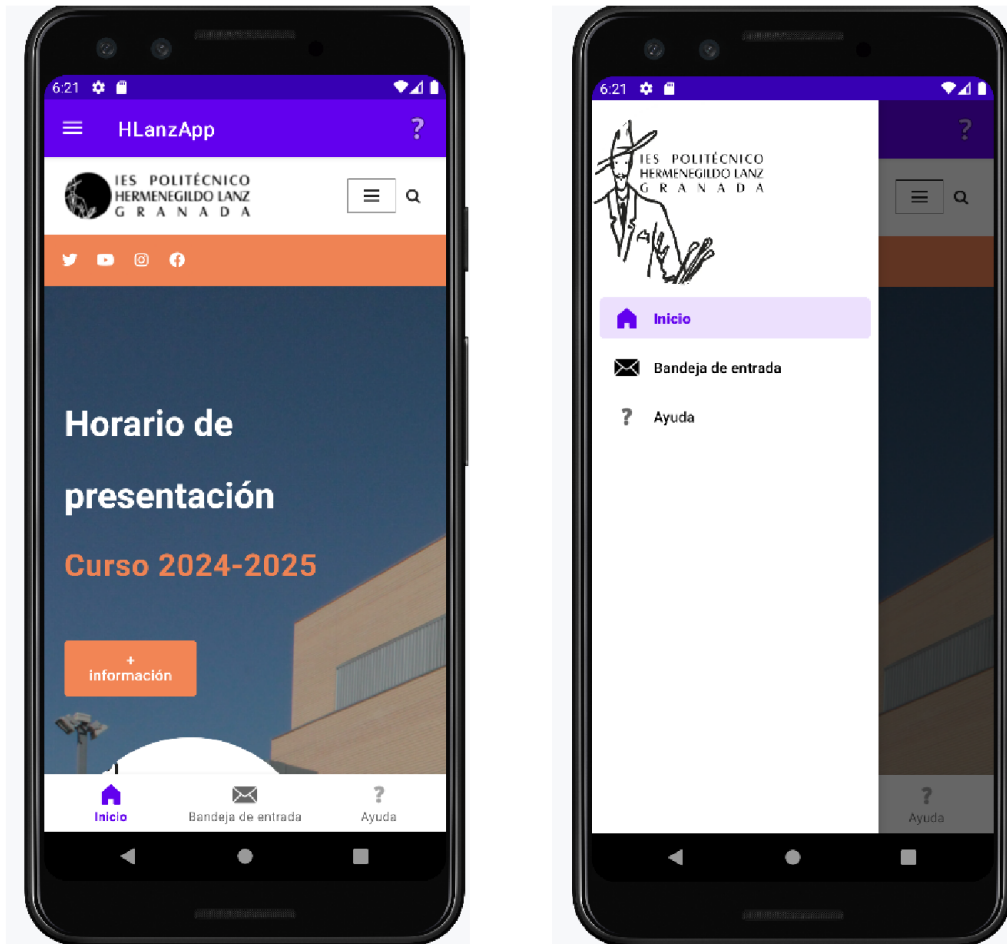


PROYECTO 7

HLANZ APP



En este proyecto haremos una app que tendrá varias pantallas por las que podemos navegar usando varias opciones (barra de navegación, barra de navegación en el fondo y navigation drawer). Veremos cómo incrustar un navegador en la interfaz, mostrar una barra de desplazamiento en la interfaz y poner una splash screen básica.

Durante su desarrollo se tratarán estos conceptos:

- WebView
- HTML en un TextView
- Navegación con la barra de tareas
- Botón de atrás en la barra de tareas
- BottomNavigationView
- NavigationDrawer
- ViewPager2
- TabLayout
- Splash Screen
- Introducción a las coroutines

1 – WebView

Un **WebView** es un componente de la interfaz que muestra un navegador que puede mostrar cualquier tipo de contenido web.

- Crea un proyecto y añade un **Fragment** llamado **InicioFragment**, eliminando del código fuente generado por Android Studio las partes obsoletas.
- Abre el archivo **AndroidManifest.xml** y añade la siguiente línea, para indicar que nuestra app necesita permiso de acceso a Internet:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

```
1. <manifest xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools">
3.     <application
4.         android:allowBackup="true"
5.         android:dataExtractionRules="@xml/data_extraction_rules"
6.         android:fullBackupContent="@xml/backup_rules"
7.         android:icon="@mipmap/ic_launcher"
8.         android:label="@string/app_name"
9.         android:roundIcon="@mipmap/ic_launcher_round"
10.        android:supportRtl="true"
11.        android:theme="@style/Theme.HLaznApp"
12.        tools:targetApi="31">
13.        <activity
14.            android:name=".MainActivity"
15.            android:exported="true">
16.            <intent-filter>
17.                <action android:name="android.intent.action.MAIN" />
18.                <category android:name="android.intent.category.LAUNCHER" />
19.            </intent-filter>
20.        </activity>
21.    </application>
22.    <uses-permission android:name="android.permission.INTERNET"/>
23. </manifest>
```

Todos los permisos que requiere nuestra app (acceso a Internet, al GPS, NFC, almacenamiento externo, etc) se indican en el archivo **AndroidManifest.xml**.

Al ejecutar la app, el dispositivo pregunta al usuario si se concede el permiso requerido. Sin embargo, el acceso a Internet es tan frecuente, que no se pregunta y se concede siempre

- Modifica el archivo **fragment_inicio.xml** para que tenga un **FrameLayout** con id **frmInicio** que contenga un **WebView** con id **webHlanz** y ancho y alto toda la pantalla

```
1. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:id="@+id/frmInicio"
4.     android:layout_width="match_parent"
5.     android:layout_height="match_parent"
6.     android:orientation="vertical"
7.     tools:context=".InicioFragment">
8.     <WebView
9.         android:id="@+id/webHlanz"
10.        android:layout_width="match_parent"
11.        android:layout_height="match_parent"/>
12. </LinearLayout>
```

- Modifica el archivo **activity_main.xml** para que contenga un **LinearLayout** con un **FragmentContainerView** que muestre en su interior a **InicioFragment**

```

1. <LinearLayout
2.     xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     android:id="@+id/frnPrincipal"
5.     android:orientation="vertical"
6.     android:layout_width="match_parent"
7.     android:layout_height="match_parent">
8.     <androidx.fragment.app.FragmentContainerView
9.         android:id="@+id/fragment_container_view"
10.        android:layout_width="match_parent"
11.        android:layout_height="match_parent"
12.        android:name="dam.moviles.hlaznapp.InicioFragment"/>
13. </LinearLayout>

```

- Habilita el **view binding** en el proyecto e inicialízalo en **InicioFragment**

```

1. class InicioFragment : Fragment() {
2.     private var _binding: FragmentInicioBinding? = null
3.     private val binding: FragmentInicioBinding
4.         get()= checkNotNull(_binding) { "uso incorrecto del objeto binding"}
5.     override fun onCreateView(
6.         inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View? {
7.         inicializarBinding(inflater,container)
8.         return binding.root
9.     }
10.    private fun inicializarBinding(inflater: LayoutInflater, container: ViewGroup?){
11.        _binding= FragmentInicioBinding.inflate(inflater,container,false)
12.    }
13.    override fun onDestroyView() {
14.        super.onDestroyView()
15.        _binding=null
16.    }
17. }

