Tabla de contenido

[Navigation UI 3](#_Toc212559741)

[1. Introduccion a NavigationUI 3](#_Toc212559742)

[1.1. Vincular destinos a elementos de menú 4](#_Toc212559743)

[2. Creación del proyecto 5](#_Toc212559744)

[2.1. Configuración inicial 5](#_Toc212559745)

[2.2. Añadir dependencias 5](#_Toc212559746)

[2.3. Habilitar ViewBinding 5](#_Toc212559747)

[3. Creación de las actividades y código inicial 6](#_Toc212559748)

[4. Creación de un NavController 8](#_Toc212559749)

[4.1. Uso de NavController con FragmentContainerView 8](#_Toc212559750)

[4.2. Uso de Fragment vs. FragmentContainerView 8](#_Toc212559751)

[5. Barra superior de la app (TopAppBar) 9](#_Toc212559752)

[5.1. Estructura 9](#_Toc212559753)

[5.2. Creación de barra de herramientas (Toolbar) 10](#_Toc212559754)

[5.3. AppBarConfiguration 11](#_Toc212559755)

[6. Actividad principal 12](#_Toc212559756)

[6.1. XML. Incluir el toolbar y diseñar la actividad 12](#_Toc212559757)

[6.2. Java. Asignar soporte al ToolBar 13](#_Toc212559758)

[6.3. Java. Definir comportamiento 14](#_Toc212559759)

[7. Actividad del OptionsMenu 15](#_Toc212559760)

[7.1. XML. Incluir el Toolbar 15](#_Toc212559761)

[7.2. XML. Grafo de navegación 15](#_Toc212559762)

[7.3. XML. Creación del menú 18](#_Toc212559763)

[7.4. Java. Asignar soporte al ToolBar 20](#_Toc212559764)

[7.5. Java. Definir AppBarConfiguration 20](#_Toc212559765)

[7.6. Java. Configurar NavigationUI 21](#_Toc212559766)

[7.7. Java. Incluir el menú. 22](#_Toc212559767)

[8. Actividad del NavigationView 23](#_Toc212559768)

[8.1. XML. Incluir el Toolbar 23](#_Toc212559769)

[8.2. XML. Grafo de navegación 24](#_Toc212559770)

[8.3. XML. Creación del menú 27](#_Toc212559771)

[8.4. XML. Diseño de la cabecera del Drawer 28](#_Toc212559772)

[8.5. XML. Incluir el NavigationView 29](#_Toc212559773)

[8.6. Java. Asignar soporte al ToolBar 31](#_Toc212559774)

[8.7. Java. Definir AppBarConfiguration 31](#_Toc212559775)

[8.8. Java. Configurar NavigationUI 32](#_Toc212559776)

[9. Actividad del BottomNavigationView 33](#_Toc212559777)

[9.1. XML. Incluir el Toolbar 33](#_Toc212559778)

[9.2. XML. Grafo de navegación 33](#_Toc212559779)

[9.3. XML. Creación del menú 36](#_Toc212559780)

[9.4. XML. Incluir el BottomNavigationView 37](#_Toc212559781)

[9.5. Java. Asignar soporte al ToolBar 37](#_Toc212559782)

[9.6. Java. Configurar NavigationUI 38](#_Toc212559783)

[10. Actividad de TabLayout y ViewPager2 39](#_Toc212559784)

[10.1. XML. Incluir el Toolbar 41](#_Toc212559785)

[10.2. XML. Incluir el TabLayout 42](#_Toc212559786)

[10.3. XML. Incluir el ViewPager2 43](#_Toc212559787)

[10.4. XML. Definir los fragmentos 44](#_Toc212559788)

[10.5. Java. Asignar soporte al ToolBar 44](#_Toc212559789)

[10.6. Java. Crear un Adapter 45](#_Toc212559790)

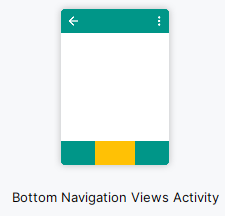
[10.7. Java. Relacionar adapter y fragmentos 46](#_Toc212559791)

# Navigation UI

## Introduccion a NavigationUI

El componente [Navigation](https://developer.android.com/guide/navigation/integrations/ui?hl=es-419) incluye una clase llamada [NavigationUI](https://developer.android.com/guide/navigation/navigation-ui), que proporciona métodos estáticos para facilitar la navegación a través de distintos elementos de la interfaz de usuario, como la barra superior de la app (Top App Bar), el panel lateral de navegación (NavigationView) y la barra de navegación inferior (BottomNavigationView).

Es importante destacar que el NavigationDrawer es un patrón de diseño que se implementa mediante un DrawerLayout, el cual permite mostrar el NavigationView como un panel lateral deslizable.



La NavigationUI actualiza automáticamente el contenido de la barra superior a medida que los usuarios navegan por la app. Por ejemplo, utiliza las etiquetas de destino definidas en el gráfico de navegación para ajustar el título de la barra superior según la página que se esté visualizando.

<navigation>  
 <fragment ...  
 android:label="Título de la página">  
 ...  
 </fragment>  
</navigation>

Además de los elementos mencionados, también se usará la navegación mediante pestañasa (Tabs) utilizando TabLayout y ViewPager2, aunque estos componentes no forman parte del paquete Navigation.

### Vincular destinos a elementos de menú

NavigationUI proporciona métodos útiles para vincular elementos de un menú con destinos definidos en un gráfico de navegación (navGraph.xml). Uno de los métodos clave es onNavDestinationSelected(), que permite asociar un MenuItem a un destino en el NavController.

Cuando el id del MenuItem coincide con el id de un destino en el gráfico de navegación, el NavController realiza automáticamente la navegación hacia ese destino.

A continuación, se muestra cómo definir un menú y un destino con el mismo id:

**menu.xml**

<menu xmlns:android="http://schemas.android.com/apk/res/android">  
 ...  
 <item  
 android:id="@+id/detalles\_pagina\_fragment"  
 android:icon="@drawable/ic\_detalles"  
 android:title="@string/detalles" />  
</menu>

**nav\_graph.xml**

<navigation xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 xmlns:android="http://schemas.android.com/apk/res/android" ... >  
 ...  
 <fragment android:id="@+id/detalles\_pagina\_fragment"  
 android:label="@string/detales"  
 android:name="com.example.android.myapp.DetallesFragment" />  
</navigation>

Para que los elementos del menú interactúen con el NavController, debes sobrescribir el método onOptionsItemSelected() en tu actividad y utilizar onNavDestinationSelected() de **NavigationUI**. Este método verifica si el id del elemento de menú coincide con el id de un destino en el gráfico de navegación, y si coindice, realiza la navegación automáticamente.

Fíjate en el siguiente ejemplo:

@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 NavController navController = Navigation.*findNavController*(this,  
 R.id.nav\_host\_fragment);  
 return NavigationUI.*onNavDestinationSelected*(item, navController)  
 || super.onOptionsItemSelected(item);  
}

Cuando un usuario haga clic en el elemento de menú con android:id="@+id/detalles\_pagina\_fragment", la app navegará automáticamente al fragmento con el mismo id definido en el gráfico de navegación.

## Creación del proyecto

En este proyecto se usará la Toolbar que integra mejor los elementos de navegación. Para crear el proyecto en Android Studio, sigue los siguientes pasos.

### Configuración inicial

Selecciona File > New > Activity > Empty Views Activity como plantilla para la MainActivity.

Verifica que la ActionBar (barra superior) está deshabilitada. En el archivo res/values/themes/themes.xml, comprueba que el tema de la aplicación hereda de un tema que deshabilita el ActionBar (\*.NoActionBar), como por ejemplo:

<resources xmlns:tools="http://schemas.android.com/tools"><style name="Base.Theme.NavigationUI"  
 parent="Theme.Material3.DayNight.NoActionBar"></style>  
  
 <style name="Theme.NavigationUI" parent="Base.Theme.NavigationUI" />  
</resources>

Si el tema de la aplicación necesita incluir el ActionBar, crea un estilo para la actividad sin el ActionBar:

<resources xmlns:tools="http://schemas.android.com/tools">...  
<style name="ActivityWithoutActionBar" parent="Theme.NavigationUI">  
 <item name="windowActionBar">false</item>  
 <item name="windowNoTitle">true</item>  
</style>  
</resources>

En el archivo de AndroidManifest.xml tendríamos que especifica el estilo creado:

<activity  
 android:name=".MainActivity"  
 android:exported="true"  
 android:theme="@style/ActivityWithoutActionBar">  
 ...  
</activity>

### Añadir dependencias

Añade las dependencias para incluir el componente Navigation, ViewPager2 y Material.

*dependencies* {  
 *implementation*("androidx.navigation:navigation-fragment:2.8.2")  
 *implementation*("androidx.navigation:navigation-ui:2.8.2")  
 *implementation*("androidx.viewpager2:viewpager2:1.1.0")  
 *implementation*("com.google.android.material:material:1.12.0")  
}

### Habilitar ViewBinding

Habilita el ViewBinding en el archivo build.gradle del módulo:

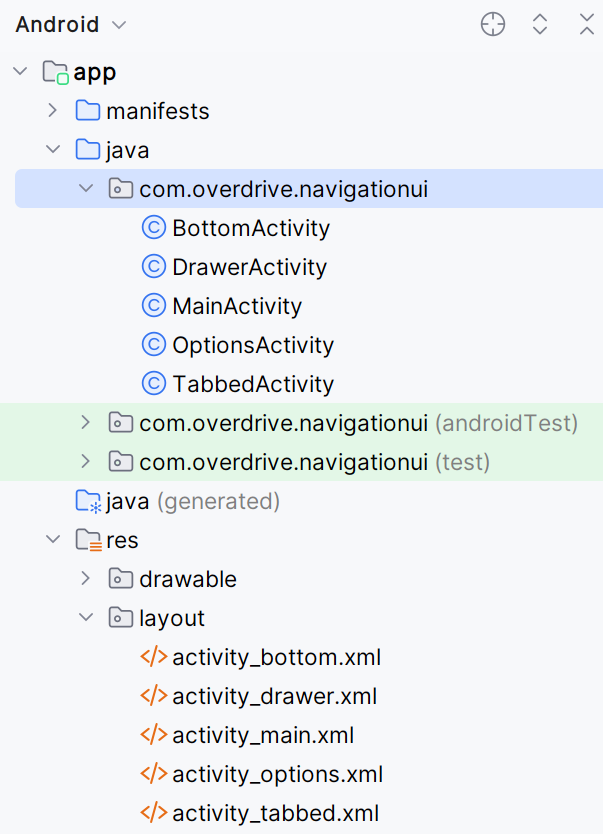
android {  
 buildFeatures {  
 viewBinding true  
 }  
}

## Creación de las actividades y código inicial

Vamos a usar el MainActivity para lanzar diferentes Activities. Cada Activity implementará un tipo distinto de elemento de navegación: NavigationView, BottomNavigationView, OptionsMenu y TabLayout+ViewPager2.

Las activities tendrán los siguientes nombres: DrawerActivity, BottomActivity, OptionsActivity y TabbedActivity.

Para crear una Activity en Android Studio selecciona New > Activity > Empty Activity:



Al usar el binding tenemos que inflar nuestros archivos de diseño de la siguiente manera:

**MainActivity.java**

public class MainActivity extends AppCompatActivity {  
 ActivityMainBinding binding;  
   
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 setContentView((binding =   
 ActivityMainBinding.*inflate*(getLayoutInflater())).getRoot());  
 }  
}

**OptionsActivity.java**

public class OptionsActivity extends AppCompatActivity {  
 ActivityOptionsBinding binding;  
   
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 setContentView((binding =   
 ActivityOptionsBinding.*inflate*(getLayoutInflater())).getRoot());  
 }  
}

**DrawerActivity.java**

public class DrawerActivity extends AppCompatActivity {  
 ActivityDrawerBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 setContentView((binding =   
 ActivityDrawerBinding.*inflate*(getLayoutInflater())).getRoot());  
 }  
}

**BottomActivity.java**

public class BottomActivity extends AppCompatActivity {  
 ActivityBottomBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
   
 setContentView((binding =   
 ActivityBottomBinding.*inflate*(getLayoutInflater())).getRoot());  
 }  
}

**TabbedActivity.java**

public class TabbedActivity extends AppCompatActivity {  
 ActivityTabbedBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 setContentView((binding =   
 ActivityTabbedBinding.*inflate*(getLayoutInflater())).getRoot());  
 }  
}

## Creación de un NavController

El NavController es uno de los [conceptos clave](https://developer.android.com/guide/navigation?hl=es-419#types) en el sistema de navegación de Android. Este controla el gráfico de navegación y proporciona los métodos necesarios para que tu app pueda desplazarse entre los destinos definidos en dicho gráfico.

