Capitulo 4

UD04. NAVEGACIÓN UI

Resultados de avaliación

RA1. Aplica tecnoloxías de desenvolvemento para dispositivos móbiles, e avalía as súas características e as súas capacidades.

RA2. Desenvolve aplicacións para dispositivos móbiles, para o que analiza e emprega as tecnoloxías e as librarías específicas.

Criterios de avaliación

- CA1.7 Analizouse a estrutura de aplicacións existentes para dispositivos móbiles, e identificáronse as clases utilizadas.
- CA1.8 Realizáronse modificacións sobre aplicacións existentes.
- CA1.9 Utilizáronse emuladores para comprobar o funcionamento das aplicacións.
- CA2.1 Xerouse a estrutura de clases necesaria para a aplicación.
- CA2.4 Utilizáronse as clases necesarias para a conexión e a comunicación con dispositivos sen fíos.
- CA2.5 Utilizáronse as clases necesarias para o intercambio de mensaxes de texto e multimedia.
- CA2.8 Realizáronse probas de interacción entre o usuario e a aplicación para mellorar as aplicacións desenvolvidas a partir de emuladores.
- CA2.9 Empaquetáronse e despregáronse as aplicacións desenvolvidas en dispositivos móbiles reais.
- CA2.10 Documentáronse os procesos necesarios para o desenvolvemento das aplicacións

BC1. Análise de tecnoloxías para desenvolvemento de aplicacións en dispositivos móbiles.

- Estrutura dunha aplicación para dispositivo móbil.
- Modificación de aplicacións existentes.
- Uso do contorno de execución do administrador de aplicacións.
- Contornos integrados de traballo.
- Módulos para o desenvolvemento de aplicacións móbiles.
- Emuladores.
- Ferramentas e fases de construción.
- Eventos da interface.
- Probas de interacción.
- Empaquetaxe e distribución.
- Documentación do desenvolvemento das aplicacións.
- Comunicacións: clases asociadas. Tipos de conexións.

Última actualización: 08.02.2024

SUBSECCIONES DE UD04. NAVEGACIÓN UI

CREAR PROYECTO

Para ver cómo funcionan los ciclos de vida en Android vamos a generar un proyecto sobre el que veremos los distintos estados de forma práctica. Así, lo primero que haremos será generar nuestro protecyo en Android Studio siguiendo los siguientes pasos:

1. **Abrir Android Studio**: Abre Android Studio en tu ordenador.
2. **Crear un Nuevo Proyecto**:
    - Selecciona “Start a new Android Studio project” en la pantalla de inicio.
    - Elige “Phone and Tablet” como tipo de dispositivo.
    - Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.
3. **Configuración del Proyecto**:
    - En la siguiente pantalla, completa la información básica sobre el proyecto:
        - **Name**: Ingresa “UF1_UD04_1_CatChat”.
        - **Package name**: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.
        - **Save location**: Elige la ubicación donde deseas guardar el proyecto en tu sistema.
        - **Language**: Selecciona “Kotlin” como lenguaje de programación.
        - **Minimum API level**: Selecciona “API 35: Android 7.0 (Nougat)” como SDK mínimo.
4. **Finalizar Configuración**:
    - Revisa la configuración y ajusta cualquier otra opción según tus preferencias.
    - Haz clic en “Finish” para crear el proyecto.

Android Studio generará automáticamente la estructura básica del proyecto, incluyendo los archivos necesarios para la actividad principal que has creado. Puedes comenzar a desarrollar tu aplicación agregando código a la actividad `MainActivity.kt` y diseñando la interfaz de usuario en el archivo de diseño correspondiente.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

Capitulo 1

NAVEGACIÓN UI

Continuaremos profundizando en el componente de **navegación** y en el **desplazamiento** de los distintos fragmentos que conforman nuestra aplicación. En este punto, nos enfocaremos en ciertos elementos que nos permitirán crear interfaces de usuario más enriquecidas y proporcionar mecanismos más flexibles de desplazamiento.

Hasta ahora, los proyectos que hemos desarrollado tenían una **navegación lineal** en la que el desplazamiento entre ellos siempre seguía el mismo orden. Ahora exploraremos una serie de

componentes que nos permitirán desplazarnos **libremente** entre estos distintos elementos que forman parte de nuestra aplicación. Específicamente, veremos cómo aplicar **temas** a nuestra aplicación, y en algunos de estos temas, cómo se definen el uso de determinadas barras de aplicación, como la que aparece en la parte superior de todos los proyectos que hemos estado creando hasta ahora, donde se muestra el título de la aplicación, por ejemplo. También aprenderemos a reemplazar estas barras por una barra de botones personalizada, que contendrá botones específicos para saltar a fragmentos particulares o para retroceder, entre otras funcionalidades.

Además, abordaremos la creación de **menús** que incluirán una serie de elementos que nos permitirán navegar hacia los diversos destinos que tengamos definidos en nuestra aplicación. Por último, exploraremos la implementación de una barra de navegación inferior, un elemento común en las aplicaciones de Android.

Autor/a: Sabela Sobrino Última actualización: 20.01.2025

SUBSECCIONES DE NAVEGACIÓN UI NUEVO DISEÑO

Vamos a crear una nueva aplicación con la siguiente apariencia:

Al iniciar un nuevo proyecto, automáticamente se genera una **barra de navegación** en la parte **superior** que muestra el **título** de la aplicación. Si ejecutamos la aplicación tal como está, notaremos que esta barra superior ya incluye el título de la aplicación y tiene un color específico.

Este comportamiento se debe al hecho de que, al crear nuestro proyecto, se aplica un tema específico que define el diseño de la aplicación. En este caso, hemos utilizado un tema

basado en **Material Design de Google**, que asegura un diseño coherente. El objetivo es que todas las aplicaciones que desarrollemos para el sistema Android tengan una **apariencia similar**, lo que facilita la experiencia del usuario al encontrar interfaces familiares. Puedes obtener más información sobre el diseño Material Design aquí.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

TEMAS Y ESTILOS

En nuestro archivo Gradle de la aplicación, podemos observar que se ha incluido una dependencia:

```
implementation("com.google.android.material:material:1.10.0")
```

Esta dependencia indica que estamos incorporando este componente en nuestra aplicación de manera predeterminada.

Además, en el archivo de **manifiesto** de la aplicación, hemos especificado el tema o estilo que nuestra aplicación utilizará, definiendo aspectos como el **color** primario y secundario, entre otros:

```
android:theme="@style/Theme.UF1_UD04_1_CatChat"
```

El tema “UF1_UD04_1_CatChat” está definido en un archivo de recursos ubicado en la carpeta `res -> values -> themes`. En esta ubicación, encontramos **dos temas** que se aplicarán en función de la configuración de **día** o **noche**. Este tema hereda de un tema predefinido en Material Design:

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.UF1_UD04_1_CatChat" parent="Theme.Material3.DayNight.NoActionBar">
        <!-- Personaliza tu tema claro aquí. -->
        <!-- <item name="colorPrimary">@color/my_light_primary</item> -->
    </style>

    <style name="Theme.UF1_UD04_1_CatChat" parent="Base.Theme.UF1_UD04_1_CatChat" />
</resources>
```

En este punto, es importante destacar que no se ha definido una barra de acción (`NoActionBar`). Sin embargo, si modificamos esta línea para que herede de la clase “DarkActionBar”:

```
<style name="Base.Theme.UF1_UD04_1_CatChat"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
```

y ejecutamos nuestra aplicación, notaremos que ahora se incluye una barra de navegación en la parte superior de la pantalla.

Es posible realizar ajustes en ciertos parámetros, como colores y estilos, que se pueden encontrar en la documentación oficial de Android. No obstante, generalmente se recomienda no modificar estos parámetros en exceso para mantener la coherencia en el estilo de las aplicaciones.

Por ejemplo:

```
<item name="colorPrimary">@color/my_light_primary</item>
```

En un tema que admite modos día y noche, es importante aplicar las modificaciones en ambos archivos de tema correspondientes a cada modo.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

TOOLBAR PERSONALIZADA

Ahora procederemos a reemplazar la barra por defecto, heredando del tema base "NoActionBar" (que es el tema predeterminado).

Para garantizar que ambos archivos de temas heredan de "NoActionBar", asegurémonos de que ambos tienen la siguiente definición:

```
<style name="Base.Theme.UF1_UD04_1_CatChat" parent="Theme.Material3.DayNight.NoActionBar">
```

Por el momento, no realizaremos modificaciones en los colores que vienen por defecto. No obstante, si en el futuro deseáramos modificar algún color específico, podemos definirlos en el archivo `colors.xml``.

Componente Toolbar

Vamos a agregar un nuevo componente a nuestro archivo principal (activity_main.xml), específicamente, vamos a incorporar un componente "Toolbar". Para lograr esto, podemos utilizar un simple "LinearLayout". En primer lugar, eliminamos el contenido existente en activity_main.xml y creamos un "LinearLayout" vacío:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
>
</LinearLayout>
```

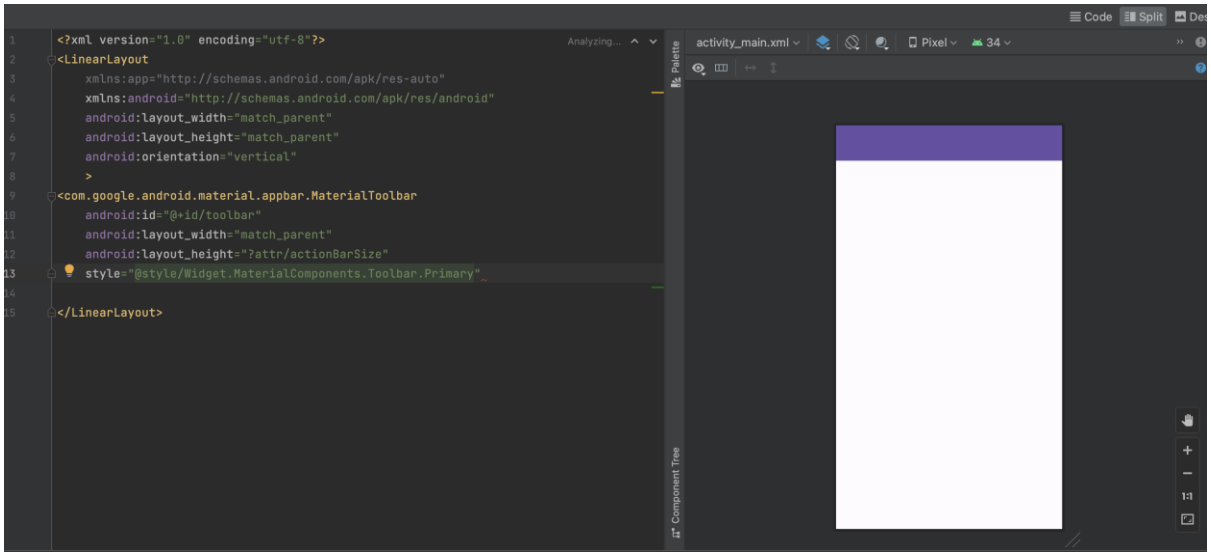
Dentro de este "LinearLayout", vamos a introducir un nuevo componente "**MaterialToolbar**" y ajustaremos su ancho al tamaño de la pantalla, y su altura la configuraremos para que coincida con el valor predeterminado de la barra del tema actual. Para lograr esto, utilizaremos el atributo de altura con el valor "?attr/actionBarSize":

```
<com.google.android.material.appbar.MaterialToolbar
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:id="@+id/toolbar"
    style="@style/Widget.MaterialComponents.Toolbar.Primary"
/>
```

Además, es necesario asignarle un identificador (en este caso, “toolbar”) para que podamos hacer referencia a él desde nuestro código.

Con el atributo “style,” aplicamos el estilo definido en el tema por defecto, accediendo a él mediante la referencia `@style/Widget.MaterialComponents.Toolbar.Primary`.

Al visualizar la pestaña de diseño, ya podremos ver la barra de navegación con el color de la barra primaria:



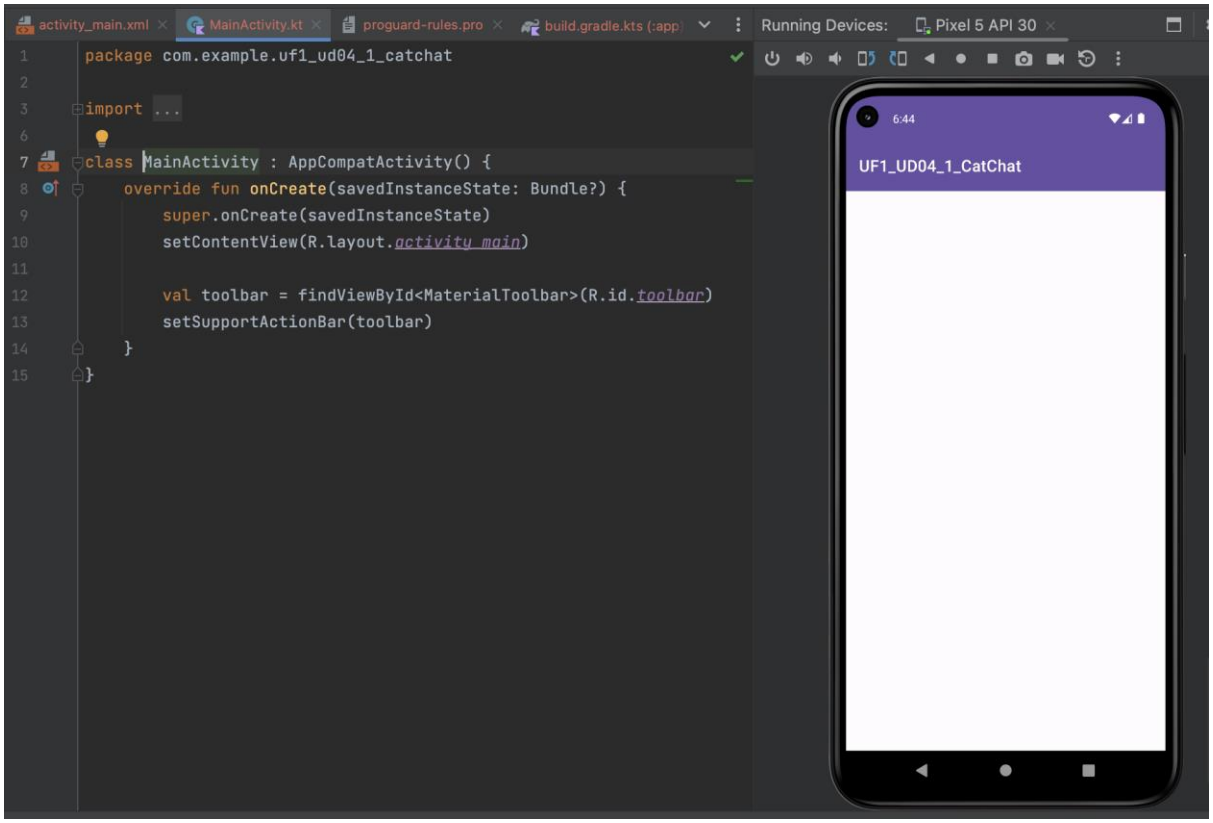
Es importante mencionar que, en este momento, la barra todavía carece de contenido, ya que acabamos de crearla dentro del diseño.

Asignar Activity

En este punto, necesitamos asignar la **barra de navegación** que hemos creado como la barra de **aplicación principal**. Para lograr esto, en nuestra actividad principal (ActivityMain.kt), debemos recuperar la instancia de la barra que acabamos de crear y especificar que la utilice como la barra de aplicación principal:

```
val toolbar = findViewById<MaterialToolbar>(R.id.toolbar)
setSupportActionBar(toolbar)
```

Una vez completado este paso, si ejecutamos la aplicación, veremos que la barra de navegación se ha personalizado y que ahora podemos agregar nuevos elementos a ella:



Esta personalización nos permite integrar nuevos elementos en la barra y adaptarla a las necesidades de nuestra aplicación.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

FRAGMENTOS

Antes de poder añadir elementos a la barra de navegación, vamos a crear los fragmentos que vamos a utilizar. Ambos fragmentos son muy simples y consisten en una etiqueta que indica en qué pantalla nos encontramos.

El resultado final debe coincidir con la siguiente apariencia:

Fragmentos Inbox

Primero, creamos un fragmento llamado “InboxFragment.” En este fragmento, agregamos un simple diseño utilizando un `LinearLayout`:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Inbox"
        android:textSize="20dp"
        />

</LinearLayout>
```

Fragmentos Help

De manera similar, creamos otro fragmento llamado “HelpFragment.” Al igual que en el caso anterior, utilizamos un `LinearLayout` para un diseño simple:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
```

```
        android:layout_height="match_parent"
        android:text="Help"
        android:textSize="20dp"
        />

</LinearLayout>
```

Fragmento Sent Items

De manera similar, creamos otro fragmento llamado “MessagesFragment.” Al igual que en el caso anterior, utilizamos un `LinearLayout` para un diseño simple:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Sent Items"
        android:textSize="20dp"
        />

</LinearLayout>
```

Una vez completados los fragmentos, editamos el archivo Kotlin asociado al fragmento y mantenemos solo el método `onCreateView`:

```
class MessageFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflar el diseño (layout) para este fragmento
        return inflater.inflate(R.layout.fragment_message, container, false)
    }
}
```

Estos fragmentos simples servirán como las pantallas de “Inbox”, “Help” y “Message” en nuestra aplicación.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

GRAFO DE NAVEGACIÓN

Además de los fragmentos, necesitamos crear un nuevo grafo de navegación que incluya las pantallas que estamos desarrollando, siguiendo el mismo enfoque que vimos en la unidad anterior.

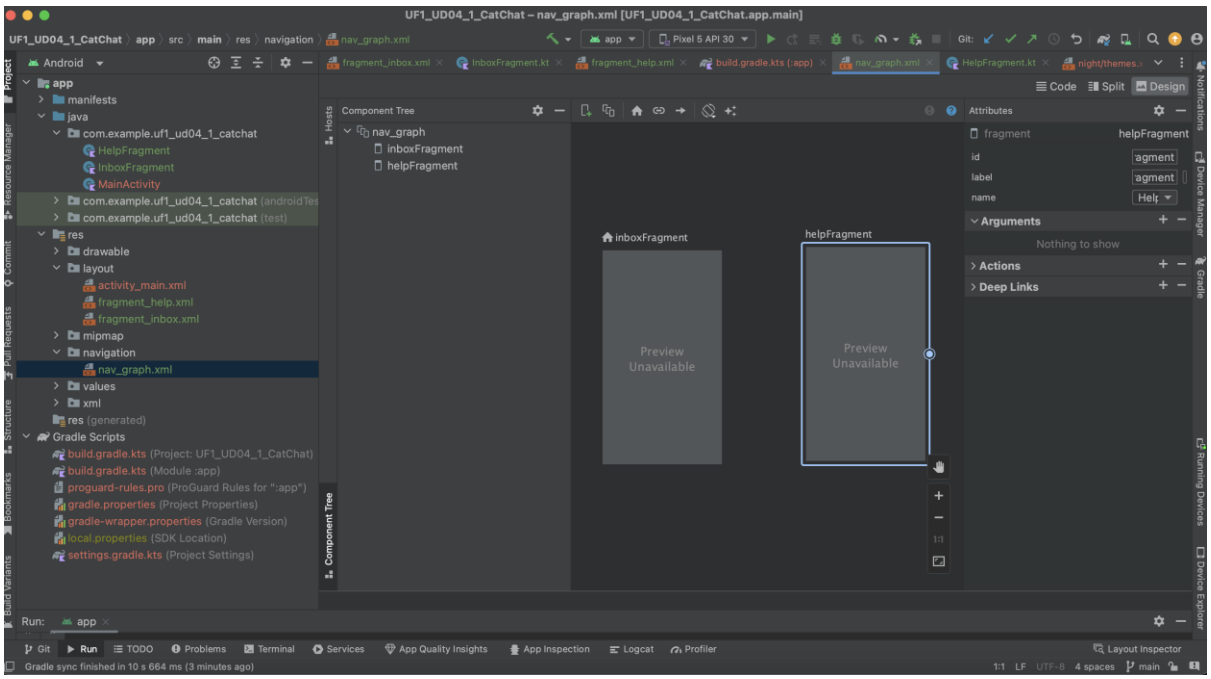
Para comenzar, debemos agregar la dependencia de navegación en el archivo Gradle:

```
implementation("androidx.navigation:navigation-fragment-ktx:2.7.4")
```

Después de agregar esta línea, es importante hacer clic en el botón “Sync Now” para actualizar las dependencias de Gradle.

Una vez que tengamos la dependencia añadida, podemos proceder a crear el grafo de navegación. Como hemos visto, el grafo de navegación es un recurso XML que se crea de manera sencilla. Para ello, creamos un nuevo **“Archivo de Recurso de Android”** desde la carpeta de Recursos, seleccionamos el tipo “Navigation,” y le damos el nombre “nav_graph.xml.”

Dentro de este archivo, agregamos los dos fragmentos que hemos creado, pero por ahora no necesitamos crear acciones para la navegación entre ellos. El grafo debe verse de la siguiente manera:



En el código, notamos que a cada fragmento se le ha asignado un identificador:

```
<fragment
   android:id="@+id/inboxFragment"
   android:name="com.example.uf1_ud04_1_catchat.InboxFragment"
   android:label="Inbox" />
<fragment
   android:id="@+id/helpFragment"
   android:name="com.example.uf1_ud04_1_catchat.HelpFragment"
   android:label="Help" />
```

Después de definir los destinos en el grafo, es importante que modifiquemos el diseño de la actividad principal (`main_activity.xml`) para agregar un contenedor de fragmentos, ajustando su alto y ancho al tamaño de la pantalla y asignándole un identificador:

```
<androidx.fragment.app.FragmentContainerView
   android:id="@+id/nav_host_fragment"
   android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

En este caso, no tendremos un único fragmento en nuestro contenedor de pantalla, sino **varios fragmentos** definidos en el grafo de navegación. Para lograr esto, debemos definir un **"host de navegación"** en el contenedor de fragmentos, especificando el grafo que acabamos de crear:

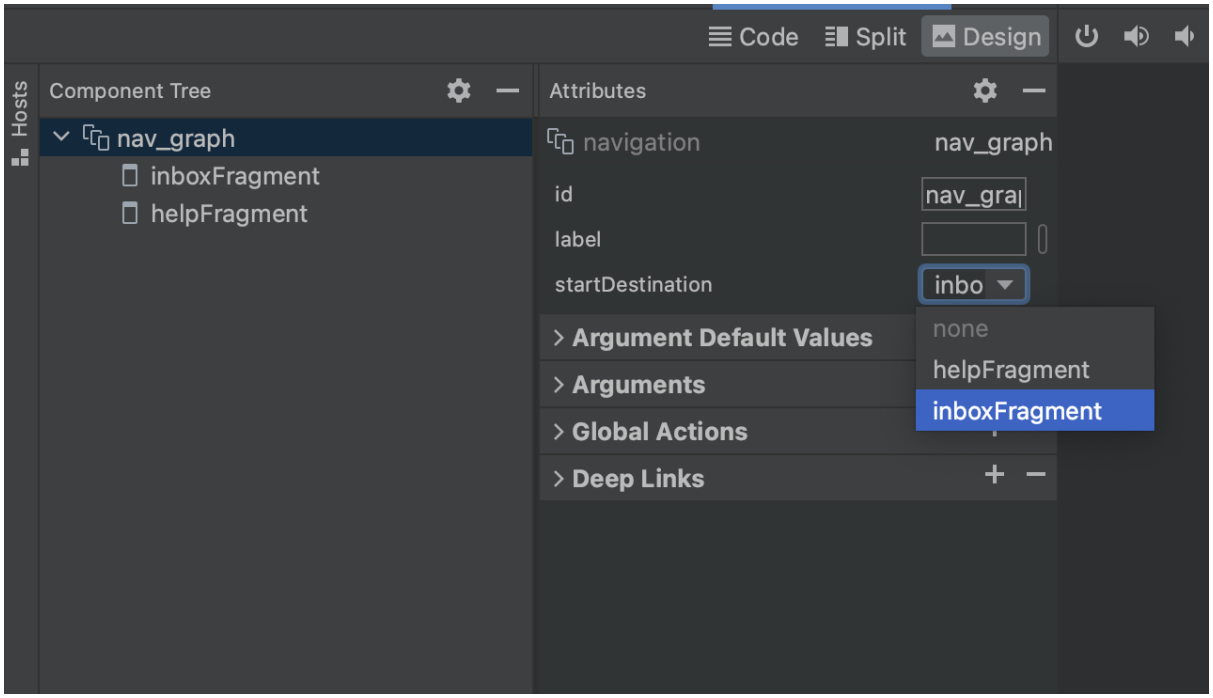
```
android:name="androidx.navigation.fragment.NavHostFragment"
app:navGraph="@navigation/nav_graph"
```

De esta forma, indicamos que este contenedor contendrá fragmentos que están definidos en el grafo de navegación. Además, como hicimos en el proyecto anterior, debemos establecer la propiedad para que el botón "atrás" funcione correctamente:

```
app:defaultNavHost="true"
```

Finalmente, podemos ejecutar la aplicación para verificar que se carga correctamente:

Comprobamos que la pantalla de "Inbox" se carga, ya que así lo hemos definido en el grafo de navegación:



Este conjunto de pasos nos permite configurar la navegación en nuestra aplicación.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

Capitulo 2

TOOLBAR

Para agregar elementos en la barra de navegación que permitan navegar entre los distintos fragmentos de la aplicación, necesitamos crear un recurso de tipo **menú**, que es otro tipo de recurso que se puede definir en Android.

Dentro de este menú, agregaremos diversos **elementos**, en este caso, botones que nos permitirán cargar los diferentes fragmentos. Es importante que el ID de cada **ítem** coincida con el ID del **fragmento de destino**, el cual se encuentra definido en el **grafo de navegación**.

Una vez que tengamos este menú creado, para añadir los elementos del menú a la barra de navegación, debemos sobrescribir el método `onCreateOptionsMenu` en la actividad principal.

Luego, para responder a los eventos de clic en los elementos del menú, sobrescribiremos el método `onOptionsItemSelected` en la actividad principal. Este método nos permite manejar las acciones que ocurren cuando se hace clic en los elementos del menú.

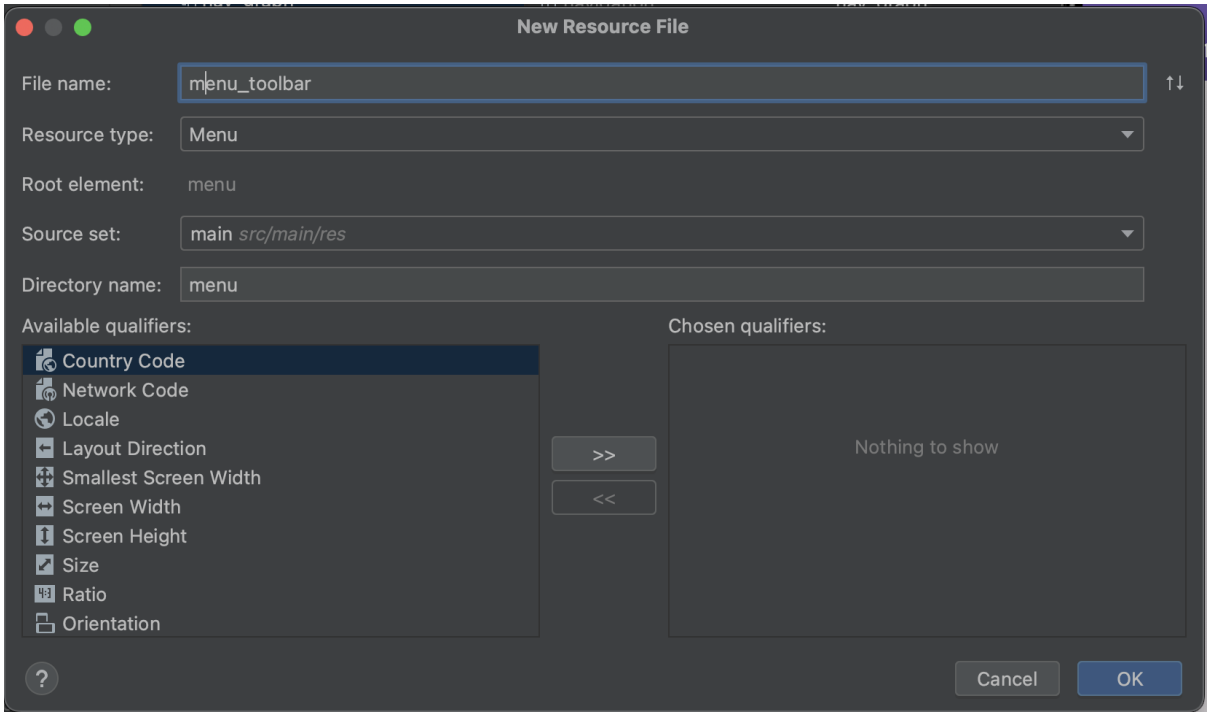
Por último, para mostrar el label del fragmento (definido en el grafo de navegación) y un botón que nos permita retroceder, podemos utilizar la clase `AppBarConfiguration` proporcionada por la biblioteca de navegación, que es una forma sencilla de lograrlo.

Autor/a: Sabela Sobrino Última actualización: 20.01.2025

SUBSECCIONES DE TOOLBAR

CREACIÓN DEL MENÚ

Lo primero que haremos es crear el menú de opciones que vamos a asignar al toolbar. Para hacer esto, creamos un nuevo recurso llamado “menu_toolbar” de tipo **“Android Resource File”** de tipo **Menú**, tal como se muestra en la imagen:



Dentro de este menú, podemos definir diferentes opciones, denominadaos “**items**”. Comenzaremos por agregar la primera opción, “Help”. Para ello, agregamos un nuevo elemento con su **identificador**. Es importante que este ID sea el mismo que el del fragmento al que queremos navegar, ya que al hacer clic en este ícono, queremos saltar a la pantalla “HelpFragment,” por lo que seleccionamos el mismo ID que está definido en el grafo de navegación:

```
<?xml version="1.0" encoding="utf-8">
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/helpFragment"
```

Otra configuración que deseamos aplicar es el **icono**. Existen varios íconos predefinidos para diferentes usos dentro de la propiedad “drawable,” y seleccionamos el ícono de ayuda:

```
    android:icon="@android:drawable/ic_menu_help"
```

Además, proporcionamos un texto descriptivo, un **título**:

```
    android:title="Help"
```

Por defecto, el menú se configura como un menú **desplegable**, lo cual no es lo que deseamos. Podemos modificar este comportamiento utilizando la propiedad “**showAsAction**” del espacio de nombres de la app. Primero, definimos el espacio de nombres de la app:

```
<?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">
```

Luego, dentro del elemento del ítem, configuramos “showAsAction” como “always” para mostrarlo como un botón en lugar de una lista desplegable de opciones de menú:

```
    <item
        android:id="@+id/helpFragment"
        android:icon="@android:drawable/ic_menu_help"
        android:title="Help"
        app:showAsAction="always" />
```

Aunque en el diseño del menú aparezca configurado correctamente, al ejecutar la aplicación, no vemos el botón generado en ningún lugar, como se muestra en la imagen:

Esto sucede porque la actividad principal (MainActivity) no está creando estos elementos del menú. Para resolver esto, debemos ir a la actividad principal y utilizar los métodos que mencionamos al principio.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

INFLADO DE ITEMS

Dentro de la actividad principal (MainActivity.kt), realizaremos el inflado de los elementos del menú utilizando la función `onCreateOptionsMenu`:

```
class MainActivity : AppCompatActivity() {

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

        val toolbar = findViewById<MaterialToolbar>(R.id.toolbar)
        setSupportActionBar(toolbar)
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        menuInflater.inflate(R.menu.menu_toolbar, menu)
        return super.onCreateOptionsMenu(menu)
    }
}
```

De manera similar a cómo inflamos el diseño XML en la actividad principal, utilizamos la clase `menuInflater` con el método `inflate` para inflar el diseño del menú. Le pasamos el diseño de nuestro menú y el propio menú que recibe como parámetro para que lo “pinte” en la barra de navegación.

Al ejecutar la aplicación, veremos que ahora muestra el botón que diseñamos:

Aunque por ahora, el botón no realiza ninguna acción.

Prueba: Puedes eliminar la propiedad `app:showAsAction="always"`, y comprobarás cómo se muestra de manera diferente, posiblemente como un ítem desplegable en lugar de un botón.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

ACCIONES

Para responder a las acciones de clic sobre los botones del menú, debemos sobrescribir el método `onOptionsItemSelected`, donde obtendremos información sobre el elemento del menú que se presionó.

Primero, necesitamos obtener una referencia al controlador de navegación. Para ello, utilizamos la siguiente función:

```
val navController = findNavController(R.id.nav_host_fragment)
```

Para vincular el botón sobre el cual se hizo clic con el evento del controlador de navegación, debemos hacer uso de una nueva función, pero primero debemos agregar una nueva dependencia:

```
implementation("androidx.navigation:navigation-ui-ktx:2.7.4")
```

Después de agregar esta línea, hacemos clic en el botón “Sync Now” para actualizar las dependencias.

Ahora, en lugar de simplemente devolver el `itemSelected`, realizamos una acción utilizando el componente `NavigationUI`:

```
return NavigationUI.onNavDestinationSelected(item, navController) || super.onOptionsItemSelected(item)
```

Este método, `onNavDestinationSelected`, toma dos parámetros: el elemento del menú que se accionó y el controlador de navegación. También devuelve un valor booleano, que utilizamos en una operación lógica “or” y devolvemos al método padre.

La llamada a `onNavDestinationSelected` establece una relación entre el botón seleccionado y el host de navegación correspondiente.

El método `onOptionsItemSelected` debería verse así:

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    val navController = findNavController(R.id.nav_host_fragment)
    return NavigationUI.onNavDestinationSelected(item, navController) || super.onOptionsItemSelected(item)
}
```

Ahora, si ejecutamos la aplicación, comprobaremos que al hacer clic en el botón de ayuda, nos llevará a la pantalla correcta.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

Capitulo 3

BOTTON NAVIGATION BAR

La barra de **navegación inferior**, conocida como Bottom Navigation Bar, es un componente de interfaz de usuario que se coloca en la parte inferior de la pantalla.

Para agregarla al diseño de la aplicación, se utiliza la vista `BottomNavigationView`, a la cual se le asigna el recurso de menú correspondiente mediante la propiedad “menu”.

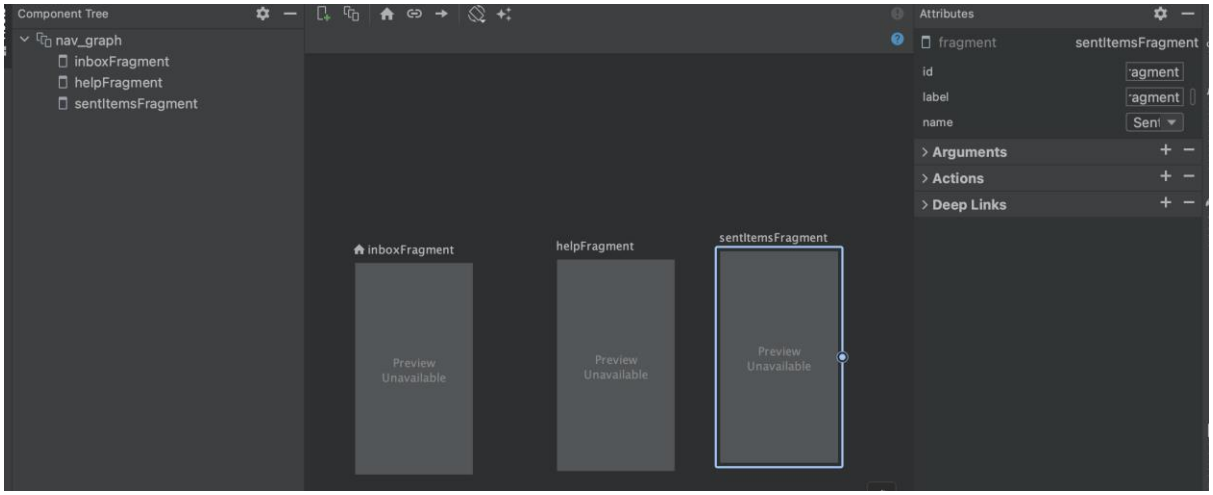
Es importante tener en cuenta que el BottomNavigationView tiene una **limitación**, por la cual es que solo puede contener un máximo de 5 elementos de navegación.

Autor/a: Sabela Sobrino Última actualización: 20.01.2025

SUBSECCIONES DE BOTTON NAVIGATION BAR

GRAFO DE NAVEGACIÓN

A continuación, debemos agregar este fragmento al grafo de navegación como se muestra en la imagen:



Además, modificaremos la etiqueta para que muestre un texto específico:

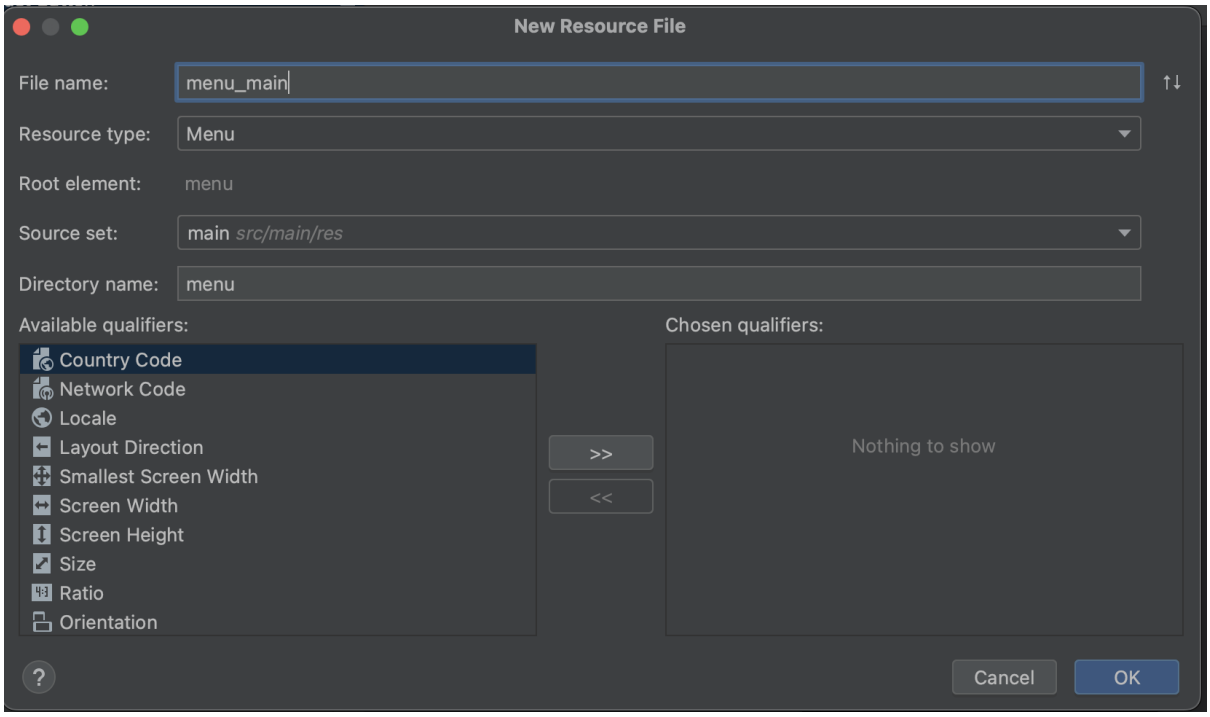
```
<fragment
    android:id="@+id/sentItemsFragment"
    android:name="com.example.uf1_ud04_1_catchat.SentItemsFragment"
    android:label="Sent Items" />
```

Esto agregará la capacidad de navegar a la pantalla “Sent Items” en nuestra aplicación.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

MENÚ

Vamos a crear el menú inferior. Para ello, necesitamos agregar un nuevo archivo de recurso de tipo menú con el nombre “menu_main,” como se muestra en la imagen:



Este menú consta de tres elementos que corresponden a las tres opciones del menú que deseamos crear:

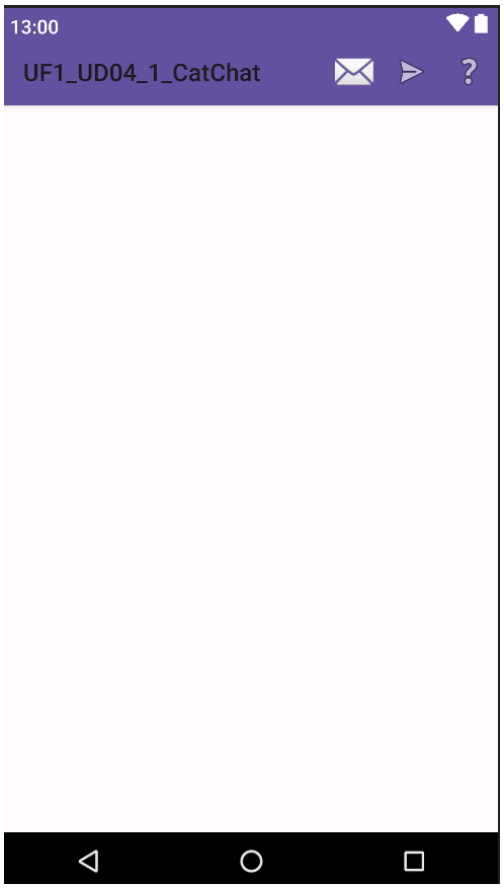
```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/inboxFragment"
        android:icon="@android:drawable/ic_dialog_email"
        android:title="Inbox"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/sentItemsFragment"
        android:icon="@android:drawable/ic_menu_send"
        android:title="Sent Items"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/helpFragment"
        android:icon="@android:drawable/ic_menu_help"
        android:title="Help"
        app:showAsAction="always"></item>
</menu>
```

Esto generará tres opciones en el menú inferior:

Este menú consta de tres elementos que corresponden a las tres opciones del menú que deseamos crear:

```xml
<?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:id="@+id/inboxFragment"
        android:icon="@android:drawable/ic_dialog_email"
        android:title="Inbox"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/sentItemsFragment"
        android:icon="@android:drawable/ic_menu_send"
        android:title="Sent Items"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/helpFragment"
        android:icon="@android:drawable/ic_menu_help"
        android:title="Help"
        app:showAsAction="always"></item>
</menu>
```

Esto generará tres opciones en el menú inferior:



BOTTON NAVIGATION BAR

Para agregar el componente de la barra de navegación **inferior** a nuestro diseño, debemos realizar los siguientes pasos en el archivo `activity_main.xml`. Primero, agregamos un

componente de tipo `BottomNavigationView`, estableciendo un ID, el ancho y el alto, y asignando el menú que acabamos de crear:

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

Sin embargo, al observar el diseño de nuestra aplicación, es posible que el menú inferior no se muestre, esto se debe a que el contenedor de fragmentos ocupa todo el espacio en el diseño. Para que se muestre el menú inferior, podemos asignar un peso de 1 al contenedor de fragmentos, lo que permite que el espacio se distribuya entre los componentes. Modificamos el **FragmentContainerView** de la siguiente manera:

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/nav_host_fragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="androidx.navigation.fragment.NavHostFragment"
    app:navGraph="@navigation/nav_graph"
    android:layout_weight="1"
    app:defaultNavHost="true" />
```

Ahora podemos verificar que el menú inferior se muestra correctamente en el diseño.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

CONTROLADOR DE NAVEGACIÓN

Para habilitar la funcionalidad de los botones, debemos vincular la **barra de navegación inferior** (`BottomNavigation`) con el **sistema de navegación**. Para lograr esto, acudimos al archivo `MainActivity.kt` y necesitamos dos cosas:

1. Obtener una referencia al nuevo componente en el diseño.

```
val bottomNav = findViewById<BottomNavigationView>(R.id.bottom_navigation)
```

2. Luego, utilizamos esta referencia para vincularla al controlador de navegación:

```
bottomNav.setupWithNavController(navController)
```

En este caso, el proceso es más sencillo que en el caso anterior. Si ejecutamos la aplicación en este punto, comprobamos que todo funciona correctamente:

NAVIGATION DRAWER

Si necesitamos incluir más de 5 elementos en nuestra barra de navegación inferior y superar esta limitación, una solución es sustituirla por un “navigation drawer,” que no tiene esta restricción. El “navigation drawer” es un panel **ocultable** deslizable con **capacidad** para más elementos de menú.

Para implementarlo, agregaremos un componente `"DrawerLayout"` a la raíz del diseño de nuestra actividad principal. Este “DrawerLayout” contendrá dos vistas principales:

1. Una vista (layout) que representa la pantalla principal, generalmente el “navigation host”.
2. Un “NavigationView” que contendrá el contenido del panel de navegación.

Cuando el panel de navegación está cerrado, el “DrawerLayout” se comporta como una actividad normal, a excepción de un ícono que permite abrir el panel.

Cuando el panel de navegación está abierto, se desliza sobre el contenido de la actividad principal, revelando el menú de acciones correspondiente.

Para personalizar el menú del “navigation drawer,” podemos realizar las siguientes acciones:

- Agregar elementos de **menú** con títulos y, si es necesario, separadores.
- Agrupar elementos utilizando el elemento “**group**”. Esto permite definir comportamientos comunes, como el resaltado del elemento seleccionado (atributo “android:checkableBehavior”).
- Asignar un recurso de tipo menú al **panel de navegación**.

Para construir el panel del “navigation drawer,” podemos definir una **cabecera** (header) que nos permita agregar una imagen u otro recurso visual. Esto se logra mediante un layout que se integra en el panel de navegación.

Finalmente, para que el “navigation drawer” sea completamente funcional, debemos realizar dos pasos importantes:

1. Agregarlo a la barra de herramientas (toolbar) de la aplicación para que el usuario pueda abrir y cerrar el panel.
2. Vincularlo con el controlador de navegación, de manera que al seleccionar un elemento del menú, la aplicación navegue a la pantalla correspondiente.

Esta implementación del “navigation drawer” brinda una mayor flexibilidad en la cantidad de elementos que podemos incluir en la barra de navegación y ofrece una experiencia de usuario más completa en aplicaciones con una amplia variedad de secciones o características.

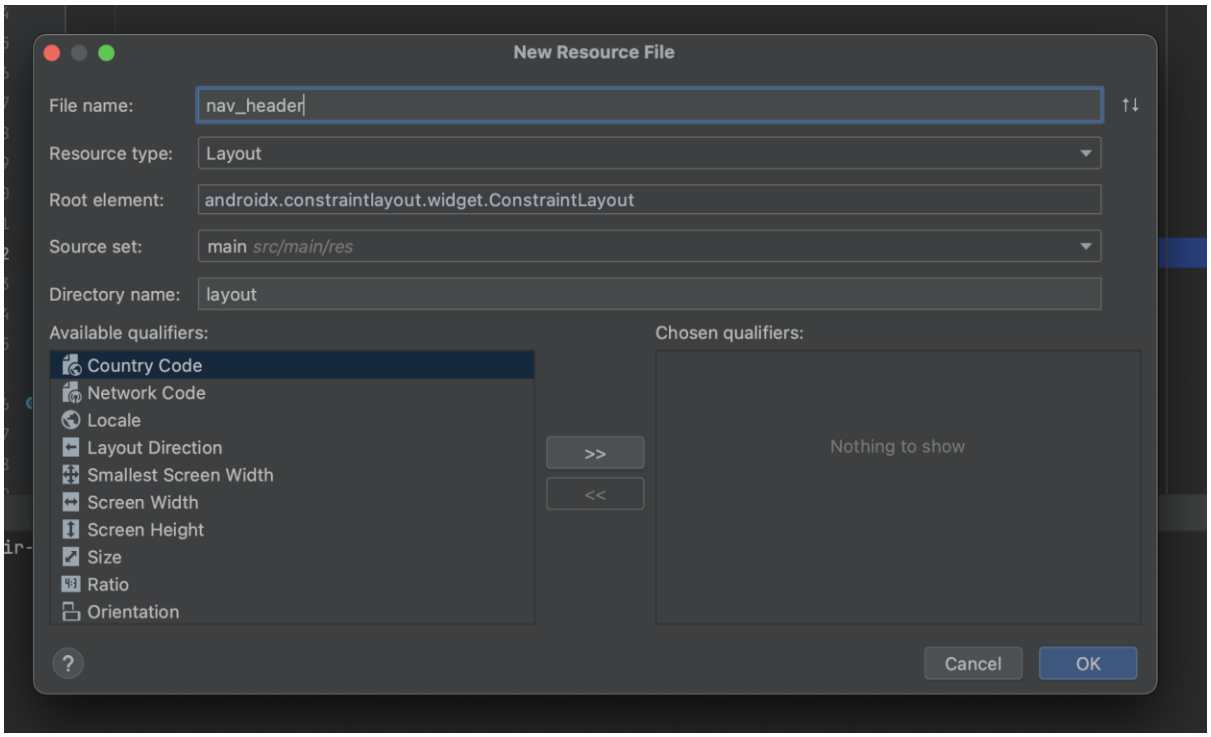
Última actualización: 08.02.2024

SUBSECCIONES DE NAVIGATION DRAWER

HEADER

Primero, crearemos el diseño de la **cabecera del menú**. Aunque esta cabecera es **opcional**, la utilizaremos en este ejemplo. Comenzaremos por descargar la imagen que se utilizará en la cabecera y la añadiremos a la carpeta “drawable” de nuestro proyecto.

Luego, crearemos un nuevo archivo de recursos de Android de tipo Layout con el nombre “nav_header”:

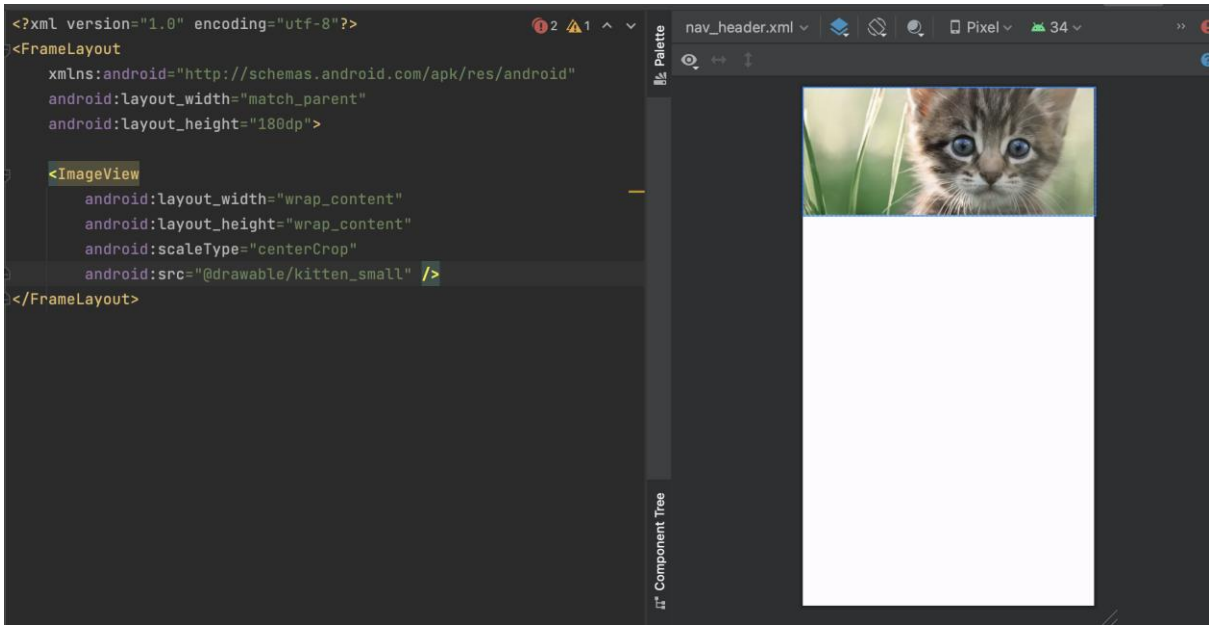


Al crear el archivo, se generará un diseño basado en ConstraintLayout. Sin embargo, para este caso, queremos un diseño más sencillo, por lo que cambiaremos el ConstraintLayout a un `FrameLayout`. Ajustaremos el ancho para que ocupe todo el contenedor, pero estableceremos una altura fija de 180dp para que no ocupe toda la pantalla. El diseño contendrá una `ImageView` para mostrar la imagen que hemos añadido previamente.

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="180dp">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/kitten_small" />
</FrameLayout>
```

Con este código, obtendremos el diseño de la cabecera con la imagen que hemos añadido.



Este diseño se usará como la **cabecera** de nuestro menú de navegación. Puedes personalizarlo más según tus preferencias o necesidades, como añadir texto o estilos adicionales.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

MENÚ

Vamos a aprovechar el diseño del **menú de la barra inferior** que ya tenemos para crear el menú lateral (navigation drawer). Para hacerlo, necesitamos agregar **separadores** para organizar las acciones en secciones dentro del menú. Para ello, modificamos el archivo "menu_main.xml" y crear un **separador** para la sección de ayuda. Esto nos permitirá agrupar elementos en un submenú específico.

Añadiremos un nuevo elemento con el título correspondiente, que en este caso será "Support". Este elemento creará una sección donde incluiremos un submenú con la opción de ayuda:

```
<item android:title="Support">
    <menu>
        <item
            android:id="@+id/helpFragment"
            android:icon="@android:drawable/ic_menu_help"
            android:title="Help"
            app:showAsAction="always"></item>
    </menu>
</item>
```

Con esta estructura, estamos creando un separador que contiene un submenú con la opción "Help".

En cuanto a los otros **elementos** dentro de este menú, vamos a agregarlos a un **grupo** para que podamos aplicar propiedades comunes a todos ellos. En este caso, aplicaremos un

comportamiento que permite resaltar el elemento seleccionado. Para lograr esto, crearemos un grupo con la etiqueta “group”, incluiremos los diferentes elementos y estableceremos la propiedad “checkableBehavior” en “single”:

```
<group android:checkableBehavior="single">
    <item
        android:id="@+id/inboxFragment"
        android:icon="@android:drawable/ic_dialog_email"
        android:title="Inbox"
        app:showAsAction="always"></item>
    <item
        android:id="@+id/sentItemsFragment"
        android:icon="@android:drawable/ic_menu_send"
        android:title="Sent Items"
        app:showAsAction="always"></item>
</group>
```

Este grupo nos permite aplicar un comportamiento de **selección única**, lo que significa que solo un elemento a la vez se resaltará como seleccionado en el menú lateral.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

DRAWER LAYOUT

Vamos a configurar nuestro diseño principal en el archivo `activity_main.xml` El elemento raíz de este diseño será un “DrawerLayout” para implementar el navigation drawer:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/drawer_layout" >
```

1. **Diseño de la pantalla principal:** Conservaremos el “LinearLayout” existente con los elementos de la pantalla principal. Sin embargo, eliminaremos el menú inferior que ya no necesitamos.

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

    <!-- Material Toolbar -->
    <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        style="@style/Widget.MaterialComponents.Toolbar.Primary" />

    <!-- Fragment Container -->
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/nav_host_fragment"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="androidx.navigation.fragment.NavHostFragment"
        app:navGraph="@navigation/nav_graph"
        android:layout_weight="1"
        app:defaultNavHost="true" />

    <!-- Bottom Navigation Bar
    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottom_navigation"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:menu="@menu/menu_main" />  -->
</LinearLayout>
```

2. **NavigationView para el panel de navegación:** Agregamos un "NavigationView" que contendrá el panel lateral de navegación. Le damos un ancho ajustado al contenido y una altura que ocupe toda la pantalla. Además, utilizamos la propiedad "layout_gravity" para posicionarlo en la parte izquierda (start) de la pantalla (ten en cuenta que dependerá de la configuración de idioma). También especificamos un identificador, la cabecera que creamos previamente y el menú que configuramos.

```
<com.google.android.material.navigation.NavigationView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:id="@+id/nav_view"
    app:headerLayout="@layout/nav_header"
    app:menu="@menu/menu_main" />
```

Con estos cambios, hemos integrado el NavigationView en el DrawerLayout. Sin embargo, para ver el menú lateral en la aplicación, debemos enlazarlo con la barra de herramientas (toolbar) y agregar la funcionalidad de navegación.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

INTEGRAR EN TOOLBAR

Para integrar el **panel lateral** con la barra de herramientas (toolbar), necesitamos realizar algunos ajustes en el código de la actividad principal (MainActivity):

1. Obtener una referencia al "DrawerLayout" que hemos creado en el diseño de la actividad principal:

```
val drawerLayout = findViewById<DrawerLayout>(R.id.drawer_layout)
```

2. Utilizar un método específico de la clase "AppBarConfiguration.Builder" para indicar que el "DrawerLayout" es parte de la configuración de la barra de herramientas:

```
val builder = AppBarConfiguration.Builder(navController.graph)
builder.setOpenableLayout(drawerLayout)
val appBarConfiguration = builder.build()
```

Con estos ajustes, cuando la barra de herramientas se configure, incluirá un **botón** que permitirá abrir y cerrar el panel lateral. Una vez que realices estos cambios, al ejecutar la aplicación, podrás comprobar que el panel lateral está presente en la interfaz, aunque aún no esté completamente funcional ya que las funciones en el panel lateral aún no están vinculadas con el sistema de navegación.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

ENLAZAR CONTROLADOR

Para enlazar los elementos del panel lateral con el sistema de navegación, procedemos de manera similar a como lo hicimos con la barra de navegación inferior (BottomNavigationView). Tenemos el componente `NavigationView` en el archivo `activity_main.xml`, que representa el panel lateral. Para enlazarlo, seguimos estos pasos:

1. Obtener una referencia al componente `NavigationView` en la actividad principal (MainActivity):

```
val navigationView = findViewById<NavigationView>(R.id.nav_view)
```

2. Usar el método `setupWithNavController` para enlazar el `NavigationView` con el controlador de navegación:

```
navigationView.setupWithNavController(navController)
```

Una vez realizados estos pasos, los botones en el panel lateral estarán vinculados al sistema de navegación y funcionarán correctamente.

Al ejecutar la aplicación, podrás comprobar que los botones del panel lateral funcionan como se esperaba.

Autor/a: Sabela Sobrino Última actualización: 08.02.2024

PRACTICA

Descripción de la Práctica:

En esta práctica, crearás una aplicación de Android UD04_2_Space que permitirá a los usuarios ver elementos visuales como imágenes de galaxias, planetas y estrellas, junto con colores oscuros y acentos luminosos que representen el misterio del cosmos.

Fragmento 1: “Planetas del Sistema Solar”

Este fragmento mostrará información sobre los planetas que componen nuestro sistema solar. Incluirá características principales como tamaño, distancia al Sol, composición atmosférica y temperatura.

Fragmento 2: “Estrellas y Constelaciones”

Este fragmento estará dedicado a las estrellas y las constelaciones más conocidas. Incluirá: representación visual de algunas de las constelaciones más reconocidas, como Orión, la Osa Mayor, Casiopea, etc.

- **Optativo: Botón para ver Leyendas y mitología**: historias detrás de las constelaciones y cómo han sido interpretadas en diferentes culturas a lo largo de la historia.
- **Optativo: Botón Tips para observación**: recomendaciones sobre cómo y cuándo observar ciertas constelaciones, con sugerencias de aplicaciones para astronomía o telescopios básicos.

Fragmento 3: "Exploración de la Luna y Marte"

En este fragmento se presentará información sobre la exploración espacial de la Luna y Marte. Incluirá detalles sobre las misiones Apollo, la llegada del ser humano a la Luna y descubrimientos relevantes.

- **Misiones en Marte**: información sobre las misiones más importantes, como el rover Curiosity, Perseverance y las futuras misiones planificadas para colonización.
- **Optativo: Galería multimedia**: imágenes y videos de las misiones espaciales más relevantes.

Fragmento 4: "Tecnología Espacial y Misiones Futuras"

Este fragmento abordará las innovaciones tecnológicas que han hecho posible la exploración espacial y lo que está por venir. Incluirá detalles sobre el funcionamiento de cohetes, satélites y estaciones espaciales como la ISS.

Menús

Nuestro aplicación tendrá:

Sidebar

Requisitos Técnicos:

- Cada fragmento proporcionará una experiencia educativa y visual atractiva para los usuarios interesados en la astronomía y la exploración del espacio.
- Deberás utilizar fragmentos para implementar cada una de las pantallas.
- La aplicación debe tener un diseño atractivo.
- Se deben utilizar menús.

Consideraciones Adicionales:

- Puedes utilizar recursos como imágenes o descripciones para cada temática de cuento.
- Asegúrate de que la aplicación sea fácil de usar y que los usuarios puedan seguir el flujo lógico de introducir su nombre, seleccionar una temática y leer su cuento personalizado.