```

- Crea un método llamado **inicializarWebView** en el que configuraremos el **WebView** para que cargue la página <https://www.ieshlanz.es>. Llamaremos a dicho método dentro de **onCreateView**

```

1. class InicioFragment : Fragment() {
2.     // resto de la clase omitido
3.     override fun onCreateView(
4.         inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle? ): View? {
5.         inicializarBinding(inflater,container)
6.         inicializarWebView()
7.         return binding.root
8.     }
9.     private fun inicializarWebView(){
10.        binding.webHLanz.webViewClient = WebViewClient()
11.        binding.webHLanz.loadUrl("https://www.ieshlanz.es")
12.    }
13. }

```

El **WebView** tiene estas propiedades y métodos:

- **javascriptEnabled:** Se pone a **true** para habilitar JavaScript en el **WebView**
- **webViewClient:** Es conveniente inicializarlo para asegurarnos de que si una web nos redirige a otra, esta se abra en nuestro **WebView**
- **loadUrl:** Hace que el **WebView** navegue al sitio web que indicamos

2 – HTML en un TextView

El **TextView** que hemos usado hasta ahora también es capaz de mostrar contenido básico en HTML (para cosas más complejas, es necesario recurrir al **WebView**)

- Añade al archivo **Strings.xml** el siguiente String llamado **ayuda** cuyo valor es una página web con un mensaje de ayuda

```
1. <resources>
2.     <string name="app_name">HLanz APP</string>
3.     <string name="hello_blank_fragment">Hello blank fragment</string>
4.     <string name="ayuda">
5.         <![CDATA[
6.             <html>
7.                 <body>
8.                     <h1>Ayuda</h1><br>
9.                     <p>La HLANZ APP está diseñada para navegar cómodamente
10.                    con la barra de tareas (Toolbar) y el menú inferior
11.                    de la pantalla (Bottom Navigation Bar)</p><br>
12.                    <p>También está a disposición del usuario un
13.                    Navigation Drawer en el que aparecen las mismas opciones, por
14.                    si prefieres acceder a ellas de esa forma</p><br>
15.                    <ul>
16.                        <li><b>Inicio</b>: Aquí verás la web del instituto</li>
17.                        <li><b>Bandeja de entrada</b>: Aquí accederás a tus mensajes</li>
18.                        <li><b>Ayuda</b>: Aquí encontrarás cómo usar la app</li>
19.                    </ul>
20.                </body>
21.            </html>
22.        ]]>
23.    </string>
24.    <string name="sin_mensajes">No hay mensajes en la bandeja de entrada</string>
25. </resources>
```

Cuando en un xml queremos escribir texto literal que contenga símbolos no permitidos en xml (como los < >), se usa una sección **CDATA** para encerrarlo

- Añade también al archivo **strings.xml** los siguientes strings:

Clave	String
sin_mensajes	No hay mensajes recibidos
sin_spam	No hay mensajes de spam
sin_papelera	No hay mensajes en la papelera

- Añade al proyecto un nuevo **Fragment** llamado **AyudaFragment**, y elimina el código fuente redundante.
- Modifica el archivo **fragment_ayuda.xml** para que su contenedor principal sea un **FrameLayout** que contenga un **TextView** que muestre el mensaje **ayuda**, usando como ancho y alto toda la pantalla. No importa el mensaje que se ponga en dicho **TextView**

```

1. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:padding="16dp"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent">
5.     <TextView
6.         android:id="@+id/txtAyuda"
7.         android:layout_width="match_parent"
8.         android:layout_height="match_parent"
9.         android:text="Aquí vendrá el mensaje de ayuda en formato HTML" />
10. </FrameLayout>

```

Como el texto que se va a mostrar en el **TextView** está en HTML, no es posible añadirlo tal cual, sino que deberemos añadirlo en el código fuente de **AyudaFragment**

- Añade a **frmAyuda** la variable **binding** e inicialízala en un método **inicializarBinding**

```

1. class AyudaFragment : Fragment() {
2.     private var _binding: FragmentAyudaBinding? = null
3.     private val binding: FragmentAyudaBinding
4.         get()= checkNotNull(_binding){"Uso incorrecto del objeto binding"}
5.     override fun onCreateView(
6.         inflater: LayoutInflater, container: ViewGroup?,
7.         savedInstanceState: Bundle?
8.     ): View? {
9.         inicializarBinding(inflater,container)
10.        return binding.root
11.    }
12.    private fun inicializarBinding(inflater: LayoutInflater, container: ViewGroup?){
13.        _binding= FragmentAyudaBinding.inflate(inflater,container,false)
14.    }
15.    override fun onDestroyView() {
16.        super.onDestroyView()
17.        _binding=null
18.    }
19. }

```

- Añade a **AyudaFragment** un método llamado **inicializarTexto** en el que se mostrará en **txtAyuda** el mensaje **ayuda**, pero usando formato HTML.

```

1. class AyudaFragment : Fragment() {
2.     // resto del código omitido
3.     override fun onCreateView(
4.         inflater: LayoutInflater, container: ViewGroup?,
5.         savedInstanceState: Bundle?
6.     ): View? {
7.         inicializarBinding(inflater,container)
8.         inicializarTexto()
9.         return binding.root
10.    }
11.    private fun inicializarTexto(){
12.        val texto:String = getString(R.string.ayuda)
13.        binding.txtAyuda.text = Html.fromHtml(texto,Html.FROM_HTML_MODE_COMPACT)
14.    }
15. }

```

Para mostrar el texto de ayuda en formato HTML, primero recuperamos su valor, y luego lo pasamos al método **HTML.fromHtml** para que este se muestre correctamente.

- Añade un **Fragment** llamado **BandejaEntradaFragment** que simplemente tenga un texto con el mensaje referenciado en el string **sin_mensajes**

```

1. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent" android:layout_height="match_parent">
4.     <TextView
5.         android:layout_width="match_parent" android:layout_height="match_parent"
6.         android:textSize="20sp"
7.         android:text="@string/sin_mensajes"/>
8. </FrameLayout>

```

- **Habilita view binding en BandejaEntradaFragment**

```

1. class BandejaEntradaFragment : Fragment() {
2.     var _binding:FragmentBandejaEntradaBinding? = null
3.     val binding:FragmentBandejaEntradaBinding
4.         get() = checkNotNull(_binding) {"Uso incorrecto del objeto binding"}
5.     override fun onCreateView(
6.         inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
7.     ): View? {
8.         inicializarBinding(inflater,container)
9.         return binding.root
10.    }
11.    private fun inicializarBinding(inflater: LayoutInflater, container: ViewGroup?){
12.        _binding = FragmentBandejaEntradaBinding.inflate(inflater,container,false)
13.    }
14.    override fun onDestroyView() {
15.        super.onDestroyView()
16.        _binding=null
17.    }

```

- **Añade tres Fragments**, cada uno con un un **EditText** con estos textos:

Fragment	Mensaje
MensajesRecibidosFragment	@string/sin_mensajes
SpamFragment	@string/sin_spam
PapeleraFragment	@string/sin_papelera

3 – Navegación con la barra de tareas

La barra de tareas puede contener iconos y menús con los que podemos cambiar el **Fragment** que se ve en el **FragmentContainerView**.

La librería **Navigation Component** nos ayudará a hacer esto.