Cuando utilizas el [componente **Navigation**](https://developer.android.com/reference/androidx/navigation/package-summary?hl=es-419), el [**NavController**](https://developer.android.com/reference/androidx/navigation/NavController?hl=es-419) se crea automáticamente. Para agregar un gráfico de navegación al NavController, consulta [Cómo diseñar tu gráfico de navegación](https://developer.android.com/guide/navigation/design?hl=es-419). Además, el NavController ofrece múltiples maneras de navegar a los destinos del gráfico. Puedes encontrar más detalles en [Cómo navegar a un destino](https://developer.android.com/guide/navigation/use-graph/navigate?hl=es-419).

Cada NavHostFragment que crees tendrá su propio NavController correspondiente. Este se encarga de gestionar la navegación dentro de su gráfico asociado.

Para [recuperar el **NavController**](https://developer.android.com/guide/navigation/navcontroller?hl=es-419#views) en el framework de vistas (Views), puedes usar uno de los siguientes métodos según el contexto:

* [NavHostFragment.findNavController(Fragment)](https://developer.android.com/reference/androidx/navigation/fragment/NavHostFragment?hl=es-419#findNavController(androidx.fragment.app.Fragment))
* [Navigation.findNavController(Activity, int viewId)](https://developer.android.com/reference/androidx/navigation/Navigation?hl=es-419#findNavController(android.app.Activity,%20int))
* [Navigation.findNavController(View)](https://developer.android.com/reference/androidx/navigation/Navigation?hl=es-419#findNavController(android.view.View))

### Uso de NavController con FragmentContainerView

Al utilizar FragmentContainerView para alojar un NavHostFragment, la inicialización del fragmento ocurre dentro del ciclo de transacciones del FragmentManager. Si intentas recuperar el NavController en el método onCreate(), es posible que dicha transacción aún no haya finalizado, lo que puede provocar errores de tiempo de ejecución.

Un ejemplo típico que puede fallar es:

NavController navController = Navigation.*findNavController*(  
 this, R.id.*nav\_host\_fragment*);

La solución recomendada por Android consiste en obtener primero el NavHostFragment y luego su NavController, como se muestra en el siguiente código:

NavHostFragment navHostFragment = (NavHostFragment) getSupportFragmentManager()  
 .findFragmentById(R.id.*nav\_host\_fragment*);  
  
NavController navController = navHostFragment.getNavController();

### Uso de Fragment vs. FragmentContainerView

También puedes declarar un NavHostFragment directamente como un elemento <fragment> en el layout:

<fragment  
 android:id="@+id/nav\_host\_fragment\_content\_main"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 ...  
 app:navGraph="@navigation/mobile\_navigation" />

De esta manera, no obtendrías fallos al usar el método estático findNavController de la claseNavigation.

Sin embargo, Android recomienda utilizar FragmentContainerView en los layouts modernos por su flexibilidad y compatibilidad con fragmentos administrados dinámicamente. Por lo tanto, para evitar problemas de inicialización, siempre utiliza el enfoque de NavHostFragment y NavController indicado anteriormente.

## Barra superior de la app (TopAppBar)

Las AppBars son elementos de diseño introducidos con **Material Design**, diseñadas para proporcionar un espacio coherente donde mostrar información y acciones relacionadas con la pantalla actual. Estas se implementan principalmente con los widgets: TopAppBar y BottomAppBar.

* TopAppBar: generalmente está representado por un ToolBar o MaterialToolbar. Se utiliza para mostrar la marca, títulos, navegación y acciones. También puede transformarse en una **barra de acción contextual** para ofrecer opciones específicas al seleccionar elementos.
* BottomAppBar: una barra ubicada en la parte inferior de la pantalla que permite la navegación.

Las **barras de acción contextual** permiten mostrar acciones específicas para los elementos seleccionados. La TopAppBar puede convertirse en una barra de acción contextual y permanecer activa hasta que se realice una acción o se descarte.

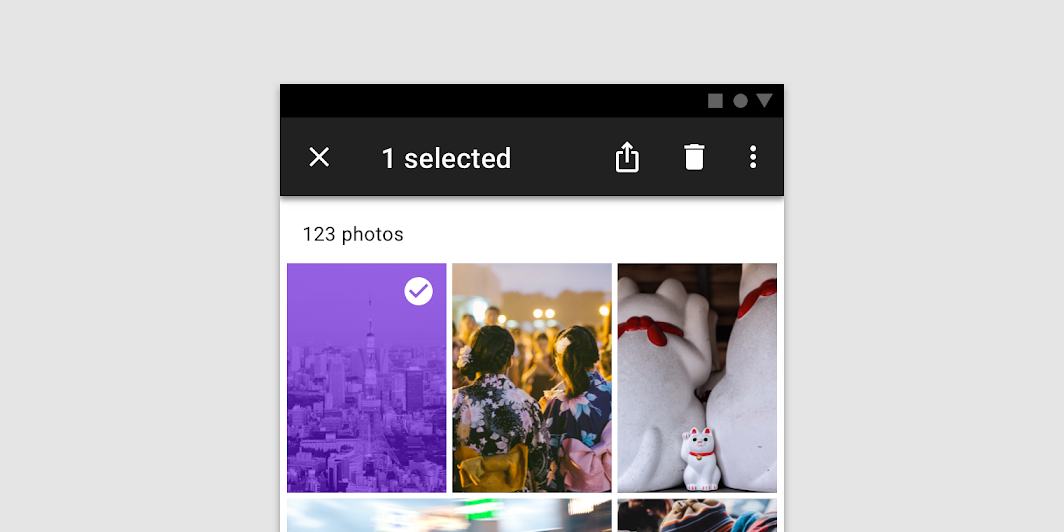
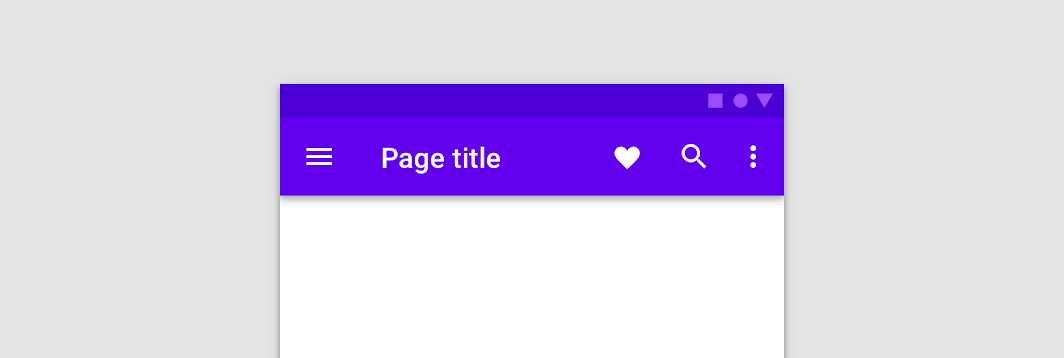


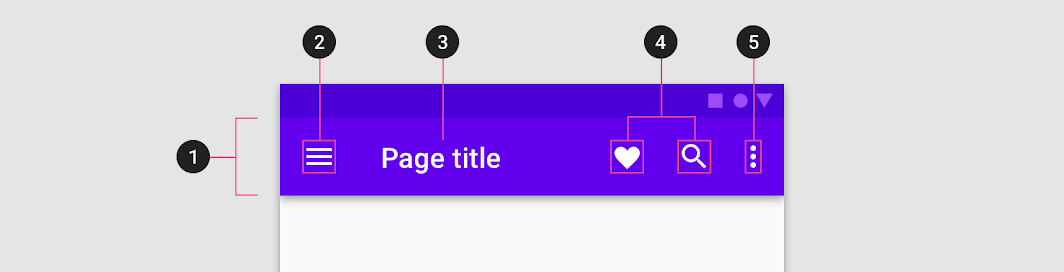
Ilustración 1. Appbar normal

Ilustración 2. Appbar contextual

La TopAppBar es un concepto implementado comúnmente en Android mediante el widget Toolbar. Introducida en la API 21 (Android 5.0), la Toolbar es más flexible y moderna que la ActionBar, lo que permite personalizaciones avanzadas y una mejor integración con **Material Design**.

En esencia, una Toolbar es una vista que hereda de ViewGroup y actúa como un contenedor, ofreciendo un espacio adaptable para incluir diversos elementos.

### Estructura



1. Contenedor (Container)
2. Botón de Navegación o botón "hamburger" (Navigation button, opcional)
3. Título (opcional)
4. Ítems de acción (Action ítems, opcionales)
5. Menú desplegable (Overflow menú, opcional). El término utilizado en el código de Android para nombrar a este elemento es OptionsMenu.

### Creación de barra de herramientas (Toolbar)

Para crear interacciones avanzadas y animaciones en la interfaz de usuario, como una Collapsing Toolbar (barra de herramientas colapsable), se recomienda usar la Toolbar junto con CoordinatorLayout y AppBarLayout.

Este patrón se utiliza comúnmente en aplicaciones modernas para mejorar la experiencia del usuario, especialmente en pantallas con imágenes o encabezados grandes que se reducen a medida que el usuario hace scroll.

Para modularizar el código creamos un archivo toolbar.xml. En su interior, definimos un AppBarLayout y, dentro de este, una Toolbar:

<com.google.android.material.appbar.AppBarLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:id="@+id/appBarLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">  
  
 <androidx.appcompat.widget.Toolbar  
 android:id="@+id/toolbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="?attr/colorPrimary"  
 app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />  
  
</com.google.android.material.appbar.AppBarLayout>

El siguiente código XML muestra una implementación básica de un CoordinatorLayout donde importamos la Toolbar mediante el nodo <include>. Le asignamos un id para poder hacer referencia a la ToolBar.

<androidx.coordinatorlayout.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">  
  
 <include android:id="@+id/include\_toolbar"   
 layout="@layout/toolbar"/>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 *<!-- Aqui va el contenido de la activity -->* </androidx.constraintlayout.widget.ConstraintLayout>  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

El CoordinatorLayout es un contenedor avanzado que coordina las interacciones entre vistas secundarias. Proporciona soporte para comportamientos dependientes, lo que significa que las vistas pueden reaccionar al comportamiento de otras vistas dentro del mismo CoordinatorLayout.

El AppBarLayout es un contenedor especial que aloja elementos de la [AppBar](https://developer.android.com/develop/ui/compose/components/app-bars?hl=es-419), como la Toolbar. Este componente es clave cuando queremos crear una barra de herramientas colapsable o deslizante, ya que puede expandirse y contraerse en función del desplazamiento de las vistas dentro del CoordinatorLayout. También se puede combinar con otros componentes como CollapsingToolbarLayout para mejorar aún más las transiciones visuales.

Dentro del AppBarLayout, encontramos la Toolbar, que actúa como la barra de acción o barra de herramientas de la aplicación. La Toolbar tiene un tema emergente (popupTheme) que afecta el estilo de los menús desplegables, como el menú de opciones o ítems de acción.

Finalmente, el contenido principal de la actividad está envuelto en un ConstraintLayout, que es un contenedor flexible para colocar las vistas. La clave aquí es el atributo app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior", que le indica al CoordinatorLayout cómo debe comportarse el ConstraintLayout durante el desplazamiento respecto del AppBarLayout. Este comportamiento permite que el contenido se desplace correctamente mientras interactúa con la barra de herramientas.

### AppBarConfiguration

La NavigationUI usa un objeto AppBarConfiguration para gestionar el comportamiento del botón de navegación ubicado en la esquina superior izquierda de la TopAppBar. Este botón cambia de apariencia y comportamiento según si el usuario está en un destino principal o en un destino más profundo dentro de la jerarquía de la navegación.

Un destino principal es como una "raíz" o el punto de partida en la aplicación, que no tiene pantallas superiores a las cuales regresar. Es decir, son aquellos que están al nivel superior de la jerarquía de navegación.

El comportamiento del botón de navegación varía dependiendo de si el destino principal está asociado a un DrawerLayout (menú lateral):

1. Con DrawerLayout: el botón de navegación se muestra como el ícono de "hamburger" (☰). Al tocarlo, abre el menú lateral que contiene las opciones de navegación.
2. Sin DrawerLayout: en un destino de nivel superior no se muestra ningún botón porque no hay pantalla anterior a la que regresar. Si el usuario navega a destinos secundarios, el botón cambia automáticamente a una flecha ← (Up Button), que permite volver a la pantalla previa.

