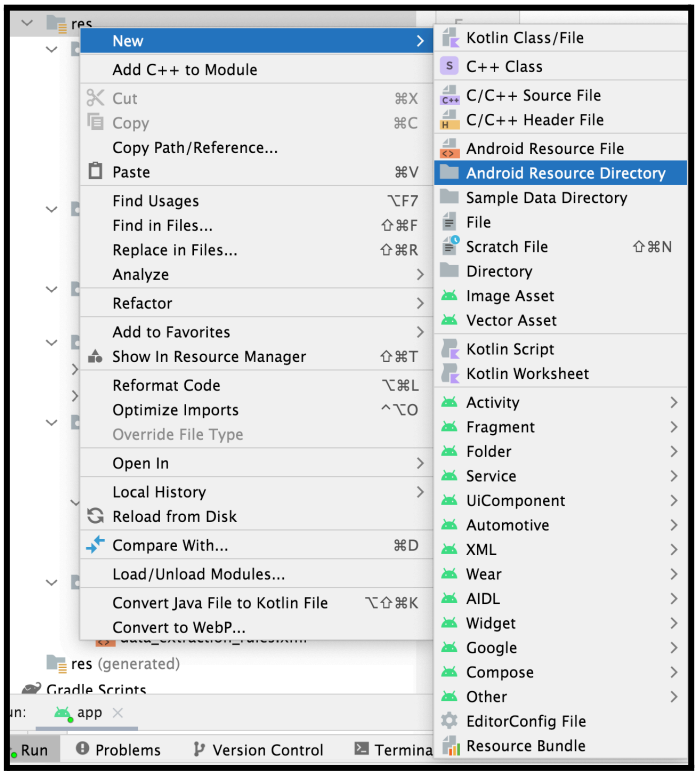
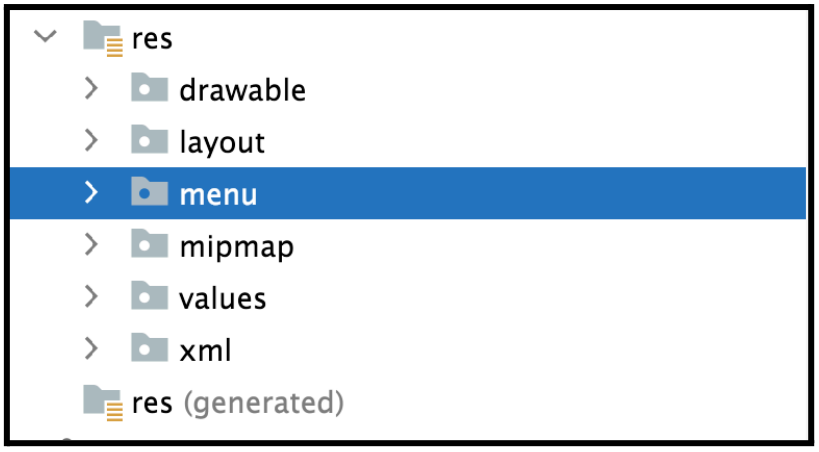
Tema 8. Menús y preferencias de usuario

8.1 Definición de un menú XML

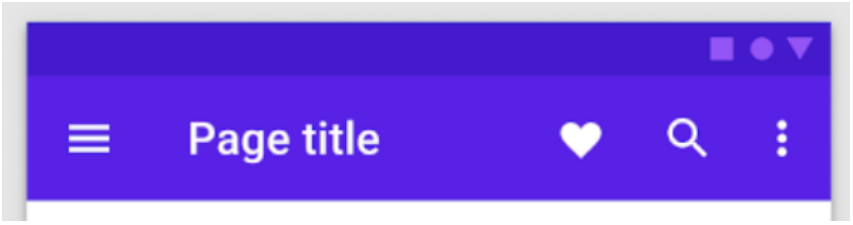
Los menús son un componente común de la interfaz de usuario en muchos tipos de aplicaciones. Para proporcionar una experiencia de usuario conocida y uniforme, debes usar las API de ***Menu*** a fin de presentar al usuario acciones y otras opciones en las actividades.

Los menús se incluyen en la carpeta de recursos de Android Studio **/res/menu**. Esta carpeta no está creada de inicio, por lo que deberás añadirla.



