.support.design.widget.AppBarLayout>

<android.support.v4.widget.NestedScrollView
        android:id="@+id/scroll_space"
        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="wrap_content"
            android:orientation="vertical">

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/carpetas"
                android:textColor="@color/textColorSecundary"
                android:textSize="17sp"
                android:paddingLeft="72dp"
                android:paddingTop="16dp"
                android:paddingBottom="16dp"/>

            <android.support.v7.widget.RecyclerView
                android:id="@+id/rc_directorio"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"/>

            <View
                android:background="@color/colorDivider"
                android:layout_width="match_parent"
```

```
                android:layout_marginLeft="72dp"
                android:layout_height="1dp" />

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:text="@string/archivos"
                android:textColor="@color/textColorSecundary"
                android:textSize="17sp"
                android:paddingLeft="72dp"
                android:paddingTop="16dp"
                android:paddingBottom="16dp"/>

            <android.support.v7.widget.RecyclerView
                android:id="@+id/rc_archivo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"/>

        </LinearLayout>

    </android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

Dentro de `NestedScrollView`, agregamos dos RecyclerViews. El primero para mostrar la lista de directorios y el segundo para mostrar la lista de archivos.

FloatingActionButton

Después de `NestedScrollView`, agregamos un `FloatingActionButton`, como se muestra a continuación:

```
…
</android.support.v4.widget.NestedScrollView>

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="10dp"
    android:src="@drawable/ic_add_black_24dp"
    app:borderWidth="2dp"
    app:fabSize="normal"
    app:layout_anchor="@+id/app_bar_space"
    app:layout_anchorGravity="bottom|left"
    app:layout_collapseMode="parallax" />
</android.support.design.widget.CoordinatorLayout>
```

El FloatingActionButton es usado para hacer notar una acción importante dentro de la aplicación. En app:fabSize se establecen dos valores: normal y mini.

El atributo `app:layout_anchor` indica el `view` que se va a anclar en el `FloatingActionButton`. En este ejemplo, ese `view` es el `AppBarLayout` con el ID `app_bar_space`. Luego de que se indica el elemento al cual se va a fijar, hay que especificar la posición en que se va a colocar con respecto al `view` ancla. Con `app:layout_anchorGravity="bottom|left"` indicamos que el `FloatingActionButton` se ubicará en el lado izquierdo inferior de `app_bar_space`.

Ítem de la Lista

Para crear el ítem para la lista de archivos, damos clic secundario en "res/layout", luego navegamos hasta "New>Android Resource File" y establecemos el nombre de `item_archivo`. Una vez creado el layout, agregamos los siguientes elementos:

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="72dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp">

    <ImageView
        android:id="@+id/img_archivo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:paddingBottom="36dp"
        android:tint="@color/textColorSecundary"
        app:srcCompat="@drawable/ic_folder_black_24dp" />
    <TextView
        android:id="@+id/txt_archivo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:paddingLeft="56dp"
        android:textColor="@color/textColorPrimary"
        android:textSize="16sp" />
    <TextView
        android:id="@+id/txt_fecha"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/textColorSecundary"
        android:layout_below="@id/txt_archivo"
        android:paddingLeft="56dp"
```

```
        android:textSize="14sp" />
    <ImageView
        android:id="@+id/img_info"
        app:srcCompat="@drawable/ic_info_black_18dp"
        android:paddingBottom="36dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:tint="@color/textColorSecundary"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true" />
</RelativeLayout>
```

Cada ítem de la lista de archivos utilizará el `layout item_archivo`. Se establecieron las métricas de Material Design de la sección Métricas de Material Design para una lista de dos líneas con imágenes. El elemento raíz del ítem es un `RelativeLayout`, este tiene un `paddingRight` y un `paddingLeft` de 16dp, y un `layout_height` de 72dp. Los `ImageViews` `img_archivo` y `img_info`, los colocamos en el centro del `RelativeLayout`, con un `paddingBottom` de 36dp. El `txt_archivo` es el `Primary Text`, con un tamaño de 16sp. Por su parte, `txt_fecha` es el `Secondary Text`, con un tamaño de 14sp.

Las Clases de Dominio

Los archivos tendrán una descripción, una fecha de creación o modificación, un tipo y un ícono. Dentro del proyecto, creamos una carpeta llamada `dominio` y allí, agregamos una clase llamada `Archivo` con los atributos `icono`, `descripcion`, `fecha` y `tipo`, con sus respectivos métodos `getters` y `setters`.

```
package deldisenoalcodigo.appbarconespacioflexible.dominio;

public class Archivo {

    private int icono;
    private String descripcion;
    private String fecha;
    private int tipo;

    public Archivo(int icono, String descripcion, String fecha, int
tipo) {
        this.icono = icono;
        this.descripcion = descripcion;
        this.fecha = fecha;
        this.tipo = tipo;
    }
        // getters setters
}
```

Clase Singleton

Para proveer de objetos `archivo` a la aplicación, utilizaremos el patrón de diseño singleton. Agregamos una clase dentro de la carpeta `dominios` llamada `ArchivoSingleton`. Establezca el siguiente código:

```
package deldisenoalcodigo.appbarconespacioflexible.dominio;

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

import deldisenoalcodigo.appbarconespacioflexible.R;

public class ArchivoSingleton {

    private static ArchivoSingleton carpetaSingleton;

    private List<Archivo> carpetas = new ArrayList<>();

    private ArchivoSingleton(){
        carpetas.add(new
Archivo(R.drawable.ic_folder_black_24dp,"Facebook","30, Nov",0));
        carpetas.add(new
Archivo(R.drawable.ic_folder_black_24dp,"Galeria","29, Nov",0));
        carpetas.add(new
Archivo(R.drawable.ic_folder_black_24dp,"General","30, Dic",0));
        carpetas.add(new
Archivo(R.drawable.ic_folder_black_24dp,"HangOut","20, Dic",0));
        carpetas.add(new
Archivo(R.drawable.ic_folder_black_24dp,"Recientes","22, Dic",0));
        carpetas.add(new
Archivo(R.drawable.ic_folder_black_24dp,"Whatsapp","15, Dic",0));

        carpetas.add(new
Archivo(R.drawable.ic_insert_drive_file_black_24dp,"notas.txt","1,
Ene",1));
        carpetas.add(new
Archivo(R.drawable.ic_insert_drive_file_black_24dp,"query.sql","1,
Ene",1));
        carpetas.add(new
Archivo(R.drawable.ic_insert_drive_file_black_24dp,"informe.docx","1,
Ene",1));

    }

    public static ArchivoSingleton get(){
        if(carpetaSingleton==null)
            carpetaSingleton = new ArchivoSingleton();
        return carpetaSingleton;
```

```
    }

    public List<Archivo> getArchivos(int id){
        List<Archivo> result =new ArrayList<>();
        for(Archivo carpeta:carpetas){
            if(carpeta.getTipo()==id)
                result.add(carpeta);
        }
        return result;
    }
}
```

En el constructor de la clase `ArchivoSingleton`, agregamos objetos `Archivo` en la lista `carpetas`. En los constructores de `Archivo`, asignamos los valores del recurso imagen en `imagen`, `descripcion`, `fecha` y `tipo`.

En la variable `tipo` establecemos 0 para los directorios y 1 para los archivos. De esta manera, el método `getArchivos` retornará una lista de archivos filtradas por el tipo.

MainActivity

Como se mencionó anteriormente, tenemos dos `RecyclerViews` para mostrar los directorios y los archivos. También tendremos dos instancias `RecyclerView.Adapter` para cada `RecyclerView`.

Cada ítem de la lista de archivos tiene dos imágenes y dos textos. Una imagen es estática y la otra cambiará dependiendo si es un archivo o un directorio. A continuación, creamos una subclase interna `ViewHolder` llamada `ArchivoHolder`. Esta clase, tendrá una variable `ImageView` y dos variables `TextView`.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    private class ArchivoHolder extends RecyclerView.ViewHolder{
        private ImageView imgArchivo;
        private TextView txtArchivo;
        private TextView txtFecha;

        public ArchivoHolder(View itemView) {
            super(itemView);
            imgArchivo = (ImageView)
itemView.findViewById(R.id.img_archivo);
            txtArchivo = (TextView)
itemView.findViewById(R.id.txt_archivo);
```

```
            txtFecha = (TextView)
itemView.findViewById(R.id.txt_fecha);
        }

        private void bind(Archivo carpeta){

imgArchivo.setImageDrawable(getResources().getDrawable(carpeta.getIcon
o()));
            txtArchivo.setText(carpeta.getDescripcion());
            txtFecha.setText(carpeta.getFecha());
        }
    }
}
```

Creamos otra clase interna llamada `ArchivoAdapter` que hereda de `RecyclerView.Adapter`, tal como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    …
    public class ArchivoAdapter extends
RecyclerView.Adapter<ArchivoHolder> {

        private List<Archivo> archivos;

        public ArchivoAdapter(List<Archivo> archivos){
            this.archivos = archivos;
        }

        @Override
        public  ArchivoHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
            LayoutInflater inflater =
LayoutInflater.from(MainActivity.this);
            View view = inflater.inflate(R.layout.item_archivo,
parent, false);
            return new MainActivity.ArchivoHolder(view);
        }

        @Override
        public void onBindViewHolder(ArchivoHolder holder, int
position) {
            Archivo carpeta = archivos.get(position);
            holder.bind(carpeta);
        }

        @Override
```

```
        public int getItemCount() {
            return archivos.size();
        }
    }
}
```

En el método `onCreateViewHolder` inflamos el layout `item_archivo` en la variable `view`. Después, enviamos `view` a una instancia `ArchivoHolder`. `item_archivo` tiene un `ImageView` y dos `TextView` que son instanciadas en `ArchivoHolder`.

En el método `onCreate` instanciamos el `Adapter directorioAdapter` que almacene los directorios y otro `Adapter archivoAdapter` que almacene los archivos. Luego, inflamos dos `RecyclerViews` y le establecemos los `Adapter` antes creados.

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ArchivoAdapter directorioAdapter = new
ArchivoAdapter(ArchivoSingleton.get().getArchivos(0));
        RecyclerView rcDirectorio = (RecyclerView)
findViewById(R.id.rc_directorio);
        rcDirectorio.setLayoutManager(new
LinearLayoutManager(MainActivity.this));
        rcDirectorio.setAdapter(directorioAdapter);

        ArchivoAdapter archivoAdapter = new
ArchivoAdapter(ArchivoSingleton.get().getArchivos(1));
        RecyclerView rcArchivo = (RecyclerView)
findViewById(R.id.rc_archivo);
        rcArchivo.setLayoutManager(new
LinearLayoutManager(MainActivity.this));
        rcArchivo.setAdapter(archivoAdapter);

    }
    …
}
```

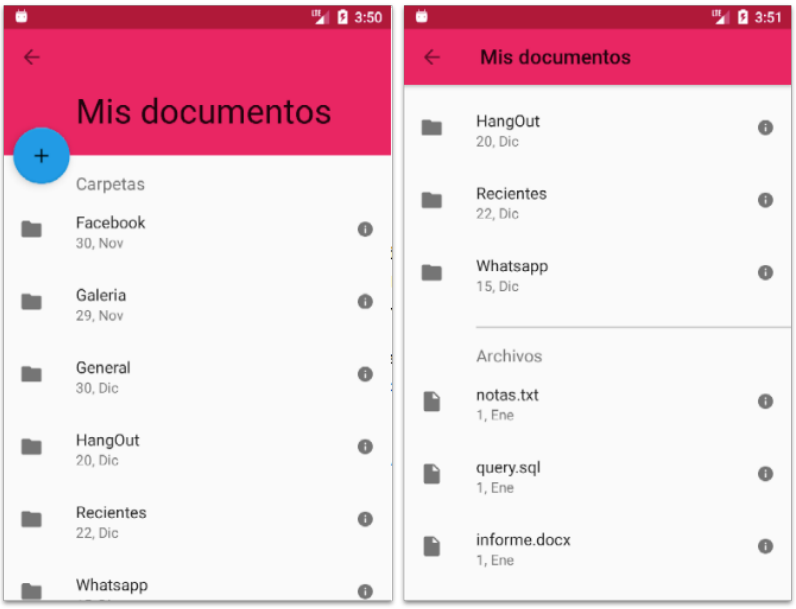
Por último, inflamos un `Toolbar`, un `ActionBar` y un `FloatingActionButton` en el método `onCreate`. En la instancia del `FloatingActionButton` establecemos el evento clic. Cada vez que se presiona este botón aparecerá el mensaje "Hacer algún proceso".