Por defecto, solo el destino de inicio de la aplicación se considera un **destino principal**, y se configura así:

AppBarConfiguration appBarConfiguration =  
 new AppBarConfiguration.Builder(navController.getGraph()).build();

En este caso, el gráfico de navegación (navController.getGraph()) se pasa al AppBarConfiguration, lo que indica que solo el destino de inicio es de nivel superior.

En algunas aplicaciones, puedes tener varios destinos de nivel superior. Por ejemplo, en aplicaciones con una barra de navegación inferior (BottomNavigationView), diferentes pantallas pueden considerarse de nivel superior, aunque no estén jerárquicamente conectadas.

Para definir varios destinos de nivel superior, puedes pasar un conjunto de IDs de destino:

AppBarConfiguration appBarConfiguration =  
 new AppBarConfiguration.Builder(R.id.*main*, R.id.*profile*).build();

En este caso, tanto R.id.main como R.id.profile son considerados destinos de nivel superior. Cuando el usuario está en uno de estos destinos, no se mostrará el icono de flecha de izquierda ← (botón up). Si hay un DrawerLayout, se mostrará el ícono de panel lateral (☰).

## Actividad principal

El siguiente paso es añadir 4 botones en la MainActivity para que cada uno nos lleve a una de las otras Activities.

### XML. Incluir el toolbar y diseñar la actividad

En la sección anterior se ha descrito como añadir un ToolBar junto con AppBarLayout y CoordinatorLayout. A partir de ese código, en el <ConstraintLayout> añadimos cuatro botones que nos redirijen al resto de actividades.

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.coordinatorlayout.widget.CoordinatorLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include  
 android:id="@+id/include\_toolbar"  
 layout="@layout/toolbar" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <Button  
 android:id="@+id/btn\_goto\_drawerActivity"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Drawer"  
 app:layout\_constraintBottom\_toTopOf="@id/btn\_goto\_bottomActivity"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />  
  
 <Button  
 android:id="@+id/btn\_goto\_bottomActivity"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Bottom"  
 app:layout\_constraintBottom\_toTopOf="@id/btn\_goto\_optionsActivity"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/btn\_goto\_drawerActivity" />  
  
 <Button  
 android:id="@+id/btn\_goto\_optionsActivity"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Options"  
 app:layout\_constraintBottom\_toTopOf="@id/btn\_goto\_tabbedActivity"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/btn\_goto\_bottomActivity" />  
   
 <Button  
 android:id="@+id/btn\_goto\_tabbedActivity"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Tabs"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/btn\_goto\_optionsActivity" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

### Java. Asignar soporte al ToolBar

En el método onCreate() de la Activity, tenemos que añadir el soporte para la Toolbar que hemos creado mediante el método setSupportActionBar().

public class MainActivity extends AppCompatActivity {  
  
 ActivityMainBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView((binding =  
 ActivityMainBinding.*inflate*(getLayoutInflater())).getRoot());  
  
 setSupportActionBar(binding.includeToolbar.toolbar);  
 }  
}

Este método realiza varias acciones:

* **Establece la** ToolBar **como** ActionBar. Esto implica que cualquier funcionalidad del ActionBar, como mostrar título, agregar menú de opciones, y manejar la navegación, será gestionada por el ToolBar.
* **Habilita las funcionalidades de la** Toolbar. Permite personalizar la barra, incorporar botones de navegación y añadir menús mediante el método onCreateOptionsMenu().

### Java. Definir comportamiento

Ahora implementamos el comportamiento del evento clic, añadiendo un OnClickListener en cada botón, de manera que cuando se haga clic en un botón se iniciará la activity correspondiente.

Para iniciar una Activity hay que llamar al método startActivity() y pasarle un objeto de clase Intent, indicando en los parámetros del constructor el objeto de la Activity en la que estamos y la clase de la Activity que queremos iniciar.

Añade el siguiente código a la clase MainActivity:

public class MainActivity extends AppCompatActivity {  
  
 ActivityMainBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityMainBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeToolbar.toolbar);  
  
 binding.btnGotoDrawerActivity.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 startActivity(new Intent(MainActivity.this, DrawerActivity.class));  
 }  
 });  
  
 binding.btnGotoBottomActivity.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 startActivity(new Intent(MainActivity.this, BottomActivity.class));  
 }  
 });  
  
 binding.btnGotoOptionsActivity.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 startActivity(new Intent(MainActivity.this, OptionsActivity.class));  
 }  
 });  
  
 binding.btnGotoTabbedActivity.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 startActivity(new Intent(MainActivity.this, TabbedActivity.class));  
 }  
 });  
 }  
}

## Actividad del OptionsMenu

OptionsMenu es el menú desplegable (Overflow Menu) que aparece en la parte superior derecha cuando se hace clic en el icono de tres puntos ( ⋮ ).

Definiremos tres opciones en el menú, y cada una de ellas estará asociada a un destino en un grafo de navegación. En Android, no es obligatorio usar un grafo de navegación para crear un menú, pero facilita la gestión de la navegación dentro de la aplicación y permite definir transiciones entre pantallas (o fragmentos) de forma declarativa (usando XML).

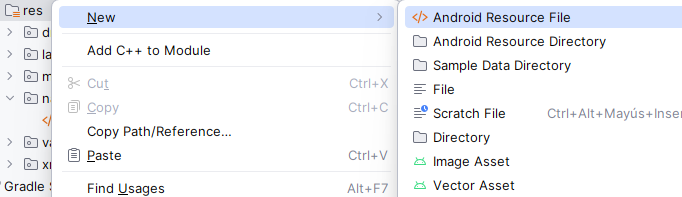
### XML. Incluir el Toolbar

Modificamos el layout de la actividad activity\_options.xml para insertar el ToolBar como se hizo con la actividad principal:

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include android:id="@+id/include\_options\_toolbar"   
 layout="@layout/toolbar"/>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 *<!-- Aqui va el contenido de la activity -->*  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

### XML. Grafo de navegación

Para crear un grafo de navegación, haz clic derecho sobre la carpeta res del proyecto y selecciona New > Android Resource File.

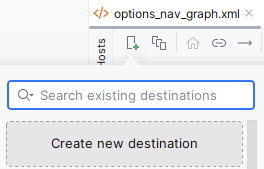


Introduce el nombre del fichero del grafo, por ejemplo: options\_nav\_graph. Y en el Resource type selecciona Navigation:

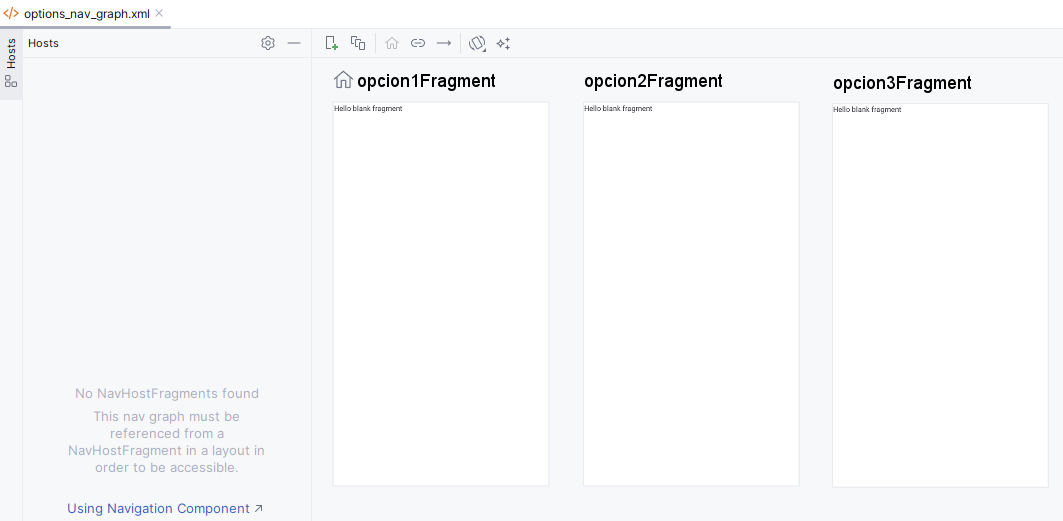


Si no existía la carpeta Navigation, entonces la creará junto el archivo options\_nav\_graph.xml.

En el grafo de navegación definimos tres destinos: opcion1Fragment, opcion2Fragment y opcion3Fragment. Haz clic en el icono de un rectángulo con un símbolo **+** para añadir cada uno de los destinos, y para cada destino eligimos un fragmento en blanco.



Después añadir los destinos, en el panel izquierdo de los **Hosts**, aparece un mensaje de Android Studio: "No se ha encontrado ningún NavHostFragment. Para que este grafo de navegación sea accesible debe ser referenciado en un NavHostFragment dentro de un diseño (layout)".



El mensaje indica que el NavHostFragment, que es el contenedor de gestionar la navegación entre fragmentos mediante el **grafo de navegación**, debe estar declarado dentro de un archivo layout XML en la carpeta res/layout. En ese archivo, el grafo de navegación será referenciado para que las transiciones entre fragmentos se realicen correctamente.

En nuestro caso, tenemos que añadir el contenedor NavHostFragment en el diseño de activity\_options.xml.

Tipicamente, para implementar el NavHostFragment, utilizaremos un FragmentContainerView especificando el atributo android:name con la clase java NavHostFragment:

android:name="androidx.navigation.fragment.NavHostFragment"

La implementación básica de un NavHostFragment es la siguiente:

<androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_options\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />

Este elemento proporciona un área dentro del diseño donde se gestionará la navegación entre fragmentos de manera independiente.

A continuación, necesitamos especificar el grafo de navegación en el NavHostFragment, para lo cual utilizaremos el atributo app:navGraph:

app:navGraph="@navigation/options\_nav\_graph"

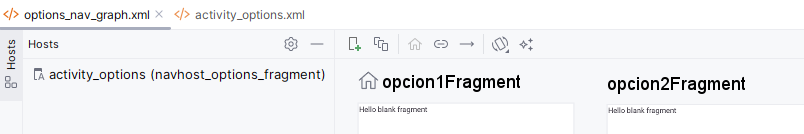
El layout activity\_options.xml, con el NavHostFragment completo, debería verse así:

<androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_options\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:defaultNavHost="true"  
 app:navGraph="@navigation/options\_nav\_graph" />

El atributo app:defaultNavHost="true" indica que deseas que este sea el NavHost que intercepte el botón de retroceso de tu dispositivo.

Cada NavHostFragment tiene un NavController que define la navegación dentro del host. Los NavHostFragments registran su NavController en la raíz de su subárbol de vistas, de manera que cualquier descendiente pueda obtener la instancia del NavController utilizando los métodos de la clase Navigation, como: Navigation.findNavController.

Si volvemos al archivo options\_nav\_graph.xml, ahora veremos en el panel de **Hosts** el NavHostFragments que acabamos de definir:

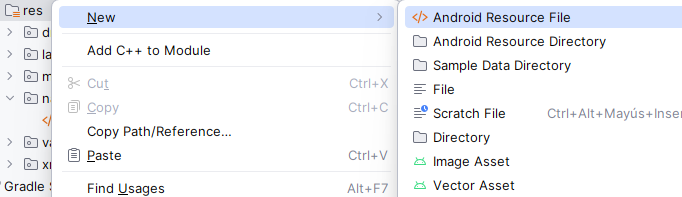


En este caso, no es necesario añadir acciones entre fragmentos, ya que la navegación será controlada por cada uno de los ítems del menú. Y el archivo options\_nav\_graph.xml se vería así:

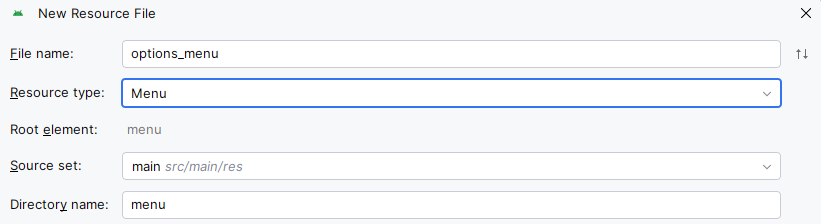
*<?*xml version="1.0" encoding="utf-8"*?>*<navigation xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/options\_nav\_graph"  
 app:startDestination="@id/opcion1Fragment">  
  
 <fragment  
 android:id="@+id/opcion1Fragment"  
 android:name="com.overdrive.navigationui.options.opcion1Fragment"  
 android:label="@string/opcion\_1"  
 tools:layout="@layout/fragment\_opcion1" />  
 <fragment  
 android:id="@+id/opcion2Fragment"  
 android:name="com.overdrive.navigationui.options.opcion2Fragment"  
 android:label="@string/opcion\_2"  
 tools:layout="@layout/fragment\_opcion2" />  
 <fragment  
 android:id="@+id/opcion3Fragment"  
 android:name="com.overdrive.navigationui.options.opcion3Fragment"  
 android:label="@string/opcion\_3"  
 tools:layout="@layout/fragment\_opcion3" />  
</navigation>