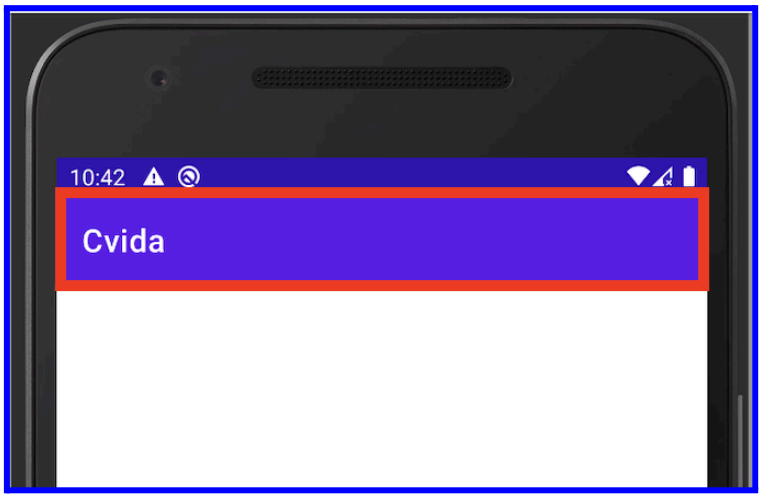


8.2 Menú de opciones y barra de app



La primera parte de la configuración de la ***ToolBar*** consiste en anular la ***ActionBar***, y posteriormente insertar la ***ToolBar*** como un objeto del *Activity*. Puedes ver los pasos en la documentación de material design.

- ***Paso 1. Anular la ActionBar***



La ***ActionBar*** fue introducida en versiones más antiguas de Android (antes de la versión 3.0, Honeycomb). Es parte integral de la actividad y se obtiene mediante actionBar en la actividad.

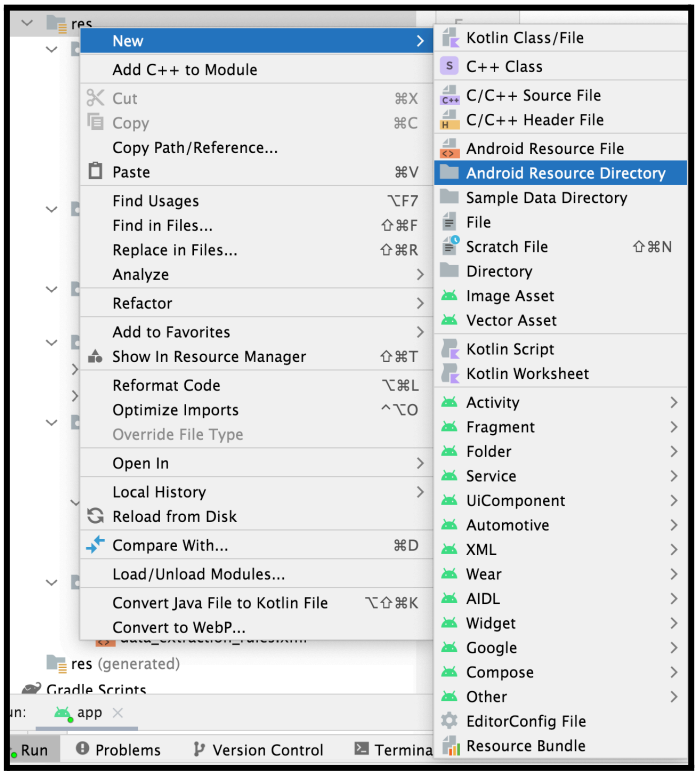
Tiene funcionalidades predeterminadas, como la capacidad de mostrar el título de la actividad, un icono de aplicación y acciones contextuales.

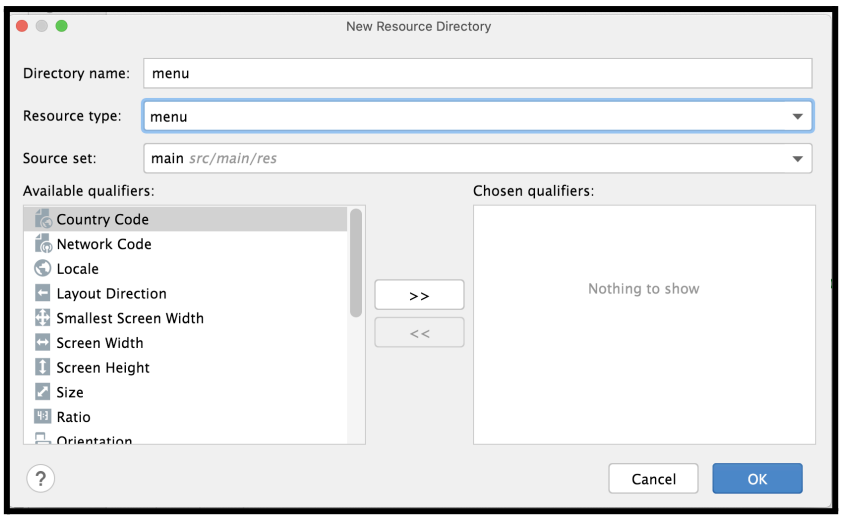
La personalización de ***ActionBar*** es posible, pero puede ser limitada en comparación con Toolbar.