- **Inicializa el view binding en MainActivity**

```

1. class MainActivity : AppCompatActivity() {
2.     lateinit var binding:ActivityMainBinding
3.     override fun onCreate(savedInstanceState: Bundle?) {
4.         super.onCreate(savedInstanceState)
5.         inicializarBinding()
6.         setContentView(binding.root)
7.     }
8.     private fun inicializarBinding(){
9.         binding=ActivityMainBinding.inflate(layoutInflater)
10.    }
11. }

```

- Abre el archivo `/res/values/themes/themes.xml`, busca el elemento **style** y cambia su atributo **parent** por:

```
1. <style
2.     name="Base.Theme.HLaznApp"
3.     parent="Theme.MaterialComponents.DayNight.NoActionBar">
```

Con esta acción, hemos puesto un estilo a nuestra app que oculta la barra de tareas que las apps de Android traen por defecto, y que es antigua.

- Añade a **main_activity.xml** una **MaterialToolbar** con estas características:
 - id: **material_toolbar**
 - anchura: la de su padre
 - altura: La definida en la variable del tema **actionBarSize**
 - estilo: **Widget.MaterialComponents.ToolBar.Primary**

```
1. <com.google.android.material.appbar.MaterialToolbar
2.     android:id="@+id/material_toolbar"
3.     android:layout_width="match_parent"
4.     android:layout_height="?attr/actionBarSize"
5.     style="@style/Widget.MaterialComponents.ToolBar.Primary"/>
```

- Abre el archivo **MainActivity** y crea un método llamado **configurarToolbar**, donde vamos a hacer que nuestra **MaterialToolBar** reemplace a la **ActionToolBar** de las versiones antiguas de Android. Este método será llamado en **onCreate**

```
1. class MainActivity : AppCompatActivity() {
2.     // resto omitido
3.     override fun onCreate(savedInstanceState: Bundle?) {
4.         super.onCreate(savedInstanceState)
5.         inicializarBinding()
6.         configurarToolbar()
7.     }
8.     private fun configurarToolbar() {
9.         setSupportActionBar(binding.materialToolbar)
10.    }
11. }
```

El método **setSupportActionBar** hace que la **ActionToolBar** de las versiones antiguas de Android sea reemplazado por la toolbar que le pasamos por parámetro (en nuestro caso la **MaterialToolbar**).

- Ejecuta la app y comprueba que aparece una barra de tareas en la parte superior de la pantalla.
- Añade al proyecto la librería **Navigation Component**
- Crea un **navigation graph** en un archivo llamado **nav_graph.xml**
- Usando el diseñador gráfico, añade **InicioFragment**, **AyudaFragment** y **BandejaEntradaFragment** al **nav_graph.xml**

- Modificamos el **activity_main.xml** para que el **Fragment** que se muestre en su interior sea el **NavHostFragment**, que como sabemos, es un **Fragment** que permite la navegación indicada en el **nav_graph.xml**

```

1. <androidx.fragment.app.FragmentContainerView
2.     android:id="@+id/frmNavHostFragment"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     android:name="androidx.navigation.fragment.NavHostFragment"
6.     app:defaultNavHost="true"
7.     app:navGraph="@navigation/nav_graph"/>

```

- Ejecuta la app y comprueba que todo se sigue viendo igual
- Pulsa el botón derecho sobre **/res** y elige **new** → **android resource file**
- Rellena la ventana que aparece con esta información:
 - o Nombre: **menu_barra_tareas**
 - o Tipo de recurso: **Menu**
- Observa que se crea una carpeta llamada **menú** y dentro de ella hay un archivo llamado **menu_barra_tareas.xml**

En el archivo **menu_barra_tareas.xml** vamos a dar forma al menú que se va a mostrar en la barra de tareas

- Abre **menú_barra_tareas.xml** con la vista de diseñador
- Arrastra al menú un **MenuItem** con estas características:
 - o id: **ayudaFragment**
 - o icono: **@android:drawable/ic_menu_help**
 - o title: **Ayuda**
 - o showAsAction: **always**
 - o Dentro de la etiqueta **menú** añade una opción que al pulsarla nos lleve a **AyudaFragment**
- Abre **menu_barra_tareas.xml** con la vista xml y comprueba que su código es este:

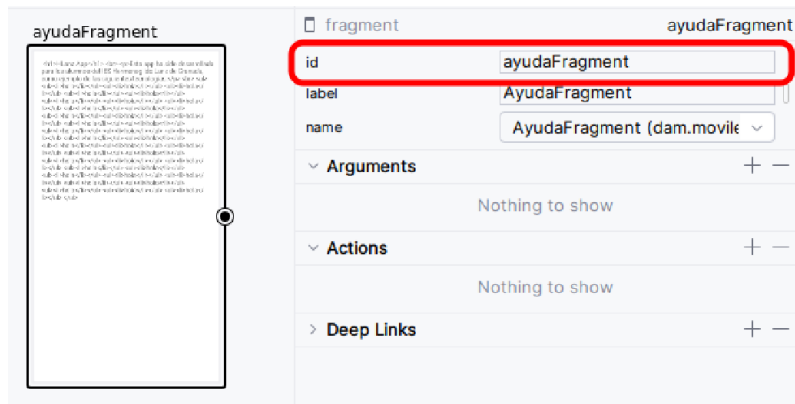
```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android"
2.     xmlns:app="http://schemas.android.com/apk/res-auto">
3.     <item
4.         android:id="@+id/ayudaFragment"
5.         android:icon="@android:drawable/ic_menu_help"
6.         android:title="Ayuda"
7.         app:showAsAction="always"/>
8. </menu>
9.

```


Las opciones de los menús se definen con la etiqueta **ítem** y sus atributos son:

- **android:id** → Es el id del **Fragment** que se mostrará al pulsar la opción, tal y como aparece en el **navigation graph**



- **android:icon** → Es el icono que tendrá la opción. Puede usarse cualquier icono que pongamos en nuestra carpeta **/res/drawable**, o podemos elegir iconos que trae Android si usamos **@android:drawable**
- **android:title** → Es el texto que se mostrará en la opción del menú
- **app:showAsAction** → Si ponemos **true**, la opción siempre se mostrará, mientras que si ponemos **false**, la opción se agrupará con otras en un submenú en el caso de que no haya espacio suficiente en la pantalla.

- Para vincular el archivo **menú_barra_tareas.xml** con nuestra **MaterialToolbar**, abre el archivo **MainActivity** y dentro del método **configurarToolbar** llamamos al método **addMenuProvider**, al que pasaremos un objeto que implementa la interfaz **MenuProvider**

```

1. private fun configurarToolbar(){
2.     materialToolbar=findViewById(R.id.material_toolbar)
3.     setSupportActionBar(materialToolbar)
4.     addMenuProvider( object:MenuProvider{
5.         override fun onCreateMenu(menu: Menu, menuInflater: MenuInflater) {
6.             // aquí cargamos el archivo R.menu.menu_toolbar y se lo ponemos a "menú"
7.         }
8.         override fun onOptionsItemSelected(menuItem: MenuItem): Boolean {
9.             // aquí programaremos lo que sucede al pulsar en la opción "menuItem"
10.            return true
11.        }
12.    })
13. }

```

- Programa **onCreateMenu** para que se llame al método **inflate** del objeto **menuInflater** y cargue el menú cuyo id es **R.menu.menu_barra_tareas**

```

1. override fun onCreateMenu(menu: Menu, menuInflater: MenuInflater) {
2.     menuInflater.inflate(R.menu.menu_barra_tareas,menu)
3. }

```

- Ejecuta la app y comprueba que en la toolbar aparece un icono de ayuda

- Programa **onMenuItemSelected** para que cuando se pulse una opción se navegue hacia el **Fragment** cuyo id (en el **navigation graph**) es el de esa opción.

```
1. override fun onMenuItemSelected(menuItem: MenuItem): Boolean {
2.     val navController = findNavController(R.id.fragment_container_view)
3.     menuItem.onNavDestinationSelected(navController)
4.     return true
5. }
```

El método **onNavDestinationSelected** recibe el **NavController** y hace que este navegue al **Fragment** cuyo id (en el **navigation graph**) es el mismo del **MenuItem**

- Ejecuta la app y comprueba que:
 - Al pulsar el botón de ayuda, la pantalla cambia a **AyudaFragment**
 - Al pulsar el botón de atrás del dispositivo, volvemos a **InicioFragment**

4 – Botón de atrás en la barra de tareas

Es habitual que en las barras de tareas se muestre un icono que nos permita volver hacia atrás, para no tener que pulsar el botón de atrás del dispositivo.

- Añade a **MainActivity** el siguiente método, que nos permite obtener el **NavController**

```
1. fun getNavController():NavController{
2.     val navHostFragment = supportFragmentManager
3.         .findFragmentById(R.id.fragment_container_view) as NavHostFragment
4.     return navHostFragment.navController
5. }
```

Cuando programamos **MainActivity** tenemos a nuestra disposición a **supportFragmentManager**, que conoce a todos los **Fragment** que hay en la **Activity**.

Dicho objeto posee un método llamado **findFragmentById**, que recibe el id de un **Fragment** y nos devuelve dicho objeto **Fragment**. Es necesario un **casting** a **NavHostFragment**, para acceder a sus métodos.

- En **MainActivity** crea un objeto de tipo **AppBarConfiguration**, que sirve para configurar la barra de tareas. Dicho método recibe el **NavController**

```
1. private fun configurarMaterialToolbar() {
2.     // código previo del método omitido
3.     val navController = getNavController()
4.     val configuracion = AppBarConfiguration.Builder(navController.graph).build()
5. }
```

La clase **AppBarConfiguration.Builder** sirve para obtener un objeto **AppBarConfiguration** mediante su método **build**.

Por defecto, la configuración incluye un icono de atrás, que nos permite volver a la pantalla previa.

- Por último, ponemos a la barra de tareas la configuración y el **NavigationController** con el método **NavigationUI.setupWithNavController**:

```

1. private fun configurarToolbar() {
2.     // código previo del método omitido
3.     val navController = getNavController()
4.     val configuracion = AppBarConfiguration.Builder(navController.graph).build()
5.     NavigationUI.setupWithNavController(binding.materialToolbar, navController, configuracion)
6. }

```

- Ejecuta la app y comprueba que:
 - Al pulsar el botón de ayuda, aparece una flecha hacia atrás en la barra de tareas, que nos permite volver hacia la pantalla de inicio.
 - En la barra de tareas aparece el nombre del **Fragment** que se muestra en la pantalla, pero ese nombre puede no ser el que más nos convenga, como ocurre cuando al volver atrás se muestra **fragment_inicio**
- Abre con el diseñador el archivo **nav_graph.xml** y selecciona **fragmentInicio**.
- En la zona de atributos, cambia **label** y pon “**HLanz App**”
- Repite lo mismo en **fragmentAyuda** para que su atributo **label** ponga **Ayuda**
- Repite lo mismo en **fragmentBandejaEntrada** para que su atributo **label** ponga **Bandeja de entrada**
- Ejecuta la app y comprueba que ahora todo funciona correctamente.

5 – BottomNavigationView

Una **bottom navigation bar** es una barra de navegación que aparece en la parte inferior de la pantalla y que puede mostrar hasta 5 items.

Aunque para el usuario su funcionamiento es similar a la barra de tareas, su programación es diferente.

- Haz clic con el botón derecho del ratón en **/menu** y elige **new** → **Menu resource file**
- Crea un archivo de tipo **Menu** llamado **menu_bottom_navigation_bar**

Al igual que sucedía en la barra de tareas, la **bottom navigation bar** también define sus opciones en un archivo de menú, que tiene la misma estructura.

- Descarga de Internet el archivo **ic_menu_home.xml** y cópialo en la carpeta **res/drawable**
- Abre el archivo **menu_bottom_navigation_bar.xml** en vista de código fuente y añade las tres opciones que se muestran a continuación:

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android">
2.     <item
3.         android:id="@+id/inicioFragment"
4.         android:icon="@drawable/ic_menu_home"
5.         android:title="Inicio"/>
6.     <item
7.         android:id="@+id/bandejaEntradaFragment"
8.         android:icon="@android:drawable/ic_dialog_email"
9.         android:title="Bandeja de entrada"/>
10.    <item
11.        android:id="@+id/ayudaFragment"
12.        android:icon="@android:drawable/ic_menu_help"
13.        android:title="Ayuda"/>
14. </menu>

```

Es muy importante que el id de los elementos del menú sea el id de los **Fragment** a los que va a navegar la app al pulsar en ellos.

- Abre el archivo **activity_main.xml** y añade al final (antes de cerrar el **LinearLayout**) un elemento **BottomNavigationView** con las siguientes características:
 - Id: **menuInferior**
 - Anchura: la de su contenedor
 - Altura: La de su contenido
 - **app:menu** → El archivo **menu_bottom_navigation_bar**

```

1. <com.google.android.material.bottomnavigation.BottomNavigationView
2.     android:layout_width="match_parent"
3.     android:layout_height="wrap_content"
4.     android:id="@+id/bottom_navigation_bar"
5.     app:menu="@menu/menu_bottom_navigation_bar"/>

```

- Modifica el **FragmentContainerView** para que su altura no sea la de su contenedor, sino que ocupe el espacio libre que quede una vez que todos los demás elementos hayan sido dibujados (*si no se hace así, la **BottomNavigationView** que va después, se saldrá fuera de la pantalla*)

```

1. <androidx.fragment.app.FragmentContainerView
2.     android:id="@+id/frnNavHostFragment"
3.     android:layout_width="match_parent"
4.     android:layout_height="0dp"
5.     android:layout_weight="1"
6.     android:name="androidx.navigation.fragment.NavHostFragment"
7.     app:defaultNavHost="true"
8.     app:navGraph="@navigation/nav_graph"/>

```

- Por último, vamos a poner a la **BottomNavigationView** el **NavigationController** que hay en el **NavHostFragment**, para que al pulsar las opciones, naveguemos automáticamente hacia el **Fragment** correspondiente. Esto se hace con el método **setupWithNavController** que ya vimos para la toolbar.

```


1. class MainActivity : AppCompatActivity() {
2.     // resto del código omitido
3.     override fun onCreate(savedInstanceState: Bundle?) {
4.         super.onCreate(savedInstanceState)
5.         setContentView(R.layout.activity_main)
6.         inicializarBinding()
7.         configurarToolbar()
8.         configurarBottomNavigationBar()
9.     }
10.    private fun configurarBottomNavigationBar(){
11.        val navController = getNavController()
12.        NavigationUI.setupWithNavController(binding.bottomNavigationBar,navController)
13.    }
14. }

```

Cuando asociamos un **NavController** con la **BottomNavigationView**, cada vez que pulsemos en una opción, navegaremos hacia el **Fragment** que en el **navigation graph** tiene el id de esa opción.

- Ejecuta la app y comprueba que aparece un menú de navegación en la parte inferior de la pantalla, y que al pulsar en cada opción se nos muestra el **Fragment** correspondiente.

6 – NavigationDrawer

El **navigation drawer** es un menú emergente que aparece cuando pulsamos las tres barras (también llamado icono hamburguesa)  y cuyas opciones nos llevan a los **Fragment** correspondientes, de la misma forma que hemos visto para los menús anteriores.

- Pulsa el botón derecho del ratón sobre **/res** y pulsa **new** → **Android resource file**
- Rellena la pantalla que aparece con estos datos:
 - o Archivo: **menu_navigation_drawer**
 - o Tipo de recurso: **Menu**

El menú que se muestra en el **navigation drawer** es también un archivo **xml**, de la misma forma que en los menús de los apartados anteriores

- Abre el archivo **menu_navigation_drawer.xml** en la vista de código fuente y añade las mismas opciones que ya añadimos para la **bottom navigation bar** (más adelante añadiremos más cosas a este archivo. Por ese motivo, estamos haciendo un archivo nuevo y no reutilizamos el que ya teníamos)

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android">
2.     <item
3.         android:id="@+id/inicioFragment"
4.         android:icon="@drawable/ic_menu_home"
5.         android:title="Inicio"/>
6.     <item
7.         android:id="@+id/bandejaEntradaFragment"
8.         android:icon="@android:drawable/ic_dialog_email"
9.         android:title="Bandeja de entrada"/>
10.    <item
11.        android:id="@+id/ayudaFragment"
12.        android:icon="@android:drawable/ic_menu_help"
13.        android:title="Ayuda"/>
14. </menu>

```

- Pulsa el botón derecho sobre **/res/layout** y elige **new → Android resource file**
- Crea un archivo llamado Archivo: **cabecera_navigation_drawer.xml**

La parte superior del **navigation drawer** es su cabecera, y en ella podemos mostrar lo que queramos. Debajo de la cabecera, aparecerán los elementos del menú.

La cabecera está diseñada en su propio archivo xml en la carpeta **layout**

- Abre el archivo **cabecera_navigation_drawer.xml** y coloca en ella un **FrameLayout** con altura **180dp**, que contenga un **ImageView** con estas características:
 - o Imagen: la del archivo **logo.webp**, que deberá encontrarse en la carpeta **/res/drawable**
 - o Ancho y alto: **180dp**

```

1. <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2.     android:layout_width="match_parent"
3.     android:layout_height="180dp">
4.     <ImageView
5.         android:layout_width="180dp"
6.         android:layout_height="180dp"
7.         android:src="@drawable/logotipo"/>
8. </FrameLayout>

```

Cuando una **Activity** tiene un **NavigationDrawer**, la estructura de su archivo xml de layout es la siguiente:

```

1. <androidx.drawerlayout.widget.DrawerLayout
2.     xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     android:id="@+id/navigation_drawer"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent">
7.     <!-- Vista principal de la interfaz -->
8.     ...
9.     <!-- Configuración del Navigation Drawer -->
10.    <com.google.android.material.navigation.NavigationView
11.        android:id="@+id/navigation_view"
12.        android:layout_width="wrap_content"
13.        android:layout_height="match_parent"
14.        android:layout_gravity="start"
15.        app:headerLayout="@layout/cabecera_navigation_drawer"
16.        app:menu="@menu/menu_navigation_drawer"/>
17. </androidx.drawerlayout.widget.DrawerLayout>

```

- Abre el archivo **activity_main.xml** y encierra todo su contenido entre un elemento **DrawerLayout**, que pasará a ser el elemento raíz de la interfaz. Dicho **DrawerLayout** tendrá como id **drawerLayout**

```

1. <androidx.drawerlayout.widget.DrawerLayout
2.     xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     android:id="@+id/navigation_drawer" android:layout_width="match_parent"
5.     android:layout_height="match_parent">
6.     <LinearLayout
7.         android:id="@+id/frnPrincipal"
8.         android:orientation="vertical" android:layout_width="match_parent"
9.         android:layout_height="match_parent">
10.        <com.google.android.material.appbar.MaterialToolbar
11.            android:id="@+id/material_toolbar"
12.            android:layout_width="match_parent"
13.            android:layout_height="?attr/actionBarSize"
14.            style="@style/Widget.MaterialComponents.Toolbar.Primary"/>
15.        <androidx.fragment.app.FragmentContainerView
16.            android:id="@+id/frnNavHostFragment"
17.            android:layout_width="match_parent"
18.            android:layout_height="0dp"
19.            android:layout_weight="1"
20.            android:name="androidx.navigation.fragment.NavHostFragment"
21.            app:defaultNavHost="true"
22.            app:navGraph="@navigation/nav_graph"/>
23.        <com.google.android.material.bottomnavigation.BottomNavigationView
24.            android:layout_width="match_parent"
25.            android:layout_height="wrap_content"
26.            android:id="@+id/menuInferior"
27.            app:menu="@menu/menu_bottom_navigation_bar"/>
28.    </LinearLayout>
29. </androidx.drawerlayout.widget.DrawerLayout>

```

- Tras el **LinearLayout**, añade un elemento **NavigationView** con estas características:
 - Id: **navigation_drawer**
 - Anchura: la de su contenido
 - Altura: la de su contenedor
 - **android:layout_gravity** → **start** (para que el **navigation drawer** se muestre a la izquierda de la pantalla)
 - **app:header_layout** → **cabecera_navigation_drawer.xml**
 - **app:menu** → **menu_navigation_drawer.xml**

```

1. <com.google.android.material.navigation.NavigationView
2.     android:id="@+id/navigation_view"
3.     android:layout_width="wrap_content"
4.     android:layout_height="match_parent"
5.     android:layout_gravity="start"
6.     app:headerLayout="@layout/cabecera_navigation_drawer"
7.     app:menu="@menu/menu_navigation_drawer"/>

```

El **NavigationView** configura el **navigation drawer** y es un elemento que se coloca justo al final del **DrawerLayout**. Sus atributos son:


- **app:headerLayout** → Es el archivo donde está el diseño de la cabecera
- **app:menu** → Es el archivo donde está definido el menú que se mostrará en el **navigation drawer**

- Para que el **navigation drawer** funcione, debemos irnos al método **configurarToolbar** y añadirlo cuando creamos el **AppBarConfiguration**

```

1. private fun configurarToolbar(){
2.     // inicio del método omitido
3.     val navController = getNavController()
4.     val configuracion = AppBarConfiguration
5.         .Builder(navController.graph)
6.         .setOpenableLayout(binding.navigationDrawer)
7.         .build()
8.     NavigationUI.setupWithNavController(binding.materialToolbar,navController,configuración)
9. }

```

- Ejecuta la app y comprueba que la barra de tareas tiene el icono , y que al pulsarlo, se abre el **navigation drawer**, pero sus opciones no funcionan.