### XML. Creación del menú

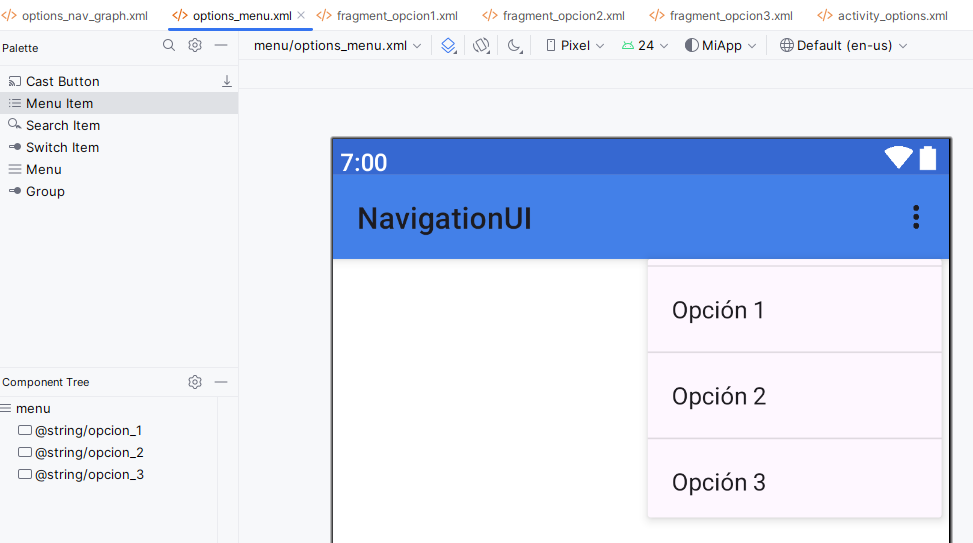
Para crear un menú, haz clic derecho sobre la carpeta res del proyecto y selecciona New > Android Resource File.



Introduce el nombre, por ejemplo: options\_menu. Y en el Resource type selecciona Menu:



En el diseño que se mostrará, añadimos tres ítems al menú y les damos los nombres: **Opción 1**, **Opción 2** y **Opción 3**.

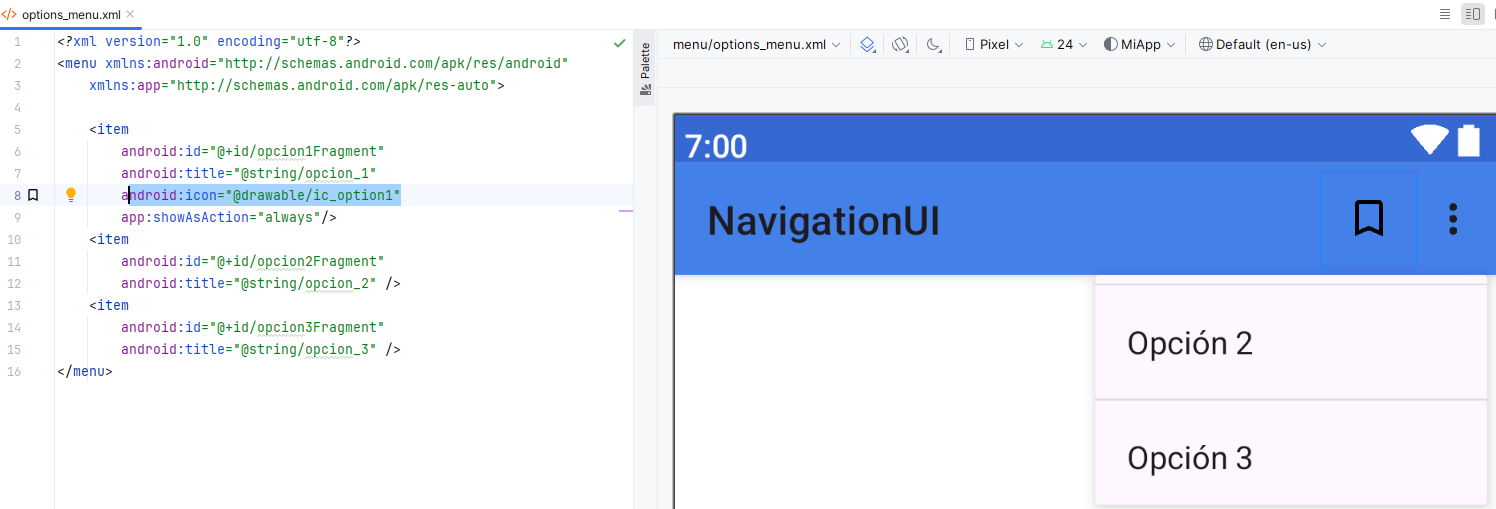


El código generado para este menú básico es:

*<?*xml version="1.0" encoding="utf-8"*?>*<menu xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:android="http://schemas.android.com/apk/res/android">  
  
 <item android:title="@string/opcion\_1" />  
 <item android:title="@string/opcion\_2" />  
 <item android:title="@string/opcion\_3" />  
</menu>

Ahora vamos a relacionarlo con nuestros fragmentos, y para eso tenemos que especificar el atributo android:id con el nombre del id de cada fragmento declarado en el grafo de navegación (options\_nav\_graph.xml):

*<?*xml version="1.0" encoding="utf-8"*?>*<menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto">  
  
 <item  
 android:id="@+id/opcion1Fragment"  
 android:title="@string/opcion\_1"  
 android:icon="@drawable/ic\_option1"  
 app:showAsAction="always"  
 />  
 <item  
 android:id="@+id/opcion2Fragment"  
 android:title="@string/opcion\_2" />  
 <item  
 android:id="@+id/opcion3Fragment"  
 android:title="@string/opcion\_3" />  
</menu>



El atributo app:showAsAction del primer item se muestra como una acción dentro del ToolBar. Para evitar que salga la opción con letras, le asignamos un icono mediante el atributo android:icon.

### Java. Asignar soporte al ToolBar

En el método onCreate() de la Activity, tenemos que añadir el soporte para la Toolbar que hemos creado mediante el método setSupportActionBar().

public class OptionsActivity extends AppCompatActivity {  
 ActivityOptionsBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityOptionsBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeOptionsToolbar.toolbar);  
 }  
}

### Java. Definir AppBarConfiguration

AppBarConfiguration se utiliza para definir el comportamiento de la Toolbar dependiendo de los destinos de navegación a los que el usuario llegue.

@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityOptionsBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeOptionsToolbar.toolbar);  
  
 *//AppBarConfiguration* AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(  
 R.id.*opcion1Fragment*, R.id.*opcion2Fragment*)  
 .build();  
}

En el constructor del método estático Builder() de la clase AppBarConfiguration, se pasan como parámetros los destinos principales de la navegación: R.id.opcion1Fragment y R.id.opcion2Fragment.

Estos destinos no mostrarán el ícono de flecha izquierda ← (botón up) en la Toolbar, sino el icono de "hamburger" (☰) o ningún ícono si no hay menú.

Si ejecutamos la aplicación, al hacer clic en la opción tres del menú veremos como se ha incluido en la ToolBar la fecha izquierda ← (botón up):



Si hacemos clic en la flecha izquierda ←, nos redirije a la **Opción 1**, que es la opción definida en el atributo app:startDestination del grafo de navegación.

<navigation 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/options\_nav\_graph"  
 app:startDestination="@id/opcion1Fragment">

### Java. Configurar NavigationUI

El siguiente código se encarga de obtener una referencia al NavHostFragment, que es una referencia a un FragmentContainerView que se ha enlazado a través de ViewBinding. Y a partir del NavHostFragment se extrae el NavController.

El método setupActionBarWithNavController() vincula el Toolbar con el NavController para que el título se actualice automáticamente cuando cambie el destino, suponiendo que en el grafo se haya definido el atributo android:label. Además, hace que gestione la navegación hacia atrás.

NavHostFragment navHostFragment = (NavHostFragment)   
 binding.navhostOptionsFragment.getFragment();  
navController = navHostFragment.getNavController();  
NavigationUI.*setupActionBarWithNavController*(this, navController, appBarConfiguration);

Ahora sobrescribimos el método onSupportNavigateUp():

@Override  
public boolean onSupportNavigateUp() {  
 NavController navController = Navigation.*findNavController*(this,  
 R.id.*navhost\_options\_fragment*);  
 return NavigationUI.*navigateUp*(navController,   
 new AppBarConfiguration.Builder(navController.getGraph()).build()) ||  
 super.onSupportNavigateUp();  
}

Este método se utiliza para manejar la acción de navegación hacia arriba (icono ←) en la jerarquía de actividades.

Ver anexo sobre los problemas que podemos tener con NavController.

### Java. Incluir el menú.

Para incluir el menú en el Toolbar, debemos sobrescribir dos métodos: onCreateOptionsMenu() y onOptionsItemSelected().

@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(R.menu.*options\_menu*, menu);  
 return true;  
}  
  
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
 return NavigationUI.*onNavDestinationSelected*(item, navController)  
 || super.onOptionsItemSelected(item);  
}

El primer método infla el menú de opciones, que se pasa como parámetro, a partir del layout XML del menú (options\_menu.xml). Esto significa que los elementos del menú definidos en ese archivo se agregarán al menú de la actividad.

El segundo método se utiliza para manejar la navegación en base a la selección de elementos del menú, con la ayuda del NavController.

## Actividad del NavigationView

El NavigationView, más conocido como "DrawerLayout", es el menú de navegación que suele aparecer en el lateral izquierdo de la app cuando se pulsa el icono hamburger (☰).

En el archivo DrawerActivity.java de esta app, usaremos un NavigationView para navegar a 3 destinos (fragments).

Imagen que contiene Texto

Descripción generada automáticamente

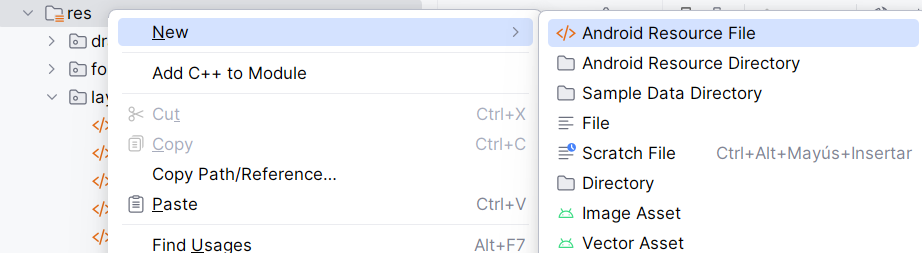
### XML. Incluir el Toolbar

Modificamos el layout de la actividad activity\_drawer.xml para insertar el ToolBar, como se hizo con la actividad principal:

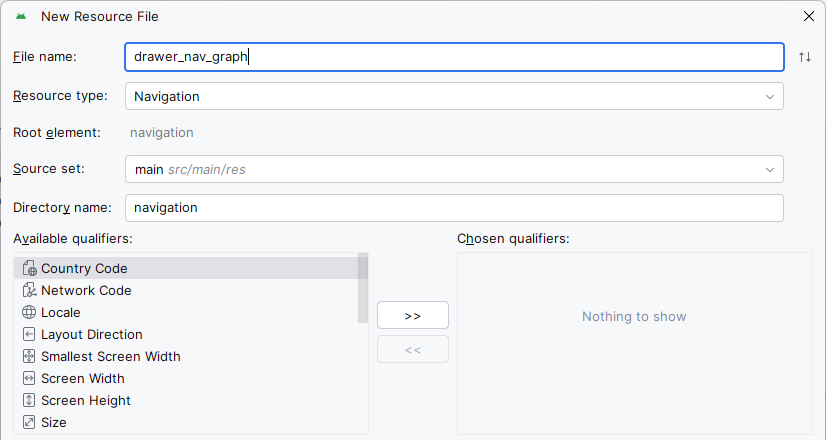
*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include   
 android:id="@+id/include\_drawer\_toolbar"   
 layout="@layout/toolbar"/>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 *<!-- Aqui va el contenido de la activity -->*  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

### XML. Grafo de navegación

Para crear un grafo de navegación, haz clic derecho sobre la carpeta res del proyecto y selecciona New > Android Resource File.