Para anular esta barra, debemos cambiar el tema en el fichero *AndroidManifest.xml*, por un tema con el sufijo ***NoActionBar***.

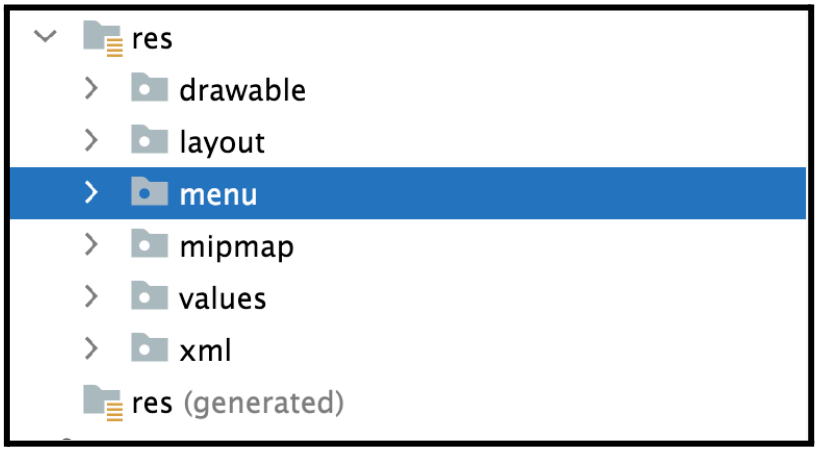
```
android:theme="@style/Theme.MaterialComponents.DayNight.NoActionBar"
```

- ***Paso 2. Añadimos la carpeta de recursos MENU***

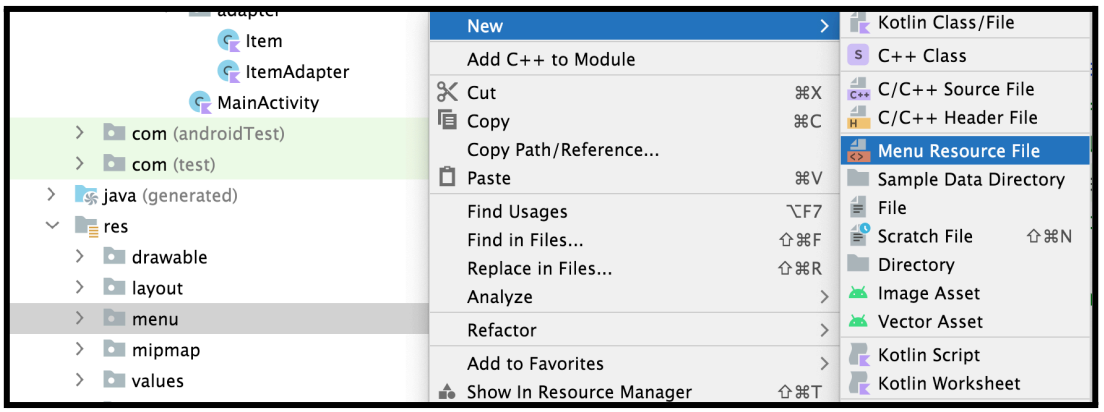




Nuestra carpeta de recursos ahora incluye la subcarpeta */menu*.



- ***Paso 3. Añadimos un fichero de menú a la carpeta MENU***



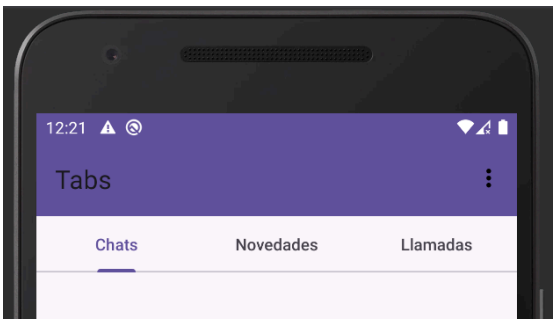
- ***Paso 4. Insertamos el siguiente código en el fichero XML "top_app_bar.xml"***

```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/action_overflow"
            android:icon="@drawable/ic_more"
            android:title="more"
            app:showAsAction="always">
        <menu>
            <!-- Opciones dentro del menú de desbordamiento -->
            <item
                    android:id="@+id/menu_option1"
                    android:title="Opción A" />
            <item
                    android:id="@+id/menu_option2"
                    android:title="Opción B" />
        </menu>
    </item>

</menu>
```

- ***Paso 5. Y en el fichero activity_main.xml se añade:***

```xml
    <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            android:elevation="4dp"
            android:theme="?attr/actionBarTheme"/>
```

Ahora faltaría programar el comportamiento de la ***ToolBar***.