El motivo de que no funcionen las opciones es que falta vincular el **NavigationView** que hay dentro del **navigation drawer** con el **NavController**

- Añade a **MainActivity** un método **configurarNavigationDrawer** en el que usarás el método **NavigationUI.setupWithNavController** para vincular el **NavigationView** con el **NavController**

```

1. class MainActivity : AppCompatActivity() {
2.     // resto del código omitido
3.     override fun onCreate(savedInstanceState: Bundle?) {
4.         super.onCreate(savedInstanceState)
5.         inicializarBinding()
6.         setContentView(binding.root)
7.         configurarToolbar()
8.         configurarBottomNavigationBar()
9.         configurarNavigationDrawer()
10.    }
11.    private fun configurarNavigationDrawer(){
12.        NavigationUI.setupWithNavController(binding.navigationView, getNavController())
13.    }
14. }

```

- Ejecuta la app y comprueba que todo funciona correctamente, pero de forma jerárquica, es decir, las opciones no son independientes unas de otras, sino que **InicioFragment** es el **Fragment** principal, y cuando navegamos a otra opción, nos aparece el botón de atrás para poder volver a **InicioFragment**
- Vamos a hacer que la navegación no sea jerárquica, y para eso modifica el método **configurarNavigationDrawer** para pasar al constructor del **AppBarConfiguration.Builder** la lista de **Fragments** principales (aquellos que no sean principales, serán tratados de forma jerárquica y al mostrarse se verá un botón de atrás que volverá a un **Fragment** principal).

```

1. private fun configurarToolbar(){
2.     // inicio del método omitido
3.     val navController = getNavController()
4.     val configuracion = AppBarConfiguration
5.         .Builder(setOf(R.id.inicioFragment,R.id.bandejaEntradaFragment))
6.         .setOpenableLayout(binding.navigationDrawer)
7.         .build()
8.     NavigationUI.setupWithNavController(binding.materialToolbar,navController,configuración)
9. }

```


- Ejecuta la app y comprueba que el **navigation drawer** se mantiene cuando visitamos **FragmentInicio** y **BandejaEntradaFragment**, pero aparece el botón de atrás cuando visitamos **AyudaFragment**, debido a que este último no es principal.

Por último, vamos a mejorar la apariencia del menú del **navigation drawer** haciendo una **sección** que contenga los botones de inicio y bandeja de entrada.

- Abre el archivo **menu_navigation_drawer.xml** y añade etiquetas **item**, **menú** y **group** para agrupar a los ítems de inicio y bandeja de entrada, así:

```

1. <menu xmlns:android="http://schemas.android.com/apk/res/android">
2.     <item android:title="Opciones">
3.         <menu android:checkableBehavior="single">
4.             <group android:checkableBehavior="single">
5.                 <item
6.                     android:id="@+id/inicioFragment"
7.                     android:icon="@drawable/ic_menu_home"
8.                     android:title="Inicio"/>
9.                 <item
10.                    android:id="@+id/bandejaEntradaFragment"
11.                    android:icon="@android:drawable/ic_dialog_email"
12.                    android:title="Bandeja de entrada"/>
13.            </group>
14.        </menu>
15.    </item>
16.    <item
17.        android:id="@+id/ayudaFragment"
18.        android:icon="@android:drawable/ic_menu_help"
19.        android:title="Ayuda"/>
20. </menu>

```

Los elementos introducidos funcionan de esta forma:

- **item (anterior a menu):** Sirve para definir una separación de opciones etiquetada con un texto
- **menu:** Encierra las opciones de una sección precedida con **<item>**
- **group:** Agrupa las opciones que se muestran con un color diferente cuando son pulsadas

El valor **single** del atributo **android:checkableBehavior** sirve para que cuando el usuario pulse en una opción, se muestre un color de fondo en la opción.

7 – ViewPager2

ViewPager2 es un componente de Android que nos permite mostrar distintos **Fragments** desplazándolos de izquierda a derecha.

Para usarlo, es necesario crear una clase hija de la clase abstracta **FragmentStateAdapter**, que se encargará de crear los **Fragment** que usará el **ViewPager2**

- Abre el archivo **fragment_bandeja_entrada.xml** con su vista xml, bórralo todo y añade un **LinearLayout** vertical con un **ViewPager2** con estas características:
 - o **Id: view_pager**
 - o **Ancho y alto: el de su contenedor**
 - o **Padding: 16dp**

```

1. <LinearLayout
2.     xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     android:orientation="vertical"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent">
7.     <androidx.viewpager2.widget.ViewPager2
8.         android:layout_width="match_parent"
9.         android:layout_height="match_parent"
10.        android:padding="16dp"
11.        android:id="@+id/view_pager"/>
12. </LinearLayout>

```

- Crea una clase llamada **BandejaEntradaViewPagerAdapter** que herede de la clase **FragmentStateAdapter** y que implemente sus dos métodos abstractos

```

1. class BandejaEntradaViewPagerAdapter(fragment: Fragment) : FragmentStateAdapter(fragment) {
2.     override fun getItemCount(): Int {
3.         // devuelve el número de Fragments que se mostrarán en el ViewPager2
4.     }
5.     override fun createFragment(position: Int): Fragment {
6.         // crea el Fragment de una posición
7.     }
8. }

```

Los métodos abstractos de **FragmentStateAdapter** son:

- o **getItemCount**: Devuelve el número de **Fragment** que hay en el **ViewPager2**
- o **createFragment**: Crea el **Fragment** de la posición recibida como parámetro

- Programa el método **getItemCount** para que devuelva el 3 (ya que solo habrá 3 **Fragment** en el **ViewPager2**)

```

1. class BandejaEntradaViewPagerAdapter(fragment: Fragment) : FragmentStateAdapter(fragment) {
2.     override fun getItemCount(): Int = 3
3.     // resto omitido
4. }

```

- Programa el método **createFragment** para que cree los fragments **MensajesRecibidosFragment**, **PapeleraFragment** y **SpamFragment** según se pase como parámetro 0,1 o 2

```

1. override fun createFragment(position: Int): Fragment = when (position) {
2.     0 -> MensajesRecibidosFragment()
3.     1 -> PapeleraFragment()
4.     2 -> SpamFragment()
5.     else -> throw Exception("posición incorrecta")
6. }

```

- Ejecuta la app y comprueba que una vez que accedes a la bandeja de entrada, puedes deslizar con el dedo para desplazar de izquierda a derecha los tres **Fragment** que hay en el **ViewPager2**.

8 – TabLayout

Un **TabLayout** es un componente que nos permite mostrar pestañas con etiquetas, de forma que cada pestaña muestra un contenido.

En este proyecto vamos a integrar un **TabLayout** con el **ViewPager2**, de forma que aparezcan etiquetas en los distintos **Fragments** del **ViewPager2**

- Abre **fragment_bandeja_entrada.xml** con la vista de código xml y realiza las siguientes modificaciones en él:
 - Pon un elemento **TabLayout** justo encima del **ViewPager2**, con estas características:
 - Id: **tab_layout**
 - Ancho: El de su contenedor
 - Alto: Su contenido
 - **tabMode: fixed**
 - Modifica el **ViewPager2** para que su altura sea **0dp** y se extienda todo el espacio libre de la pantalla

```

1. <LinearLayout
2.     xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:tools="http://schemas.android.com/tools"
4.     android:orientation="vertical"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent">
7.     <com.google.android.material.tabs.TabLayout
8.         android:id="@+id/tab_layout"
9.         android:tabMode="fixed"
10.        android:layout_width="match_parent"
11.        android:layout_height="wrap_content"/>
12.     <androidx.viewpager2.widget.ViewPager2
13.         android:layout_width="match_parent"
14.         android:layout_height="0dp"
15.         android:layout_weight="1"
16.         android:id="@+id/view_pager"/>
17. </LinearLayout>