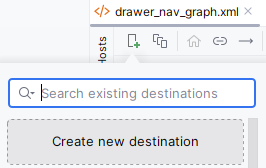


Introduce el nombre del fichero del grafo, por ejemplo: drawer\_nav\_graph. Y en el Resource type selecciona Navigation:

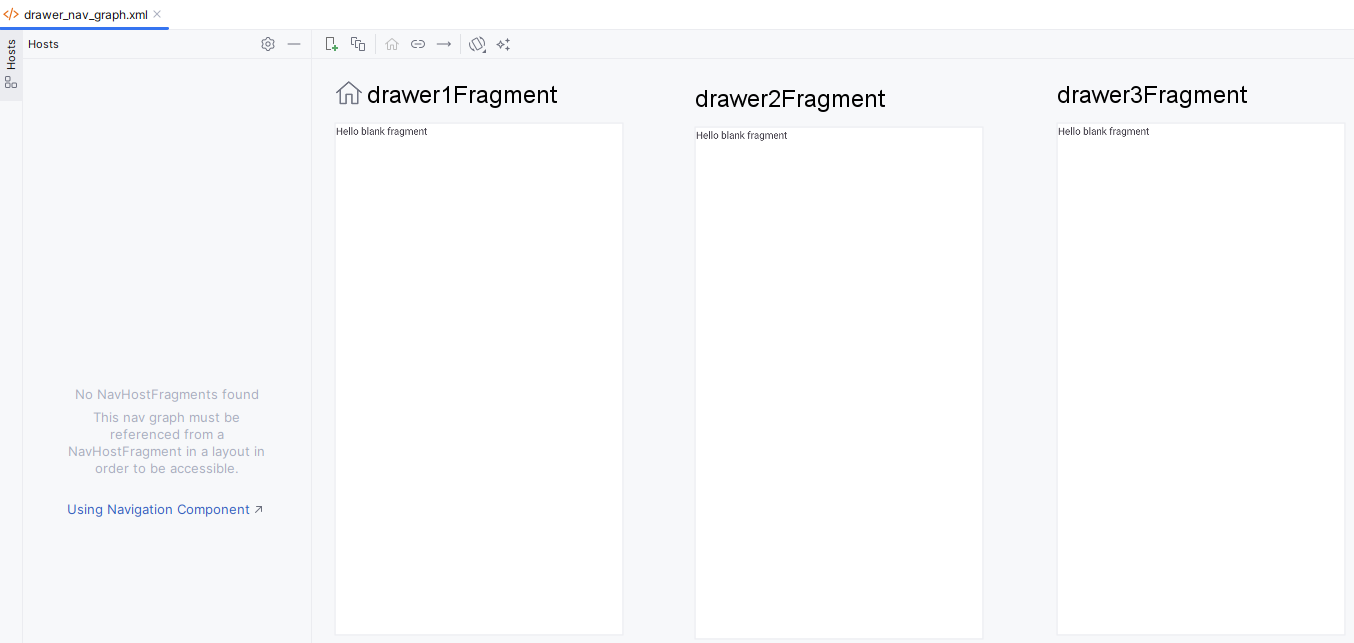


Si no existía la carpeta Navigation, entonces la creará junto el archivo drawer\_nav\_graph.xml.

En el grafo de navegación definimos tres destinos: drawer1Fragment, drawer2Fragment y drawer drawer3Fragment. Haz clic en el icono de un rectángulo con un símbolo **+** para añadir cada uno de los destinos y selecciona un fragmento en blanco para cada uno.



Después añadir los destinos, en el panel izquierdo de los **Hosts**, aparece un mensaje de Android Studio: "No se ha encontrado ningún NavHostFragment. Para que este grafo de navegación sea accesible debe ser referenciado en un NavHostFragment dentro de un diseño (layout)".



El mensaje indica que el NavHostFragment, que es el contenedor de gestionar la navegación entre fragmentos mediante el **grafo de navegación**, debe estar declarado dentro de un archivo layout XML en la carpeta res/layout. En ese archivo, el grafo de navegación será referenciado para que las transiciones entre fragmentos se realicen correctamente.

En nuestro caso, tenemos que añadir el contenedor NavHostFragment en el diseño de activity\_drawer.xml.

Tipicamente, para implementar el NavHostFragment, utilizaremos un FragmentContainerView especificando el atributo android:name con la clase java NavHostFragment:

android:name="androidx.navigation.fragment.NavHostFragment"

La implementación básica de un NavHostFragment es la siguiente:

<androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_drawer\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />

Este elemento proporciona un área dentro del diseño donde se gestionará la navegación entre fragmentos de manera independiente. A continuación, necesitamos especificar el grafo de navegación en el NavHostFragment, para lo cual utilizaremos el atributo app:navGraph:

app:navGraph="@navigation/drawer\_nav\_graph"

El layout activity\_drawer.xml, con el NavHostFragment completo, debería verse así:

<androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_drawer\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:defaultNavHost="true"  
 app:navGraph="@navigation/drawer\_nav\_graph" />

El atributo app:defaultNavHost="true" indica que deseas que este sea el NavHost que intercepte el botón de retroceso de tu dispositivo.

Cada NavHostFragment tiene un NavController que define la navegación dentro del host. Los NavHostFragments registran su NavController en la raíz de su subárbol de vistas, de manera que cualquier descendiente pueda obtener la instancia del NavController utilizando los métodos de la clase Navigation, como: Navigation.findNavController.

Si volvemos al archivo drawer\_nav\_graph.xml, ahora veremos en el panel de **Hosts** el NavHostFragments que acabamos de definir:

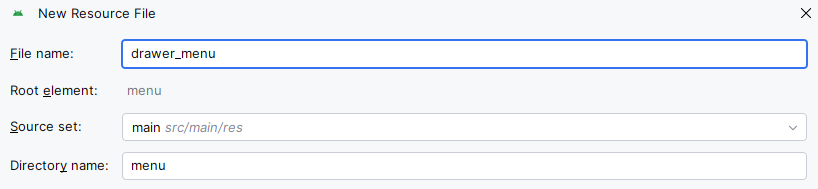


En este caso, no es necesario añadir acciones entre fragmentos, ya que los tres destinos son independientes entre sí, es decir, no se navega de ninguno a otro, sino que son accesibles desde el menú **drawer**. Y el archivo drawer\_nav\_graph.xml se vería así:

*<?*xml version="1.0" encoding="utf-8"*?>*<navigation 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/drawer\_nav\_graph"  
 app:startDestination="@id/drawer1Fragment">  
  
 <fragment  
 android:id="@+id/drawer1Fragment"  
 android:name="com.overdrive.navigationui.drawer.drawer1Fragment"  
 android:label="@string/drawer\_1"  
 tools:layout="@layout/fragment\_drawer1" />  
 <fragment  
 android:id="@+id/drawer2Fragment"  
 android:name="com.overdrive.navigationui.drawer.drawer2Fragment"  
 android:label="@string/drawer\_2"  
 tools:layout="@layout/fragment\_drawer2" />  
 <fragment  
 android:id="@+id/drawer3Fragment"  
 android:name="com.overdrive.navigationui.drawer.drawer3Fragment"  
 android:label="@string/drawer\_3"  
 tools:layout="@layout/fragment\_drawer3" />  
</navigation>

### XML. Creación del menú

Para crear un menú, haz clic derecho sobre la carpeta res del proyecto y selecciona New > Android Resource File. Introduce el nombre, por ejemplo: drawer\_menu. Y en el Resource type selecciona Menu:



Editamos el código XML asociado para incluir el siguiente:

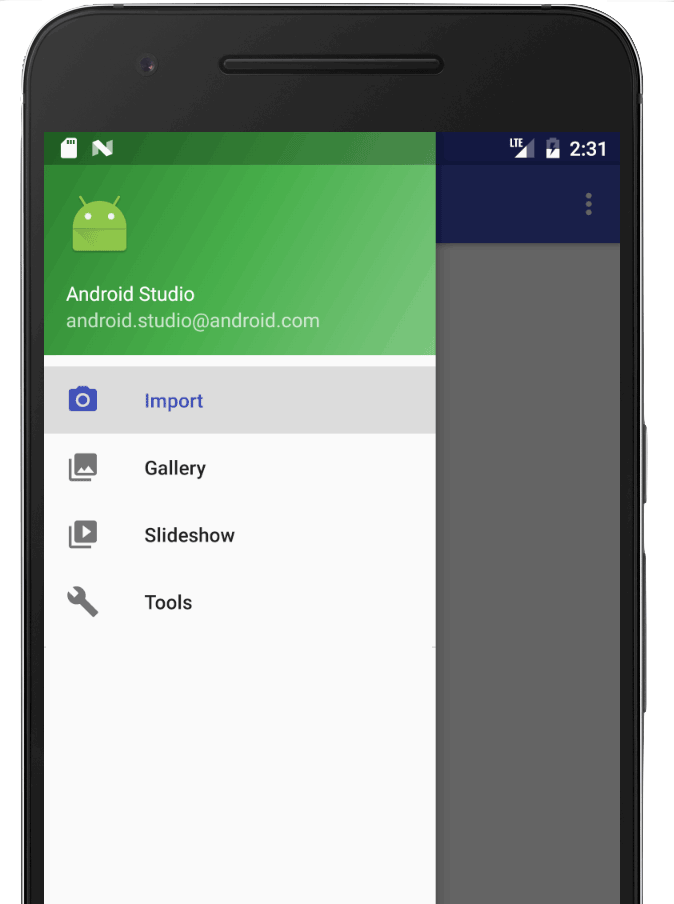
<menu xmlns:android="http://schemas.android.com/apk/res/android">  
 <item  
 android:id="@+id/drawer1Fragment"  
 android:icon="@drawable/ic\_drawer\_camera"  
 android:title="@string/drawer\_1" />  
  
 <item  
 android:id="@+id/drawer2Fragment"  
 android:icon="@drawable/ic\_drawer\_slideshow"  
 android:title="@string/drawer\_2" />  
  
 <item  
 android:id="@+id/drawer3Fragment"  
 android:icon="@drawable/ic\_drawer\_bluetooth"  
 android:title="@string/drawer\_3" />  
</menu>

De esta manera, hemos asociado los fragmentos del drawer\_nav\_graph.xml a cada item de menú mediante el atributo android:id. Para asignar iconos personalizados hay que definir el atributo android:icon.

El panel lateral de navegación se compone tanto del Menu como del Header que vamos a definir:

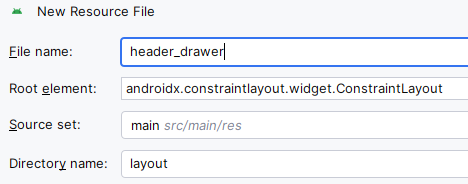
**Header**

**Menu**



### XML. Diseño de la cabecera del Drawer

El panel lateral está compuesto por una cabecera (header) y un menú. Para personalizar la cabecera tenemos que crear un recurso XML en res/layout, que llamaremos header\_drawer.xml:

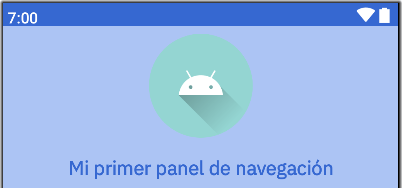


En el layout, incluiremos una imagen y un texto cualquiera:

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="#ACC4F4">  
  
 <ImageView  
 android:id="@+id/imageView"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="8dp"  
 android:adjustViewBounds="true"  
 android:background="@drawable/circular\_image"  
 android:backgroundTint="#94D5D2"  
 android:cropToPadding="true"  
 android:scaleType="centerCrop"  
 app:circularflow\_defaultAngle="@android:dimen/app\_icon\_size"  
 app:circularflow\_defaultRadius="@android:dimen/app\_icon\_size"  
 app:circularflow\_radiusInDP="20dp"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:srcCompat="@drawable/ic\_launcher\_foreground" />  
  
 <TextView  
 android:id="@+id/tvHeaderDrawer"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="16dp"  
 android:layout\_marginBottom="8dp"  
 android:fontFamily="@font/anuphan"  
 android:text="@string/header\_drawer"  
 android:textColor="@color/blue\_700"  
 android:textSize="20sp"  
 android:textStyle="bold"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@+id/imageView" />  
</androidx.constraintlayout.widget.ConstraintLayout>