- ***Paso 6. Programar el comportamiento de la ToolBar desde el fichero MainActivity.kt:***

En primer lugar, debemos añadir la siguiente línea en el método ***OnCreate()***:

```
class MainActivity : AppCompatActivity()
{
    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //Establecemos nuestra ToolBar en lugar de la ActionBar
        setSupportActionBar(findViewById(R.id.toolbar))
```

Además, se deben redefinir los métodos ***onCreateOptionsMenu()*** y ***onOptionsItemSelected()***.

El primer método, infla la vista de la ***ToolBar***, tal como se haría con un ***fragment***.

```
    //Infla la ToolBar
    override fun onCreateOptionsMenu(menu: Menu?): Boolean
    {
        menuInflater.inflate(R.menu.top_app_bar, menu)
        return true
    }
```

El segundo método, gestiona el evento *onClick()* sobre cada uno de los elementos de la ***ToolBar***.

```
    //Gestionar los eventos onClick sobre los elementos de la ToolBar
    override fun onOptionsItemSelected(item: MenuItem): Boolean
    {
        when (item.itemId)
        {
            R.id.action_overflow->{
                // Acción cuando se selecciona el primer elemento del menú
                return true
            }

            R.id.menu_option1 ->{

                Toast.makeText( context: this,  text: "Has pulsado la opción A",
                    Toast.LENGTH_LONG).show()

                return true
            }

            R.id.menu_option2 ->{

                Toast.makeText( context: this,  text: "Has pulsado la opción B",
                    Toast.LENGTH_LONG).show()

                return true
            }

            // Agrega más casos según tus necesidades
            else -> return super.onOptionsItemSelected(item)
        }
    }
```

8.2.1 SearchView

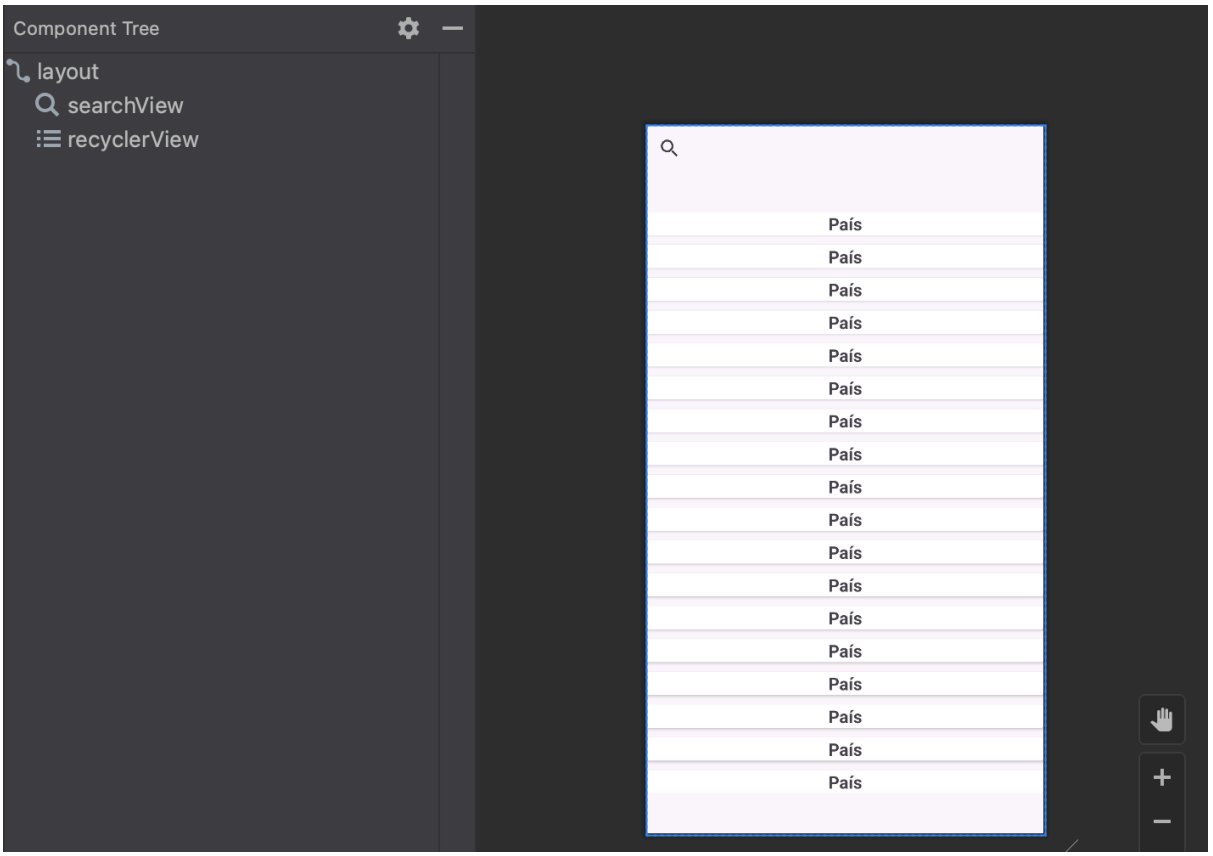
Es un widget que proporciona una interfaz de usuario para que los usuarios ingresen y realicen búsquedas en una aplicación. Este widget generalmente se coloca en la barra de acción (ActionBar) o en la barra de herramientas (***Toolbar***) de una actividad.