```

El atributo **android:tabMode** nos indica si queremos que las pestañas del **TabLayout** entren en la pantalla (**fixed**), el usuario pueda desplazarse por ellas (**scrollable**) o que el sistema decida automáticamente (**auto**)

- Abre el código fuente de **BandejaEntradaFragment** y añade un método llamado **configurarTabLayout** que cree un **TabLayoutMediator** que integre el **TabLayout** con el **ViewPager2**, así:

```

1. private fun configurarTabLayout(){
2.     TabLayoutMediator(binding.tabLayout,binding.viewPager){ tab,posicion ->
3.         tab.text = when(posicion){
4.             0 -> "Recibidos"
5.             1 -> "Papelera"
6.             2 -> "Spam"
7.             else -> throw Exception("Posición no válida")
8.         }
9.     }.attach()
10. }

```

El constructor del **TabLayoutMediator** recibe tres parámetros:

- o El **TabLayout**
- o El **ViewPager2**
- o Una expresión lambda que devuelve el texto que le corresponde a la pestaña que se pasa como parámetro

Una vez construido el **TabLayoutMediator**, su método **attach** produce la integración entre el **TabLayout** y el **ViewPager2**

- Ejecuta la app y comprueba que al acceder a la bandeja de entrada ahora podemos desplazarnos por las pestañas de “recibidos”, “papelera” y “spam”

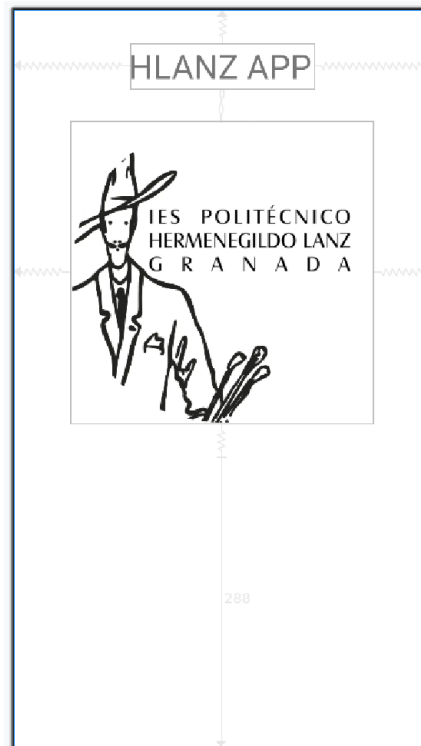
9 – Splash screen

Una **splash screen** es una pantalla de bienvenida que se inicia cuando se abre por primera vez la app y en ella se suele mostrar el logotipo de la app, la empresa desarrolladora, etc.

Hay ocasiones en las que durante la **splash screen** se produce la carga de recursos (imágenes de internet, conexión a bases de datos, etc) y se muestra una barra de progreso mientras se cargan.

En este proyecto crearemos una **splash screen** de adorno, usando el **navigation component** y usaremos **coroutines**

- Crea un **Fragment** llamado **SplashScreenFragment**
- Abre el archivo **fragment_splash_screen.xml** y realiza las siguientes acciones con el diseñador:
 - o Cambia el elemento principal para que sea un **ConstraintLayout**
 - o Arrastra un **TextView** y ponle de texto “**HLANZ APP**” con un tamaño de **36dp**
 - o Arrastra un **ImageView** y ponle la imagen **drawable/logotipo** con una anchura y altura de **300dp**
 - o Centra horizontalmente el **TextView** y el **ImageView**
 - o Haz una cadena vertical con el **TextView** y el **ImageView**



- Abre el archivo **activity_main.xml** y realiza los siguientes cambios en él:
 - Añade el atributo **android:visibility="gone"** al **FragmentManager**, la **MaterialToolbar** y el **BottomNavigationView**
 - Añade un segundo **FragmentManager** con estas características:
 - Id: **fragment_splash_screen**
 - name: La clase **FragmentSplashScreen**

```

1. <com.google.android.material.appbar.MaterialToolbar
2.     android:layout_width="match_parent"
3.     android:layout_height="?attr/actionBarSize"
4.     android:id="@+id/material_toolbar"
5.     android:visibility="gone"
6.     style="@style/Widget.MaterialComponents.Toolbar.Primary"/>
7. <androidx.fragment.app.FragmentManager
8.     android:layout_width="match_parent"
9.     android:layout_height="0dp"
10.    android:layout_weight="1"
11.    android:visibility="gone"
12.    android:id="@+id/fragment_container_view"
13.    app:defaultNavHost="true"
14.    app:navGraph="@navigation/nav_graph"
15.    android:name="androidx.navigation.fragment.NavHostFragment"/>
16. <androidx.fragment.app.FragmentManager
17.     android:layout_width="match_parent"
18.     android:layout_height="0dp"
19.     android:layout_weight="1"
20.     android:id="@+id/fragment_splash_screen"
21.     android:name="dam.moviles.repasohlanzapp.SplashScreenFragment"/>
22. <com.google.android.material.bottomnavigation.BottomNavigationView
23.     android:layout_width="match_parent"
24.     android:layout_height="wrap_content"
25.     android:visibility="gone"
26.     android:id="@+id/bottom_navigation_bar"
27.     app:menu="@menu/menu_bottom_navigation_bar"/>
28. </LinearLayout>

```

El atributo **android:visibility** nos indica si un elemento es visible o no, admitiendo:

- o **visible**: El elemento es perfectamente visible en la pantalla
- o **invisible**: El elemento es invisible, lo que significa que no se ve, pero ocupa el espacio que le corresponde en la pantalla
- o **gone**: El elemento existe, pero está fuera de la pantalla, por lo que no ocupa espacio en ella y el resto de la interfaz se adapta a su ausencia.

- Abre el **MainActivity** y en **onCreate** elimina las llamadas a **configurarToolbar**, **configurarBottomNavigationBar** y **configurarNavigationDrawer**
- Ejecuta la app y comprueba que aparece la **splash screen**, pero todo queda ahí

10 – Introducción a las coroutines

Los dispositivos móviles, al igual que los ordenadores, poseen **hilos (threads)** que ejecutan código fuente en segundo plano.

Una app posee un hilo (**Android UI Thread**) que encarga de dibujar la interfaz, esperar a que el usuario haga algo y ejecutar el código fuente de los eventos que se van produciendo, como por ejemplo, pulsar un botón.

Cuando una tarea consume tiempo y el hilo de Android se pone con ella, el resultado es que no se puede dibujar la interfaz durante ese tiempo y la interfaz no responde, dando mala imagen al usuario. Para evitar este problema, aparecen las **coroutines**, que es un enfoque para hacer multitarea sin usar los hilos directamente.

Una **coroutine** es un trozo de código fuente (método, expresión lambda, etc) que en su interior posee **puntos de suspensión** (que se identifican por el símbolo **→**). Un punto de suspensión puede ser, por ejemplo, una llamada a un método que ejecuta una tarea que tarda mucho tiempo en completarse. Normalmente, los puntos de suspensión inician algo que se ejecuta en segundo plano por un hilo diferente al que está ejecutando la coroutine.

Cuando el hilo que ejecuta la coroutine llega a un punto de suspensión, interrumpe la ejecución de la coroutine y se pone a hacer otra cosa (por ejemplo, ejecutar otra coroutine). Cuando pasa un tiempo y la tarea que dio lugar al punto de suspensión finaliza, un hilo (que puede ser el mismo que inició la coroutine, u otro hilo diferente), continúa ejecutando la coroutine.

Para lanzar una coroutine se necesita un objeto capaz de ponerlas en marcha. Dicho objeto se denomina **CoroutineScope**. En Android podemos obtenerlo así:

- En **MainActivity** → **lifecycleScope**
- En un **Fragment** → **lifecycleScope** o **viewLifecycleOwner.lifecycleScope**
- En un **ViewModel** → **viewModelScope**

- Abre el código fuente de **SplashScreenFragment** y añade un método llamado **esperarInicioApp**, que será llamado en **onCreateView**

```

1. class SplashScreenFragment : Fragment() {
2.     override fun onCreateView(
3.         inflater: LayoutInflater, container: ViewGroup?,
4.         savedInstanceState: Bundle?
5.     ): View? {
6.         esperarInicioApp()
7.         return inflater.inflate(R.layout.fragment_splash_screen, container, false)
8.     }
9.     private fun esperarInicioApp(){
10.        // aquí vamos a lanzar una coroutine que espere 1 segundo, y a su
11.        // finalización, muestre la toolbar, el bottom navigation bar
12.        // y navegue a InicioFragment
13.    }
14. }

```

- Dentro del método **esperar5segundos** usa el método **launch** de **lifecycleScope** para poner en marcha una coroutine

```

1. private fun esperar5segundos(){
2.     lifecycleScope.launch {
3.         // esto ya es una coroutine y podemos llamar aquí
4.         // a métodos que sean puntos de suspensión
5.     }
6. }

```


El método **launch** del **CoroutineScope** pone en marcha una coroutine definida mediante una expresión lambda.

- Dentro del método **launch**, llama al método **delay**, que es un punto de suspensión que hace que el hilo que ejecuta la coroutine (el hilo de Android) se vaya a hacer otra cosa (como dibujar la pantalla y seguir procesando los eventos del usuario) hasta que pasen 5 segundos. Al pasar los 5 segundos, el hilo de Android volverá a retomar la ejecución de la coroutine.

```

22         private fun esperarInicioApp(){
23             lifecycleScope.launch {
24                 delay( timeMillis: 1000)
25                 iniciarInterfazPrincipal()
26             }
27         }

```

Observa que **delay** es un punto de suspensión, porque lleva el símbolo . Esto hace que el hilo que ejecuta la coroutine se quede libre para hacer otras cosas, y que tras la pausa de 5 segundos, vuelva y continúe la ejecución de la coroutine por donde se quedó.

- Termina de programar la coroutine llamando a un método **iniciarInterfazPrincipal** que programaremos en **MainActivity** (ahora mismo dará error porque aún no está programado)

```

22     private fun esperarInicioApp(){
23         lifecycleScope.launch {
24             ↪      delay( timeMillis: 1000)
25             val mainActivity = activity as MainActivity
26             mainActivity.iniciarInterfazPrincipal()
27         }
28     }

```

- Abre **MainActivity** y programa el método **iniciarInterfazPrincipal** de forma que al llamarlo:
 - Se vuelva invisible el **FragmentSplashScreen**
 - Se vuelvan visibles el **FragmentContainerView**, la **MaterialToolbar** y el **BottomNavigationBar**
 - Se llamen a los métodos **configurarToolbar**, **configurarBottomNavigationBar** y **configurarBottomNavigationDrawer**

```

1. class MainActivity : AppCompatActivity() {
2.     // resto del código omitido
3.     override fun onCreate(savedInstanceState: Bundle?) {
4.         super.onCreate(savedInstanceState)
5.         inicializarBinding()
6.         setContentView(binding.root)
7.     }
8.     fun iniciarInterfazPrincipal(){
9.         configurarToolbar()
10.        configurarBottomNavigationBar()
11.        configurarNavigationDrawer()
12.        binding.fragmentSplashScreen.visibility=View.GONE
13.        binding.fragmentContainerView.visibility=View.VISIBLE
14.        binding.materialToolbar.visibility = View.VISIBLE
15.        binding.bottomNavigationBar.visibility = View.VISIBLE
16.    }
17. }

```

- Ejecuta la app y comprueba que todo funciona correctamente, pero la desaparición de la **splash screen** es un poco brusca
- Modifica el código fuente de la coroutine para que tras pasar 500 milisegundos, llame a un método llamado **iniciarFadeOut**

```

1. private fun esperarInicioApp(){
2.     lifecycleScope.launch {
3.         delay(500)
4.         iniciarFadeOut()
5.         delay(500)
6.         val mainActivity = activity as MainActivity
7.         mainActivity.iniciarInterfazPrincipal()
8.     }
9. }

```

- Programa el método **iniciarFadeOut** para que en él se inicie un efecto de fade-out (desaparecer poco a poco cambiando la transparencia de 1 hasta 0) que desvanezca **FragmentSplashScreen** en 500 milisegundos.


```

1. fun iniciarFadeOut(){
2.     val mainActivity = activity as MainActivity
3.     mainActivity.binding.fragmentSplashScreen.animate()
4.         .alpha(0f)
5.         .setDuration(500)
6. }

```

- Ejecuta la app y comprueba que al pasar 500 milisegundos, se desvanece la **splash screen** y tras eso, se inicia la app con normalidad, pero que si giras el móvil vuelve a aparecer la **splash screen**.

Nuestra app no resiste a los cambios estructurales y vamos a arreglarlo con un **view model**

- Crea una clase llamada **MainActivityViewModel** que herede de **ViewModel** y añádele una variable de instancia **iniciado** que valdrá **false** si la app aún no ha superado la **splash screen**

```

1. class MainActivityViewModel : ViewModel(){
2.     var iniciado:Boolean = false
3. }

```

- Añade a **MainActivity** un **MainActivityViewModel** e inicialízalo en un método llamado **inicializarViewModel** que se llama en **onCreate**

```

1. class MainActivity : AppCompatActivity() {
2.     // resto de la clase omitido
3.     lateinit var viewModel:MainActivityViewModel
4.     override fun onCreate(savedInstanceState: Bundle?) {
5.         super.onCreate(savedInstanceState)
6.         inicializarBinding()
7.         inicializarViewModel()
8.     }
9.     fun inicializarViewModel(){
10.         viewModel=ViewModelProvider(this).get(MainActivityViewModel::class.java)
11.     }
12. }

```

- Modifica el método **iniciarInterfazPrincipal** para que la variable **iniciado** del **viewModel** valga **true**

```

1. fun iniciarInterfazPrincipal(){
2.     configurarToolBar()
3.     configurarBottomNavigationBar()
4.     configurarNavigationDrawer()
5.     binding.fragmentSplashScreen.visibility=View.GONE
6.     binding.fragmentContainerView.visibility=View.VISIBLE
7.     binding.materialToolBar.visibility = View.VISIBLE
8.     binding.bottomNavigationBar.visibility = View.VISIBLE
9.     viewModel.iniciado=true
10. }

```

- Programa el método **onStart** en **MainActivity** de forma que si la variable **iniciado** del **viewModel** es **true**, se llame al método **iniciarInterfazPrincipal**

```
1. override fun onStart() {  
2.     super.onStart()  
3.     if(viewModel.iniciado) {  
4.         iniciarInterfazPrincipal()  
5.     }  
6. }
```

- Ejecuta la app y comprueba que al girar el móvil ya no vuelve a salir la **splash screen**