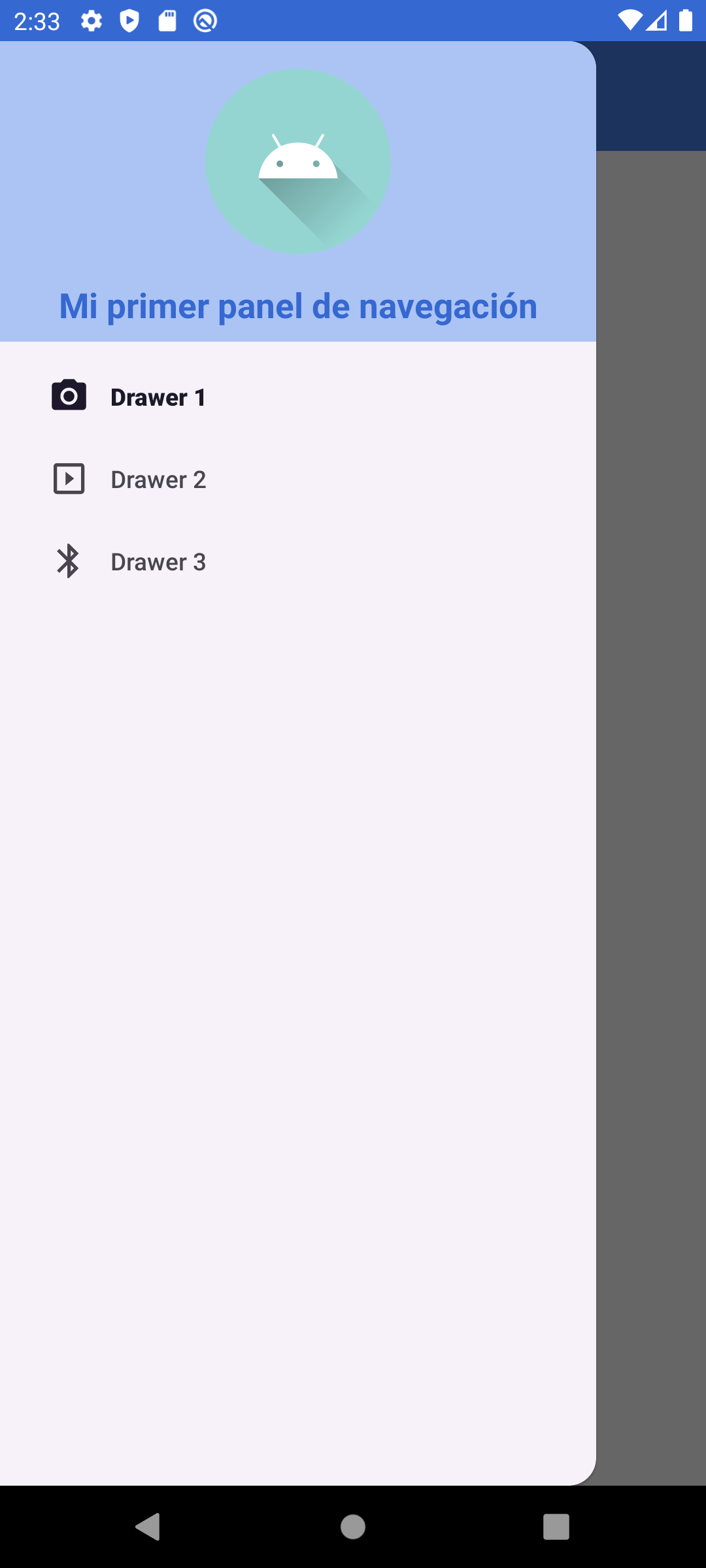
El atributo android:background="@drawable/circular\_image" se corresponde con un elemento creado en la carpeta res/drawable para crear bordes circulares:

*<?*xml version="1.0" encoding="utf-8"*?>*<shape xmlns:android="http://schemas.android.com/apk/res/android"  
 android:shape="oval">  
 <solid android:color="#FFFFFF"/> *<!-- Color de relleno -->* <size  
 android:width="200dp"  
 android:height="200dp"/> *<!-- Ajusta las dimensiones de la imagen -->*</shape>



### XML. Incluir el NavigationView

El panel lateral de navegación muestra el menú principal de tu app. El panel aparece cuando el usuario toca el ícono del panel lateral (☰) en la barra de la app, o cuando se desliza un dedo desde el borde izquierdo de la pantalla.



Para agregar un panel lateral de navegación, lo primero que debes hacer es declarar un DrawerLayout como la vista raíz del layout de la actividad. Le asignamos un id para poder hacer referencia al objeto desde código.

El primer elemento hijo de este layout será el contenido principal de la actividad.

Luego, añadimos un NavigationView como segundo hijo del contenedor DrawerLayout.

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.drawerlayout.widget.DrawerLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app=<http://schemas.android.com/apk/res-auto>  
 android:id="@+id/drawer\_layout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 *<!-- CONTENIDO DE LA PANTALLA DEL DRAWER -->* <androidx.coordinatorlayout.widget.CoordinatorLayout  
  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include  
 android:id="@+id/include\_drawer\_toolbar"  
 layout="@layout/toolbar" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 <androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_drawer\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:defaultNavHost="true"  
 app:navGraph="@navigation/drawer\_nav\_graph" />  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
 </androidx.coordinatorlayout.widget.CoordinatorLayout>  
  
 *<!-- DEFINICION DEL MENU DRAWER -->* <com.google.android.material.navigation.NavigationView  
 android:id="@+id/nav\_view"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_gravity="start"  
 android:fitsSystemWindows="true"  
 app:headerLayout="@layout/header\_drawer"  
 app:menu="@menu/drawer\_menu"/>  
  
</androidx.drawerlayout.widget.DrawerLayout>

De este modo, el DrawerLayout contiene tanto el contenido principal de la actividad (que incluye el Toolbar y el NavHostFragment) como el NavigationView, el cual alberga el panel lateral de navegación. El NavigationView es un hijo del DrawerLayout, y se desliza desde el lado izquierdo cuando se pulsa el ícono hamburger o se realiza un gesto de deslizamiento.

### Java. Asignar soporte al ToolBar

En el método onCreate() de la Activity, tenemos que añadir el soporte para la Toolbar que hemos creado mediante el método setSupportActionBar().

public class DrawerActivity extends AppCompatActivity {  
 ActivityDrawerBinding binding;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityDrawerBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeDrawerToolbar.toolbar);  
 }  
}

### Java. Definir AppBarConfiguration

AppBarConfiguration se utiliza para definir el comportamiento de la Toolbar dependiendo de los destinos de navegación a los que el usuario llegue.

public class DrawerActivity extends AppCompatActivity {  
 ActivityDrawerBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityDrawerBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeDrawerToolbar.toolbar);  
  
 AppBarConfiguration appBarConfiguration = new AppBarConfiguration.Builder(  
 *// Top-level destinations:* R.id.*drawer1Fragment*, R.id.*drawer2Fragment* ).setOpenableLayout(binding.drawerLayout)  
 .build();  
 }  
}

En el constructor del método estático Builder() de la clase AppBarConfiguration, se pasan como parámetros los destinos principales de la navegación: R.id.drawer1Fragment y R.id.drawer2Fragment.

Estos destinos no mostrarán el ícono de flecha izquierda ← (botón up) en la Toolbar, sino el icono de "**hamburger**" (☰).

Además, se hace una llamada al método setOpenableLayout(binding.drawerLayout) para especificar que el DrawerLayout debe ser controlado por la Toolbar. Al hacerlo, el icono del menú **hamburger** (☰) abrirá o cerrará el DrawerLayout.

### Java. Configurar NavigationUI

El siguiente código se encarga de obtener una referencia al NavHostFragment, que es una referencia a un FragmentContainerView que se ha enlazado a través de ViewBinding. Y a partir del NavHostFragment se extrae el NavController que es responsable de gestionar la navegación entre los destinos.

La clase de la actividad tendrá el siguiente código:

public class DrawerActivity extends AppCompatActivity {  
 private ActivityDrawerBinding binding;  
 private NavController navController;  
 private AppBarConfiguration appBarConfiguration;  
 private NavHostFragment navHostFragment;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityDrawerBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 Toolbar toolbar = binding.includeDrawerToolbar.toolbar;  
 setSupportActionBar(toolbar);  
  
 appBarConfiguration = new AppBarConfiguration.Builder(  
 *// Top-level destinations:* R.id.*drawer1Fragment*, R.id.*drawer2Fragment* ).setOpenableLayout(binding.drawerLayout)  
 .build();  
  
 navHostFragment = (NavHostFragment) getSupportFragmentManager()  
 .findFragmentById(R.id.*navhost\_drawer\_fragment*);  
 navController = navHostFragment.getNavController();  
NavigationUI.*setupWithNavController*(binding.navView, navController);NavigationUI.*setupActionBarWithNavController*(this, navController,  
 appBarConfiguration);  
 }  
@Override  
 public boolean onSupportNavigateUp() {  
 return NavigationUI.*navigateUp*(navController, appBarConfiguration)  
 || super.onSupportNavigateUp();  
 }  
}

Mediante el método NavigationUI.setupWithNavController(binding.navView, navController) se vincula el NavigationView con el NavController. De este modo, cuando se selecciona un elemento del menú del drawer, el NavController navega al fragmento correspondiente definido en el gráfico de navegación.

El método NavigationUI.setupActionBarWithNavController() vincula el ActionBar con el NavController y el AppBarConfiguration. Esto asegura que el título del ActionBar se actualice automáticamente.

Por último, se sobrescribe el método onSupportNavigateUp que le permite al usuario regresar a destinos de nivel superior dentro del gráfico de navegación.

## Actividad del BottomNavigationView

El BottomNavigationView es la barra de menú que se muestra en la parte inferior de la pantalla. La añadiremos en la actividad BottomActiviy para navegar entre 3 destinos.

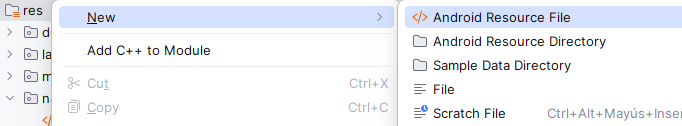
### XML. Incluir el Toolbar

Modificamos el layout de la actividad activity\_bottom.xml para insertar el ToolBar, tal como se hizo en la actividad principal:

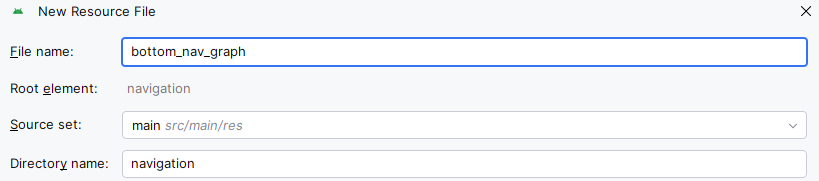
*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include   
 android:id="@+id/include\_bottom\_toolbar"   
 layout="@layout/toolbar"/>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 *<!-- Aqui va el contenido de la activity -->*  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

### XML. Grafo de navegación

Para crear un grafo de navegación, haz clic derecho sobre la carpeta res del proyecto y selecciona New > Android Resource File.

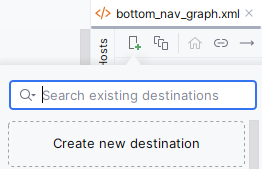


Introduce el nombre del fichero del grafo, por ejemplo: bottom\_nav\_graph. Y en el Resource type selecciona Navigation:

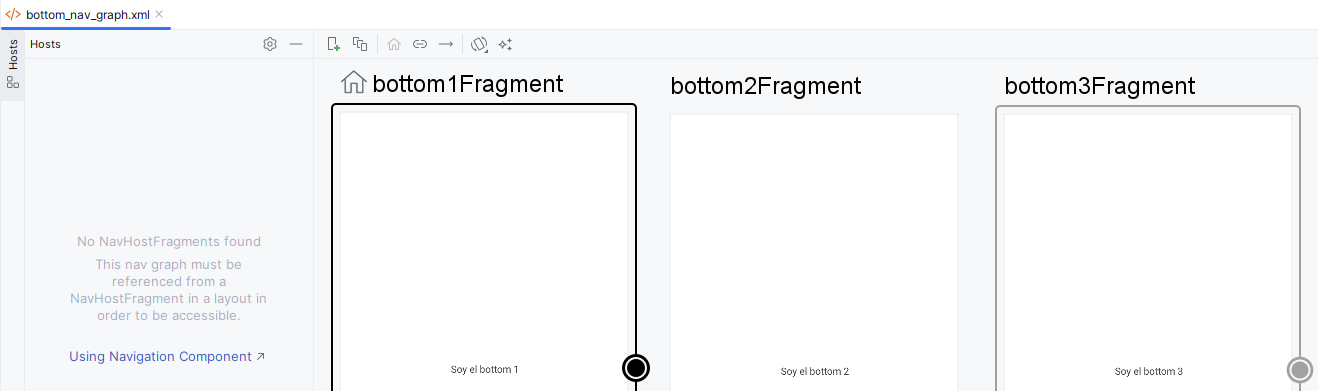


Si no existía la carpeta Navigation, entonces la creará junto el archivo bottom\_nav\_graph.xml.