Al usar ***SearchView***, los usuarios pueden ingresar consultas de búsqueda y ver los resultados de la búsqueda directamente en la interfaz de usuario de la aplicación.

El widget ***SearchView*** es bastante flexible y ofrece varias opciones de personalización para adaptarse al diseño y estilo de la aplicación.

Vamos a ver un ejemplo a continuación.

En primer lugar, se ha diseñado la vista del activity, donde se tiene un ***ConstraintLayout***, que contiene una ***SearchView*** y un ***RecyclerView***, sobre el que haremos el filtrado de elementos.



Cada uno de los ítems del ***RecyclerView***, es simplemente un elemento de tipo ***TextView***, para hacerlo lo más sencillo posible.

El código necesario para resolver el funcionamiento de la ***SearchView*** es:

```
class MainActivity : AppCompatActivity() {

    private lateinit var searchView: SearchView
    private lateinit var recyclerView: RecyclerView

    private val allCountries = listOf(
        Country( name: "Argentina"),
        Country( name: "Brazil"),
        Country( name: "Canada"),
        Country( name: "Germany"),
        Country( name: "India"),
        Country( name: "Japan"),
        Country( name: "United States"),
        // Add more countries as needed
    )

    private var filteredCountries = allCountries.toMutableList()

    override fun onCreate(savedInstanceState: Bundle?)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        searchView = findViewById(R.id.searchView)
        recyclerView = findViewById(R.id.recyclerView)

        recyclerView.layoutManager = LinearLayoutManager( context: this)
        val adapter = CountryAdapter(filteredCountries)
        recyclerView.adapter = adapter
```

Rescatamos los elementos de la vista

Añadimos varios elementos al RecyclerView

Esta lista se usará para almacenar los elementos que cumplen los criterios de filtrado

Rescatamos los elementos de la vista

Configuración del RecyclerView

```
        searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {

            override fun onQueryTextSubmit(query: String?): Boolean
            {
                return false
            }

            override fun onQueryTextChange(newText: String?): Boolean
            {
                filterCountries(newText)
                return true
            }
        })
    }
```

Este es el Listener a implementar, para las búsquedas por texto

Se captura este evento, cada vez que se añade o elimina un carácter en la SearchView

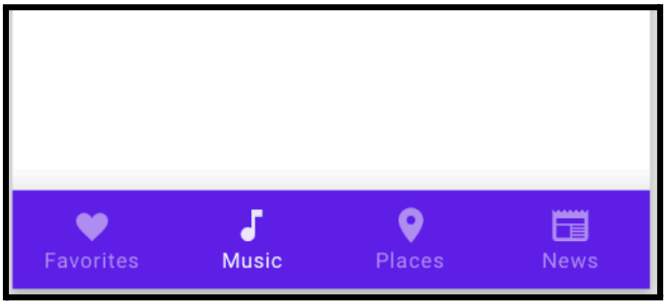
Por último, nos faltaría implementar el método **filterCountries**, que selecciona aquellos países que cumplen con el patrón de texto introducido en la ***SearchView***.

```
private fun filterCountries(query: String?)
{
    filteredCountries.clear()

    if (query.isNullOrEmpty()) {
        filteredCountries.addAll(allCountries)
    } else {
        val lowercaseQuery = query.lowercase()
        filteredCountries.addAll(
            allCountries.filter { country ->
                country.name.lowercase().contains(lowercaseQuery)
            }
        )
    }

    recyclerView.adapter?.notifyDataSetChanged()
}
```

El método ***notifyDataSetChanged()*** actualiza la vista del ***RecyclerView***, para que solo se muestren los elementos de la lista que cumplen con el texto filtrado.