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        …
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);

        FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this,"Hace algún
proceso",Toast.LENGTH_SHORT).show();
            }
        });
    }
    …
}
```

Una vez terminado este comportamiento, ejecutamos la aplicación. Se mostrará el siguiente resultado:

Figura 3.24 Resultado final del App bar con espacio flexible



App Bar con espacio flexible con imagen

Cuando se implementa un Appbar con espacio flexible con una imagen, la altura del Appbar se expande hasta una altura determinada por el desarrollador y se contrae hasta que solo queda el Status bar.

Cuando el Appbar está expandido, en su espacio se muestra la imagen, a medida que se va contrayendo el Appbar, la imagen va desapareciendo hasta llegar a la altura mínima del mismo. Si se sigue contrayendo, el App bar desaparecerá.

Este comportamiento tiene un botón flotante ubicado en la parte inferior derecha del Appbar. Cuando el Appbar se contrae, este botón desaparecerá, y aparecerá cuando se expanda el Appbar.

Para llevar el diseño y comportamiento de un Appbar con espacio flexible y una imagen al código, tomaremos de base la aplicación desarrollada del Appbar con Tabs, y agregaremos detalles de la mascota que se selecciona de la lista.

Configuración del Proyecto App bar con espacio flexible con imagen

Para desarrollar el App bar con espacio flexible con imagen, utilizaremos el proyecto `AppBarConTabs`. Realizaremos los siguientes pasos:

1. En `res/values/colors.xml`, agregamos el color `colorDivider`.

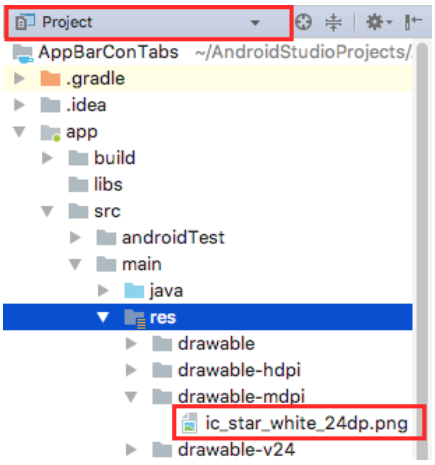
```
<resources>
    <color name="colorPrimary">#9c27b0</color>
    <color name="colorPrimaryDark">#7b1fa2</color>
    <color name="colorAccent">#ffc107</color>
    <color name="colorDivider">#bdbdbd</color>
</resources>
```

2. Agregamos los siguientes recursos String en `res/values/string.xml`.

```
<resources>
    …
    <string name="favoritos">Favoritos</string>
    <string name="visualizaciones">Visualizaciones</string>
    <string name="comentarios">Comentarios</string>
</resources>
```

3. En https://drive.google.com/drive/folders/0B0mMTdWK8rKKdDRHLVlzT1RpRnc descargue Archivo.zip.
4. Copiamos las carpetas `drawable-hdpi`, `drawable-mdpi`, `drawable-xhdpi` y `drawable-xxhdpi` y `drawable-xxxhdpi` con la imagen ic_star_white.

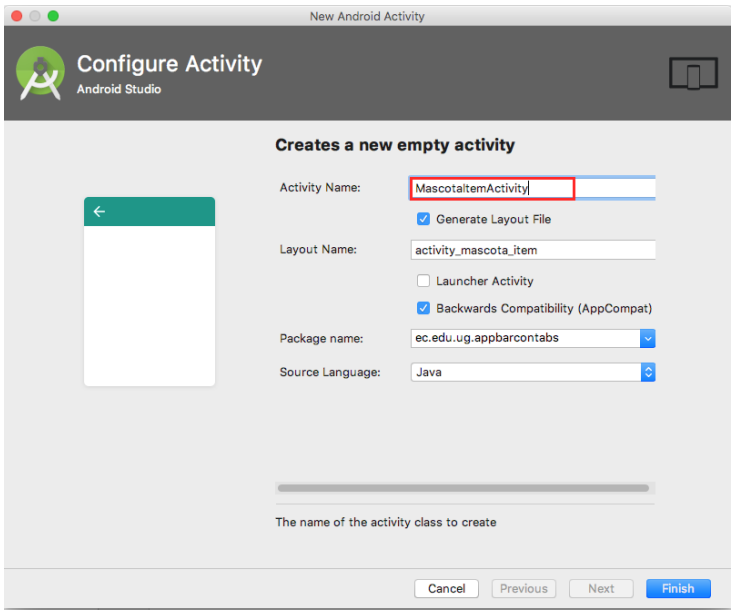
Figura 3.25 Carpetas de imágenes para el App Bar flexible con imagen



Diseño de los Layout

Creamos una `Empty` Activity llamada `MascotaItemActivity`, tal como se muestra a continuación:

Figura 3.26 Agregando nueva Activity al proyecto



En el layout `activity_mascota_item` agregamos los siguientes elementos:

```
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/coordinator"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true">

    <android.support.design.widget.AppBarLayout
        android:id="@+id/app_bar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fitsSystemWindows="true"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">

        <android.support.design.widget.CollapsingToolbarLayout
            android:id="@+id/collapser"
            android:layout_width="match_parent"
            android:layout_height="256dp"
            android:fitsSystemWindows="true"
            app:contentScrim="?attr/colorPrimary"
            app:layout_scrollFlags="scroll|enterAlways|snap">
            <ImageView
                android:id="@+id/img_mascota"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:fitsSystemWindows="true"
                android:scaleType="centerCrop"
                android:src="@drawable/dog2"
                app:layout_collapseMode="parallax"/>
```

```
            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
                app:layout_collapseMode="pin" />
        </android.support.design.widget.CollapsingToolbarLayout>

    </android.support.design.widget.AppBarLayout>
</android.support.design.widget.CoordinatorLayout>
```

En el elemento `CollapsingToolbarLayout`, el atributo android:layout_height="256dp" indica la altura máxima de expansión del Appbar.

Establecimos los valores `"scroll|enterAlways|snap"` en `app:layout_scrollFlags`. Con estos valores, el `CollapsingToolbarLayout` se contrae hasta el tamaño normal del Appbar (48dp o 56dp). Luego, si se sigue desplazando el contenido hacia arriba, el Appbar desaparecerá, quedando solamente el Status bar.

Después del `AppBarLayout`, agregamos un `NestedScrollView` con los siguientes elementos anidados:

```
…
</android.support.design.widget.AppBarLayout>

<android.support.v4.widget.NestedScrollView
        android:id="@+id/scroll"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/app_bar"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">

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

            <TextView
                android:id="@+id/txt_fecha"
                android:layout_marginLeft="68dp"
                android:layout_marginTop="16dp"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

            <View
                android:background="@color/colorDivider"
                android:layout_width="match_parent"
                android:layout_marginLeft="68dp"
                android:layout_marginTop="16dp"
                android:layout_height="1dp" />
```

```
<RelativeLayout
    android:paddingTop="16dp"
    android:paddingLeft="16dp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <ImageView
        android:id="@+id/img_favorito"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_star_white_24dp"
        android:tint="@android:color/black"/>
    <LinearLayout
        android:layout_toRightOf="@id/img_favorito"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:paddingLeft="30dp"
        android:layout_height="wrap_content">

        <TextView
            android:id="@+id/txt_favoritos"
            android:textSize="24dp"
            android:textColor="@android:color/black"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
        <TextView
            android:text="@string/favoritos"
            android:textSize="18dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <TextView
            android:paddingTop="16dp"
            android:id="@+id/txt_visto"
            android:textColor="@android:color/black"
            android:textSize="24dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <TextView
            android:text="@string/visualizaciones"
            android:textSize="18dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <TextView
            android:paddingTop="16dp"
            android:id="@+id/txt_comentarios"
            android:textColor="@android:color/black"
            android:textSize="24dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
```

```
                    <TextView
                        android:text="@string/comentarios"
                        android:textSize="18dp"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content" />
                </LinearLayout>

            </RelativeLayout>

        </LinearLayout>

    </android.support.v4.widget.NestedScrollView>

</android.support.design.widget.CoordinatorLayout>
```

Para finalizar con `activity_mascota_item`, después de `NestedScrollView`, agregamos un `FloatingActionButton`, como se muestra a continuación:

```
    …
    </android.support.v4.widget.NestedScrollView>
    <android.support.design.widget.FloatingActionButton
        android:id="@+id/floatingActionButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:src="@drawable/ic_star_white_24dp"
        app:borderWidth="2dp"
        app:fabSize="normal"
        app:layout_anchor="@+id/app_bar"
        app:layout_anchorGravity="bottom|right"
        app:layout_collapseMode="parallax" />

</android.support.design.widget.CoordinatorLayout>
```

Con el atributo `app:layout_anchor="@+id/app_bar"` indicamos que el `FloatingActionButton` se anclará al `AppBarLayout`, y con `app:layout_anchorGravity="bottom|right"` este se ubicará en la parte inferior derecha del AppBarLayout.

La clase de Dominio

En la clase `Mascota`, agregamos las propiedades `id`, `nombre`, `fecha`, `vistas`, `favoritos` y `comentarios`, tal como se muestra a continuación:

```
package deldisenoalcodigo.appbarconespacioflexibleeimagen.dominios;

import java.util.UUID;
```

```
public class Mascota {
    private UUID id;
    private String nombre;
    private int imagen;
    private int tipoMascota;
    private String fecha;
    private int vistas;
    private int favoritos;
    private int comentarios;

    public Mascota(String nombre,int imagen,int tipoMascota,
                   String fecha,int vistas,int favoritos,int
comentarios){
        this.id = UUID.randomUUID();
        this.nombre = nombre;
        this.imagen = imagen;
        this.tipoMascota = tipoMascota;
        this.fecha = fecha;
        this.vistas = vistas;
        this.favoritos = favoritos;
        this.comentarios = comentarios;
    }
    // getters setters

}
```

La propiedad `id` es de tipo `UUDI` o *universally unique identifier*, es un número de 16 bytes. En el constructor de `Mascota` se genera un valor aleatorio único, que servirá para identificar un objeto. Esto nos permitirá buscar objetos `Mascota` por el `id`.

Clase Singleton

Para proveer de objetos `Mascota`, modificaremos la clase `MascotaSingleton`. En el constructor de `MascotaSingleton` creamos objetos `Mascota` enviando los nuevos parámetros de entrada del constructor de la clase. También, agregamos el método `getMascota`, que servirá para obtener un objeto `Mascota` por su `id`.

```
package deldisenoalcodigo.appbarconespacioflexibleeimagen.dominios;

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

import deldisenoalcodigo.appbarconespacioflexibleeimagen.R;
```

```
public class MascotaSingleton {
    …
    private MascotaSingleton(){
        mascotas.add(new Mascota("Nube",R.drawable.cat1, 0,"29
May",29900,9023,7513));
        mascotas.add(new Mascota("Moncho",R.drawable.cat2, 0,"4
Jun",34989,10030,5450));
        mascotas.add(new Mascota("Sissi",R.drawable.cat3, 0,"5
Jun",40289,39002,40103));
        mascotas.add(new Mascota("Milito",R.drawable.cat4, 0,"5
Jun",54201,102,50301));
        mascotas.add(new Mascota("Milka",R.drawable.cat5, 0,"11
Jun",103982,90102,101092));
        mascotas.add(new Mascota("Olaf",R.drawable.cat6, 0,"12
Jun",402982,203982,102999));
        mascotas.add(new Mascota("Michi",R.drawable.cat7, 0,"23
Jun",505985,303933,202227));
        mascotas.add(new Mascota("Gran Gordo",R.drawable.cat8, 0,"23
Jun",606966,505985,103933));
        mascotas.add(new Mascota("Yeika",R.drawable.dog1, 1,"24
Jun",302383,103383,2441));
        mascotas.add(new Mascota("Darwin",R.drawable.dog2, 1,"25
Jun",402982,203982,102999));
        mascotas.add(new Mascota("Zara",R.drawable.dog3, 1,"1
Jul",102921,103121,12923));
        mascotas.add(new Mascota("Kappy",R.drawable.dog4, 1,"2
Jul",302333,102627,204167));
        mascotas.add(new Mascota("Mingo",R.drawable.dog5, 1,"2
Jul",302981,403572,702697));
        mascotas.add(new Mascota("Yogui",R.drawable.dog6, 1,"3
Jul",202982,403968,602699));
        mascotas.add(new Mascota("Comino",R.drawable.dog7, 1,"13
Ago",502984,203312,502229));
        mascotas.add(new Mascota("Lilly",R.drawable.dog8, 1,"14
Ago",702682,103985,402945));
        mascotas.add(new Mascota("Darth Vader",R.drawable.dog9, 1,"15
Ago",801987,203281,40221));
        mascotas.add(new Mascota("Heracles",R.drawable.dog10, 1,"16
Ago",906289,203972,102111));
        mascotas.add(new Mascota("Pícaro",R.drawable.hor1, 2,"1
Sep",505985,203982,502233));
        mascotas.add(new Mascota("Sargento",R.drawable.hor2, 2,"1
Sep",402982,203982,102988));
        mascotas.add(new Mascota("Carbonero",R.drawable.hor3, 2,"2
Sep",302982,203982,102397));
        mascotas.add(new Mascota("Pío",R.drawable.hor4, 2,"3
Sep",602942,203982,102999));
        mascotas.add(new Mascota("Octavio",R.drawable.fis1, 3,"4
Sep",603982,203982,102922));
        mascotas.add(new Mascota("Pinto",R.drawable.fis2, 3,"1
Oct",702937,203982,102999));
```

```
        mascotas.add(new Mascota("Minchi",R.drawable.fis3, 3,"2
Oct",803981,203982,102999));
        mascotas.add(new Mascota("Coco",R.drawable.fis4, 3,"2
Oct",902785,203982,102197));
        mascotas.add(new Mascota("Alcachofo",R.drawable.fis5, 3,"11
Oct",502585,103281,302339));
        mascotas.add(new Mascota("Emilio",R.drawable.fis6, 3,"15
Oct",402982,203982,102697));
    }
    …
    public Mascota getMascota(UUID id){
        for(Mascota mascota:mascotas){
            if(mascota.getId().equals(id))
                return mascota;
        }
        return null;
    }

}
```

MascotaItemActivity

Para llamar a un Activity de otra Activity, se utiliza el método estático startActivity(intent). Este método envía información a una parte del sistema operativo llamado `ActivityManager`. El `ActivityManager` crea instancias `Activity` y llama a sus métodos `onCreate(…)`.

Los programadores Android utilizan una convención para invocar una `Activity`. Se crea un método estático llamado `newIntent(…)` en las clases `Activity`. Este método recibe los parámetros requeridos por la Activity, se crea un `Intent` y se retorna este objeto a la Activity o Fragment que lo está invocando.

Un `Intent` es un objeto que puede comunicarse con el sistema operativo. Tiene algunos constructores, pero el que vamos a utilizar es el siguiente constructor:

```
public Intent(Context packageContext, Class<?> cls)
```

El argumento `Class` indica al `ActivityManager` qué `Activity` destino es invocada. El argumento `Contenxt` indica al `ActivityManager` qué `Activity` origen está invocando a la nueva `Activity`.

En la clase `MascotaItemActivity` creamos la variable `EXTRA_MASCOTA_ID` y el método `newIntent`, como se muestra a continuación:

```
public class MascotaItemActivity extends AppCompatActivity {

    private static final String EXTRA_MASCOTA_ID = "mascotaId";

    public static Intent newIntent(Context context, UUID id){
        Intent intent = new Intent(context,MascotaItemActivity.class);
        intent.putExtra(EXTRA_MASCOTA_ID,id);
        return intent;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mascota_item);
    }
}
```

Los objetos `Intent` almacenan valores clave-valor. Luego, estos valores pueden ser obtenidos en el método `onCreate`. Para obtener el `UUDI` en el método `onCreate`, agregamos el siguiente código:

```
public class MascotaItemActivity extends AppCompatActivity {

    …
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mascota_item);

        UUID id = (UUID)
getIntent().getSerializableExtra(EXTRA_MASCOTA_ID);
        Mascota mascota = MascotaSingleton.get().getMascota(id);

    }
}
```

En la variable `id` se obtiene el identificador único de un objeto `Mascota`. Después, a través de `MascotaSingleton` obtenemos el objeto `Mascota`.

Para finalizar con `MascotaItemActivity`, instanciamos los objetos necesarios para mostrar información sobre la mascota que seleccionamos.

```
public class MascotaItemActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        …
        DecimalFormat formatter = new DecimalFormat("#,###,###");

        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

```
        ImageView imgMascota = (ImageView)
findViewById(R.id.img_mascota);

imgMascota.setImageDrawable(getResources().getDrawable(mascota.getImag
en()));

        CollapsingToolbarLayout collapser = (CollapsingToolbarLayout)
findViewById(R.id.collapser);
        collapser.setTitle(mascota.getNombre());

        TextView txtFavoritos = (TextView)
findViewById(R.id.txt_favoritos);

txtFavoritos.setText(formatter.format(mascota.getFavoritos()));

        TextView txtVisto = (TextView) findViewById(R.id.txt_visto);
        txtVisto.setText(formatter.format(mascota.getVistas()));

        TextView txtComentarios = (TextView)
findViewById(R.id.txt_comentarios);

txtComentarios.setText(formatter.format(mascota.getComentarios()));

        TextView txtFecha = (TextView) findViewById(R.id.txt_fecha);
        txtFecha.setText(mascota.getFecha());

        ActionBar actionBar = getSupportActionBar();
        actionBar.setDisplayHomeAsUpEnabled(true);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem menuItem) {
        if (menuItem.getItemId() == android.R.id.home) {
            finish();
        }
        return super.onOptionsItemSelected(menuItem);
    }
}
```

En `actionBar.setDisplayHomeAsUpEnabled(true)` establecemos en la App bar una flecha de navegación hacia atrás. En el método `onOptionsItemSelected` capturamos el evento clic de la flecha de navegación, y allí finalizamos la `Activity` actual, para así retornar a la lista de mascotas.

MascotasFragment

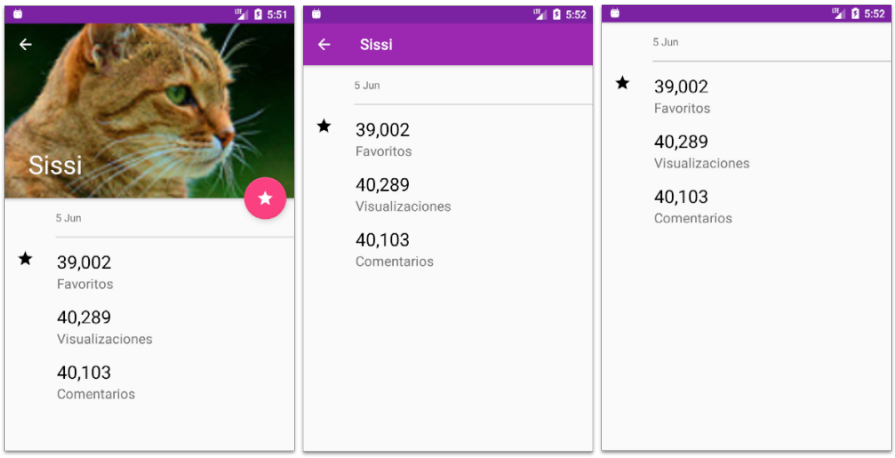
En la `Activity` o `Fragment` que invoca a otra Activity, obtenemos el Intent del método newIntent(…). Luego, enviamos como parámetro ese Intent al método `startActivity`.

Para finalizar el proyecto, agregamos el evento clic en `MascotaHolder` y llamamos a `MascotaItemActivity`. Para esto, implementamos la interfaz `View.OnclickListener` en `MascotaHolder`. `itemView` representa un ítem de la lista y establecemos el evento clic. En el método onClick llamamos a `MascotaItemActivity`.

```
public class MascotasFragment extends Fragment {

…
private class MascotaHolder extends RecyclerView.ViewHolder implements
View.OnClickListener{

        private Mascota mascota;
        private ImageView imgMascota;

        public MascotaHolder(View itemView) {
            …
            itemView.setOnClickListener(this);
            …
        }

        private void bind(Mascota mascota){
            this.mascota = mascota;

imgMascota.setImageDrawable(getResources().getDrawable(mascota.getImag
en()));
        }

        @Override
        public void onClick(View view) {
            Intent intent =
MascotaItemActivity.newIntent(getContext(),mascota.getId());
            startActivity(intent);
        }
    }
}
```

Una vez terminado este comportamiento, ejecutamos la aplicación. Se mostrará el siguiente resultado:

Figura 3.27 Resultado final del App Bar con espacio flexible y una imagen



App Bar con espacio flexible y contenido sobrepuesto

Cuando se implementa un Appbar con espacio flexible y contenido sobrepuesto, la altura del Appbar se expande hasta una altura predeterminada por el desarrollador y se contrae hasta la altura mínima de Portrait o Landscape del Appbar.

Cuando el Appbar está expandido, el contenido está encima del Appbar. A medida que se va contrayendo el contenido, el Appbar se ubica encima del contenido.

Para llevar el diseño y comportamiento de un App bar con espacio flexible y una imagen al código, tomaremos de base la aplicación desarrollada del Appbar con espacio flexible.

Configuración del Proyecto App Bar con espacio flexible con contenido sobrepuesto

Para desarrollar el Appbar con espacio flexible con imagen, agregamos todo lo que hemos hecho en el proyecto `AppBarConEspacioFlexible`. Para esto, realizamos los siguientes pasos:

1. En `res/values/colors.xml`, agregamos el color `colorCardView`.

```
<resources>
    …
    <color name="colorCardView">#eeeeee</color>
```

```
</resources>
```

2. En `Gradle Scripts/build.gradle (Module:app)`, agregamos la biblioteca de compatibilidad Design y sincronizamos.

```
dependencies {
    …
    implementation 'com.android.support:cardview-v7:27.1.1'
}
```

Diseño de los Layout

Para establecer el comportamiento del Appbar con espacio flexible y contenido sobrepuesto, modificaremos algunos widgets de `activity_main`.

En el elemento `CollapsingToolbarLayout` eliminamos el atributo `app:title`, actualizamos la altura de 150dp a 272dp en el atributo `android:layout_width="272dp"` y agregamos el atributo `app:titleEnable="true"`.

```
<android.support.design.widget.CoordinatorLayout
…
        <android.support.design.widget.CollapsingToolbarLayout
            android:layout_width="match_parent"
            android:layout_height="272dp"
            android:fitsSystemWindows="true"
            app:contentScrim="@color/colorPrimary"
            app:layout_scrollFlags="scroll|exitUntilCollapsed|snap"
            app:expandedTitleMarginStart="72dp"
            app:titleEnabled="false">
…
</android.support.design.widget.CoordinatorLayout>
```

El título en este comportamiento se mantiene fijo. Por este motivo, se elimina el título de `CollapsingToolbarLayout` y se agrega el atributo `app:title="@string/titulo"` en el `Toolbar`.

```
<android.support.design.widget.CoordinatorLayout
…
            <android.support.v7.widget.Toolbar
                android:id="@+id/toolbar"
                app:title="@string/titulo"
                android:layout_width="match_parent"
                android:layout_height="?attr/actionBarSize"
                app:layout_collapseMode="pin" />
…
</android.support.design.widget.CoordinatorLayout>
```

Para establecer la apariencia del contenido sobrepuesto en el Appbar, se agrega el atributo `app:Behavior_overlapTop="100dp"` en el elemento `NestedScrollView`. Con este valor, el contenido se sobrepondrá al Appbar con 100dp.

```
<android.support.design.widget.CoordinatorLayout

    <android.support.v4.widget.NestedScrollView
        android:id="@+id/scroll_space"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:behavior_overlapTop="100dp"
        app:layout_behavior="@string/appbar_scrolling_view_behavior">
        …

    </android.support.v4.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

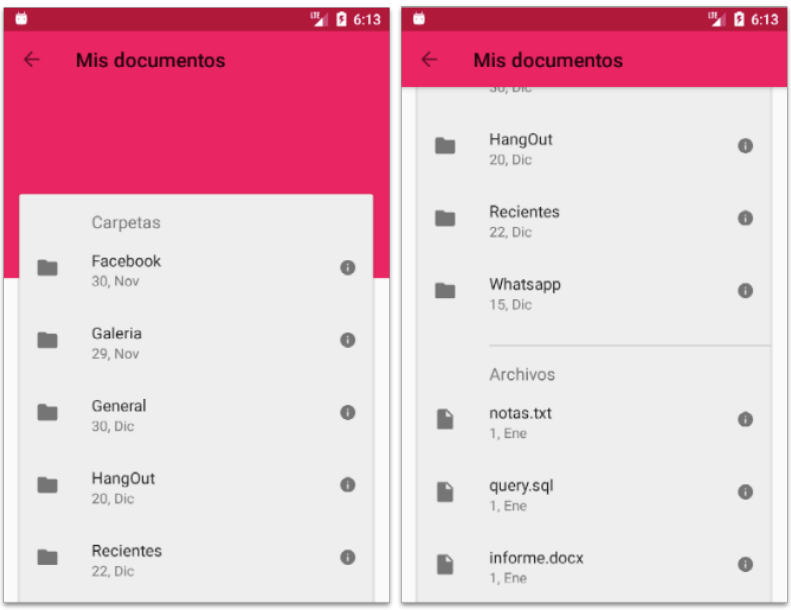
Agregamos un `CardView` como raíz del `LinearLayout`. En este elemento establecemos un margen de 16dp, una elevación y un color de fondo.

```
<android.support.design.widget.CoordinatorLayout
…

    <android.support.v4.widget.NestedScrollView
    …>
        <android.support.v7.widget.CardView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="16dp"
            app:cardElevation="@dimen/cardview_default_elevation"
            app:cardBackgroundColor="@color/colorCardView">

            <LinearLayout
            …>

                <TextView
                 …/>

                <android.support.v7.widget.RecyclerView
                …/>

                <View
                …/>

                <TextView
                …/>

                <android.support.v7.widget.RecyclerView

                …/>
```

```
            </LinearLayout>

        </android.support.v7.widget.CardView>
    </android.support.v4.widget.NestedScrollView>
</android.support.design.widget.CoordinatorLayout>
```

Una vez terminado este comportamiento, ejecutamos la aplicación. Se mostrará el siguiente resultado:

Figura 3.28 Resultado final del App Bar con espacio flexible y contenido sobrepuesto



Resumen

Es este capítulo se establecieron las métricas de Material Design para algunas listas de elementos. Se desarrolló el comportamiento del estándar Appbar, del Appbar con Tabs, del Appbar con espacio flexible con y sin imagen, y del Appbar con contenido sobrepuesto.

Capítulo 4. Navigation Drawer Y Bottom Navigation

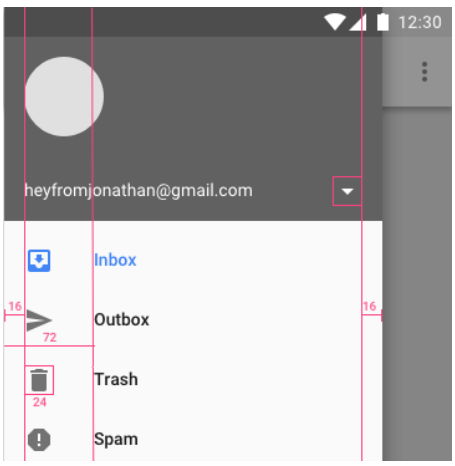
En este capítulo, desarrollaremos los comportamientos del Navigation Drawer y del Bottom Navigation. Ambos componentes definen la forma de navegación de cualquier aplicación en la plataforma Android. Utilizaremos como base el proyecto `AppbarConTab` del capítulo 2 y cambiaremos el Appbar con pestañas por un Navigation Drawer y por un Bottom Navigation.

Navigation Drawer

Aplicaciones como Gmail o Facebook han implementado este tipo de patrón de navegación. Navigation Drawer es un menú lateral que sale del lado izquierdo o del lado derecho de la aplicación cuando se desliza el dedo por la pantalla en los bordes o al hacer clic en un botón superior que normalmente tiene un ícono con tres líneas horizontales en el App Bar. Ocupa toda la pantalla, incluso el Statusbar. Todo lo que esta detrás es visible, pero está oscurecido por una malla.

A continuación se muestran las métricas por defecto de Material Design para este menú lateral.

Figura 4.1



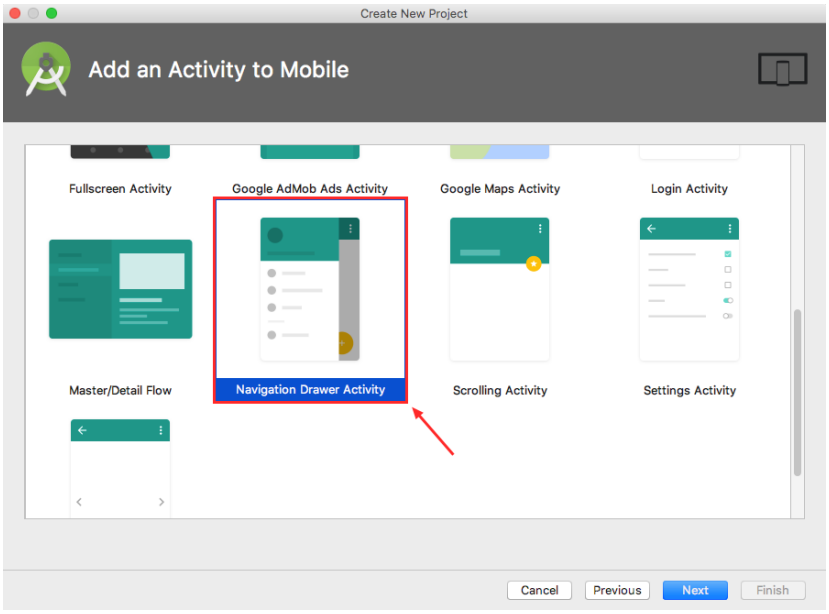
Para desarrollar el comportamiento del Navigation Drawer, utilizaremos la plantilla Navigation Drawer Activity, la cual viene en Android Studio. El código de esta plantilla está optimizado para este patrón, por lo que es una buena idea utilizarlo y luego personalizarlo a tu aplicación.

Creación y configuración de un proyecto

Para desarrollar este proyecto, realizaremos los siguientes pasos:

1. Creamos un proyecto con el nombre "NavigationDrawerInfo".
2. Elegimos el SDK mínimo, el API 21: Android 5.0.
3. Seleccionamos la plantilla "Navigation Drawer Activity".

Figura 4.2 Seleccionar la plantilla Navigation Drawer

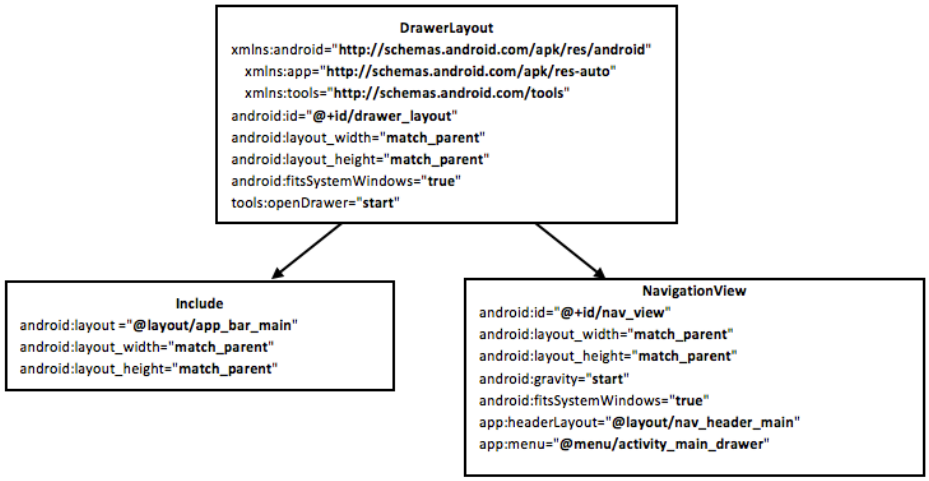


4. Creamos la `Activity` y el Layout del proyecto con el nombre `MainActivity`, y finalizamos la creación.

Layouts de la plantilla Navigation Drawer

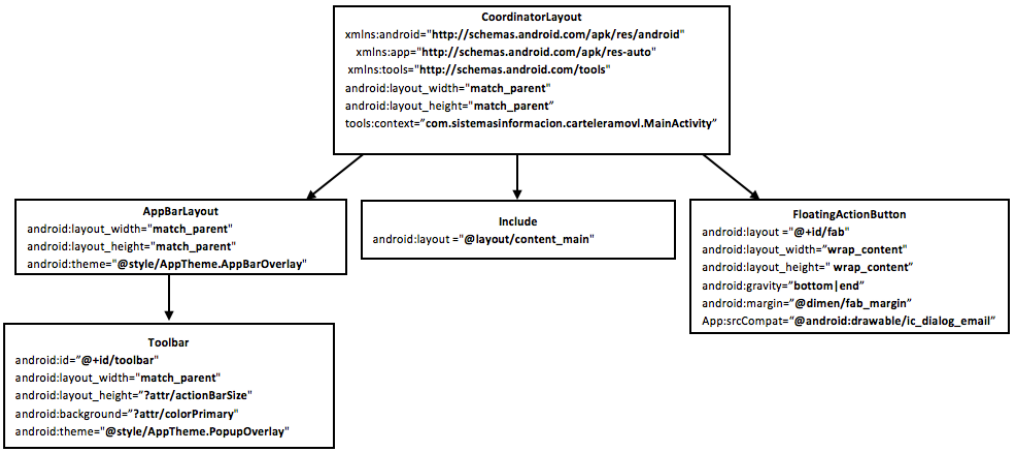
Para analizar esta plantilla, empezaremos en `app/res/layout/activity_main.xml`. A continuación se muestra una vista jerárquica de este layout:

Figura 4.3 Vista jerárquica de activity_main de la plantilla Navigation Drawer



`DrawerLayout` actúa como contenedor del componente `NavigationView`. El elemento `NavigationView` representa un menú estándar de navegación para las aplicaciones en Android. En el atributo `app:menu` se establece el recurso menú `menu/activity_main_drawer` y el atributo `app:headerLayout` coloca una cabecera en el `NavigationDrawer`. El elemento `include` sirve para colocar un layout dentro de otro layout. Para este caso, se incluye el layout `app_bar_main.xml` en `activity_main`.

Figura 4.4 Vista jerárquica de app_bar_main de la plantilla Navigation Drawer



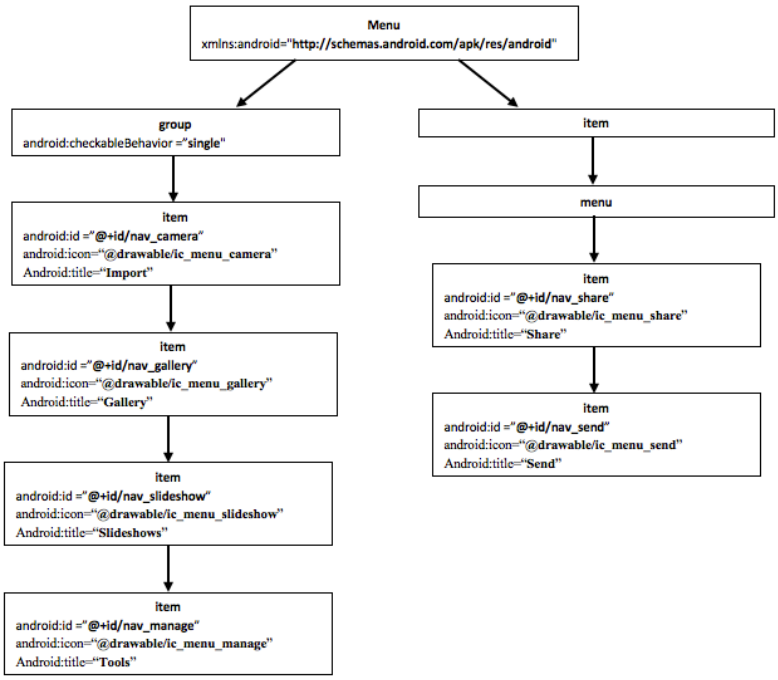
En `app_bar_main.xml`, tenemos un Appbar, un `include` y un `FloationActionButton` (FAB). El FAB es un botón redondo, colocado en la parte inferior derecha de la pantalla, a través del atributo `android:gravity="bottom|end"`. En el siguiente capítulo daremos más detalles sobre el FAB.

El elemento `include` carga el layout `content_main`. Este layout ocupa toda la pantalla de la aplicación, y es aquí donde se coloca el contenido personalizado de la aplicación.

En `"app/res/menu"`, encontramos los menús de la aplicación. Las opciones que se muestran en el menú lateral se encuentran en el menú `activity_main_drawer`. A cada

`MenuItem` se le asigna un `id`, un icono y un título. El atributo `id` servirá para inflarlos en el código fuente de la aplicación.

Figura 4.5 Vista jerárquica de activity_main_drawer de la plantilla Navigation Drawer



Código fuente del Navigation Drawer

La única clase que existe en la plantilla Navigation Drawer es `MainActivity` y tiene cinco métodos: `create`, `onBackPressed`, `onCreateOptionMenu`, `onOptionsItemSelected`, `onNavigationItemSelected`.

En el método `onCreate` se instancia un objeto `ToolBar`, un objeto `FloatingActionButton`, un objeto `NavigationView` y un objeto `DrawerLayout`, como se muesta a continuación:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Snackbar.make(view, "Replace with your own action",
Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();
        }
    });
```

```
    DrawerLayout drawer = (DrawerLayout)
findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            this, drawer, toolbar, R.string.navigation_drawer_open,
R.string.navigation_drawer_close);
    drawer.addDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = (NavigationView)
findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}
```

Observemos que inflamos los widgets de diferentes layout. El `Toolbar` y el `FloatingActionButton` están localizados en el layout `app_bar_main`, el `DrawerLayout` y el `NavigationView` están ubicados en `activity_main`. Es una buena práctica mantener componentes de interfaz de usuario que puedan ser reutilizados en diferentes `Activities`, como es el caso del Appbar.

La plantilla Navigation Drawer trae por defecto un menú en el Appbar. En el método `onCreateOptionMenu` se infla el menú `app/res/menu/main.xml`.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
```

El método `onOptionsItemSelected` se dispara cada vez que se selecciona un ítem del menu `R.menu.main`.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

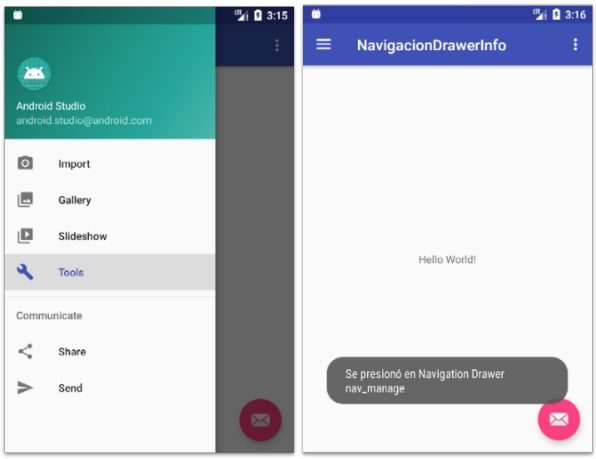
El método `onNavigationItemSelected` se activa cuando el usuario selecciona cualquier ítem del Navigation Drawer.

```
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
```

```
    int id = item.getItemId();

    if (id == R.id.nav_camera) {
        // Handle the camera action
    } else if (id == R.id.nav_gallery) {

    } else if (id == R.id.nav_slideshow) {

    } else if (id == R.id.nav_manage) {

    } else if (id == R.id.nav_share) {

    } else if (id == R.id.nav_send) {

    }

    DrawerLayout drawer = (DrawerLayout)
findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

Antes de ejecutar la aplicación, modificamos el método `onNavigationItemSelected`, como se muestra a continuación:

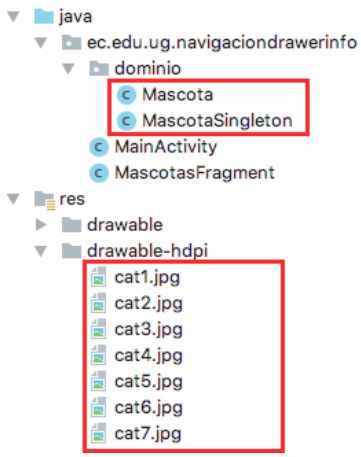
Figura 4.6 Resultado de la plantilla Navigation Drawer



Navigation Drawer con Fragment

Vamos a seguir actualizando la aplicación anterior. Tomaremos algunos componentes de la aplicación `AppbarConTabs` y la agregaremos en `NavigationDrawerInfo`. Las clases Macota, MascotaSingleton y MascotasFragment, y las imágenes de mascotas las agregamos al proyecto NavigationDrawerInfo, como se muestra a continuación:

Figura 4.7 Componentes agregados a NavigationDrawerInfo



Recordemos que en el layout `content_main` se agrega el contenido principal de la aplicación. Para cargar el fragmento de mascotas creamos un `FragmentLayout`, como se muestra a continuación:

```
<android.support.constraint.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"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/app_bar_main">

    <FrameLayout
        android:id="@+id/fragment_content"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</android.support.constraint.ConstraintLayout>
```

En la clase `MainActivity`, creamos el método setFragment para cargar un objeto `Fragment` de la clase `MascotasFragment` en el layout `fragment_content`.

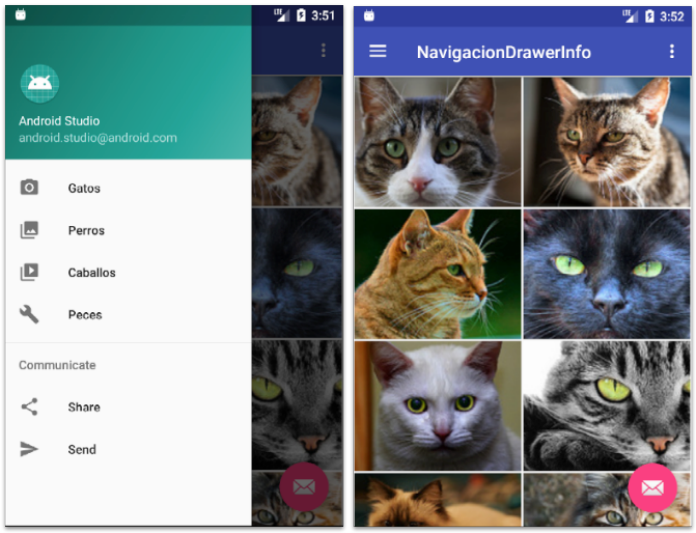
```
public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {
…
    public void setFragment(Fragment fragment){
        FragmentManager fragmentManager = getSupportFragmentManager();
        fragmentManager
                .beginTransaction()
                .replace(R.id.fragment_content, fragment)
                .commit();
    }
}
```

En `onCreate` de `MainActivity` creamos un fragmento con la lista de gatos. En `onNavigationItemSelected`, cada vez que seleccionamos un `MenuItem`, creamos un `fragment` y lo cargamos en el contenido principal de la aplicación.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    …
    Fragment fragment = MascotasFragment.newInstance(0);
    setFragment(fragment);
}

public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.nav_camera) {
        Fragment fragment = MascotasFragment.newInstance(0);
        setFragment(fragment);
    } else if (id == R.id.nav_gallery) {
        Fragment fragment = MascotasFragment.newInstance(1);
        setFragment(fragment);
    } else if (id == R.id.nav_slideshow) {
        Fragment fragment = MascotasFragment.newInstance(2);
        setFragment(fragment);
    } else if (id == R.id.nav_manage) {
        Fragment fragment = MascotasFragment.newInstance(3);
        setFragment(fragment);
    } else if (id == R.id.nav_share) {
        Toast.makeText(this,"Se presionó en Navigation Drawer
nav_share",Toast.LENGTH_LONG).show();
    } else if (id == R.id.nav_send) {
        Toast.makeText(this,"Se presionó en Navigation Drawer
nav_send",Toast.LENGTH_LONG).show();
    }

    DrawerLayout drawer = (DrawerLayout)
findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

Luego, ejecutamos la aplicación y obtenemos el siguiente resultado:

Figura 4.8 Resultado del Navigation Drawer con Fragment



Bottom Navigation

Este patrón de navegación se ha vuelto muy popular, tanto es así que aplicaciones como YouTube han implementado esta forma de navegar. Este menú permite a los usuarios cambiar de secciones sin demasiado esfuerzo, ofreciendo una gran ventaja sobre el Appbar que tiene sus opciones en la parte superior. Se recomienda tener de tres a cinco secciones para que estén siempre visibles en la pantalla.

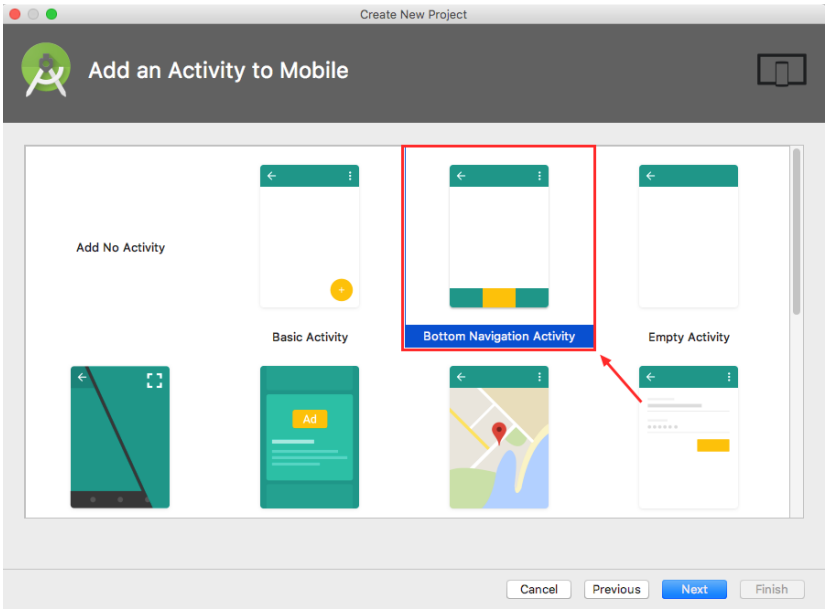
Para desarrollar el comportamiento del Bottom Navigation, utilizaremos la plantilla Bottom Navigation Activity, que viene en Android Studio y agregaremos las clases `Mascota`, `MascotaSingleton` y `MascotasFragment`, y las imágenes de mascotas del proyecto `AppbarConTabs`.

Creación y configuración de un proyecto

Para desarrollar este proyecto, realizaremos los siguientes pasos:

1. Creamos un proyecto con el nombre "BottomNavigationInfo".
2. Elegimos el SDK mínimo, el API 21: Android 5.0.
3. Seleccionamos la planatilla "Bottom Navigation Activity".

Figura 4.9 Seleccionar la plantilla Navigation Drawer



4. Creamos la `Activity` y el Layout del proyecto con el nombre `MainActivity`, y finalizamos la creación.

Desarrollando el comportamiento

El widget `BottomNavigationView` representa una barra de menú en la parte inferior de la pantalla. El contenido se puede asignar a través de un archivo de recurso menú.

Esta plantilla agrega por defecto un recurso menú llamado `navigation` con tres `MenuItem` que se utilizarán en el Bottom Navigation. Modificamos el layout `activity_main`, como se muestra a continuación:

```
<android.support.constraint.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:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <FrameLayout
        android:id="@+id/fragment_content"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <android.support.design.widget.BottomNavigationView
        android:id="@+id/navigation"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="0dp"
        android:layout_marginStart="0dp"
        android:background="?android:attr/windowBackground"
        app:layout_constraintBottom_toBottomOf="parent"
```

```
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:menu="@menu/navigation" />

</android.support.constraint.ConstraintLayout>
```

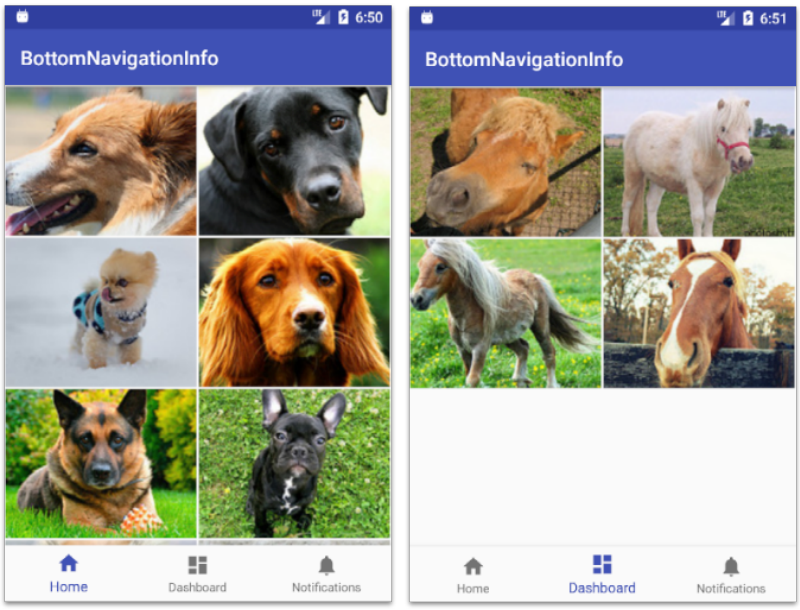
Agregamos el layout `FragmentLayout` para cargar el fragmento de mascotas y en el atributo `app:menu` del `BottomNavigationView` asignamos el menú `navigation`.

En la clase `MainActivity`, en el evento `OnNavigationItemSelectedListener` llamamos al fragmento `MascotasFragment` en cada ítem disponible en el `BottomNavigationView`.

```
public class MainActivity extends AppCompatActivity {

    private BottomNavigationView.OnNavigationItemSelectedListener
mOnNavigationItemSelectedListener
            = new
BottomNavigationView.OnNavigationItemSelectedListener() {

        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem
item) {
            switch (item.getItemId()) {
                case R.id.navigation_home:

getSupportFragmentManager().beginTransaction().replace(R.id.fragment_c
ontent, MascotasFragment.newInstance(1)).commit();
                    return true;
                case R.id.navigation_dashboard:

getSupportFragmentManager().beginTransaction().replace(R.id.fragment_c
ontent, MascotasFragment.newInstance(2)).commit();
                    return true;
                case R.id.navigation_notifications:

getSupportFragmentManager().beginTransaction().replace(R.id.fragment_c
ontent, MascotasFragment.newInstance(3)).commit();
                    return true;
            }
            return false;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        BottomNavigationView navigation = (BottomNavigationView)
findViewById(R.id.navigation);
```

```
navigation.setOnNavigationItemSelectedListener(mOnNavigationItemSelect
edListener);
    }

}
```

Figura 4.10 Resultado final del Bottom Navigation



Resumen

En este capítulo desarrollamos los comportamientos del Navigation Drawer y del Bottom Navigation. Utilizamos como base el proyecto `AppbarConTab` del capítulo 2 y cambiamos el Appbar con pestañas por un Navigation Drawer y por un Bottom Navigation.

Capítulo 5. Text Fields, SnackBars, Floating Action Buttons

El TextField es un elemento que nos permite ingresar información a través de un `EditText`. También, nos permite validar y mostrar sugerencias o mensajes de errores a los usuarios sobre la información que se ha ingresado.

El Snackbar proporciona comentarios breves sobre una operación a través de un cuadro que aparece desde la parte inferior de la pantalla.

El `FloatingActionButton` es un botón redondeado que facilita acceso rápido a las principales opciones de la pantalla actual donde esté localizado este botón.

En este capítulo, veremos las métricas de los componentes TextField, `SnackBar` y `FloatingActionButton`, desarrollaremos algunos comportamientos para validar información de entrada a través de TextFields, aplicaremos el Snackbar para mostrar mensajes consecutivos sobre un proceso, y utilizaremos componentes de terceros para implementar comportamientos del `FloatingActionButton` que no están respaldados por el Framework de Android.

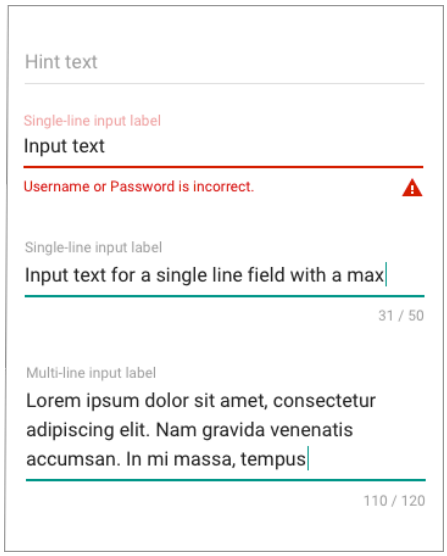
Se cubrirán los siguientes tópicos:

- TextField
- SnackBar
- FloatingActionButton con menú
- FloatingActionButton con animación

TextField

Los TextFields permiten ingresar información a la aplicación. Se puede validar información de entrada; además ayuda a los usuarios con sugerencias, mensajes de errores o autocompletado de palabras. Existen varios tipos, tales como línea simple y multilínea.

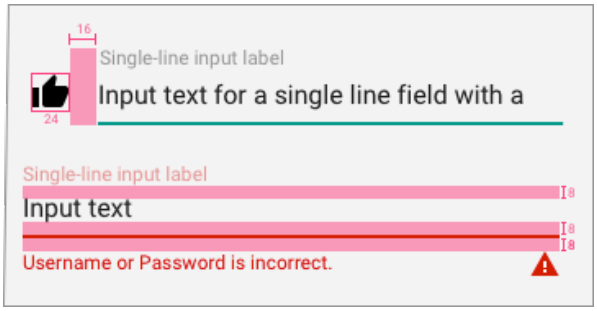
Figura 5.1 Tipos de TextField



En Android, un TextField puede estar compuesto por el widget `TextInputLayout`. Dentro de este, encontramos un `EditText`, y opcionalmente un `ImageView` con una imagen.

Material Design establece algunos valores por defecto para el TextField. Los espacios entre los Labels y el texto de entrada, el tamaño de las letras, y los colores son definidos por las guías de Material Design. El espacio entre la imagen y el `EditText` es establecido por el desarrollador.

Figura 5.2 Medidas de los TextFields



Para realizar el diseño y el comportamiento de un TextField desarrollaremos una aplicación con un formulario para ingresar información del nombre, correo electrónico y dirección de un usuario.

Creación y configuración de un proyecto con TextFields

Para desarrollar este proyecto, realizaremos los siguientes pasos:

1. Establecemos el nombre del proyecto `TextFieldsForm`.
2. Elegimos el SDK mínimo, el API 21: Android 5.0.
3. Seleccionamos un proyecto `Empty Activity`.
4. Creamos la `Activity` y el Layout del proyecto con el nombre `MainActivity`.

5. En Gradle `Scripts/build.gradle (Module:app)`, agregamos la librería de compatibilidad Design y sincronizamos.

```
…
implementation 'com.android.support:design:27.1.1'

…
```

5. En https://drive.google.com/file/d/1nsrM-w_S0rf1gYAw-BUSscgyj5wemIxd/view?usp=sharing, descargamos el Archivo.zip.
6. Copiamos las carpetas `drawable-hdpi, drawable-mdpi, drawable-xhdpi y drawable-xxhdpi y drawable-xxxhdpi` con la imagen ic_email_black.
7. Creamos los siguientes recursos String en `res/values/string.`

```
<resources>
    …

<string name="error_nombre">El Nombre es requerido</string>
<string name="error_correo_vacio">El Correo es requerido</string>
<string name="error_correo_valido">El Correo no es valido</string>

</resources>
```

Diseño del Layout

El principal componente para desarrollar los comportamientos de un TextField es el `TextInputLayout`. Dentro de este widget, se agrega un `EdtitText` y opcionalmente una imagen representativa del campo.

En el layout `activity_main`, borramos el contenido actual y agregamos los siguientes componentes:

```
<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:id="@+id/linearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:padding="16dp">

    <android.support.design.widget.TextInputLayout
        android:id="@+id/til_nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <EditText
            android:id="@+id/edt_nombre"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Nombre *"
```

```
            android:inputType="text"/>
    </android.support.design.widget.TextInputLayout>

</LinearLayout>
```

Hemos creado el primer campo del formulario a través de un TextField. El campo nombre mostrará al usuario la etiqueta "Nombre". Este valor se establece en el atributo `android:hint="Nombre *"` del `EditText`. A continuación, agregamos el campo correo.

```
…
    </android.support.design.widget.TextInputLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/ic_email_black_24dp"
            android:layout_gravity="center_vertical"/>
        <android.support.design.widget.TextInputLayout
            android:id="@+id/til_correo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="16dp">
            <EditText
                android:id="@+id/edt_correo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:hint="Correo Electrónico *" />
        </android.support.design.widget.TextInputLayout>
    </LinearLayout>
…
```

Para agregar una imagen al `TextField`, establecemos un `LinearLayout` como padre de un `ImageView` y un `TextInputLayout`. Observe que asignamos un margen izquierdo de `16dp` en el `TextInputLayout`. Agregamos el último campo en el formulario y un botón.

```
…
<android.support.design.widget.TextInputLayout
    android:id="@+id/til_direccion"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <EditText
        android:id="@+id/edt_direccion"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Dirección"
        android:inputType="text"/>
</android.support.design.widget.TextInputLayout>
```

```
<Button
    android:id="@+id/btn_guardar"
    android:text="Guardar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Desarrollando el código

En la clase `MainActivity`, inflamos los `TextInputLayout`, un `Botton` y los `EditText`, como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {

    private TextInputLayout tilNombre;
    private TextInputLayout tilCorreo;
    private EditText edtNombre;
    private EditText edtCorreo;
    private EditText edtDireccion;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tilNombre = findViewById(R.id.til_nombre);
        edtNombre = findViewById(R.id.edt_nombre);

        tilCorreo = findViewById(R.id.til_correo);
        edtCorreo = findViewById(R.id.edt_nombre);

        edtDireccion = findViewById(R.id.edt_direccion);

        Button btnGuardar = findViewById(R.id.btn_guardar);
    }

}
```

Para validar el campo "Nombre", codificamos el método `validarNombre`, que recibe un `String` como parámetro de entrada.

```
…
public boolean validarNombre(String nombre){
    if(nombre.isEmpty()) {
        tilNombre.setError(this.getResources()
                .getString(R.string.error_nombre));
        return false;
    }
    tilNombre.setError(null);
    return true;
}
```

Dentro del método `validarNombre`, si el nombre está vacío, entonces en el método `setError` del objeto `tilNombre` establecemos el mensaje `R.string.error_nombre`, caso contrario, será `null` y no se mostrará ningún mensaje de error en el formulario de la aplicación.

Para validar el correo, necesitamos verificar que este campo no esté vacío y que su contenido sea un correo electrónico válido. Creamos el método `correoVacio` y `correoValido`.

```
…
public boolean correoVacio(String correo){
    if(correo.isEmpty()) {
        tilCorreo.setError(this.getResources()
                  .getString(R.string.error_correo_vacio));
        return false;
    }
    tilCorreo.setError(null);

    return true;
}

public boolean correoValido(String correo) {
    boolean correoValido = android.util.Patterns.EMAIL_ADDRESS
                           .matcher(correo).matches();
    if(!correoValido){

tilCorreo.setError(this.getResources().getString(R.string.error_correo
_valido));
        return false;
    }
    return true;
}
…
```

En el método validarCorreo, si el parámetro `correo` es vacío se muestra el mensaje de error `R.string.error_correo_vacio` en el TextField. En el método `correoValido`, se verifica que el parámetro `correo` tenga el formato de un correo electrónico. Si este no cumple con el formato de un correo electrónico, se muestra el mensaje de error `R.string.error_correo_valido`.

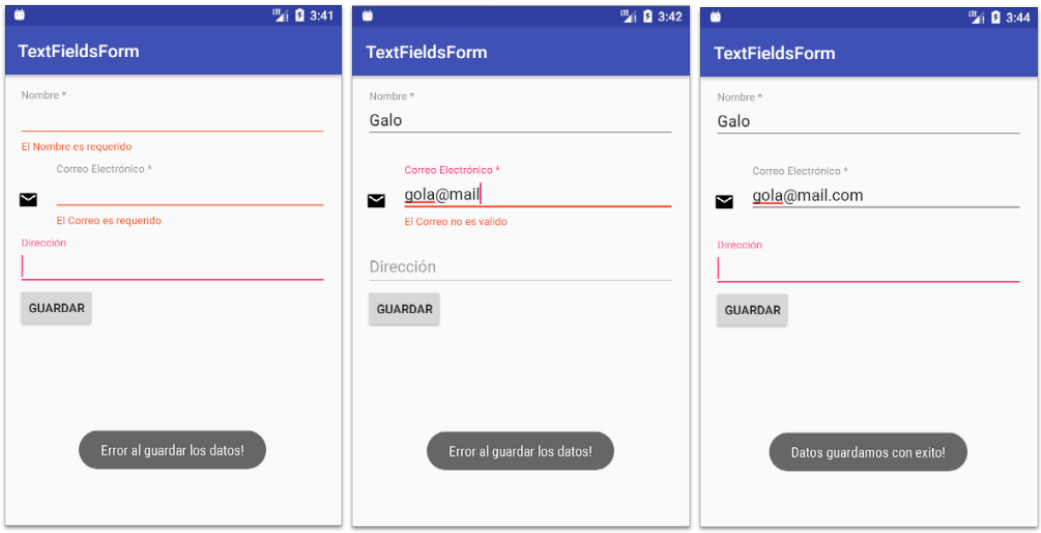
Por último, establecemos el evento clic en el botón `btnGuardar` y creamos el método `validarCampos`, como se muestra a continuación:

```
…
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    …
    Button btnGuardar = findViewById(R.id.btn_guardar);
    btnGuardar.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
        public void onClick(View view) {
            validarCampos();
        }
    });
}
…
public void validarCampos(){
    String nombre = edtNombre.getText().toString();
    String correo = edtCorreo.getText().toString();

    boolean nombreValido = validarNombre(nombre);
    boolean correoValido = correoVacio(correo);
    if(correoValido)
        correoValido = correoValido(correo);

    if(nombreValido && correoValido)
        Toast.makeText(this,"Datos guardamos con
exito!",Toast.LENGTH_LONG).show();
    else
        Toast.makeText(this,"Error al guardar los
datos!",Toast.LENGTH_LONG).show();
}
…
```

En el método `validarCampos`, se obtienen los textos de `edtNombre` y `edtCorreo`. Luego, se pasan esos valores a los métodos `validarNombre`, `correoVacio` y `correoValido`. Si todos los datos son correctos, se muestra el mensaje "Datos guardados con éxito" en un `Toast`. Si no, se muestra "Error al guardar los datos".

Figura 5.3 Resultado final del TextFieldForm



SnackBar

El Snackbar provee comentarios breves sobre una operación a través de un cuadro que aparece desde la parte inferior de la pantalla.

Figura 5.4 Medidas de un Snackbar

En Android, un Snackbar tiene predeterminado el tamaño y los espacios de las letras. Para llevar el diseño y el comportamiento del widget SnackBar, desarrollaremos una aplicación con dos SnackBars consecutivos con diferentes contenidos.

Creación y configuración de un proyecto con SnackBar

Para desarrollar este proyecto, realizaremos los siguientes pasos:

1. Establecemos el nombre del proyecto `SnackBar`.
2. Elegimos el SDK mínimo, el API 21: Android 5.0.
3. Seleccionamos un proyecto `Empty Activity`.
4. Creamos la `Activity` y el Layout del proyecto con el nombre `MainActivity`.
5. En Gradle `Scripts/build.gradle (Module:app)`, agregamos la librería de compatibilidad Design y sincronizamos.

```
…
implementation 'com.android.support:design:27.1.1'

...
```

6. Desargamos cualquier ícono en https://material.io/icons/ y lo agregamos al proyecto.

Diseño del Layout

Cuando una pantalla tiene un `FloatingActionButton` en la parte inferior derecha, Material Design sugiere que un Snackbar no se sobreponga con el `FloatingActionButton`. Para lograr este comportamiento, utilizamos el layout padre `CoordinatorLayout`. Este layout permite mover automáticamente cualquier widget hacia arriba, evitando que se sobrepongan. En `activity_main`, agregamos los siguientes componentes:

```
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/parent"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:src="@drawable/ic_add_black_18dp"
        android:layout_margin="16dp"
        android:layout_gravity="bottom|end"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</android.support.design.widget.CoordinatorLayout>
```

Para mostrar un Snackbar en la pantalla, se le debe asignar un contenedor padre, es por esta razón que asignamos un `id` al elemento `CoordinatorLayout` para inflarlo y asignarlo a los Snackbars que vamos a utilizar. El `FloatingActionButton` nos servirá para activar un SnackBar dentro de la aplicación. Este estará colocado en la parte inferior derecha de la pantalla.

Desarrollando el código

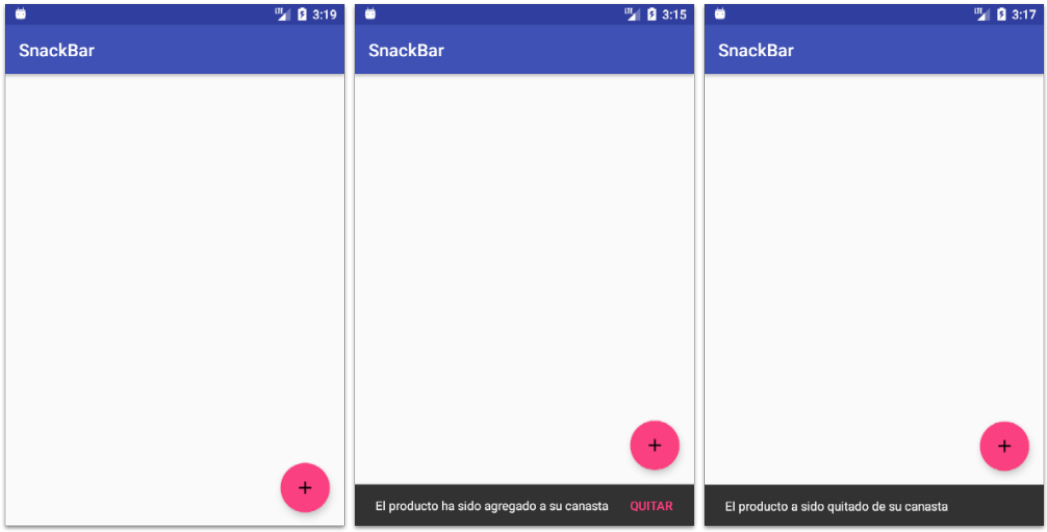
Para mostrar consecutivamente dos Snackbars, los inflamos en `MainActivity`, y establecemos los mensajes que se van a mostrar por la pantalla. Para activar un Snackbar, utilizaremos el evento clic del `FloatingActionButton fab`, como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final CoordinatorLayout parent = findViewById(R.id.parent);
```

```
        FloatingActionButton fab = findViewById(R.id.fab);

        final Snackbar snb1 = Snackbar.make(parent,"El producto ha
sido agregado a su canasta",Snackbar.LENGTH_LONG);
        final Snackbar snb2 = Snackbar.make(parent,"El producto a sido
quitado de su canasta",Snackbar.LENGTH_LONG);

        snb1.setAction("Quitar", new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                snb2.show();
            }
        });

        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                snb1.show();
            }
        });
    }
}
```

El método `make` recibe como parámetros de entrada la vista padre, el mensaje que se va a mostrar y el tiempo que tardará en estar en la pantalla el Snackbar. El método `show` muestra el Snackbar en la pantalla. En el objeto `snb1`, establecemos el evento clic. Cuando se active este evento se mostrará el Snackbar `snb2`. A continuación, se muestra el resultado de este proyecto:

Figura 5.5 Resultado final del SnackBar

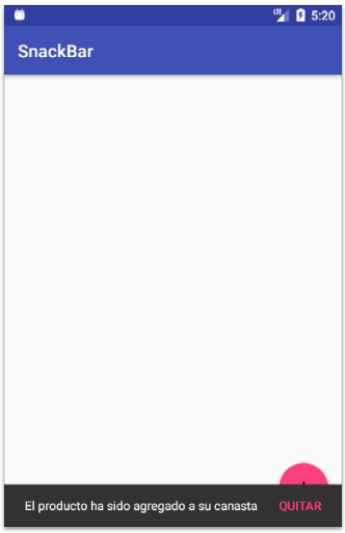


Para observar cómo se comporta el Snackbar con otros layouts, cambiamos el `CoordinatorLayout` por un `RelativeLayout`.

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/parent"
    …>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        …
        />

</RelativeLayout>
```

Al ejecutar la aplicación, podemos observar que el Snackbar se sobrepone con el `FloatingActionButton`.

Figura 5.6 Snackbar con un RelativeLayout



FloatingActionButton con menú

El comportamiento del `FloatingActionButton` con menú definido en Material Design, no está integrado en el framework de Android. Sin embargo, podemos utilizar algunos componentes de otros desarrolladores, como por ejemplo, el componente desarrollado en https://github.com/futuresimple/android-floating-action-button.

Configuración del Proyecto App Bar con espacio flexible

Para desarrollar el FloatingActionButton con menú tomaremos como proyecto base `AppBarConEspacioFlexible`. Realizamos los siguientes pasos:

1. En `Gradle Scripts/build.gradle (Module:app)`, agregamos la biblioteca de compatibilidad Design y sincronizamos.

```
dependencies {
    …
```

```
    implementation 'com.getbase:floatingactionbutton:1.10.1'
}
```

Diseño del Layout

En el layout `activity_main`, eliminamos el actual `FloatingActionButton`, y agregamos el nuevo componente `com.getbase.floatingactionbutton.FloatingActionsMenu`, como se muestra a continuación:

```
<android.support.design.widget.CoordinatorLayout
…
    <android.support.v4.widget.NestedScrollView
        …
    </android.support.v4.widget.NestedScrollView>

    <com.getbase.floatingactionbutton.FloatingActionsMenu
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:layout_margin="16dp"
        app:fab_addButtonSize="normal">

        <com.getbase.floatingactionbutton.FloatingActionButton
            android:id="@+id/fab_file"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:fab_colorNormal="@android:color/white"
            app:fab_colorPressed="@android:color/darker_gray"
            app:fab_icon="@drawable/ic_insert_drive_file_black_24dp"
            app:fab_size="mini" />

        <com.getbase.floatingactionbutton.FloatingActionButton
            android:id="@+id/fab_folder"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:fab_colorNormal="@android:color/white"
            app:fab_colorPressed="@android:color/darker_gray"
            app:fab_icon="@drawable/ic_folder_black_24dp"
            app:fab_size="mini" />
    </com.getbase.floatingactionbutton.FloatingActionsMenu>

</android.support.design.widget.CoordinatorLayout>
```

El widget `FloatingActionsMenu` actúa como contenedor de los widgets `com.getbase.floatingactionbutton.FloatingActionButton`, y será el botón principal que se muestre en la pantalla. El atributo app:fab_addButtonSize, establece el tamaño de los botones. Existen dos tamaños: normal y mini. Se establecen los atributos `id` en cada `FloatingActionButton` para inflarlos en el código fuente y asignarles un evento clic a cada uno.

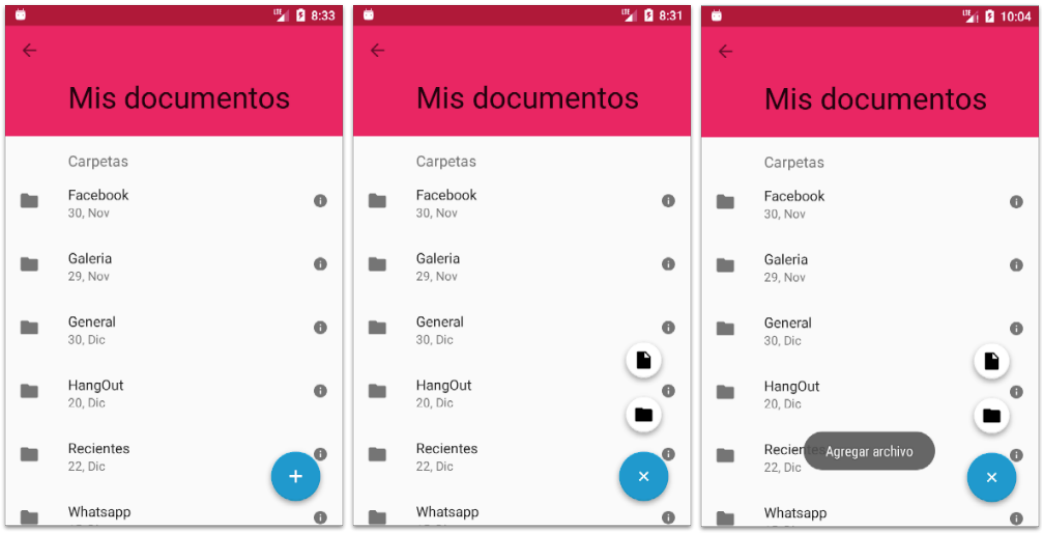
Desarrollando el código

Inflamos los `FloatingActionButtons` si establecemos el evento clic a cada uno, como se muestra a continuación:

```
…
import com.getbase.floatingactionbutton.FloatingActionButton;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        …
        FloatingActionButton fabFile = findViewById(R.id.fab_file);
        fabFile.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this,
                     "Agregar archivo",Toast.LENGTH_SHORT).show();
            }
        });

        FloatingActionButton fabFolder =
findViewById(R.id.fab_folder);
        fabFolder.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast.makeText(MainActivity.this,
                     "Agregar carpeta",Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Tenga en cuenta que el `FloatingActionButton` de `com.getbase` tiene el mismo nombre que el `FloatingActionButton` que viene en el SDK de Android. Esto puede causar errores al momento de importar el widget de `com.getbase`. Cuando ejecutamos el proyecto, tenemos el siguiente resultado:

Figura 5.7 Resultado final del FloatingActionButton con menú



FloatingActionButton con animación

Material Design recomienda que si un FloatingActionButton está presente en múltiples pantallas laterales (como las pantallas con un Appbar con pestañas), debería desaparecer brevemente, luego reaparecer si cambia su acción. Si la acción es la misma entre pantallas, el botón debe permanecer fijo.

Para llevar a cabo este comportamiento, modificaremos el proyecto AppbarConTabs, y le agregaremos un FloatingActionButton, para que cada vez que cambiemos las pestañas de las mascotas, este cambiará de ícono, desaparecerá y reaparecerá en la pantalla de la aplicación.

Configuración del Proyecto App Bar con espacio flexible

Para desarrollar el `FloatingActionButton` con animación, tomaremos como proyecto base `AppBarConTabs`. Realizamos los siguientes pasos:

1. En https://drive.google.com/file/d/1qeyMDx2l2UPYpGgmE4NOKLnwMWzPUNQT/view?usp=sharing descargar el Archivo.zip y agregamos los íconos en la aplicación.

Diseño del Layout

En el layout `activity_main`, agregamos un `FloatingActionButton`, como se muestra a continuación:

```
<android.support.design.widget.CoordinatorLayout
…>
    <android.support.design.widget.AppBarLayout
    …>

        <android.support.v7.widget.Toolbar
      …/>

        <android.support.design.widget.TabLayout
```

```
      …/>
    </android.support.design.widget.AppBarLayout>

    <android.support.v4.view.ViewPager
      …/>

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|bottom"
        android:layout_margin="16dp"
        android:src="@drawable/ic_camera" />
</android.support.design.widget.CoordinatorLayout>
```

Desarrollando el código

Para lograr que el FloatingActionButton cambie de ícono cada vez que cambiamos de pestaña, crearemos un array de ícono e inflaremos el botón, como se muestra a continuación:

```
public class MainActivity extends AppCompatActivity {

    private FloatingActionButton fab;
    int[] iconIntArray = {R.drawable.ic_camera, R.drawable.ic_send,
R.drawable.ic_photo, R.drawable.ic_share};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
      …

        fab = findViewById(R.id.fab);
    }
}
```

Para asignar animaciones al `FloatingActionButton`, crearemos el método animateFab en la clase `MainActivity`. En este método, tiene como variable de entrada la posición de la pestaña que se cambia. De esta forma, establecemos el ícono correspondiente a cada pestaña. También, agregaremos las animaciones correspondientes para que el FloatingActionButton desaparezca y reaparezca cada vez que se cambie de pestaña.

```
private void animateFab(final int position) {
    fab.clearAnimation();

    ScaleAnimation shrink =  new ScaleAnimation(1f, 0.2f, 1f, 0.2f,
Animation.RELATIVE_TO_SELF, 0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
    shrink.setDuration(150);     // animation duration in milliseconds
    shrink.setInterpolator(new DecelerateInterpolator());
    shrink.setAnimationListener(new Animation.AnimationListener() {
```

```
        @Override
        public void onAnimationStart(Animation animation) { }

        @Override
        public void onAnimationEnd(Animation animation) {
            //Establecer el ícono

fab.setImageDrawable(getResources().getDrawable(iconIntArray[position]
));

            // Animación de rotación
            Animation rotate = new RotateAnimation(60.0f, 0.0f,
                    Animation.RELATIVE_TO_SELF, 0.5f,
Animation.RELATIVE_TO_SELF,
                    0.5f);
            rotate.setDuration(150);
            rotate.setInterpolator(new DecelerateInterpolator());

            // Scale up animation
            ScaleAnimation expand = new ScaleAnimation(0.1f, 1f, 0.1f,
1f, Animation.RELATIVE_TO_SELF, 0.5f, Animation.RELATIVE_TO_SELF,
0.5f);
            expand.setDuration(150);     // animation duration in
milliseconds
            expand.setInterpolator(new DecelerateInterpolator());

            // Agregar ambas animacion a animationState
            AnimationSet s = new AnimationSet(false); //false means
don't share interpolators
            s.addAnimation(rotate);
            s.addAnimation(expand);
            fab.startAnimation(s);
        }

        @Override
        public void onAnimationRepeat(Animation animation) { }
    });
    fab.startAnimation(shrink);
}
```

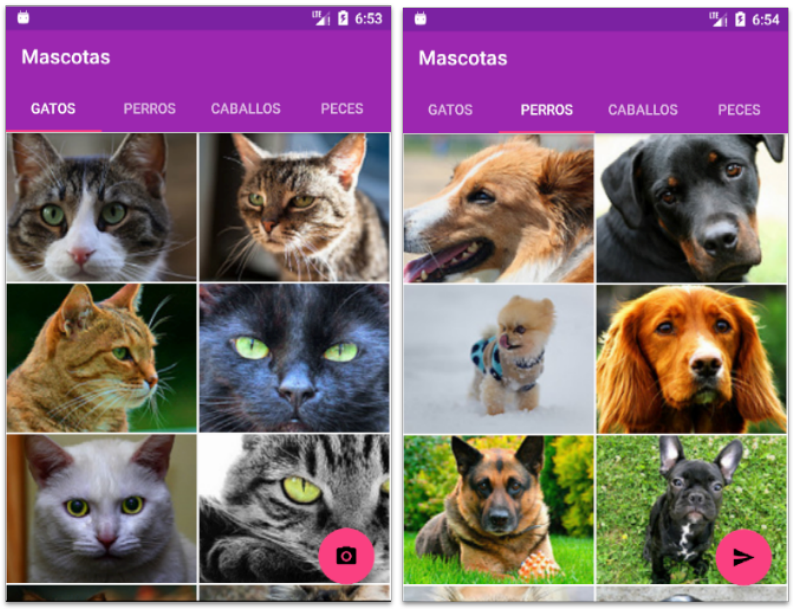
Por último, establecemos el evento `addOnTabSelectedListener` en el `tabLayout`. Este evento se dispara cada vez que se cambia de pestaña. En `onTabSelected` llamamos al método animaFab, lo que permite establecer la animación al `FloatingActionButton`.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    …

    TabLayout tabLayout = (TabLayout) findViewById(R.id.tab_layout);
    tabLayout.setupWithViewPager(viewPager);
    tabLayout.addOnTabSelectedListener(new
```

```
TabLayout.OnTabSelectedListener() {
        @Override
        public void onTabSelected(TabLayout.Tab tab) {
            animateFab(tab.getPosition());
        }

        @Override
        public void onTabUnselected(TabLayout.Tab tab) { }

        @Override
        public void onTabReselected(TabLayout.Tab tab) { }
    });
}
```

A continuación, se muestra el resultado final de este proyecto.

Figura 5.8 Resultado final del FloatingActionButton con animación



Resumen

En este capítulo vimos las métricas de los componentes TextField, `SnackBar` y `FloatingActionButton`. Desarrollamos algunos comportamientos para validar información de entrada a través de TextFields. Aplicamos el Snackbar para mostrar mensajes consecutivos sobre un proceso. Utilizamos componentes de terceros para implementar comportamiento del `FloatingActionButton` que no están respaldados por el Framework de Android.

Capítulo 6. Material Desing en las principales aplicaciones móviles de redes sociales

En este capítulo, analizaremos la navegabilidad y la forma en que se muestra la información en las aplicaciones móviles Whatsapp, Twitter, Instagram y Facebook en la plataforma Android y qué tan alineadas están a las guías de Material Design.

Se cubrirán los siguientes tópicos:

- Navegabilidad y la forma de presentar información
- Material Design y Whatsapp
- Material Design y Twitter
- Material Design e Instagram
- Material Design y Facebook

Navegabilidad y la forma de presentar información

Los elementos Appbar, Button Navigation y Navigation Drawer definen la estructura visual y los patrones de navegación de las aplicaciones móviles en la plataforma Android. Por otro lado, las listas son la mejor forma de representar información homogénea a través de un conjunto de datos, como textos o imágenes.

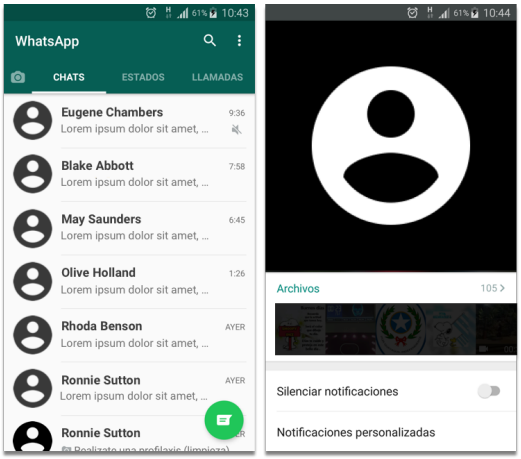
A lo largo de este libro, hemos desarrollado aplicaciones utilizando los elementos Appbar, Button Navigation, Navigation Drawer, Floating Action Button y algunas listas de contenido. También, existen otras aplicaciones que utilizan estos mismos componentes alineados a las guías de Material Design, por ejemplo, las aplicaciones móviles de las principales redes sociales. Muchas de estas aplicaciones móviles se han adaptado gradualmente a las guías de Material Design. En este capítulo, analizaremos las aplicaciones Whatsapp, Twitter, Instagram y Facebook.

Whatsapp con Material Design

Whatsapp ha ido adaptando las guías de Material Design paulatinamente desde la versión 2.12.84. En la actualidad, observamos que la pantalla principal de Whatsapp en los dispositivos Android implementa un Appbar con Tabs para dividir su contenido en chats, estados y llamadas. Además, implementa un menú con opciones personalizadas por cada

pestaña. Para mostrar los chats, estados y llamadas, la aplicación utiliza una lista de dos líneas con un ícono. Para visualizar más detalles de un contacto o grupo, la información se muestra en una ventana con un Appbar con espacio flexible e imagen.

Figura 6.1 Interfaz de usuario de Whatsapp



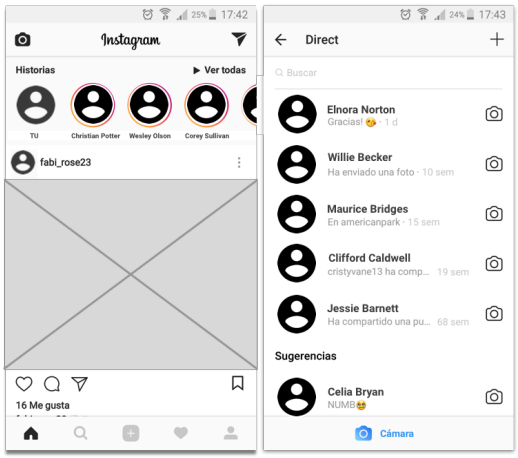
Esta aplicación utiliza un `FloatingActionButton` para resaltar las principales funcionalidades de cada sección. Está ubicada en la parte inferior derecha de la pantalla debido a su proximidad del dedo pulgar. Esto favorece un acceso rápido a estas funciones.

Normalmente, se utiliza un Appbar con espacio flexible para mostrar los detalles de un ítem de una lista, como es el caso de la información de un grupo o contacto. Cabe recalcar que Material Design no obliga a utilizar sus métricas o comportamiento. Material Design solo sugiere utilizar elementos visuales comunes con otras aplicaciones, lo que facilita una rápida adaptación de los usuarios.

Instagram con Material Design

La estructura visual en la pantalla principal de Instagram está definida por un Button Navigation. Se muestran las principales opciones de navegabilidad de esta aplicación. Utiliza algunos tipos de listas de contenidos definidas por Material Design, por ejemplo, para mostrar los contactos con su última actualización, utiliza una lista de dos líneas con dos íconos.

Figura 6.2 Interfaz de usuario de Instagram

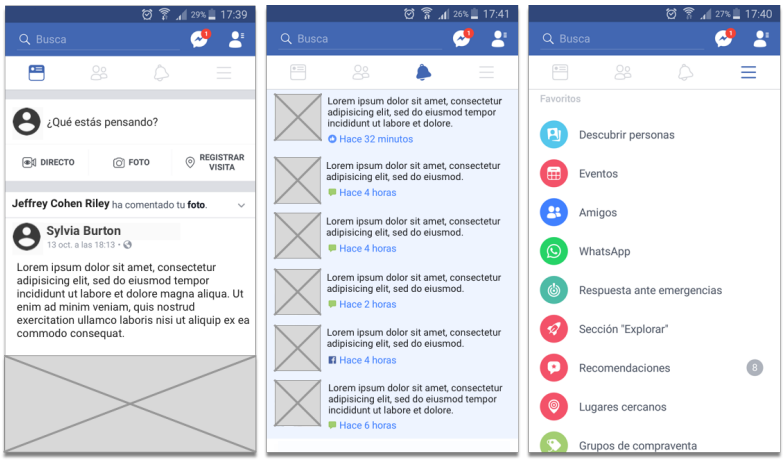


El widget Button Navigation es la forma de navegación más utilizada en las aplicaciones iOS. En Android, este componente se ha vuelto muy popular. Algunas aplicaciones como YouTube, han implementado este patrón de navegación en sus aplicaciones móviles, ya que al estar en la parte inferior de la pantalla, permite a los usuarios cambiar las secciones sin demasiado esfuerzo.

Facebook con Material Design

La aplicación móvil de Facebook, al igual que las demás aplicaciones, ha ido adaptando su diseño a las guías de Material Design. La navegabilidad de Facebook está definida por una Appbar con Tabs. Utiliza algunas listas definidas en Material Design para mostrar contactos, últimas actualizaciones, etc.

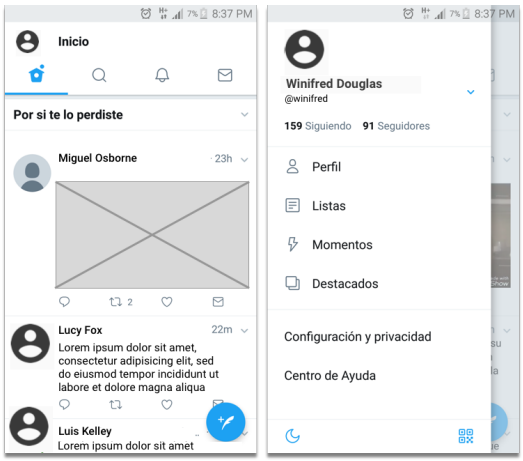
Figura 6.3 Interfaz de usuario de Facebook



Twitter con Material Design

Twitter utiliza un Appbar con pestañas, un Navigation Drawer y un Floating Action Button. En la Appbar tenemos un Home Button, una lupa que busca información dentro de la aplicación, una sección de notificaciones y una sección para mensajes. Podemos observar que esta aplicación no integra la búsqueda a través de un SearchView. Tiene una sección en el Appbar dedicada a la búsqueda y a filtrar información.

Figura 6.4 Interfaz de usuario de Twitter



Resumen

En este capítulo, analizamos la navegabilidad y cómo se muestra la información en las aplicaciones móviles Whatsapp, Twitter, Instagram y Facebook en la plataforma Android.

Bibliografía

Bill Phillips, C. S. (2015). *Android Programming The Big Nerd Ranch Guide 2ND Edition.* Atlanta: Big Nerd Ranch.

Material Design. *AppBar*. Obtenido de https://material.io/develop/android/components/app-bar-layout/

Material Design. *BottomNavigation.* Obtenido de https://material.io/develop/android/components/bottom-navigation-view/

Material Design. *BottomSheetDialog.* Obtenido de https://material.io/develop/android/components/bottom-sheet-dialog-fragment/

Material Design. *Collapsing Toolbar Layout*. Obtenido de https://material.io/develop/android/components/collapsing-toolbar-layout/

Material Design. *FloatingActionButtons*. Obtenido de https://material.io/develop/android/components/floating-action-button/

Material Design. *Navigation View*. Obtenido de https://material.io/develop/android/components/navigation-view/

Material Design. *Snackbar*. Obtenido de https://material.io/develop/android/components/snackbar/

Material Design. *TabLayout*. Obtenido de https://material.io/develop/android/components/tab-layout/

Material Design. *Text Input Layout*. Obtenido de https://material.io/develop/android/components/text-input-layout/

Pablo Perea, P. G. (2017). *UX Design for Mobile.* Birmingham: Packt Publishing Ltd.

Ruiz, A. P. (2015). *Mastering Android Application Development.* Birmingham: Packt Publishing Ltd.

César Andrés Alcívar Aray, Ingeniero en Sistemas por la Universidad Laica Eloy Alfaro de Manabí (Ecuador). Master en Gestión de Proyectos, Escuela Superior Politécnica del Litoral. Actualmente, desempeña el cargo de docente investigador en la Universidad de Guayaquil, en la carrera de Ingeniería de Sistemas en Información, donde imparte cursos de programación móvil en la plataforma Android. Posee más de cinco años de experiencia desarrollando aplicaciones móviles y es un apasionado por las nuevas tecnologías de diseño y desarrollo de aplicaciones.

Angel Marcel Plaza Vargas, Ingeniero en Computación especialización Sistemas Tecnológicos de la Escuela Superior Politécnica del Litoral - Facultad de Ingeniería Eléctrica y Computación (Ecuador). Máster en Modelado Computacional en Ingeniería, Universidad de Cádiz (España). Cursando Programa de Doctorado en Tecnologías de la información y la Comunicación en la Universidad de Granada (España). Área de trabajo en microcontroladores, electrónica, robótica, IOT, Ingeniería de software, agentes inteligentes, gamificación, procesamiento digital de imágenes y minería de datos. Proyectos: Base de datos para la definición de escenarios de la contaminación petrolera en Ecuador, Análisis antropométrico en la detección de posibles talentos deportivos ecuatorianos, Capacitación en Tecnologías de Información y Comunicación en Otavalo y Galápagos. Actualmente docente Investigador de la Facultad de Ingeniería Industrial de la Universidad de Guayaquil.

Michelle Agustina Varas Chiquito, Ingeniera en Sistemas Computacionales de la Escuela Universidad de Guayaquil - Facultad de Ciencias Matemáticas y Físicas (Ecuador). Máster en Diseño Curricular , Universidad de Guayaquil (Ecuador). Área de trabajo en Ingeniería de software, Diseño curricular, IT. Docente de la Universidad Laica Vicente Rocafuerte escuela de Ciencias Contables, Universidad de Guayaquil Facultad de Ciencias Matemáticas y Físicas carrera Ingeniería Civil, actualmente docente gestora de Educación Continua de la Facultad de Ingeniería Industrial de la Universidad de Guayaquil.

Michael Mario Melo Parrales, nació en el año 1993. Egresado de la carrera sistemas de información de la universidad de Guayaquil, tiene 3 años de experiencia desarrollando aplicaciones (web, móvil) en el lenguaje java. Su pasión por temas informáticos le ha llevado a conoces varios ámbitos como, redes, sistemas operativos, mantenimiento de hardware, telefonía IP, entre otros, una de sus cualidades más importantes es de enfocarse en un tema de su interés hasta solucionarlo.

compás

ISBN: 978-9942-770-99-8