En el grafo de navegación definimos tres destinos: bottom1Fragment, bottom2Fragment y bottom3Fragment. Haz clic en el icono de un rectángulo con un símbolo **+** para añadir cada uno de los destinos y selecciona un fragmento en blanco para cada uno.



Después añadir los destinos, en el panel izquierdo de los **Hosts**, aparece un mensaje de Android Studio: "No se ha encontrado ningún NavHostFragment. Para que este grafo de navegación sea accesible debe ser referenciado en un NavHostFragment dentro de un diseño (layout)".



El mensaje indica que el NavHostFragment, que es el contenedor de gestionar la navegación entre fragmentos mediante el **grafo de navegación**, debe estar declarado dentro de un archivo layout XML en la carpeta res/layout. En ese archivo, el grafo de navegación será referenciado para que las transiciones entre fragmentos se realicen correctamente.

En nuestro caso, tenemos que añadir el contenedor NavHostFragment en el diseño de activity\_bottom.xml.

Tipicamente, para implementar el NavHostFragment, utilizaremos un FragmentContainerView especificando el atributo android:name con la clase java NavHostFragment:

android:name="androidx.navigation.fragment.NavHostFragment"

La implementación básica de un NavHostFragment es la siguiente:

<androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_bottom\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />

Este elemento proporciona un área dentro del diseño donde se gestionará la navegación entre fragmentos de manera independiente. A continuación, necesitamos especificar el grafo de navegación en el NavHostFragment, para lo cual utilizaremos el atributo app:navGraph:

app:navGraph="@navigation/bottom\_nav\_graph"

El layout activity\_bottom.xml, con el NavHostFragment completo, debería verse así:

<androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_bottom\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:defaultNavHost="true"  
 app:navGraph="@navigation/bottom\_nav\_graph" />

El atributo app:defaultNavHost="true" indica que deseas que este sea el NavHost que intercepte el botón de retroceso de tu dispositivo.

Cada NavHostFragment tiene un NavController que define la navegación dentro del host. Los NavHostFragments registran su NavController en la raíz de su subárbol de vistas, de manera que cualquier descendiente pueda obtener la instancia del NavController utilizando los métodos de la clase Navigation, como: Navigation.findNavController.

Si volvemos al archivo bottom\_nav\_graph.xml, ahora veremos en el panel de **Hosts** el NavHostFragments que acabamos de definir:

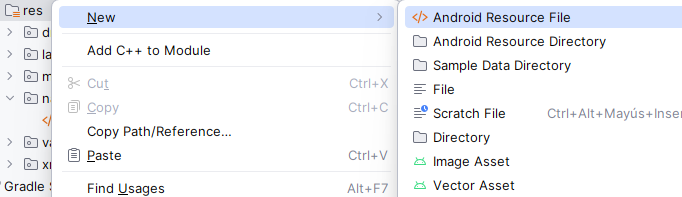


En este caso, no es necesario añadir acciones entre fragmentos, ya que los tres destinos son independientes entre sí, es decir, no se navega de ninguno a otro, sino que son accesibles desde el menú bottom. Y el archivo bottom\_nav\_graph.xml se vería así:

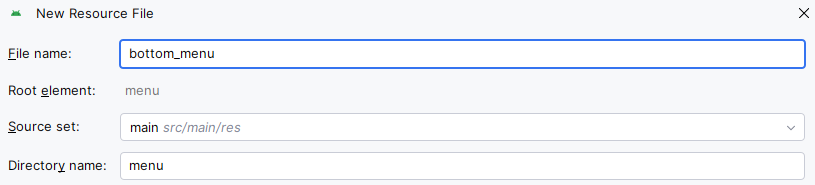
*<?*xml version="1.0" encoding="utf-8"*?>*<navigation xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/bottom\_nav\_graph"  
 app:startDestination="@id/bottom1Fragment">  
  
 <fragment  
 android:id="@+id/bottom1Fragment"  
 android:name="com.overdrive.navigationui.bottom1Fragment"  
 android:label="@string/bottom\_1"  
 tools:layout="@layout/fragment\_bottom1" />  
 <fragment  
 android:id="@+id/bottom2Fragment"  
 android:name="com.overdrive.navigationui.bottom2Fragment"  
 android:label="@string/bottom\_2"  
 tools:layout="@layout/fragment\_bottom2" />  
 <fragment  
 android:id="@+id/bottom3Fragment"  
 android:name="com.overdrive.navigationui.bottom3Fragment"  
 android:label="@string/bottom\_3"  
 tools:layout="@layout/fragment\_bottom3" />  
</navigation>

### XML. Creación del menú

Para crear un menú, haz clic derecho sobre la carpeta res del proyecto y selecciona New > Android Resource File.



Introduce el nombre, por ejemplo: bottom\_menu:



Editamos el código XML asociado para incluir el siguiente:

*<?*xml version="1.0" encoding="utf-8"*?>*<menu xmlns:android="http://schemas.android.com/apk/res/android">  
 <item  
 android:id="@+id/bottom1Fragment"  
 android:icon="@drawable/ic\_drawer\_camera"  
 android:title="@string/bottom\_1" />  
  
 <item  
 android:id="@+id/bottom2Fragment"  
 android:icon="@drawable/ic\_drawer\_slideshow"  
 android:title="@string/bottom\_2" />  
  
 <item  
 android:id="@+id/bottom3Fragment"  
 android:icon="@drawable/ic\_drawer\_bluetooth"  
 android:title="@string/bottom\_3" />  
</menu>

### XML. Incluir el BottomNavigationView

Añade el elemento <BottomNavigationView> al layout de la BottomActivity. Colócalo dentro del ConstraintLayout, justo debajo del NavHostFragment:

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include  
 android:id="@+id/include\_bottom\_toolbar"  
 layout="@layout/toolbar" />  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 <androidx.fragment.app.FragmentContainerView  
 android:id="@+id/navhost\_bottom\_fragment"  
 android:name="androidx.navigation.fragment.NavHostFragment"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:defaultNavHost="true"  
 app:navGraph="@navigation/bottom\_nav\_graph" />  
  
 <com.google.android.material.bottomnavigation.BottomNavigationView  
 android:id="@+id/bottom\_nav\_view"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:background="?android:attr/windowBackground"  
 app:menu="@menu/bottom\_menu"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintLeft\_toLeftOf="parent"  
 app:layout\_constraintRight\_toRightOf="parent" />  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

### Java. Asignar soporte al ToolBar

En el método onCreate() de la Activity, tenemos que añadir el soporte para la Toolbar que hemos creado mediante el método setSupportActionBar().

public class BottomActivity extends AppCompatActivity {  
 ActivityBottomBinding binding;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityBottomBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeBottomToolbar.toolbar);  
 }  
}

### Java. Configurar NavigationUI

El siguiente código se encarga de obtener una referencia al NavHostFragment, que es una referencia a un FragmentContainerView que se ha enlazado a través de ViewBinding. Y a partir del NavHostFragment se extrae el NavController.

El método setupActionBarWithNavController() vincula el Toolbar con el NavController, actualizando automáticamente el título se navega entre destinos.

El método setupWithNavController() enlaza el BottomNavigationView con el NavController. Al presionar una de las opciones del BottomNavigationView, se realiza la navegación al destino correspondiente.

public class BottomActivity extends AppCompatActivity {  
 ActivityBottomBinding binding;  
 private NavController navController;  
 private NavHostFragment navHostFragment;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityBottomBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeBottomToolbar.toolbar);  
  
 navHostFragment = (NavHostFragment) getSupportFragmentManager()  
 .findFragmentById(R.id.*navhost\_bottom\_fragment*);  
 navController = navHostFragment.getNavController();  
  
 NavigationUI.*setupWithNavController*(binding.bottomNavView, navController);  
 NavigationUI.*setupActionBarWithNavController*(this, navController);  
 }  
}

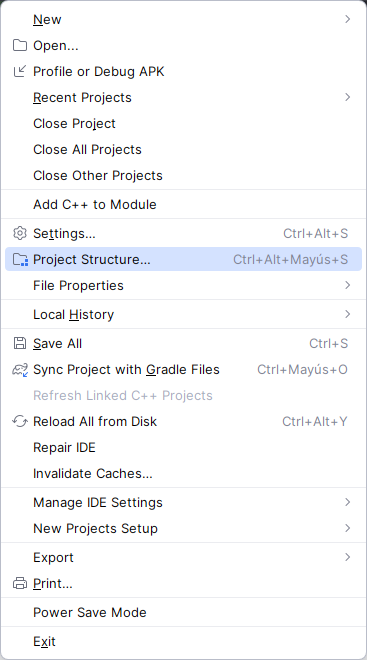
Este código permite la navegación automática a los destinos definidos en el grafo de navegación cuando se selecciona un elemento en el BottomNavigationView.

## Actividad de TabLayout y ViewPager2

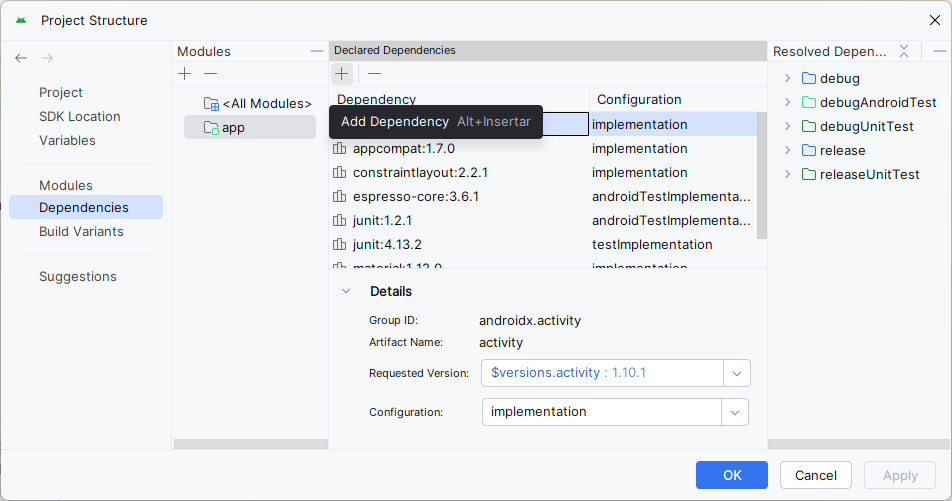
El componente ViewPager2 permite crear vistas deslizables, a las que se puede navegar deslizando el dedo. El componente TabLayout añade unas pestañas en la parte superior para facilitar la navegación entre diferentes fragmentos.

Estos componentes no forman parte del sistema de navegación de Navigation y, por lo tanto, no son gestionados por NavigationUI. ViewPager2 maneja la navegación entre fragmentos internamente mediante su adaptador (FragmentStateAdapter), y no requiere ni se beneficia de un grafo de navegación (NavGraph).

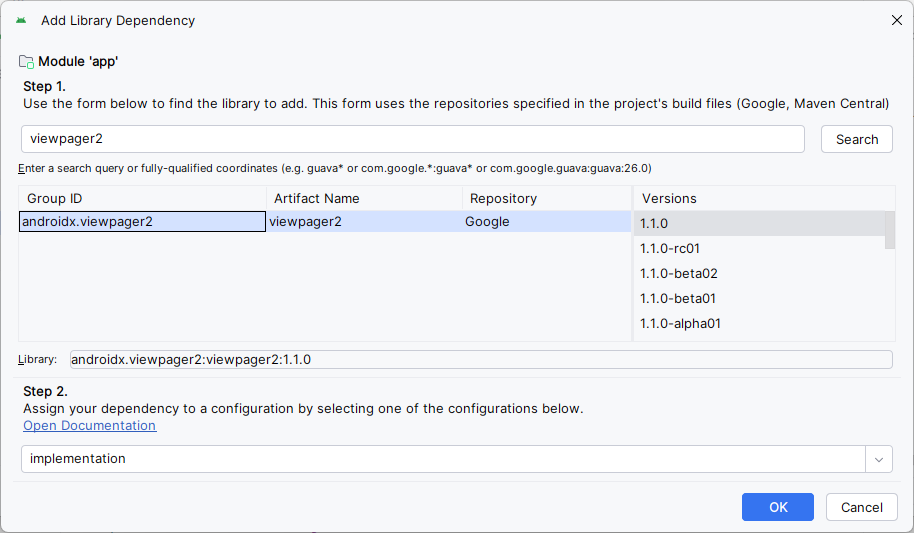
Tenemos que añadir la dependencia para ViewPager2. Navegamos por Android Studio, seguimos la siguiente ruta: **File -> Project Structure…**



Seleccionamos **Dependencies** del panel lateral izquierdo y **app** del panel central.



Hacemos clic en el símbolo de más (+), en la ventana emergente insertamos el texto de la librería que necesitamos añadir, en nuestro caso, viewpager2.



Otra opción es la definición directa en el archivo build.gradle a nivel de módulo:

*dependencies* **{**   
 *implementation*("androidx.viewpager2:viewpager2:1.1.0")  
**}**

### XML. Incluir el Toolbar

Modificamos el layout de la actividad activity\_tabbed.xml para insertar el [ToolBar](#_Creación_de_barra), como se hizo con la actividad principal:

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
 <include android:id="@+id/include\_tabbed\_toolbar"   
 layout="@layout/toolbar"/>  
  
 <androidx.constraintlayout.widget.ConstraintLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 app:layout\_behavior="@string/appbar\_scrolling\_view\_behavior">  
  
 *<!-- Aqui va el contenido de la activity -->*  
  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

En donde el archivo toolbar.xml está definido como:

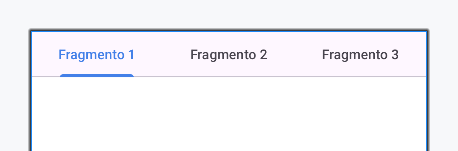
<com.google.android.material.appbar.AppBarLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 android:id="@+id/appBarLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar">  
  
 <androidx.appcompat.widget.Toolbar  
 android:id="@+id/toolbar"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="?attr/colorPrimary"  
 app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />  
  
</com.google.android.material.appbar.AppBarLayout>

### XML. Incluir el TabLayout

Dentro del diseño de la Activity, añadimos el elemento <TabLayout>, y dentro de este, definimos tres <TabItem>, uno para cada fragmento.

<androidx.constraintlayout.widget.ConstraintLayout   
 android:id="@+id/main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
  
 <com.google.android.material.tabs.TabLayout  
 android:id="@+id/tabLayout"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent">  
  
 <com.google.android.material.tabs.TabItem  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Fragmento 1" />  
  
 <com.google.android.material.tabs.TabItem  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Fragmento 2" />  
  
 <com.google.android.material.tabs.TabItem  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Fragmento 3" />  
 </com.google.android.material.tabs.TabLayout>  
  
</androidx.constraintlayout.widget.ConstraintLayout>

De esta manera, acabamos de definir tres pestañas:



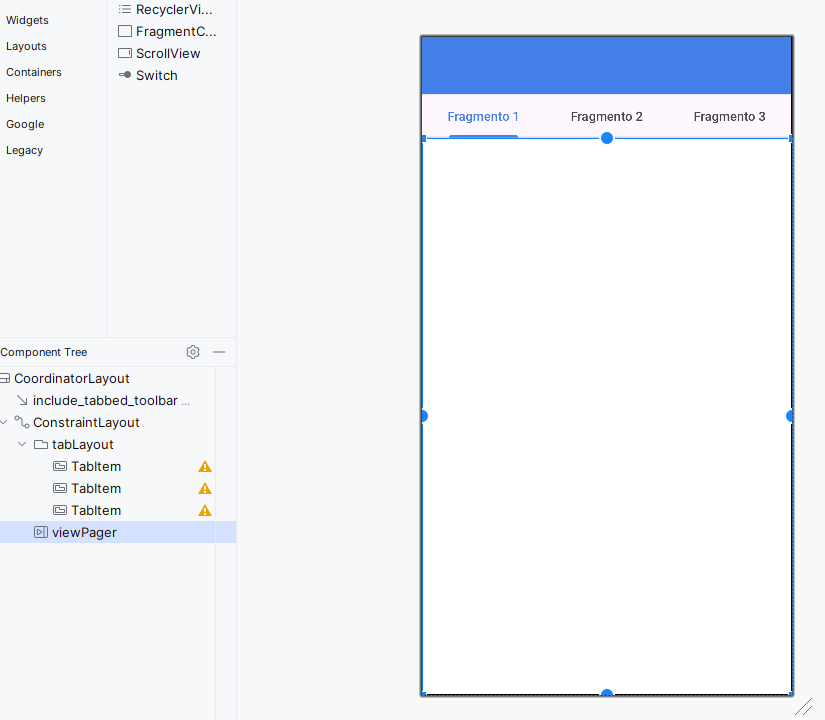
### XML. Incluir el ViewPager2

Añadimos el visor de páginas que mostrará los fragmentos, ya sea haciendo clics en los elementos del TabLayout o deslizando lateralmente.

El elemento <ViewPager2> se declara debajo del <TabLayout> en el layout de la actividad.

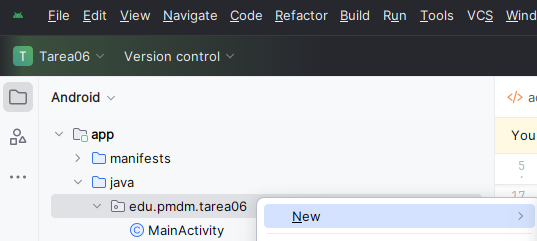
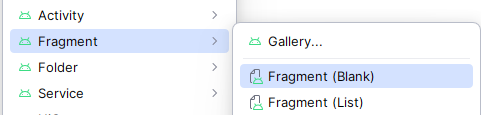
<androidx.coordinatorlayout.widget.CoordinatorLayout ... >  
  
 <include android:id="@+id/include\_tabbed\_toolbar" layout="@layout/toolbar"/>  
  
 <androidx.constraintlayout.widget.ConstraintLayout ... >  
 <com.google.android.material.tabs.TabLayout>   
 ...  
 </com.google.android.material.tabs.TabLayout>  
  
 <androidx.viewpager2.widget.ViewPager2  
 android:id="@+id/viewPager"  
 android:layout\_width="match\_parent"  
 android:layout\_height="0dp"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toBottomOf="@id/tabLayout" />  
 </androidx.constraintlayout.widget.ConstraintLayout>  
  
</androidx.coordinatorlayout.widget.CoordinatorLayout>

De esta manera, el espacio por debajo del TabLayout es ocupado por el ViewPager2:



### XML. Definir los fragmentos

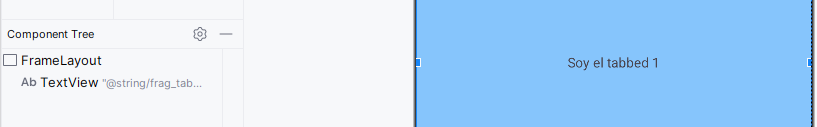
Dirígete a la carpeta **java** del proyecto y selecciona el modulo de la aplicación. haz clic derecho y selecciona File > New > Fragment > Fragment (Blank).

Creamos los siguientes fragmentos: tabbed1Fragment, tabbed2Fragment y tabbed3Fragment.

Las clases generadas contendrán código de plantilla (boilerplate) que no será necesario y se puede eliminar, dejando solo el **constructor** y los métodos onCreate y onCreateView.

Edita los fragmentos para que cada uno tenga un color de fondo (background) diferente y un TextView con el nombre del fragmento.



### Java. Asignar soporte al ToolBar

En el método onCreate() de la Activity, agrega soporte para la Toolbar mediante el método setSupportActionBar().

public class TabbedActivity extends AppCompatActivity {  
 ActivityTabbedBinding binding;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityTabbedBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeTabbedToolbar.toolbar);  
 }  
}

### Java. Crear un Adapter

Creamos una clase adaptadora que extienda de FragmentStateAdapter y que permita a ViewPager2 recuperar una instancia de cada fragmento en función de la selección actual.

public class AdaptadorFragment extends FragmentStateAdapter {  
 private final List<Fragment> listaFragmentos = new ArrayList<>();  
  
 public AdaptadorFragment(@NonNull FragmentActivity fragmentActivity) {  
 super(fragmentActivity);  
 }  
  
 @NonNull  
 @Override  
 public Fragment createFragment(int position) {  
 return listaFragmentos.get(position);  
 }  
  
 @Override  
 public int getItemCount() {  
 return listaFragmentos.size();  
 }  
  
 public void add(Fragment fragmento) {  
 listaFragmentos.add(fragmento);  
 }  
}

Hay que sobrescribir dos métodos: createFragment() y getItemCount().

El método createFragment() recupera un fragmento de la lista según la posición pasada al argumento del método.

El método getItemCount() devuelve el número de fragmentos de la lista.

El método add() nos permite agregar fragmentos a la lista.

Se pueden implementar varios [constructores](https://developer.android.com/develop/ui/views/animations/vp2-migration?hl=es-419#adapter-constructors) de la clase adaptadora que extiende de FragmentStateAdapter. El constructor más sencillo es aquel que recibe como argumento una instancia de FragmentActivity o Fragment.

Para tener un mayor control sobre el ciclo de vida de los fragmentos en el adaptador, es posible utilizar un constructor que acepte como argumentos un objeto FragmentManager y Lifecycle.

### Java. Relacionar adapter y fragmentos

El código de la actividad TabbedActivity tendrá el siguiente aspecto:

public class TabbedActivity extends AppCompatActivity {  
 ActivityTabbedBinding binding;  
 private TabLayout tabLayout;  
 private ViewPager2 viewPager2;  
 private TabbedAdapter adaptador;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 binding = ActivityTabbedBinding.*inflate*(getLayoutInflater());  
 setContentView(binding.getRoot());  
  
 setSupportActionBar(binding.includeTabbedToolbar.toolbar);  
  
 *//Establecemos el título en el ToolBar* setTitle("TabLayout y ViewPager2");

*//Recuperamos el visor de fragmentos* viewPager2 = binding.viewPager;  
  
 *//Configurar el adapter* adaptador = new TabbedAdapter(this);  
  
 *//Añadir los fragmentos* adaptador.add(new tabbed1Fragment());  
 adaptador.add(new tabbed2Fragment());  
 adaptador.add(new tabbed3Fragment());  
  
 *//Establecer el adaptador* viewPager2.setAdapter(adaptador);  
  
 tabLayout = binding.tabLayout;  
  
 *//Vincular el TabLayout con el ViewPager2*  
new TabLayoutMediator(tabLayout, viewPager2,   
 new TabLayoutMediator.TabConfigurationStrategy() {  
 @Override  
 public void onConfigureTab(@NonNull TabLayout.Tab tab, int position) { *// Configura el texto de cada tab manualmente* switch (position) {  
 case 0:  
 tab.setText("Fragmento " + (position + 1));  
 break;  
 case 1:  
 tab.setText("Fragmento " + (position + 1));  
 break;  
 case 2:  
 tab.setText("Fragmento " + (position + 1));  
 break;  
 }  
 }  
  
 }).attach();  
 }  
}

El método setTitle() se utiliza para cambiar el título de la Toolbar en la actividad. La clase AppCompatActivity hereda de Activity, y el método setTitle() cambia el título de la actividad. Si la actividad utiliza una Toolbar personalizada, que ha sido configurada con setSupportActionBar(toolbar), entonces setTitle() actualizará el título mostrado en la Toolbar.

El componente ViewPager2 es el encargado de gestionar la paginación entre los fragmentos. En este código, se obtiene la referencia al ViewPager2 desde el binding. Luego, se crea una instancia del adaptador personalizado TabbedAdapter. Este adaptador tiene la responsabilidad de gestionar y proporcionar los fragmentos que se mostrarán dentro del ViewPager2. Posteriormente, se añaden tres fragmentos (tabbed1Fragment, tabbed2Fragment, y tabbed3Fragment) al adaptador para que puedan ser visualizados al navegar entre las páginas.

Una vez que los fragmentos han sido añadidos al adaptador, se establece dicho adaptador en el ViewPager2 para gestionar la navegación entre los fragmentos. Adicionalmente, se obtiene una referencia al TabLayout utilizando el objeto binding.

Creamos un objeto TabLayoutMediator para vincular el TabLayout con el ViewPager2. Esta conexión garantiza que, al cambiar de página en el ViewPager2, las pestañas del TabLayout se actualicen automáticamente.

Dentro del TabLayoutMediator, se utiliza un objeto TabConfigurationStrategy que permite personalizar el texto de las pestañas.

En este caso, el método onConfigureTab() se encarga de asignar el texto correspondiente a cada pestaña según su posición. Aquí, las pestañas se denominan "Fragmento 1", "Fragmento 2", y "Fragmento 3", de acuerdo con la posición de cada fragmento.

Por último, se invoca el método attach() del TabLayoutMediator para completar la configuración y establecer la sincronización entre el TabLayout y el ViewPager2.