8.3 Bottom navigation menu

Las barras de navegación inferiores permiten el movimiento entre los destinos principales de una aplicación. Cada uno de los items del menú, nos llevará a un fragment distinto.

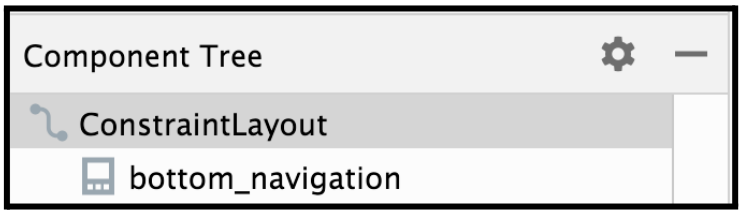


Para ver cómo implementar esta barra de navegación inferior, vamos a seguir el manual de la web https://m2.material.io/components/bottom-navigation

- *En el fichero **activity_main.xml** se añade un objeto de tipo **BottomNavigationView**:*

```
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/bottom_navigation"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:menu="@menu/bottom_navigation_menu" />
```

Quedando así nuestro árbol de componentes:

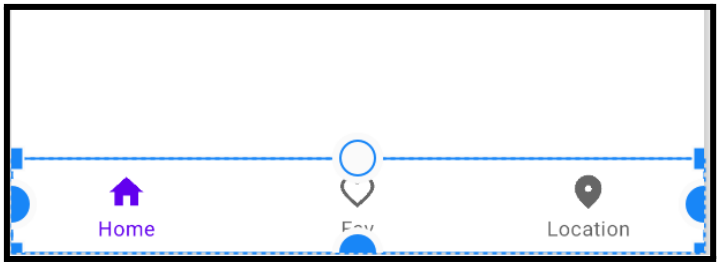


- *Como consecuencia de lo añadido en el fichero **activity_main.xml**, ahora debemos crear un nuevo fichero de menú que tenga el nombre especificado anteriormente **"bottom_navigation_menu"** :*

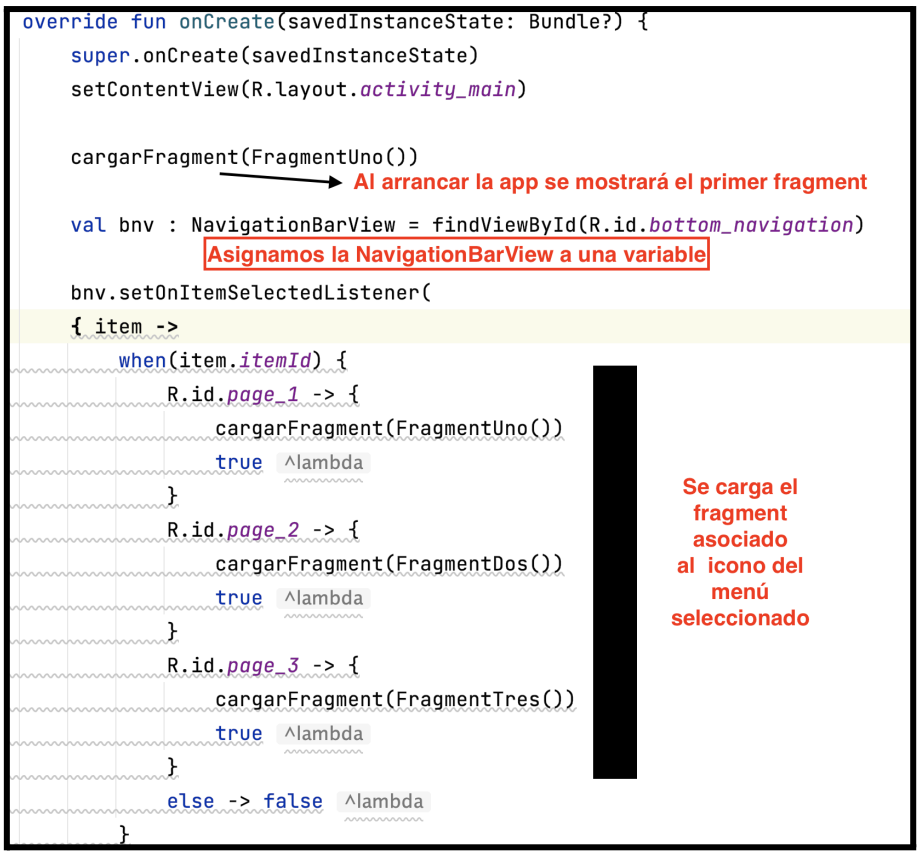
- *Este fichero va a tener el siguiente código:*

```xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/page_1"
        android:enabled="true"
        android:icon="@drawable/ic_baseline_home_24"
        android:title="Home"/>
    <item
        android:id="@+id/page_2"
        android:enabled="true"
        android:icon="@drawable/ic_baseline_favorite_border_24"
        android:title="Fav"/>

    <item
        android:id="@+id/page_3"
        android:enabled="true"
        android:icon="@drawable/ic_baseline_fmd_good_24"
        android:title="@string/icono3"/>
</menu>
```

Se han insertado tres items, el primero llamado *"Home"*, el segundo *"Fav"* y el tercero *"Location"*. El menú se ve ahora así:



- ***Ya solo falta indicar en el activity qué fragment se debe cargar en cada caso:***



8.4 Navigation drawer menu

Un ***Navigation Drawer Menu*** es un panel deslizable que se muestra desde el borde de la ventana de una aplicación, comúnmente desde el lado izquierdo, pero también puede configurarse para que aparezca desde el lado derecho. Este panel es una forma popular y eficiente de navegar entre las diferentes áreas de una aplicación móvil, especialmente en Android.

El ***Navigation Drawer*** suele contener una lista de opciones de navegación o destinos dentro de la aplicación. Estos elementos de navegación a menudo están representados por íconos y etiquetas de texto, facilitando a los usuarios la identificación de las diferentes secciones de la aplicación a las que pueden acceder desde el menú.

La implementación del ***Navigation Drawer*** en Android generalmente se realiza utilizando el componente ***DrawerLayout*** como contenedor raíz en el archivo de diseño XML de la actividad. Dentro del ***DrawerLayout***, se coloca el contenido principal de la interfaz de usuario y el diseño del menú de navegación (a menudo un ***NavigationView***), que se desliza desde el borde.

El ***Navigation Drawer*** se integra comúnmente con la barra de herramientas (***Toolbar***) de la aplicación, proporcionando un botón (a menudo el botón de hamburguesa, que consta de tres líneas horizontales) para abrir y cerrar el menú. Además, el menú puede ser deslizado hacia afuera manualmente por el usuario.

Este tipo de menú es ampliamente utilizado en aplicaciones con múltiples niveles de navegación, ya que ofrece una forma clara y accesible de explorar la aplicación sin abrumar al usuario con demasiadas opciones en la pantalla principal. Además, ayuda a mantener la interfaz de usuario limpia y organizada al ocultar las opciones de navegación cuando no se utilizan.

Los pasos para implementar este menú en nuestra aplicación son:

1. **Añadimos a nuestro fichero de *gradle* las siguientes librerías:**

```
implementation ("com.google.android.material:material:1.4.0")
implementation
("androidx.navigation:navigation-fragment-ktx:2.3.5")
implementation ("androidx.navigation:navigation-ui-ktx:2.3.5")
```

2. **Creamos el *DrawerLayout* con los elementos necesarios:**

```
<?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"
       xmlns:tools="http://schemas.android.com/tools"
       android:id="@+id/drawer_layout"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:fitsSystemWindows="true"
       tools:openDrawer="start">

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

       <androidx.appcompat.widget.Toolbar
               android:id="@+id/toolbar"
               android:layout_width="match_parent"
               android:layout_height="?attr/actionBarSize"
               android:background="?attr/colorPrimary"
               app:titleTextColor="@android:color/white" />
```

```
        <fragment
                android:id="@+id/nav_host_fragment"

android:name="androidx.navigation.fragment.NavHostFragment"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                app:defaultNavHost="true"/>

    </LinearLayout>

    <!-- NavigationView para el menú del Drawer -->
    <com.google.android.material.navigation.NavigationView
            android:id="@+id/nav_view"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="start"
            android:fitsSystemWindows="true"
            app:headerLayout="@layout/nav_header"
            app:menu="@menu/drawer_menu" />

</androidx.drawerlayout.widget.DrawerLayout>
```

3. **Creamos el menú drawer *menu*. Previo a su creación (si no lo hemos hecho antes) debemos añadir a nuestro proyecto la subcarpeta de recursos *res/menu*.**

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
                android:id="@+id/nav_home"
                android:icon="@drawable/ic_home"
                android:title="Inicio" />
        <item
                android:id="@+id/nav_gallery"
                android:icon="@drawable/ic_gallery"
                android:title="Galería" />
        <item
                android:id="@+id/nav_slideshow"
                android:icon="@drawable/ic_slideshow"
                android:title="Diapositivas" />
    </group>
</menu>
```

Aquí tenemos como se verá el menú:

4. **Además, debemos crear un *layout* asociado a la cabecera del menú.**

```
<!-- res/layout/nav_header.xml -->
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="160dp"
        android:background="?attr/colorPrimaryDark"
        android:gravity="bottom"
        android:padding="16dp"
        android:theme="@style/ThemeOverlay.AppCompat.Dark">

    <!-- Puedes incluir una imagen de perfil, por ejemplo -->
    <ImageView
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:src="@mipmap/ic_launcher"
            android:layout_gravity="bottom"
            android:layout_marginRight="8dp"/>

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Nombre del Usuario"
            android:textColor="@android:color/white"
            android:textSize="16sp"
            android:layout_gravity="bottom"/>

</LinearLayout>
```

Esta cabecera o header se verá así:

5. **El siguiente paso consistirá en crear los diferentes *fragments* que se van a ir cargando a medida que se haga clic sobre los diferentes elementos del menú. Esta parte ya se ha realizado en contenidos anteriores.**

6. **Finalmente, nos vamos al fichero kotlin, en nuestro caso MainActivity.kt, para programar el comportamiento del menú.**

```
val navController = findNavController(R.id.nav_host_fragment) //Equivalente a FrameLayout
// Menu lateral - Drawer Menu
appBarConfiguration = AppBarConfiguration(setOf(R.id.nav_home,
    R.id.nav_gallery, R.id.nav_slideshow), drawerLayout)

setupActionBarWithNavController(navController, appBarConfiguration)
navigationView.setupWithNavController(navController)


//Toggle y DrawerLayout
val toggle = ActionBarDrawerToggle(
    activity: this, drawerLayout, toolbar, R.string.navigation_drawer_open, R.string.navigation_drawer_close
)
drawerLayout.addDrawerListener(toggle)
toggle.syncState()

supportFragmentManager
    .beginTransaction()
    .replace(R.id.nav_host_fragment, FirstFragment())
    .addToBackStack( name: null)
    .commit()
```

Estas líneas, configuran los diferentes ítems del menú

Aquí se configura el botón (tres líneas horizontales)

Por último, se carga el primero de los fragments

7. **Por último, debemos solucionar un problema asociado al botón de volver atrás (*Back pressed*).**

Al utilizar un ***Drawer Menu*** en aplicaciones móviles, especialmente en plataformas como Android, un problema común que puede surgir al presionar el botón de ir hacia atrás, es que el comportamiento por defecto de este botón es cerrar la aplicación o salir de la actividad actual, en lugar de simplemente cerrar el menú desplegable (***Drawer Menu***) si está abierto.

Esto puede resultar en una experiencia de usuario **poco intuitiva**, ya que los usuarios pueden esperar que al presionar el botón de ir hacia atrás se cierre el ***Drawer Menu*** si este está abierto, y solo si el ***Drawer Menu*** está cerrado, entonces considerar salir de la aplicación o retroceder en el historial de navegación.

Para manejar adecuadamente este comportamiento, se necesita implementar una lógica específica en el método **onBackPressed()** de la actividad que contiene el ***Drawer Menu***. Esta lógica debe verificar primero si el ***Drawer Menu*** está abierto. Si es así, el método debería cerrar el ***Drawer Menu*** y no llamar a **super.onBackPressed()**, que es lo que normalmente provocaría la salida de la aplicación o el retroceso en el historial de

navegación. Si el ***Drawer Menu*** no está abierto, entonces se puede proceder con el comportamiento por defecto de **onBackPressed()**.

Este enfoque asegura que el botón de ir hacia atrás funcione de manera más intuitiva para los usuarios, mejorando la usabilidad de la aplicación.

El método **onBackPressed()** ha quedado obsoleto recientemente, por lo que debemos usar la nueva solución, que consiste en utilizar un ***CallBack***.

```
val callback = object : OnBackPressedCallback( enabled: true /* enabled by default */) {
    override fun handleOnBackPressed()
    {
        // Maneja el evento de ir hacia atrás aquí
        if (drawerLayout.isDrawerOpen(GravityCompat.START))
        {
            drawerLayout.closeDrawer(GravityCompat.START)
        }
        else
        {
            isEnabled = false
            finish()
        }
    }
}

onBackPressedDispatcher.addCallback( owner: this, callback)
```

Así se vería la aplicación en funcionamiento: