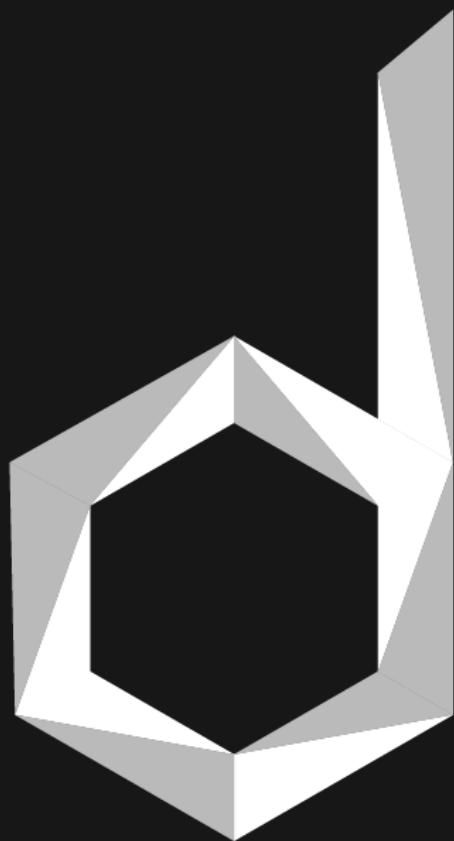


INGENIERÍA MECATRÓNICA



DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

DESARROLLO MÓVIL - ANDROID

ANDROID STUDIO

Interfaz de Usuario  
(UI) - Android Studio

## Contenido

UI (User Interface) y manejo de errores - Android Studio.....	2
Componentes de una UI .....	3
Componentes y conceptos de librerías.....	7
Librería Material Design:.....	7
Librería Android Jetpack: .....	9
Creación de recursos: app/java/res .....	9
¿Qué es LifeCycle y MVVM? .....	14
Creación de una Activity y Fragment .....	18
UI - Diseño de la interfaz.....	28
Diseño de Activities:.....	30
Activación del menú en el archivo Kotlin:.....	58
Manejo y visualización de errores .....	61
Referencias: .....	63



# UI (User Interface) y manejo de errores - Android Studio

Para poder desarrollar una interfaz de usuario, primero se debe crear una maquetación de las pantallas a desarrollar, a esta fase de desarrollo se le llama Mockups o Wireframes.

- **Mockups o Wireframes:** Es la maquetación de las pantallas a desarrollar, tomando en cuenta su flujo de ejecución. Se considera además la creación de imágenes de fondo y elementos estéticos que conformarán la aplicación.

Unos de los softwares gratuitos que se pueden usar para esto es el de Marvel, donde se pueden hacer los diseños hasta a mano:

<https://marvelapp.com/>

Otro software gratuito donde se puede hacer el maquetado de pantallas es Draw.io en la opción de Blank Diagram → More Shapes → iOS UI → iPhone (Portrait):

<https://app.diagrams.net/>

Y otro software gratuito donde se puede hacer el maquetado de pantallas es Figma:

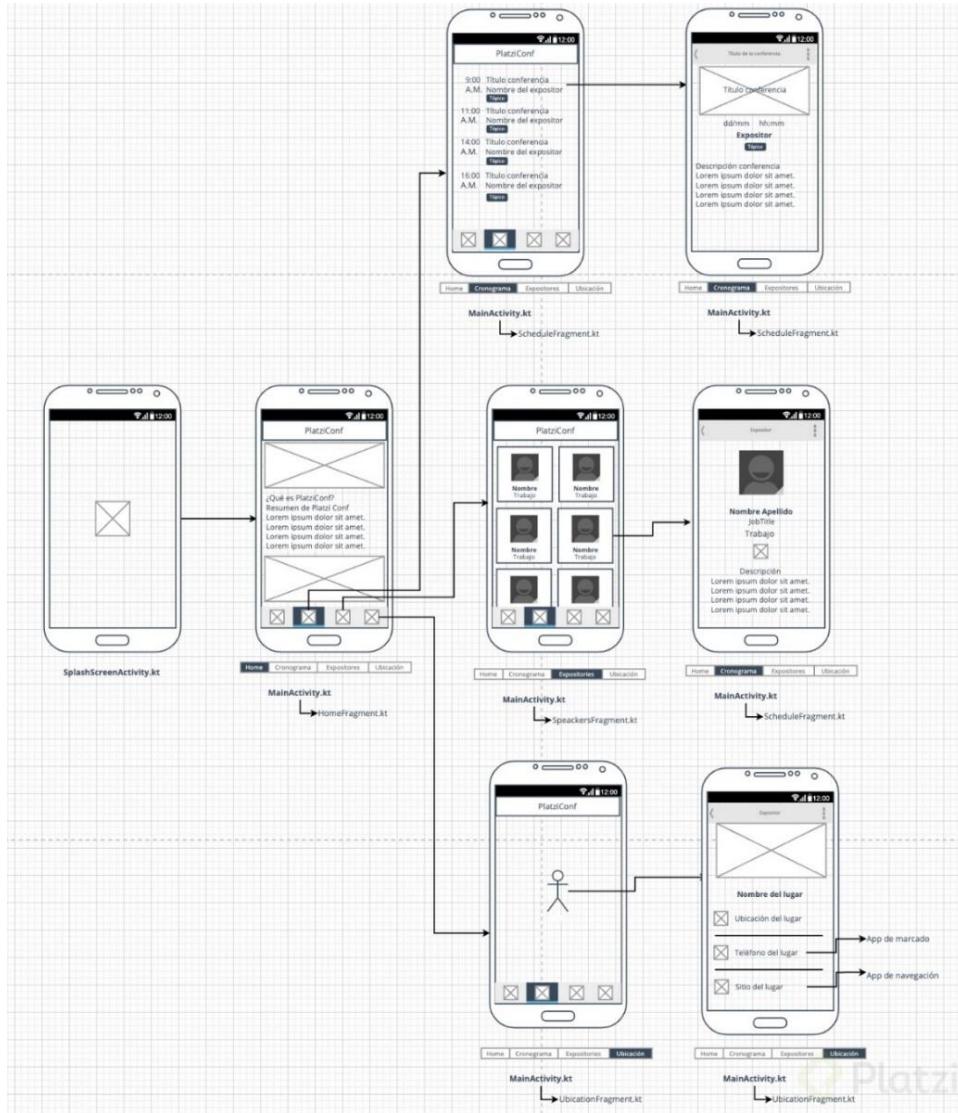
<https://www.figma.com/>

La forma en la que se estructuran los nombres y acomodo de las pantallas diseñadas en el maquetado de pantallas es la siguiente:

- Por buenas prácticas todas las **Activities** de la carpeta `app/java/com.nombreDominio.nombreProyecto` que representen las funciones de las **interfaces** creadas en la carpeta `app/res/layout` se deben nombrar empezando con una letra mayúscula, sin espacios y utilizando la palabra **Activity** hasta el final:
  - Ejemplo: `LoginActivity.kt`
- Normalmente la forma en la que se crean las apps es por medio de Activities y Fragmentos de la siguiente manera:
  1. La primera pantalla es una **Activity**, puede ser la pantalla de Login o una que tenga una animación del logo de la empresa.
  2. Luego el usuario se introduce en la pantalla principal, donde ya se muestra el menú que te permite navegar por medio de la aplicación, en esta segunda **Activity de la aplicación es donde se muestran las distintas partes de la aplicación en forma de Fragments**, que son subpantallas contenidas en una Activity y harán que se muestren las distintas pantallas a las que te dirige el menú de la app.
    - Por buenas prácticas todos los **Fragments** de la carpeta `app/java/com.nombreDominio.nombreProyecto` que representen las funciones de las subpantallas creadas para una **Activity** dentro de la carpeta `app/res/layout` se deben nombrar empezando con una letra mayúscula, sin espacios y utilizando la palabra **Fragment** hasta el final:
      - Ejemplo: `HomeFragment.kt`

En los diagramas de Mockups o Wireframes utilizados para mostrar el flujo de la aplicación Android se coloca una imagen de la pantalla mostrando más o menos que es lo que hace o contiene cada una, debajo se muestra el nombre de su Activity, y si esta contiene Fragments se muestra el nombre del

fragmento debajo del nombre de la Activity a la que pertenece, ya que el Fragmento es una subpantalla de alguna Activity.

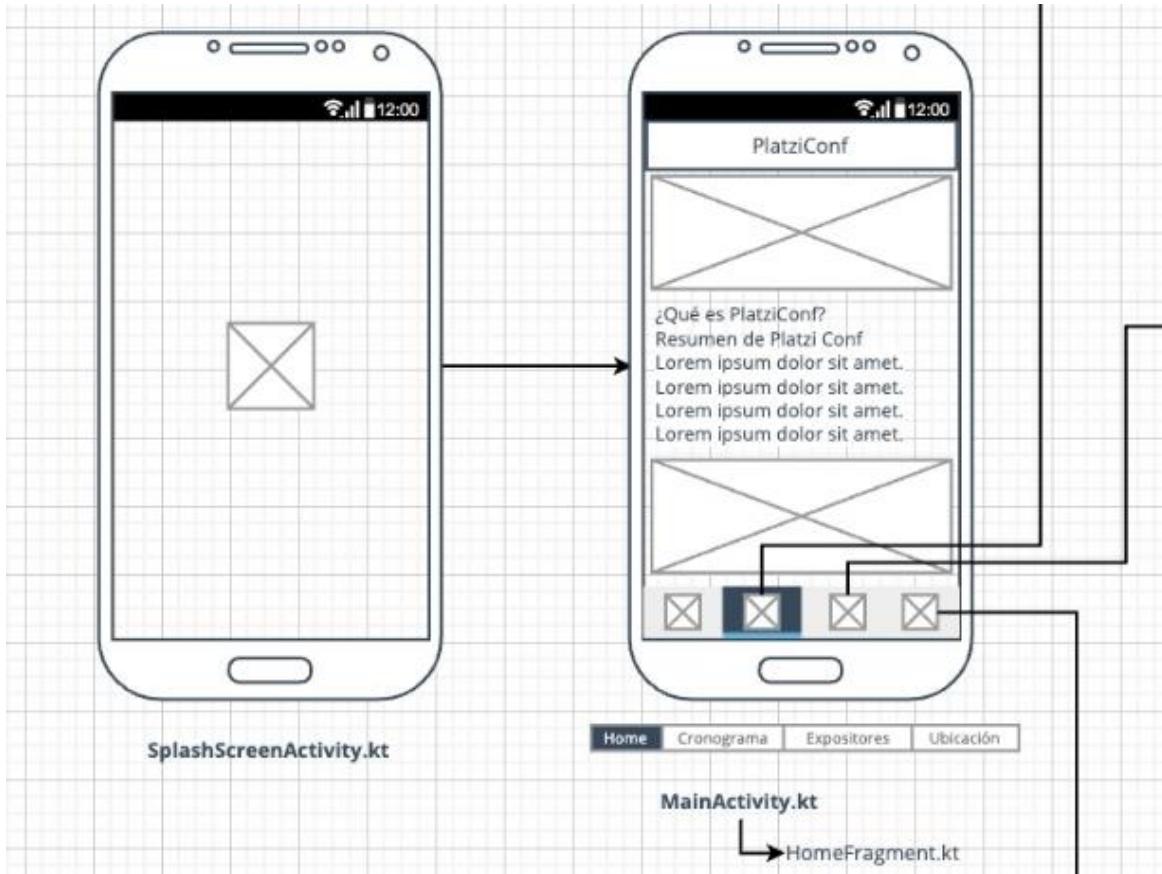


## Componentes de una UI

- **Layouts:** Son todas las pantallas que conforman las aplicaciones Android y se dividen en dos tipos:
  - **Actividad (Activity):** Representa una pantalla cualquiera de una aplicación Android, la función de una Actividad se crea con el lenguaje de programación Kotlin o Java en un archivo de la carpeta **app/java/com.nombreDominio.nombreProyecto** y su interfaz gráfica se crea con el lenguaje de programación XML en la carpeta **app/res/layout** del proyecto.
  - **Fragmento (Fragment):** Representa la subpantalla (osea una parte de la pantalla) de alguna Activity, los fragmentos son elementos dinámicos de las aplicaciones, esto implica que es un cuadro estático que cambia su contenido de forma dinámica.

Cuando aparezca el menú de la aplicación, es que se hace uso de fragmentos para mostrar las pantallas a las que te dirige cada botón del menú.

- Antes de que aparezca el menú de la aplicación se hace uso de **Actividades**.
- Cuando aparece el menú en las aplicaciones Android es cuando se hace uso de **Fragmentos** para que el menú te dirija a cada una de las distintas pantallas de la aplicación, a este tipo de fragmento que pretende actuar como si fuera una Activity se le llama **DialogFragment**.



- **item:** Es un contenedor con diseño personalizado creado específicamente para mostrar todos los elementos de una lista o grid, el diseño hecho en el item se aplicará a todos los elementos de una lista o rejilla creada con la etiqueta **RecyclerView**.
- **ViewGroups:** Es un **contenedor invisible** hecho con XML que define la estructura del diseño (más o menos como funciona la etiqueta div en diseño web), existen distintos tipos de contenedor, los cuales acomodarán de forma diferente los elementos que tengan dentro:
  - **Linear Layout:** Contenedor que ordena sus componentes de forma lineal, vertical u horizontalmente, uno arriba del otro o uno alado del otro. Permite anidación.
  - **Relative Layout:** Ordena sus elementos sobreponiéndolos uno encima de otro.
  - **Coordinator Layout:** El ViewGroup que utilice esta etiqueta se considerará como el principal, osea el que debería estar al principio del diseño y se utiliza cuando se pretenda usar varios elementos en el diseño que no queremos que se sobrepongan uno encima de otro, se quiera incluir animaciones o tener mucho dinamismo en la pantalla. Además, Posiciona los widgets de la aplicación de nivel superior, como **AppBarLayout**.

- **Constraint Layout:** Esta etiqueta XML se utiliza para crear diseños planos, no permitiendo que exista mucha anidación dentro del contenedor, es el contenedor que se crea por default en las Actividades y su peculiaridad es que permite aplicar una serie de atributos especiales para hacerlo responsivo, enlistados a continuación:
  - **layout\_constraintBottom\_toBottomOf:** Hace que el nodo inferior del ViewGroup se coloque sobre la parte inferior de otro elemento, ya sea su elemento padre o el elemento que tenga un id en específico.
  - **layout\_constraintTop\_toTopOf:** Hace que el nodo superior del contenedor se coloque sobre la parte superior de otro elemento, ya sea su elemento padre o el elemento que tenga un id en específico.
  - **layout\_constraintEnd\_toEndOf:** Hace que el nodo derecho del contenedor se coloque sobre la parte derecha de otro elemento, ya sea su elemento padre o el elemento que tenga un id en específico.
  - **layout\_constraintStart\_toStartOf:** Hace que el nodo izquierdo del contenedor se coloque sobre la parte izquierda de otro elemento, ya sea su elemento padre o el elemento que tenga un id en específico.
    - **Nota:** No es necesario colocar estas etiquetas directamente, se puede solo acceder a la ventana Design del layout y tomar los nodos que se crean en las partes superior, derecha, inferior e izquierda para colocar el View en donde sea, de esta manera también se asegura que el diseño sea responsive.
- **Views:** Es un **elemento visible** hecho con XML con el que el usuario puede interactuar y se encuentra normalmente dentro de un ViewGroup para ordenarlo de cierta manera respecto a otros Views, básicamente cada uno de los componentes colocados en la interfaz de la aplicación es llamado **View** y existen varios:
  - **TextView:** Es una caja de texto, su texto se introduce como atributo y se jala del archivo values en donde es declarado como string para que se pueda traducir a diferentes lenguajes automáticamente.
  - **ImageView:** Contenedor que jala una imagen de la carpeta **app/res/drawable**.
  - **Button:** Es un botón.
  - **EditText:** Es una caja de texto editable que permite ingresar datos a la aplicación.
  - **Checkbox:** Es una casilla de selección para cuando quiera que se puedan elegir varias opciones a la vez.
  - **RadioButton:** Es una casilla de selección para cuando quiera que solo se pueda elegir una opción a la vez.
  - **View:** Es un contenedor donde se puede poner lo que sea (semejante al div en diseño web), si está vacío puede servir para crear una línea de separación entre elementos.
- **Atributos comunes de los ViewGroups y Views:** Tanto los ViewGroups como los Views cuentan con los atributos enlistados a continuación, entre otros:
  - **android:id="@+id/nombre\_id":** El id es el identificador único que tendrá un ViewGroup o View para diferenciarse de los demás y poder referenciarlo desde cualquier parte del proyecto con el fin de darle una función o aspecto estético.
  - **android:layout\_width="ancho\_elemento":** width se refiere al ancho de un elemento cualquiera (osea un view).
  - **android:layout\_height="alto\_elemento":** height se refiere a la altura de un view.

- **Unidades de medida:** Los valores que pueden ser utilizados para dar tamaño o herencia a los atributos de los Views en la interfaz son los siguientes:

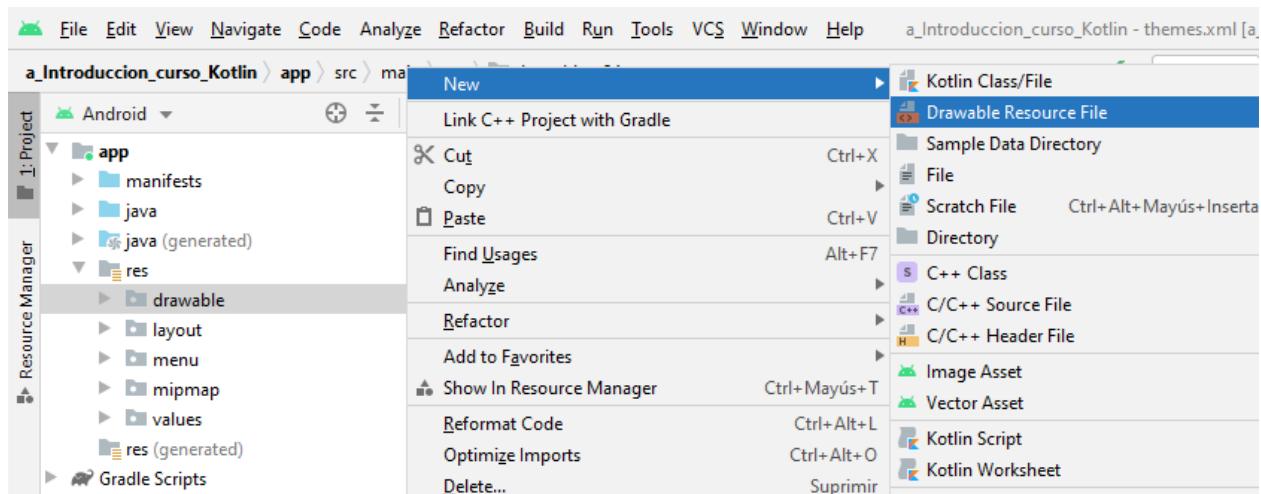
- **match\_parent:** Hace que el elemento al que se aplique tenga una dimensión fija, teniendo el ancho o alto que tiene su elemento padre (osea al de su jerarquía superior), ya sea que se encuentre dentro de un ViewGroup o que su padre sea la pantalla completa del dispositivo móvil.
- **wrap\_content:** Se utiliza este valor cuando queramos que las dimensiones del elemento no tengan una dimensión fija, sino que se adecúen de acuerdo con el contenido que tenga esa vista (View).

**En diseño de aplicaciones las medidas no pueden ser dadas en pixeles ya que la resolución entre dispositivos móviles varía.**

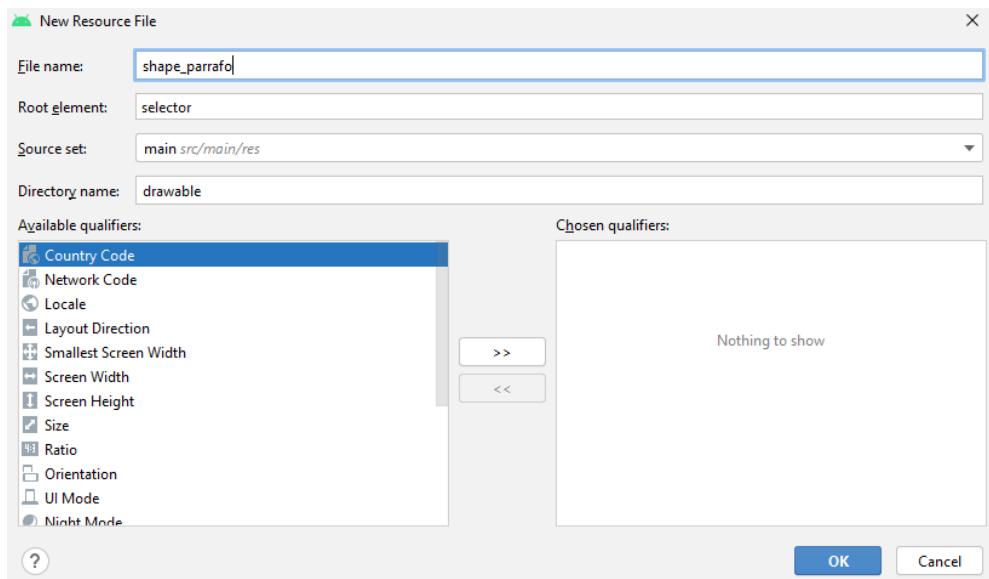
- **dp:** Es una unidad de **dimensiones** fija, que significa pixel independiente de densidad.
- **sp:** Es una unidad de **texto** fija, que significa pixel independiente de escala.

- **Shape:** Es una etiqueta especial con la que se puede personalizar el aspecto visual de los distintos Views que conforman la aplicación, este tipo de etiquetas se crean en un archivo xml distinto a donde se quieren utilizar, en específico se deben crear dentro de la carpeta drawable del proyecto.

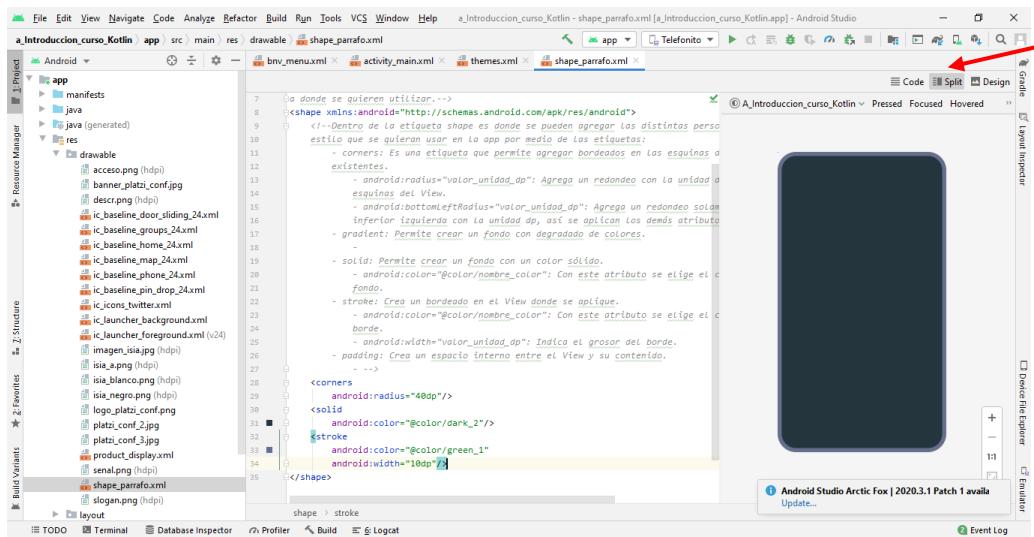
- **Corners:** Es una etiqueta que permite agregar bordeados en las esquinas de Views existentes.
- **Gradient:** Permite crear un fondo con degradado de colores.
- **Solid:** Permite crear un fondo con un color sólido.
- **Stroke:** Crea un bordeado en el View donde se aplique.
- **Padding:** Crea un espacio interno entre el View y su contenido.
  - **Este tipo de etiquetas deben ser creadas en un archivo xml distinto a donde se use, dentro de la carpeta app/res/drawable dando clic derecho en la carpeta drawable → New → Drawable Resource File:**



Por buenas prácticas el nombre de estos archivos se empieza con la palabra shape de la siguiente manera: `shape_nombreArchivo.xml`



Además, dentro de este tipo de archivos activando la ventana de Split o Design podemos observar el cambio que hará en el elemento cuando el shape se aplique a un View dentro de la app.



## Componentes y conceptos de librerías

### Librería Material Design:

- **Material Design:** Material Design no es solamente el nombre de la librería que extrae componentes estéticos para la interfaz de usuario, sino que es una guía de diseño para aplicaciones móviles, que optimiza su aspecto visual y estético.
- **Material Theming:** Es la personalización del diseño obtenido al seguir la guía de diseño Material Design.
- **Material components:** Son bloques de construcción preconstruidos para las interfaces de usuario que provienen de la librería Material Design, como lo son:

- **AppBar:** Es la barra superior de una activity y se utiliza a través de las dos siguientes etiquetas:
  - **AppBarLayout:** Es la primera etiqueta que se pone para crear una barra de navegación.
 

```
<com.google.android.material.appbar.AppBarLayout/>
```
  - **Toolbar:** Esta etiqueta se coloca dentro de **AppBarLayout** para así poder crear la barra de navegación:
 

```
<androidx.appcompat.widget.Toolbar/>
```

    - Para que solo aparezca una barra superior en la aplicación y no dos, debemos ingresar a la carpeta **app/res/values/themes** y en ambos archivos cambiar el valor del atributo parent dentro de la etiqueta style, en específico debemos cambiar el valor:

`Theme.MaterialComponents.DayNight.DarkActionBar`

**Por el valor:**

`Theme.MaterialComponents.DayNight.NoActionBar`

**Y además agregar las etiquetas:**

```
<item name="android:windowFullscreen">false</item>
<item name="android:windowNoTitle">true</item>
```

**Ya agregadas esas etiquetas, no aparecerá la barra superior que sale por defecto en la aplicación, solamente la creada con AppBarLayout y Toolbar.**

- **BottomNavigationView:** Menú inferior para la navegación entre las distintas pantallas, puede tener mínimo 3 opciones de navegación y máximo 5, hace uso de un fragmento especial llamado FragmentContainerView.

```
<com.google.android.material.bottomnavigation.BottomNavigationView/>
<androidx.fragment.app.FragmentContainerView/>
```

- El menú inferior de navegación se coloca dentro del layout principal (que es la pantalla de la activity principal de la aplicación llamada activity\_main.xml, conectada al archivo MainActivity.kt) donde después de poner un contenedor principal **CoordinatorLayout** se coloca la barra de navegación con la etiqueta **AppBarLayout** y dentro **Toolbar**, para posteriormente colocar un contenedor **RelativeLayout** (dentro del mismo CoordinatorLayout) que tendrá el contenido de los DialogFragments dentro de la etiqueta **FragmentContainerView** y finalmente se coloca el menú de navegación por medio de la etiqueta **BottomNavigationView**:

```
<CoordinatorLayout
    <AppBarLayout
        <Toolbar/>
    />
    <RelativeLayout
        <FragmentContainerView/>
        <BottomNavigationView/>
    />
```



/>

- Para que se pueda utilizar el **BottomNavigationView** no basta con solo ponerlo en el layout activity\_main.xml, sino que es necesario poner código adicional en el archivo Kotlin correspondiente (MainActivity.kt) para activar su función.

## Librería Android Jetpack:

• **AndroidX:** Es la actualización necesaria que se debe implementar para poder utilizar Jetpack, entre otras librerías. Básicamente es un espacio para las librerías Android.

• **Android Jetpack:** Sirve para añadir funcionalidad a los layouts de la aplicación Android y son un conjunto de herramientas que ayudan a acelerar el desarrollo de la aplicación por medio de los siguientes principios:

**Fundación:** Permite hacer que la aplicación pueda interactuar con el código Kotlin.

**Arquitectura:** Permite que la aplicación se adapte a cualquier tipo de arquitectura de diseño, acceda a permisos y todo lo relacionado con la arquitectura del proyecto.

**Comportamiento:** Permite realizar pruebas.

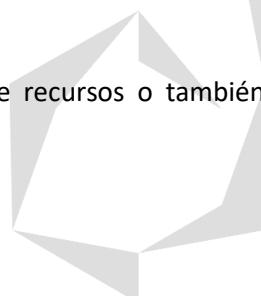
**UI:** Ayuda a la estética de la interfaz.

- **Navigation:** Es un componente de Android Jetpack que sirve para poder indicar al menú como va a pasar de un DialogFragment a otro para navegar entre las distintas páginas de la aplicación Android.
  - **NavGraph:** Es un gráfico de navegación entre fragmentos para poder observar la conexión entre DialogFragments de mejor manera.
    - **Para poder crear un NavGraph se deben agregar dos dependencias en el archivo build.gradle(Module: NombreProyecto.app).**

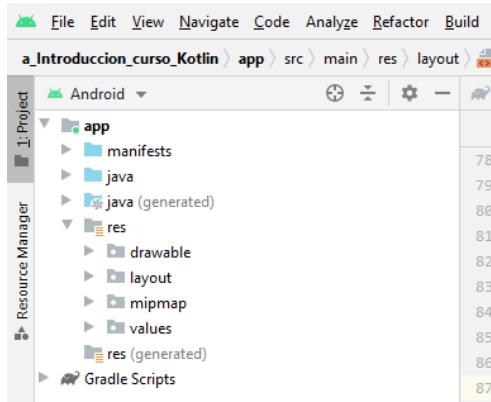
```
Implementation 'androidx.navigation:navigation-fragment-ktx:2.3.4'  
implementation 'androidx.navigation:navigation-ui-ktx:2.3.4'  
    • Y una dependencia en el archivo build.gradle(Project:  
        NombreProyecto.app)  
  
classpath 'androidx.navigation:navigation-safe-args-gradle-plugin:2.1.0'  
    • Ya habiendo dado clic en el botón de Sync Now y que se implemente la  
        dependencia en las carpetas del proyecto, para poder crear un NavGraph se  
        debe crear un nuevo recurso en la carpeta de res que sea de tipo navigation.
```

## Creación de recursos: app/java/res

En un inicio los proyectos Android solo cuentan con las siguientes carpetas de recursos o también llamados paquetes:



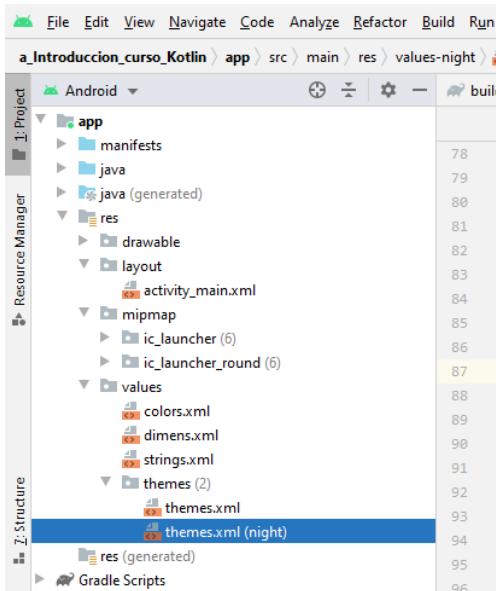
- **drawable**: Carpeta que contiene todas las imágenes y donde se crean los archivos shape para dar estilo a los Views del proyecto.
- **layout**: Es donde se crean todos los layouts (los ítems, activities y fragments del proyecto).
- **mipmap**: Contiene el ícono de la aplicación en todos los tamaños que puede ser requerido.
- **values**: Dentro tiene todos los códigos hex de los colores utilizados en la app, el título de la aplicación, las palabras que podrán ser traducidas automáticamente, las dimensiones de los textos y contenedores que se podrán repetir en varios elementos y el archivo themes en donde se aplican los colores que utiliza la app.



*Nota: Todos los textos utilizados en un View de tipo TextView utilizados en los layouts serán obtenidos del archivo strings.xml de la carpeta app/res/values.*

Adicionales a estas carpetas (o paquetes) que se crean por default, se pueden crear las carpetas de:

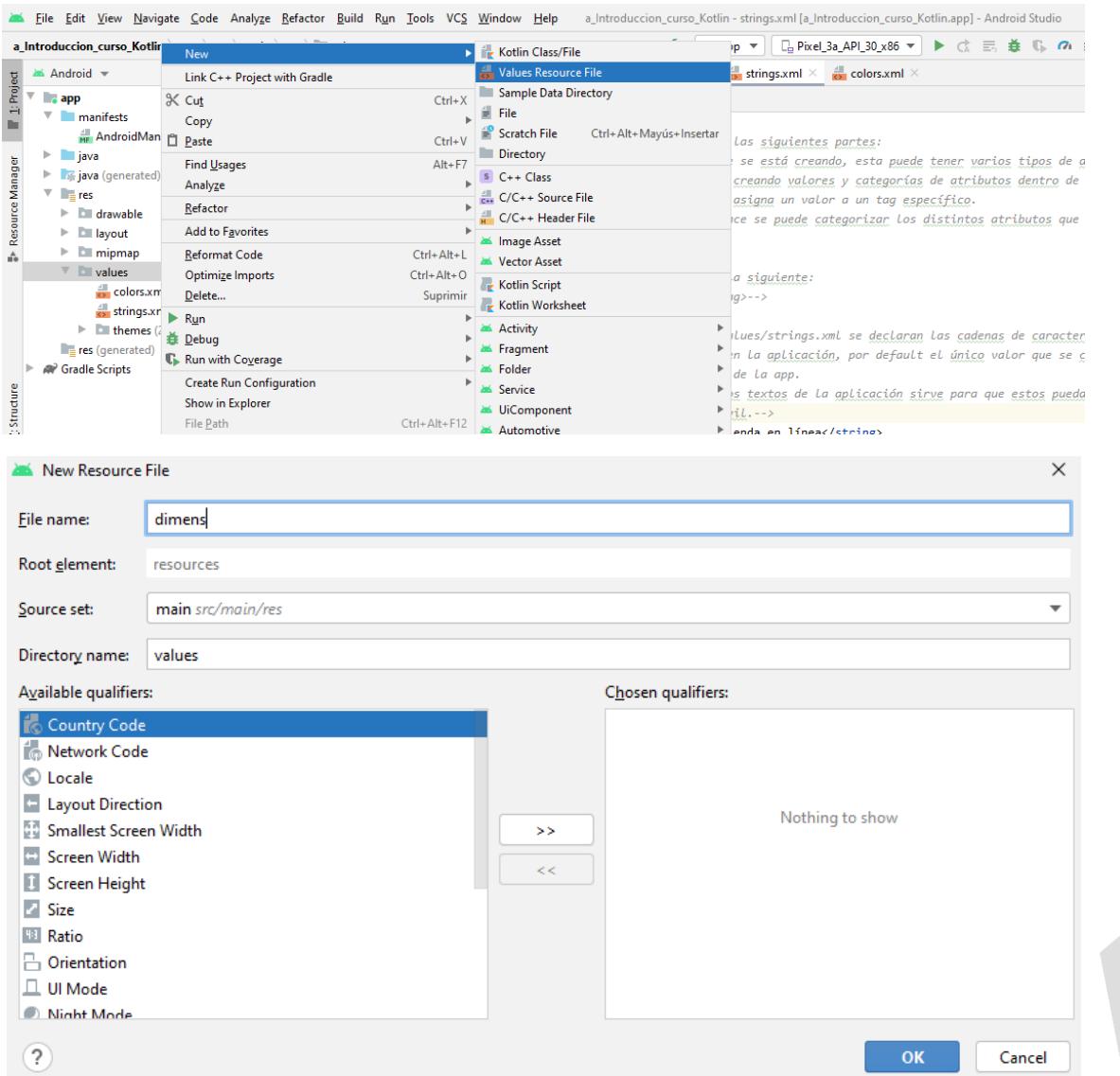
- **anim**: Donde se almacenarán las animaciones que utiliza la aplicación.
- **navigation**: En esta carpeta se indicarán las transiciones que realiza el menú entre los fragmentos que representen cada pantalla de la aplicación (DialogFragments).
- **raw**: Es donde se colocarán algunos recursos adicionales que no entren en ninguna de las categorías anteriores, como lo puede ser la personalización de un mapa de Google Maps, etc.



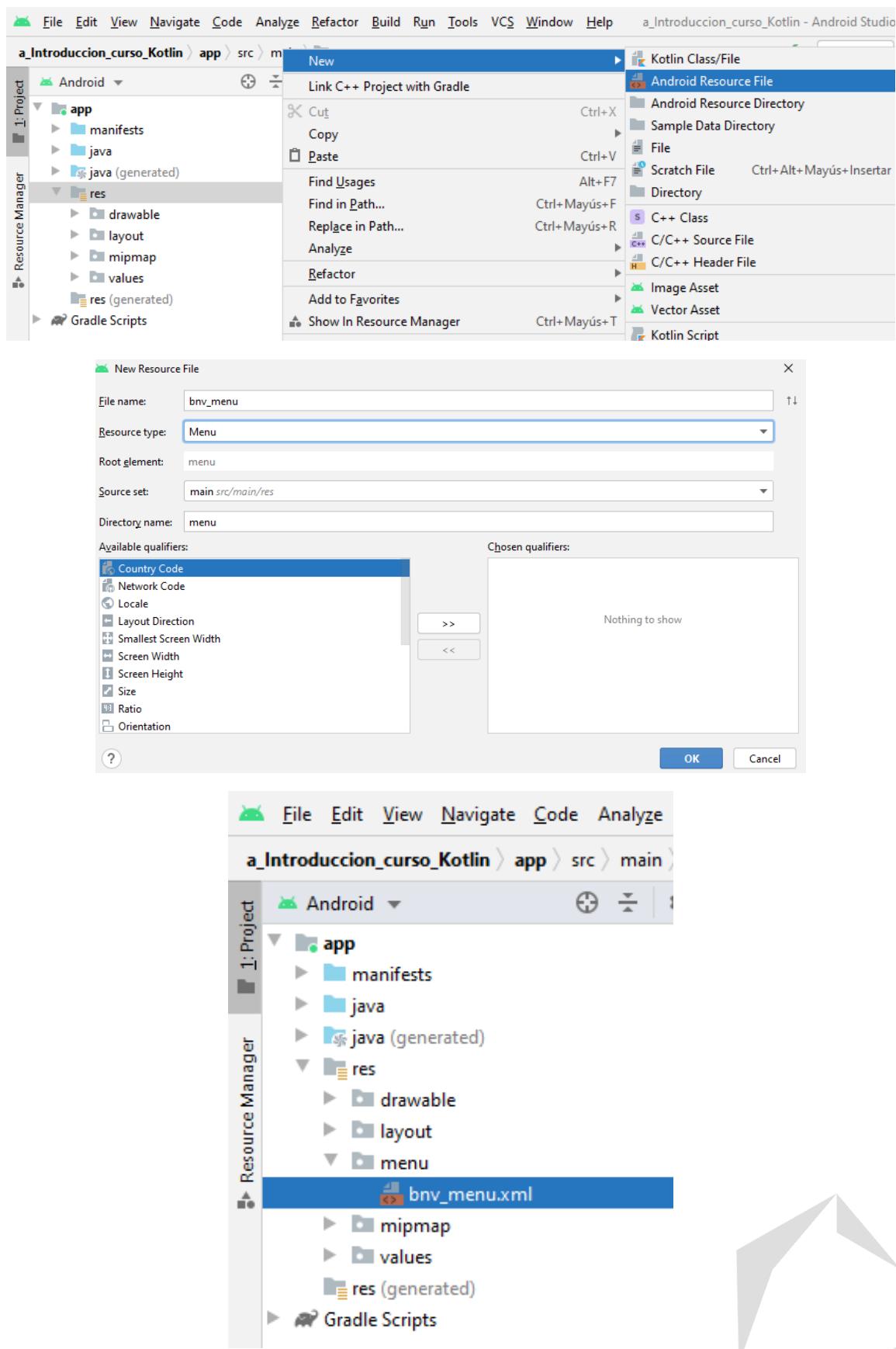
*Nota: Normalmente todos los archivos de la carpeta de recursos que no sean imágenes serán de tipo xml.*

Por buenas prácticas, todos los nombres de los layouts deben estar en minúsculas, las palabras separadas por un guión bajo, no deben empezar con un número ni usar caracteres especiales, el nombre de las actividades debe empezar con la palabra **activity\_** y los fragmentos con la palabra **fragment\_** y los archivos que conforman los recursos deben estar lo más separado posible en distintas carpetas que los clasifiquen.

Para crear un nuevo recurso en una carpeta o paquete, debemos dar clic derecho en alguna carpeta y seleccionar la opción de New → Values Resource File → County Code → Ok.

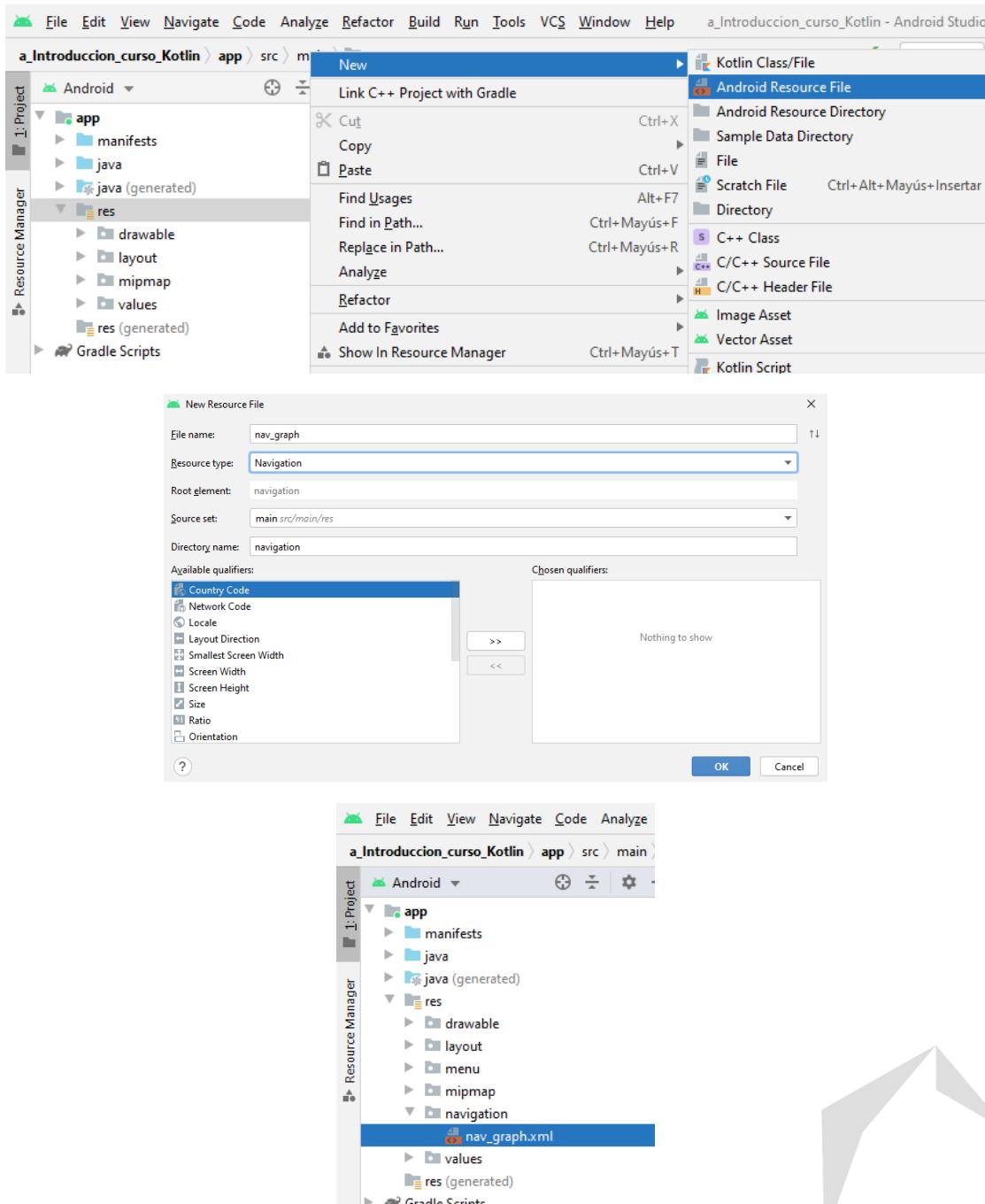


- **menu:** El menú de navegación es un caso especial ya que se crea fuera de cualquier paquete de la carpeta de recursos (res) dando clic derecho en la carpeta res → New → Android Resource File → File name: Dar nombre al layout del menú → Resource type: → Menu → Ok, y con esto se creará una carpeta nueva dentro de la carpeta de recursos llamada menú, donde estará el diseño del menú de navegación:



Dentro del layout de tipo menú se agregan las opciones del menú por medio de etiquetas item.

- **navigation:** Para poder crear este tipo de elemento, antes se debe haber instalado la dependencia de Android Jetpack, luego como el diagrama de navegación (NavGraph) es un caso especial, se crea fuera de cualquier paquete de la carpeta de recursos (res) dando clic derecho en la carpeta res → New → Android Resource File → File name: Dar nombre al layout del navgraph → Resource type: → Navigation → Ok, y con esto se creará una carpeta nueva dentro de la carpeta de recursos llamada navigation, donde estará el diseño del diagrama de navegación (NavGraph):



# ¿Qué es LifeCycle y MVVM?

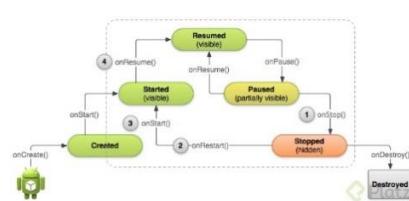
- **LifeCycle o Ciclo de vida de una Activity:** Es un proceso interno que ocurre en un proyecto Android cuando se ejecuta una Actividad y se compone de los siguientes pasos:
  - **Activity launched:** Ocurre cuando cualquier pantalla (actividad) de una aplicación es activada (osea que es puesta en primer plano), ya sea porque se hizo clic en algún botón que nos redirigió ahí (ya sea del menú o de cualquier otro botón) o porque se acaba de iniciar la aplicación. La acción de poner en primer plano una actividad ejecuta 3 métodos en el siguiente orden:
    - onCreate():
    - onStart():
    - onResume():
  - **Activity running:** Es el estado que alcanza el ciclo de vida de una pantalla (actividad) después de haberse puesto en primer plano y ejecutado los 3 métodos de inicialización en orden (onCreate, onStart y onResume), por lo que este estado se encuentra en el método onResume() y muestra la actividad en pantalla.

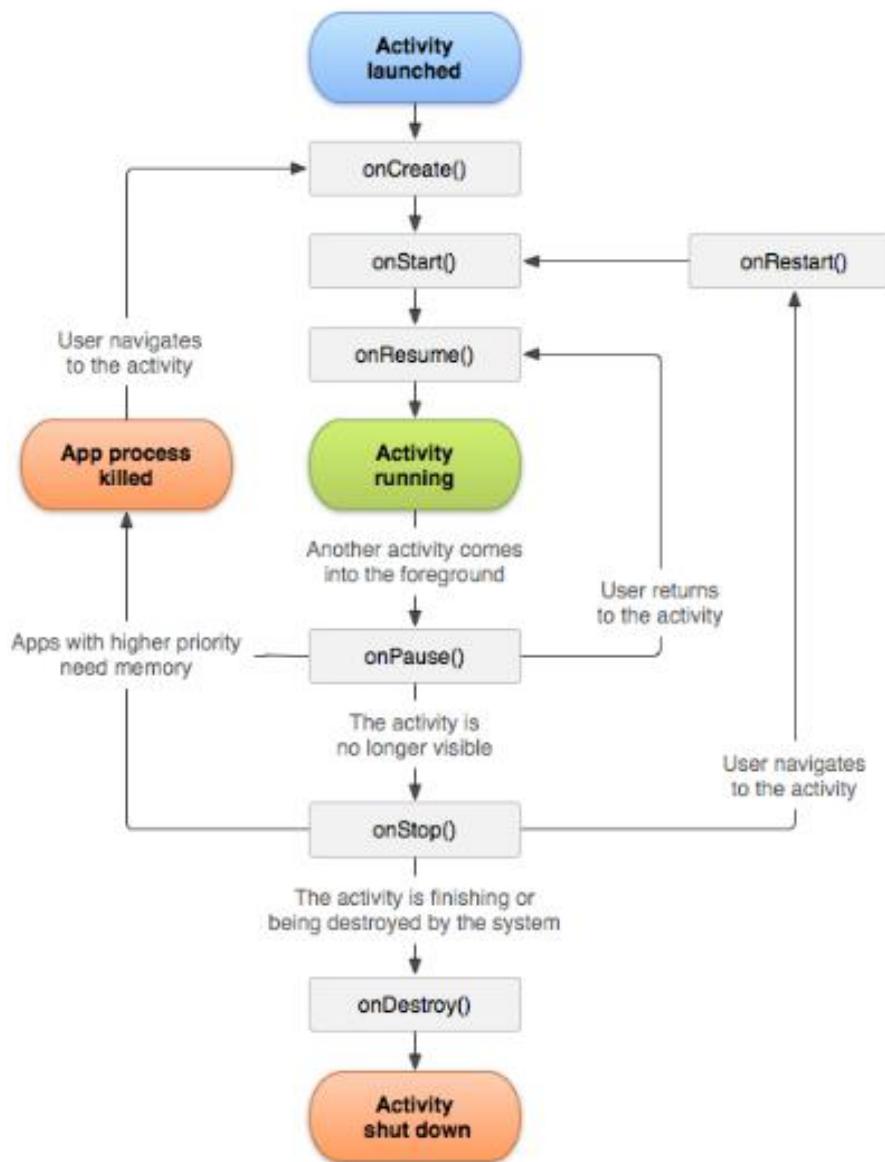
Si en este momento abrimos alguna otra pantalla de la aplicación, se ejecutarán los siguientes métodos para colocar la actividad actual en segundo plano y la nueva en primer plano:

- onPause():
- onStop():
  - Cuando alguna actividad se coloque en segundo plano y se inicialice otra, lo que estará pasando es que se creará una **pila de actividades**, poniendo la nueva pantalla inicializada encima de la actual. En la nueva actividad creada se ejecutarán los métodos del estado **Activity launched** (onCreate, onStart y onResume).

Si cerramos la pantalla actual, se ejecutarán los siguientes métodos:

- onPause():
- onStop():
- onDestroy():
- **Activity shut down:** Es el estado que alcanza el ciclo de vida de una pantalla cuando esta es cerrada, por lo cual se encuentra en el método onDestroy() y las pantallas que hayan sido puestas en segundo plano de la **pila de actividades** que actualmente se encontrarán en el estado onStop() serán reiniciadas, por lo que ejecutarán el siguiente método:
  - onRestart():
    - Este método redirige el ciclo de vida de la pantalla que se encuentre en segundo plano dentro de la pila de actividades al 2do método del estado **Activity launched** que es el de onStart(), devolviéndola a primer plano.





Todas las pantallas, osea las Actividades (Activities) que conforman la aplicación tienen un ciclo de vida, los fragmentos, aunque son parte de las Activities tienen sus propios ciclos de vida y serán descritos a continuación:

- **LifeCycle o Ciclo de vida de un Fragment:** Es un proceso interno que ocurre en un proyecto Android cuando se ejecuta una Actividad que contiene uno o varios fragmentos. Recordemos que hay dos tipos de fragmentos:
  - **Fragmento:** Es un simple contenedor que cambia su contenido de forma dinámica dentro de una pantalla o Activity.
  - **DialogFragment:** Es un tipo de fragmento cuya función es aparentar ser una Activity y mostrar las distintas pantallas de una aplicación, realiza lo mismo que el otro tipo de fragmento, mostrar contenido de forma dinámica, pero este lo realiza utilizando todo el espacio de la pantalla para aparentar ser una actividad nueva.

Aunque ambos fragmentos cumplirán las mismas reglas y pasos en su ciclo de vida, los dos dependerán del ciclo de vida de la Activity que los contiene, su ciclo de vida se compone de los siguientes pasos:

- **Fragment is added:** Ocurre cuando cualquier pantalla (actividad) que contenga un fragmento sea activada (puesta en primer plano), ya sea porque se hizo clic en algún botón que nos redirigió ahí o porque se acaba de iniciar la actividad que contiene uno o varios fragmentos. La acción de poner en primer plano el fragmento de una actividad ejecuta los 3 métodos que se ejecutaban cuando se inicializa una actividad (onCreate, onStart y onResume), pero con otros 3 añadidos (onAttach, onCreateView y onActivityCreated), y se ejecutan en el siguiente orden:
  - onAttach():
  - **onCreate():**
  - onCreateView():
  - onActivityCreated():
  - **onStart():**
  - **onResume():**
- **Fragment is active:** Es el estado que alcanza el ciclo de vida de la aplicación después de haber puesto en primer plano el o los fragmentos de una actividad, por lo que el estado del fragmento y la actividad se encuentra en el método onResume() y se muestra la actividad con su fragmento en pantalla.

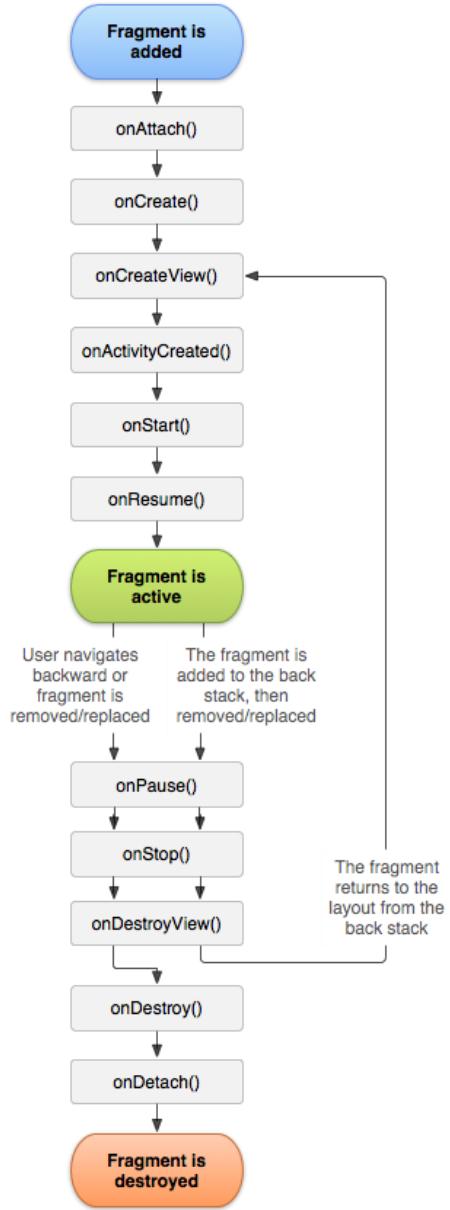
Si en este momento abrimos algún otro fragmento de la actividad por medio del menú, enseñando así una nueva “pantalla” de la aplicación, el fragmento actual se reemplazará con uno nuevo, por lo que el fragmento actual pasará a segundo plano y se ejecutarán los siguientes métodos:

- onPause():
- onStop():
  - Cuando un fragmento se coloque en segundo plano y se inicialice otro, lo que estará pasando es que se creará una **pila de fragmentos**, poniendo el nuevo fragmento inicializado encima del actual. En el nuevo fragmento creado se ejecutarán los métodos del estado **Fragment is added**.

Si cerramos el fragmento actual, se ejecutarán los siguientes métodos:

- onPause():
- onStop():
- onDestroyView():
- onDestroy():
- onDetach():

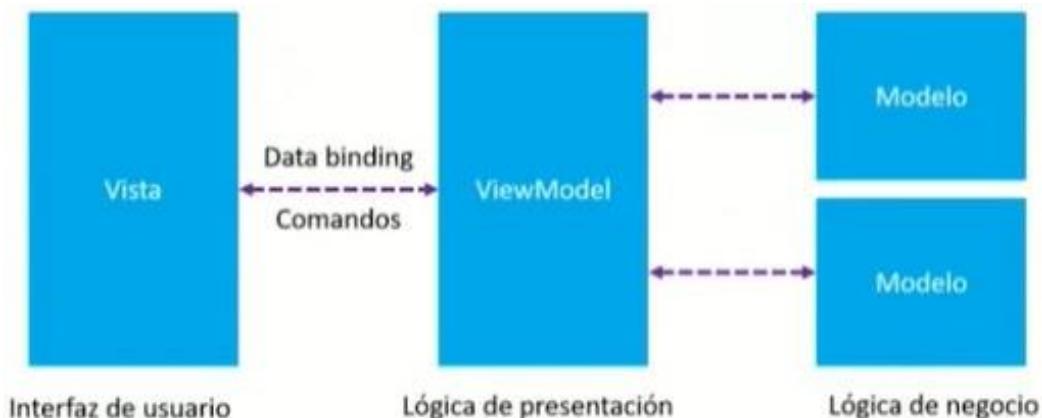
- **Fragment is destroyed:** Es el estado que alcanza el ciclo de vida de un fragmento cuando es cerrado, por lo cual se encuentra en el método onDetach() y los fragmentos que hayan sido puestos en el segundo plano de la **pila de fragmentos** serán reiniciados, por lo que regresarán al método onCreateView() del estado **Fragment is added**, devolviendo el fragmento que sigue en la pila de fragmentos a primer plano:



Todas las pantallas, osea las Actividades (Activities) que conforman la aplicación tienen un ciclo de vida, los fragmentos, aunque son parte de las Activities tienen sus propios ciclos de vida.

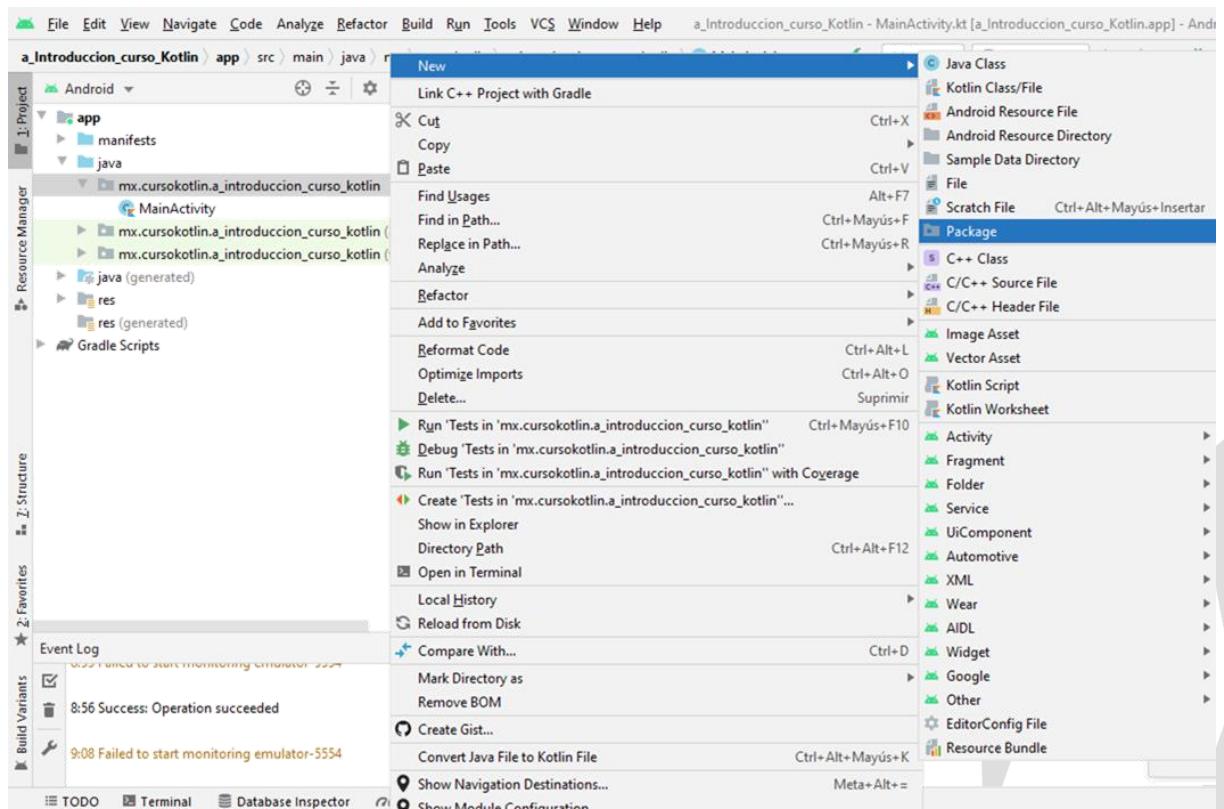
- **MVVM (Model-View-View-Model):** Es el modelo estándar para crear aplicaciones Android y se basa en dividir las aplicaciones en 3 secciones:
  - **Interfaz de usuario:** Es la vista, osea toda la interfaz de usuario (UI) de la app, usualmente manejados por los layouts hechos con lenguaje XML.
  - **Lógica de presentación:** Es la conexión entre los datos del negocio y la interfaz de usuario, para que puedan ser mostrados, usualmente manejados por los archivos Kotlin.
  - **Lógica de negocio:** Son los datos recopilados por el negocio que querrán ser mostrados en la UI, manejados por la base de datos como lo puede ser Firebase en conjunto con

los archivos Kotlin para llamar o mandar datos y acomodarlos en la base de datos. Model se refiere a las tablas de la BD.

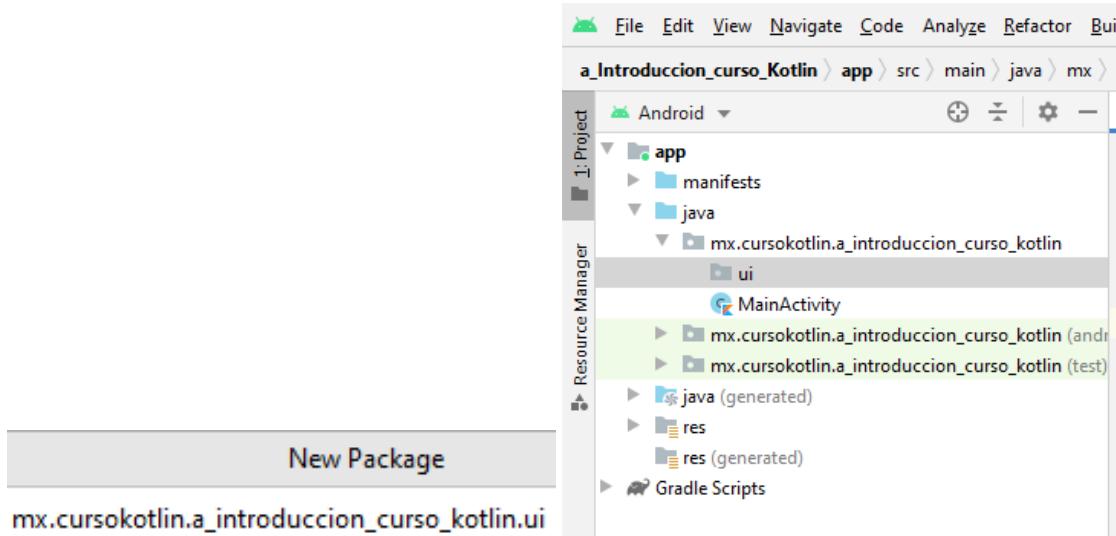


## Creación de una Activity y Fragment

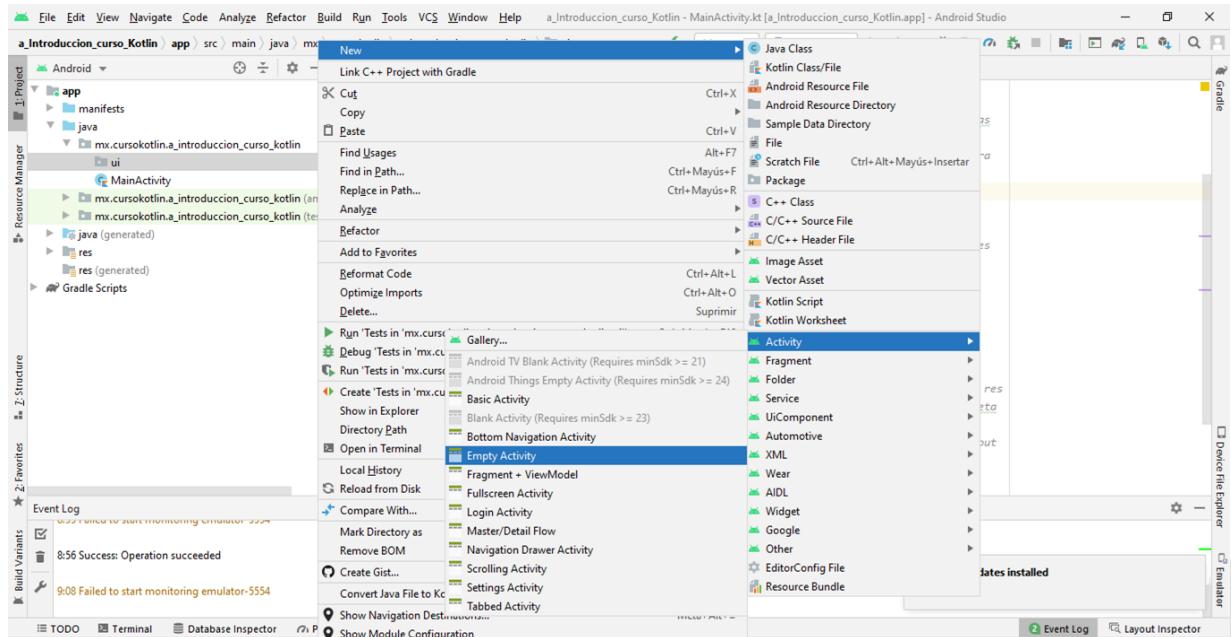
- Creación de una Activity:** Dentro de la carpeta `app/java/com.nombreDominio.nombreProyecto` no se crean solamente las Actividades, también se crean los archivos que hacen la conexión con la base de datos, etc. por lo que es necesario separar el tipo de archivos por paquetes dando clic derecho en la carpeta y seleccionando la opción de New → Package.



- Como este paquete contendrá todas las actividades que estarán conectadas con los layouts de la interfaz, le pondremos como nombre ui por User Interface.



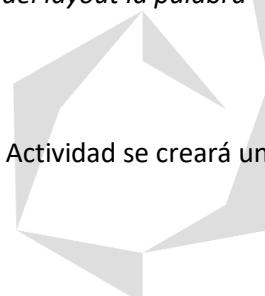
- Ahora para crear una nueva Activity daremos clic derecho en el paquete ui y seleccionaremos la opción de New → Activity → Empty Activity → Activity Name: → Poner un nombre que empiece con mayúscula y tenga la palabra Activity al final → Finish.



*Nota: Cuando se indique el nombre de la nueva Activity creada, por default se creará un layout que tenga el mismo nombre, siguiendo las buenas prácticas, por lo que en el nombre del layout la palabra activity se pondrá al inicio.*

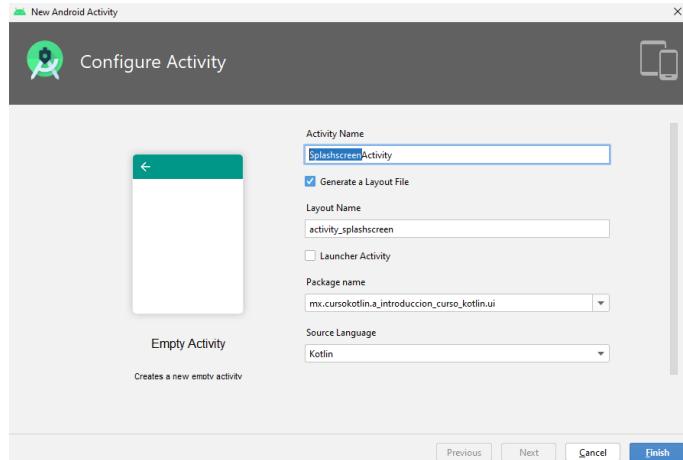
También hay dos checkbox:

- ✓ **Generate a Layout file:** Si está seleccionado el checkbox, al crear la nueva Actividad se creará un archivo layout que representará su UI.

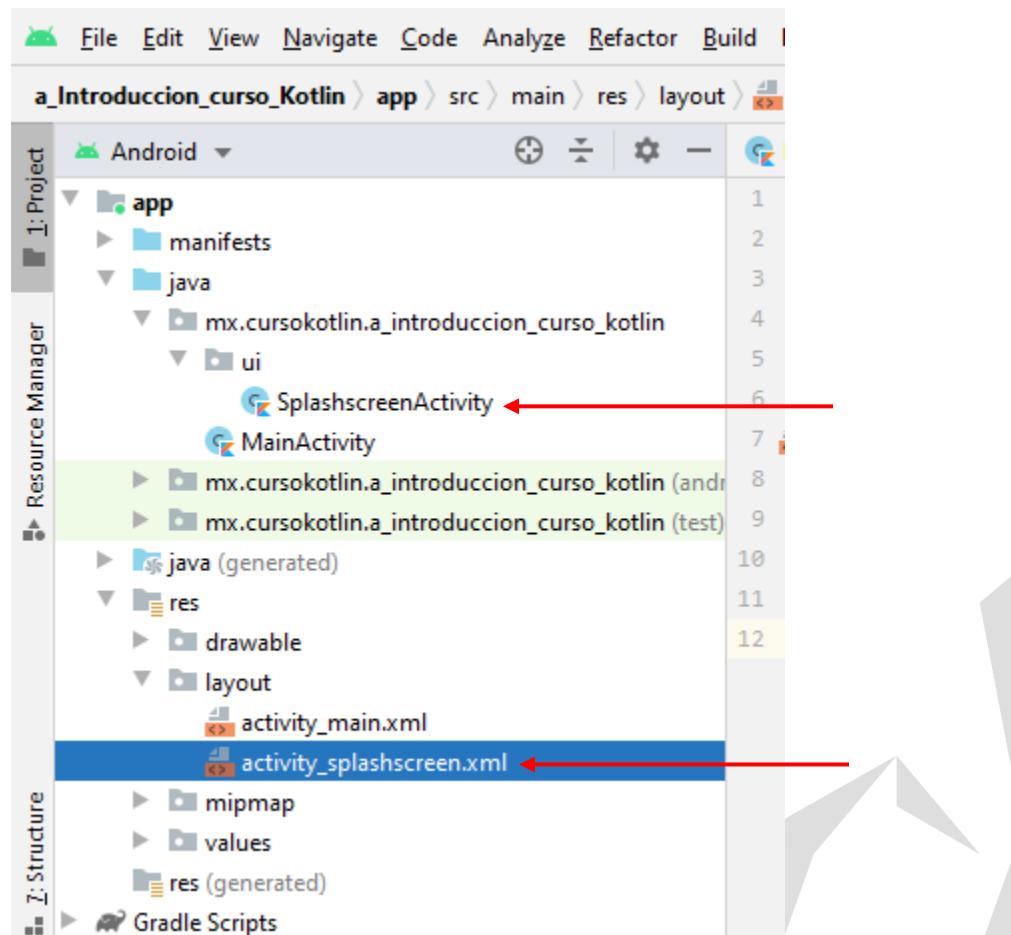


- ✓ **Launcher Activity:** Si está seleccionado el checkbox, al crear la actividad esta será la que se ejecute primero cuando sea abierta la aplicación.

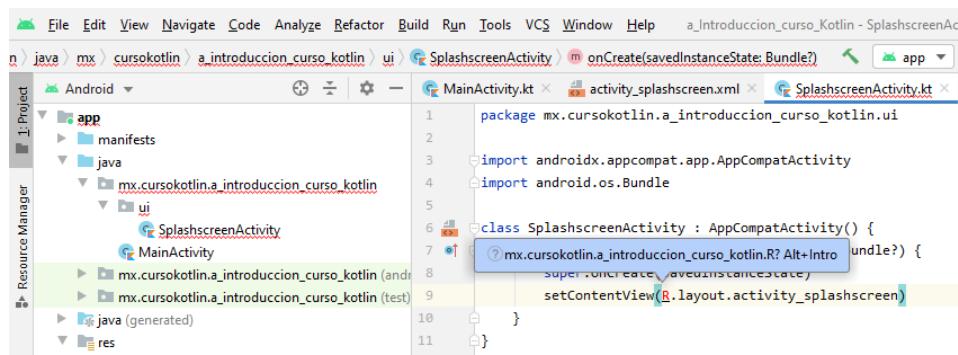
Además, se indica el paquete al que pertenece el archivo y el lenguaje de programación (Java o Kotlin) que se está utilizando en él.



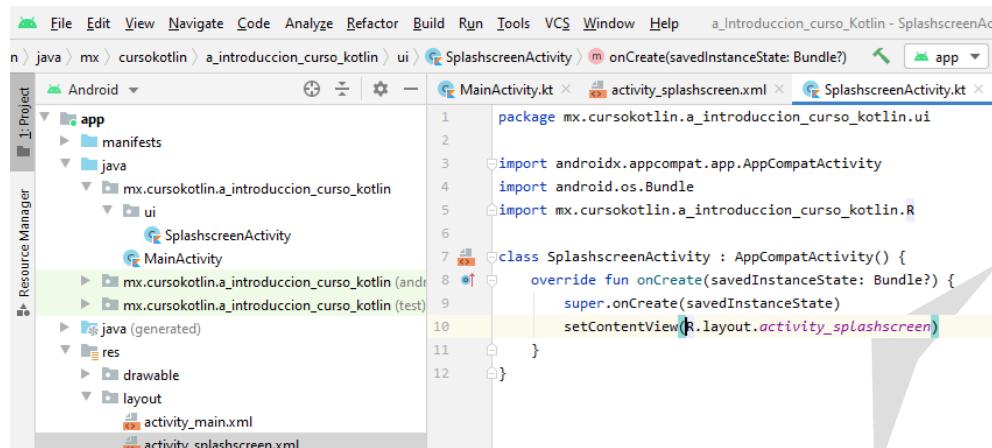
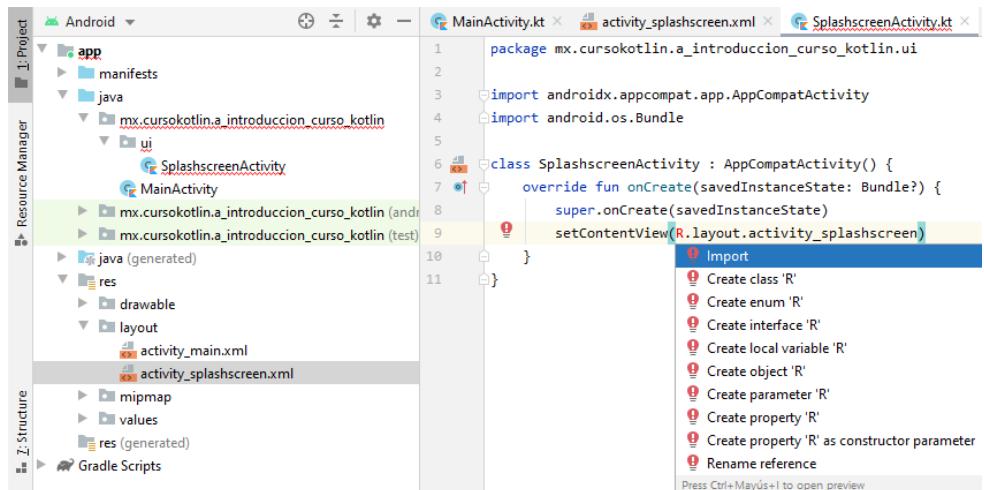
Al dar clic en el botón de Finish se habrá creado la activity con extensión Kotlin (que dará sus funciones) y su layout con extensión XML (que será la interfaz gráfica).



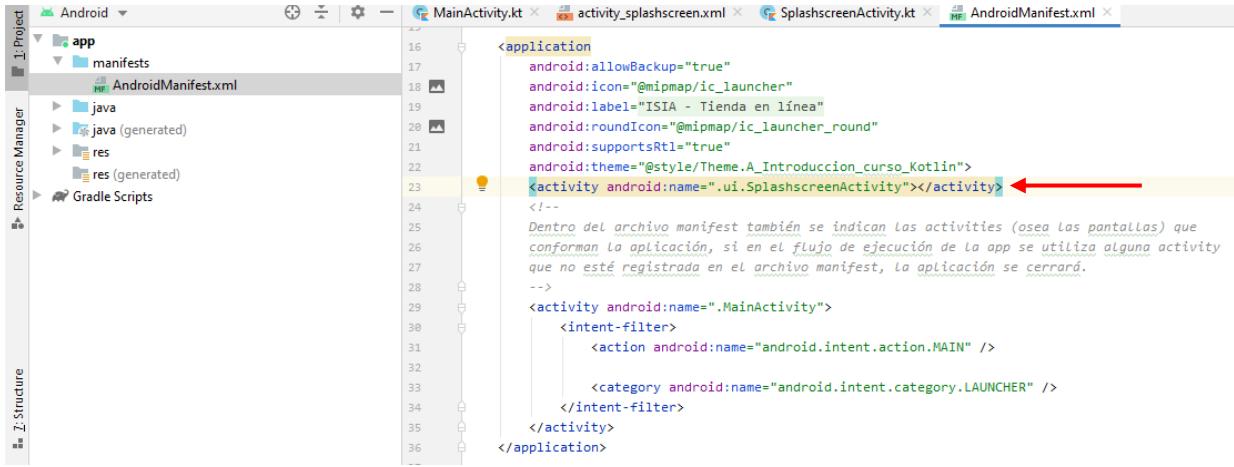
- Ya que hayamos creado la nueva Activity debemos introducirnos al archivo Kotlin e importar el paquete que incluya a su layout dando clic en la R (de Resources) del método **setContentView()** y presionando las teclas **ALT + ENTER**.
  - **setContentView():** Método que conecta al archivo Kotlin con su layout correspondiente, esto lo hace indicando su ubicación dentro de la carpeta resources (src o R), layout y el nombre del layout.



Otra manera de importar manualmente el paquete es poniendo la palabra reservada **import**, seguida del dominio del proyecto y la letra R, con eso se importa el contenido de toda la carpeta de Resources, que contiene todos los recursos del proyecto, ya sean colores, layouts, imágenes, etc.



- Por último, debemos checar en el archivo manifest que se haya incluido una etiqueta con el nombre de la nueva actividad, ya que si no es el caso cuando se quiera acceder a esta actividad, la aplicación se cerrará, todas las actividades existentes deben ser mencionadas en el archivo manifest.

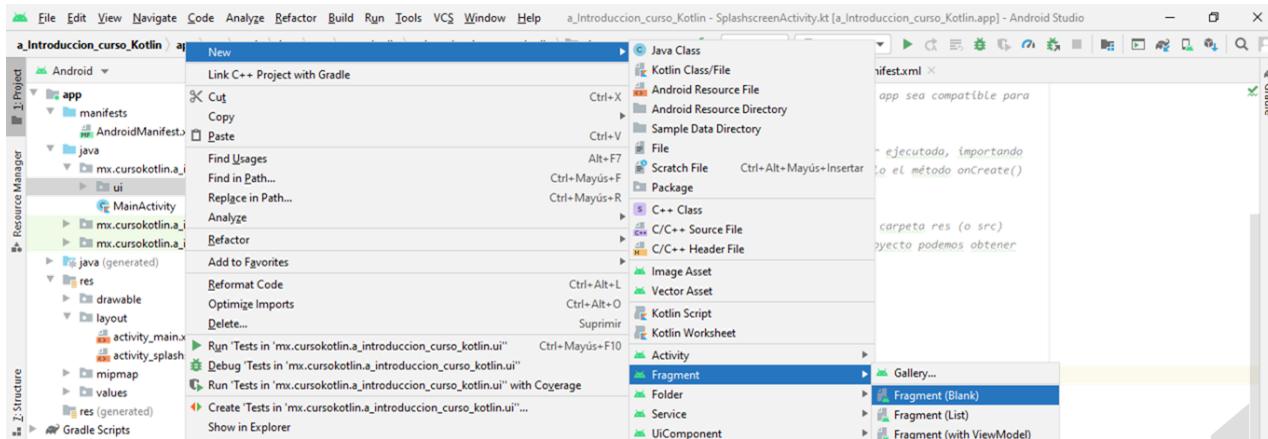


```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="ISIA - Tienda en línea"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme_A_Introduccion_curso_Kotlin">
    <activity android:name=".ui.SplashscreenActivity"></activity> ----->
    <!--
        Dentro del archivo manifest también se indican las activities (osea las pantallas) que
        conforman la aplicación, si en el flujo de ejecución de la app se utiliza alguna activity
        que no esté registrada en el archivo manifest, la aplicación se cerrará.
    -->
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

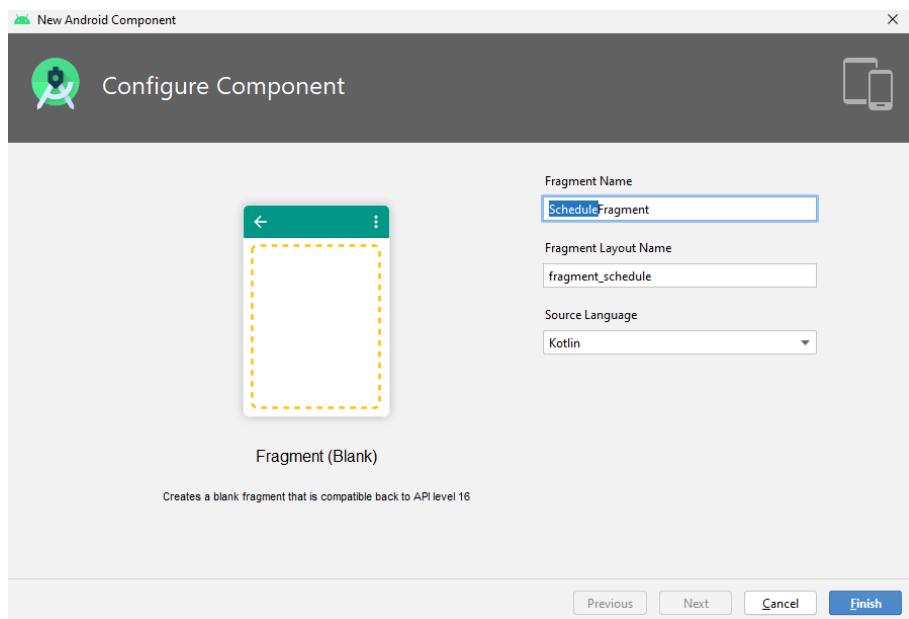
```

**•Creación de un Fragment:** En `app/java/com.nombreDominio.nombreProyecto` se crean los fragmentos dentro del mismo paquete ui creado para almacenar todas las actividades, para ello daremos clic derecho en el paquete y seleccionaremos la opción de New → Fragment → Fragment (Blank) → Fragment Name: → Poner un nombre que empiece con mayúscula y tenga la palabra Fragment al final → Finish.

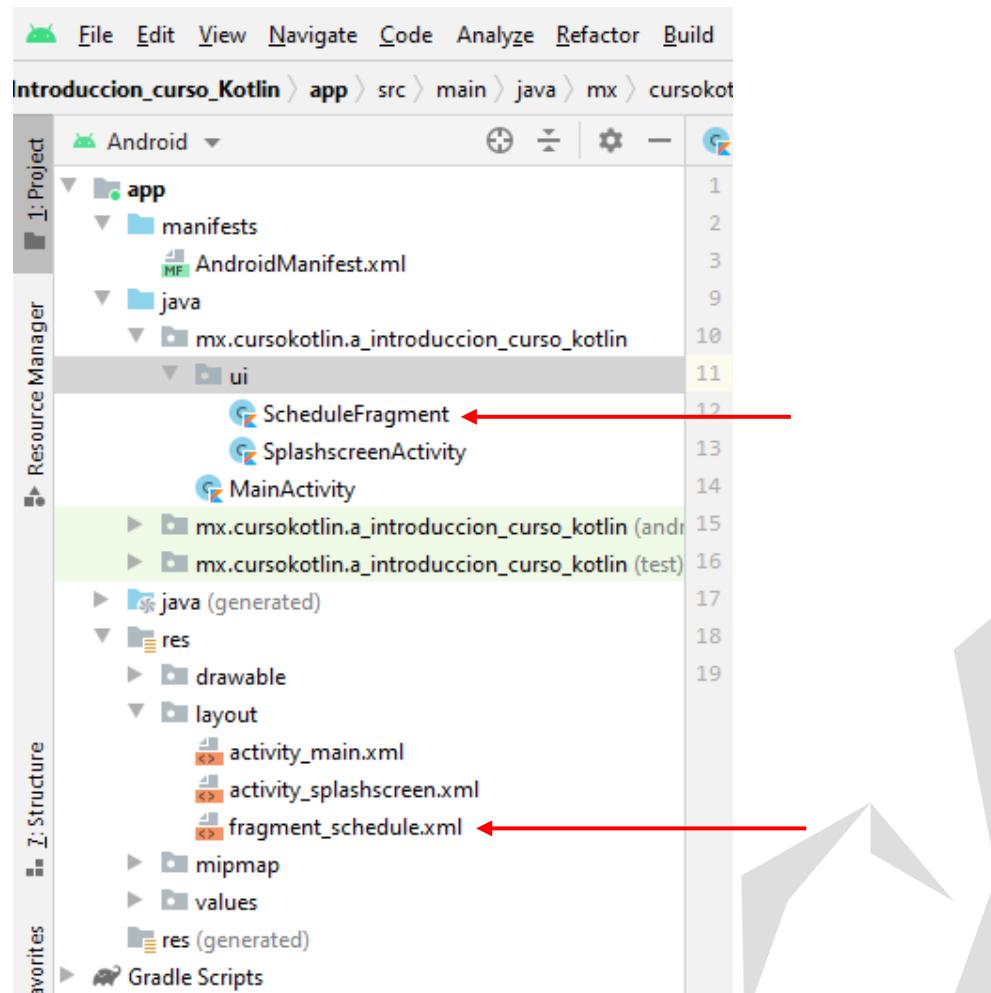


*Nota: Cuando se indique el nombre del nuevo Fragment creado, por default se creará un layout que tenga el mismo nombre, siguiendo las buenas prácticas, por lo que en el nombre del layout la palabra fragment se pondrá al inicio.*

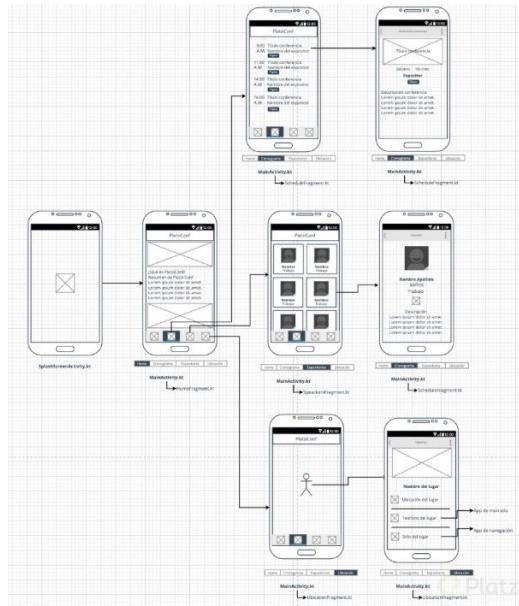
Además, se indica el lenguaje de programación (Java o Kotlin) que se está utilizando en él.



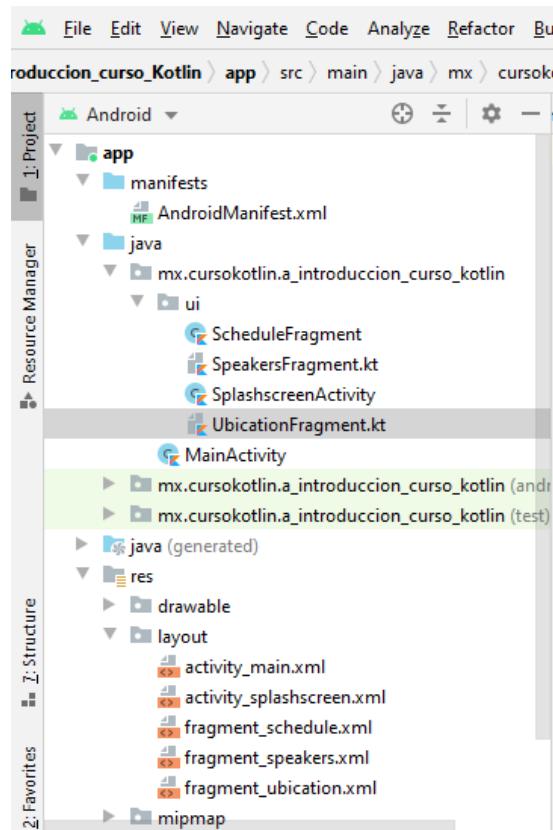
Al dar clic en el botón de Finish se habrá creado el fragment con extensión Kotlin (que dará sus funciones) y su layout con extensión XML (que será la interfaz gráfica).



- Se debe repetir este proceso con los Fragmentos que faltan y están indicados en la maquetación de pantallas, en donde se observa que solo hay 2 actividades y las otras 6 pantallas son DialogFragments.



- Ya creados los 3 dialog fragments que son: ScheduleFragment, SpeakersFragment y UbcationFragment con todo y sus respectivos layouts, la carpeta del proyecto se vería de la siguiente manera:



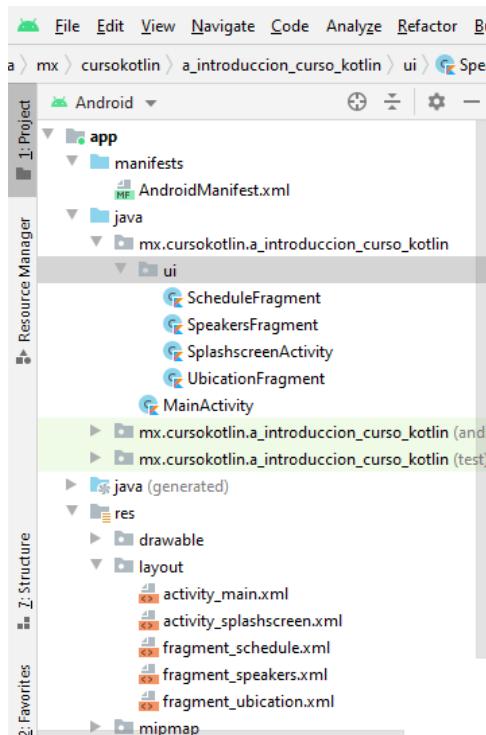
*Nota: Para que los fragmentos estén configurados desde cero como se busca, se deben borrar las constantes y métodos que están ahí por default para dejar el código de la siguiente manera:*

```

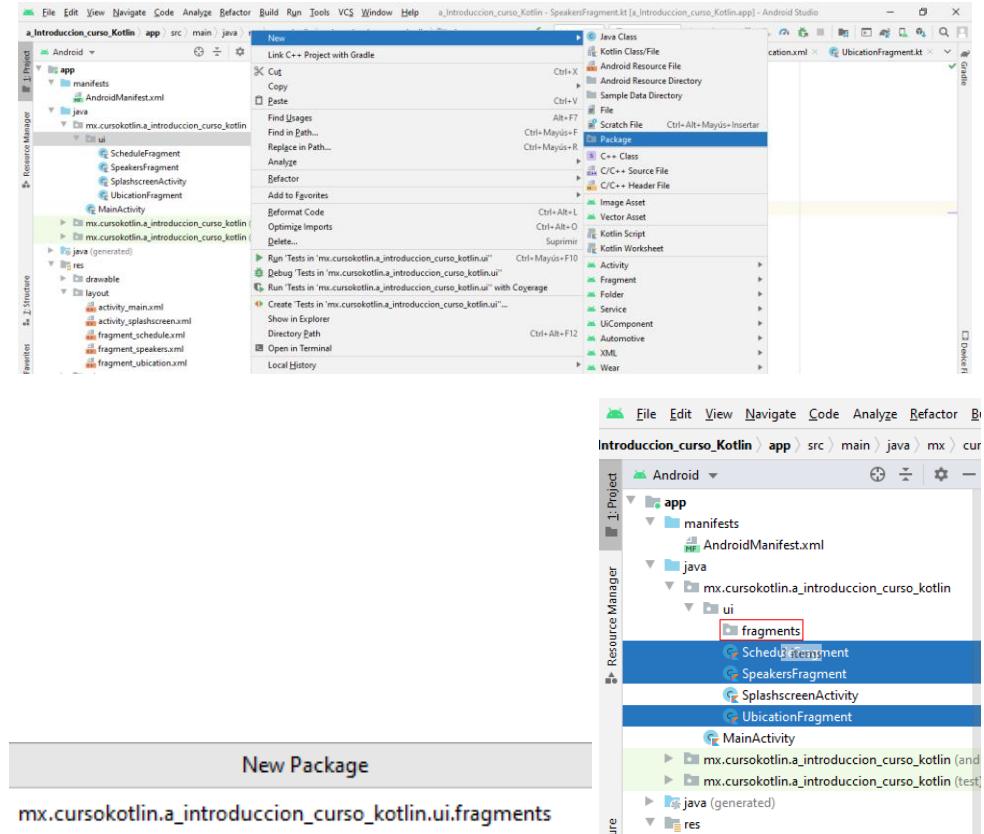
1 package mx.curso kotlin.a_introduccion_curso_kotlin.ui
2
3 import ...
4
5 // TODO: Rename parameter arguments, choose names that match
6 // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
7 private const val ARG_PARAM1 = "param1"
8 private const val ARG_PARAM2 = "param2"
9
10 /**
11  * A simple [Fragment] subclass.
12  * Use the [UbicationFragment.newInstance] factory method to
13  * create an instance of this fragment.
14  */
15 class UbicationFragment : Fragment() {
16     // TODO: Rename and change types of parameters
17     private var param1: String? = null
18     private var param2: String? = null
19
20     override fun onCreate(savedInstanceState: Bundle?) {
21         super.onCreate(savedInstanceState)
22         arguments?.let {
23             param1 = it.getString(ARG_PARAM1)
24         }
25     }
26
27     override fun onCreateView(
28         inflater: LayoutInflater, container: ViewGroup?,
29         savedInstanceState: Bundle?
30     ): View? {
31         // Inflate the layout for this fragment
32         return inflater.inflate(R.layout.fragment_ubicacion, container, false)
33     }
34 }

```

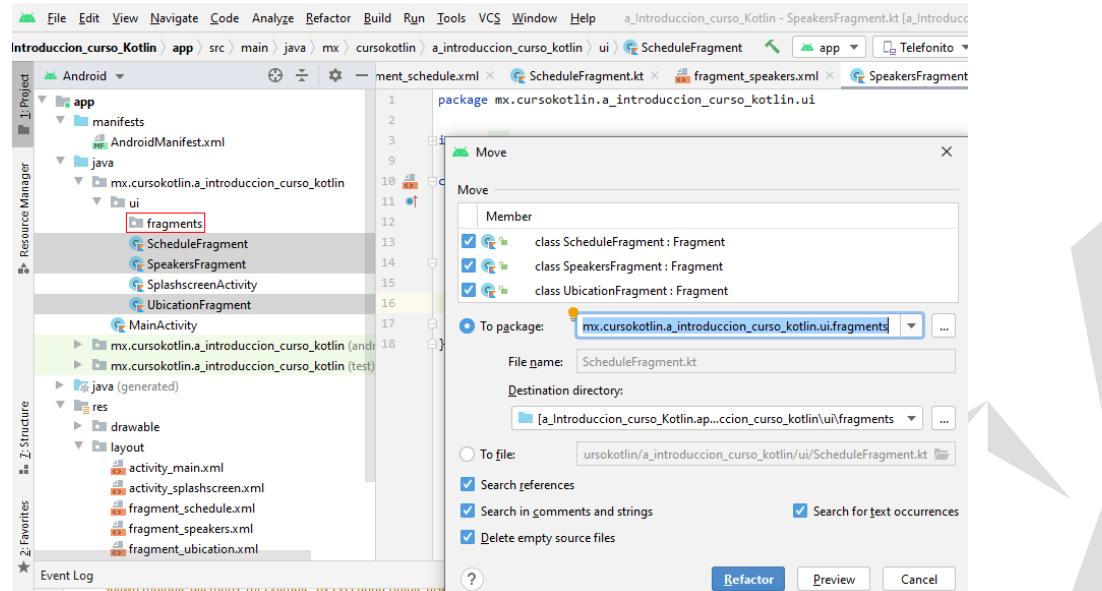
*Al hacerlo podremos ver que hasta su ícono cambia en las carpetas del proyecto, dejando al final todas las carpetas de la siguiente manera:*



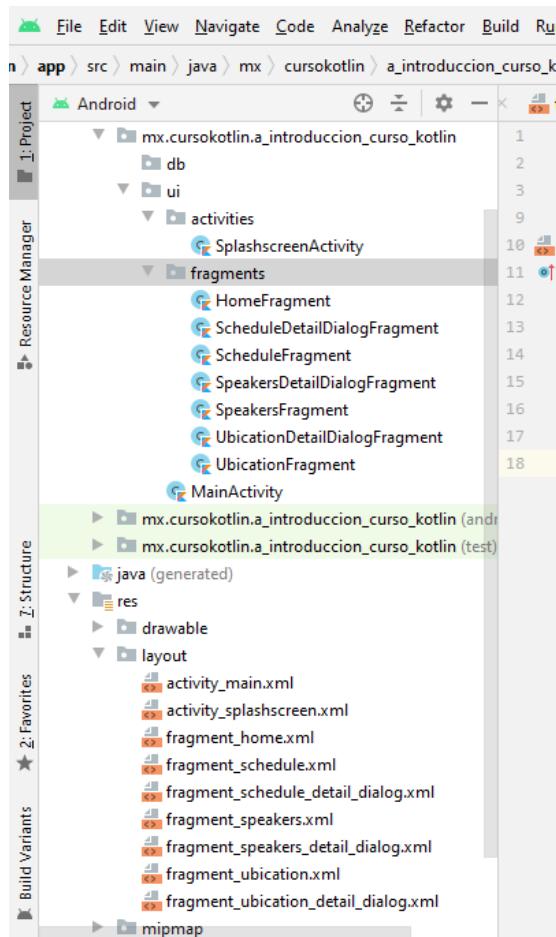
- Dentro del paquete ui se suele crear un nuevo paquete llamado fragments en donde se coloquen todos los fragmentos creados en el paquete, esto se hace dando clic derecho en el paquete ui y seleccionando la opción de New → Package.



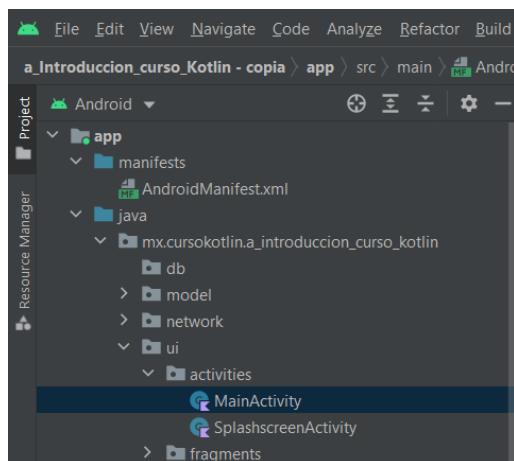
Luego lo que haremos es agarrar y arrastrar los archivos dentro del nuevo paquete y daremos clic en el botón de Refactor (se nos había olvidado crear el fragmento de Home, pero lo creamos directo en el paquete fragments).



Después de agrupar todos los fragmentos que representan una parte distinta de la aplicación en un paquete llamado fragments, se repite el mismo proceso, pero ahora creando el paquete activities para agrupar las Actividades del proyecto, y al final las carpetas quedan de la siguiente forma:



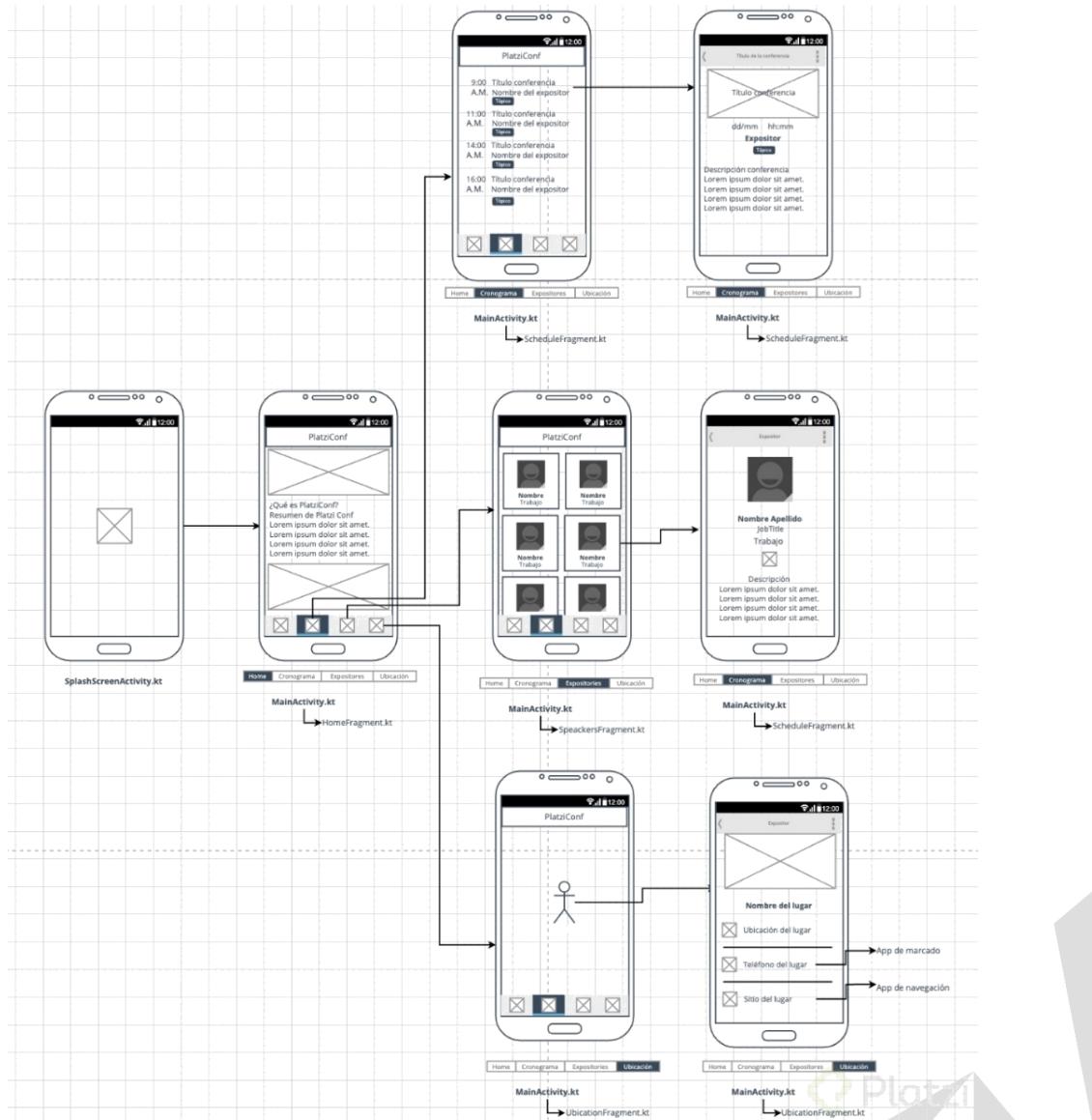
*Nota: Hasta que no haya otro paquete fuera de ui no se puede meter la MainActivity al paquete de activities, sino lo que pasará es que se fusionarán los paquetes principales y el ui, para corregir ese error se debe crear otro paquete vacío fuera de ui o para evitar el error no se debe meter la actividad MainActivity dentro del paquete ui hasta que haya otro paquete en el mismo nivel de ui.*



# UI - Diseño de la interfaz

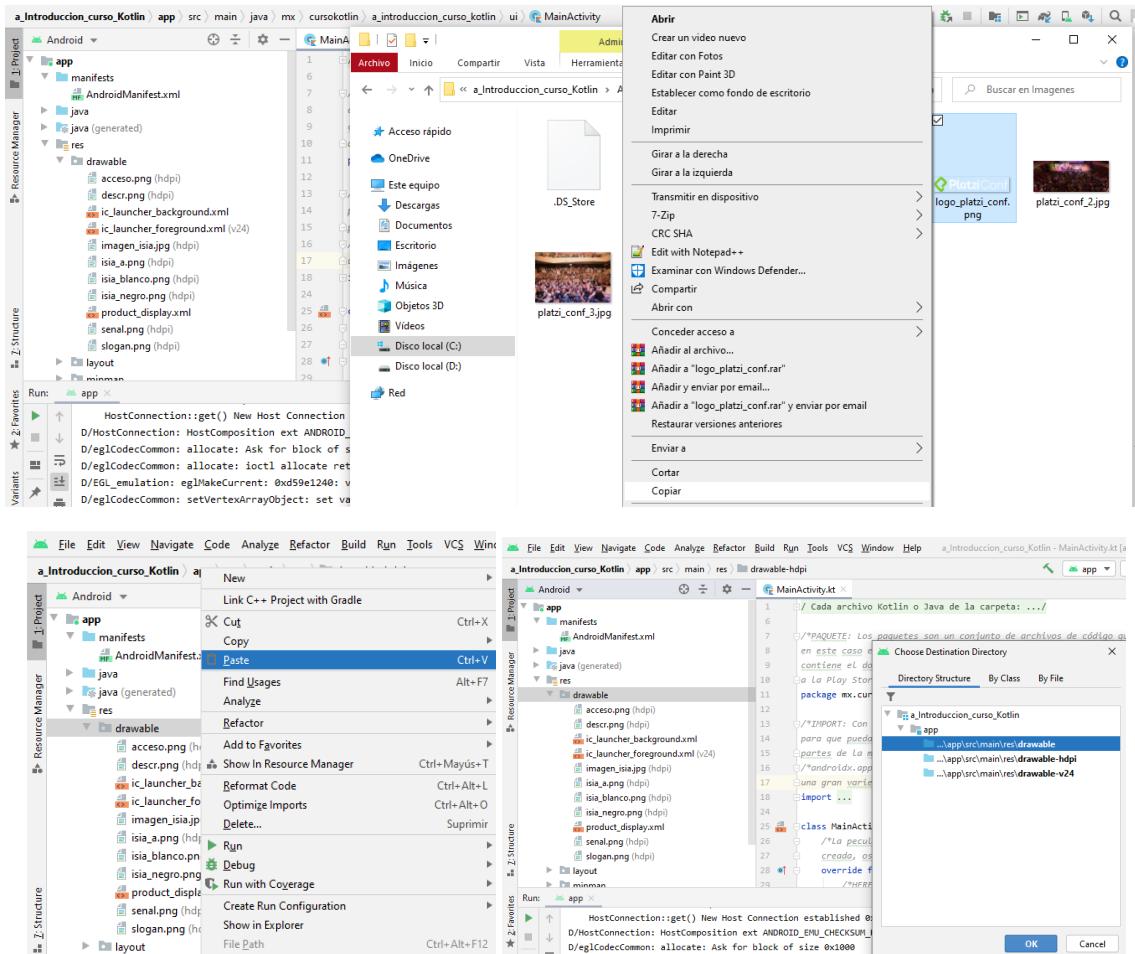
Antes de unificar las Actividades y DialogFragments (fragmentos que se hacen pasar por actividades), es necesario diseñar cada una y luego unirlas, para ello se hace uso otra vez de la maquetación de pantallas. Una vez que se hayan declarado ya todos los fragmentos y actividades que se van a utilizar en la aplicación, estas se unirán por medio del navgraph y bottom navigation menu.

Para realizar el diseño de los layouts (actividades o fragmentos) se debe contar ya con los logos, íconos y demás imágenes que se quiera utilizar, recordemos que las imágenes y demás elementos estéticos se deben colocar en la carpeta de **app/res/drawable**, esto se puede hacer copiando y pegando la imagen directo en Android Studio o simplemente arrastrándola a la carpeta donde corresponde.

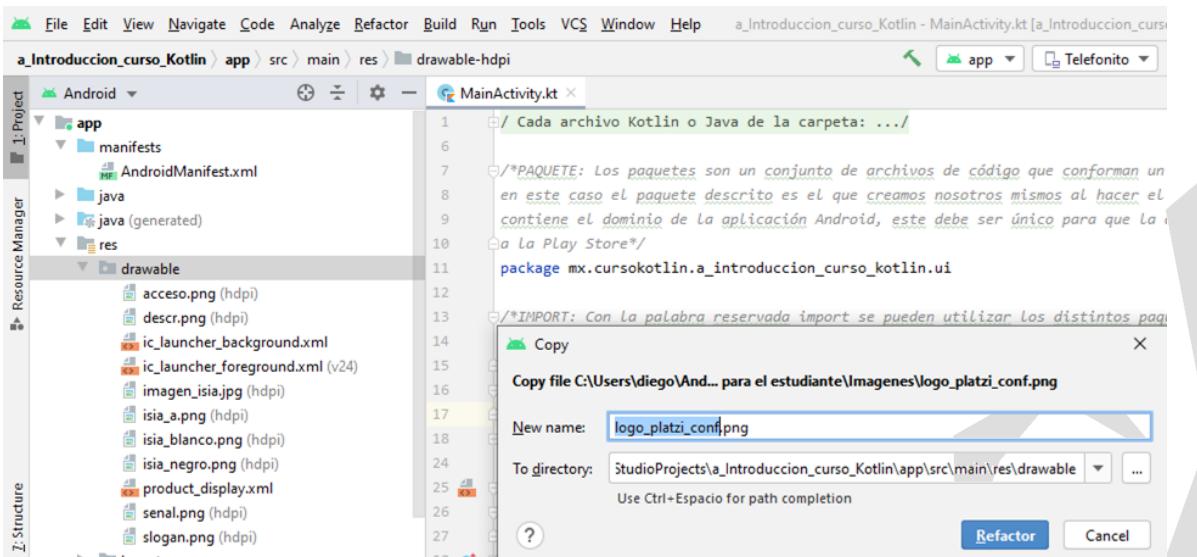


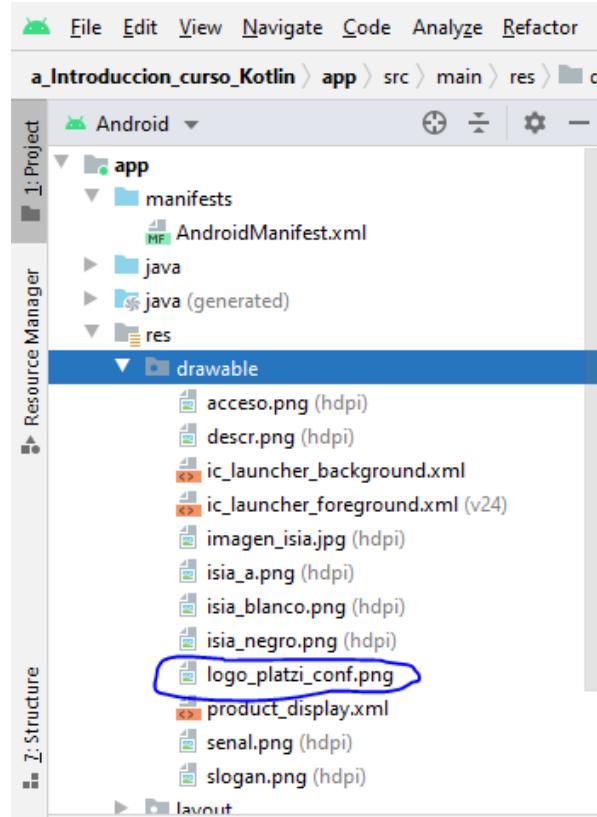
Cuando se quiera copiar y pegar una imagen en la carpeta **app/res/drawable** del proyecto, Android Studio nos dará la opción de colocarla en dos posibles carpetas, debemos colocar la imagen en la primera opción que se muestra, ya que la segunda y tercera opción que son las carpetas **drawable-hdpi**

y drawable-v24 se utilizan para cuando se quiera que la aplicación sea soportada en múltiples dispositivos con distintos tamaños de pantalla.



Dentro de Android Studio todos los nombres de imágenes deben estar en minúsculas y con palabras separadas por un guión bajo.





## Diseño de Activities:

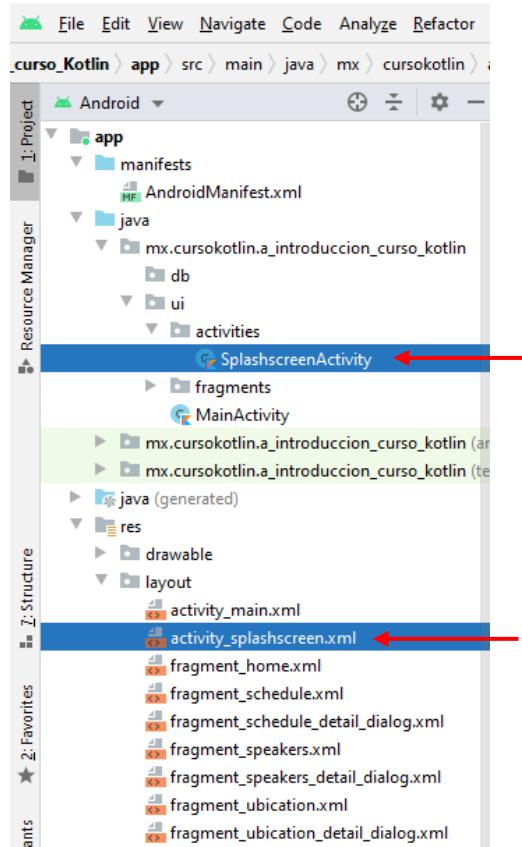
1.- Debemos dar clic en el archivo xml que represente al layout de la Actividad que queramos editar, en este caso es la actividad llamada **SplashScreenActivity.kt** con su layout correspondiente que es **activity\_splashscreen.xml**. Es la primera pantalla de la aplicación, de acuerdo con la maquetación de pantallas de ejemplo:



En la carpeta layout es donde podemos diseñar la interfaz de cada pantalla (Activity).

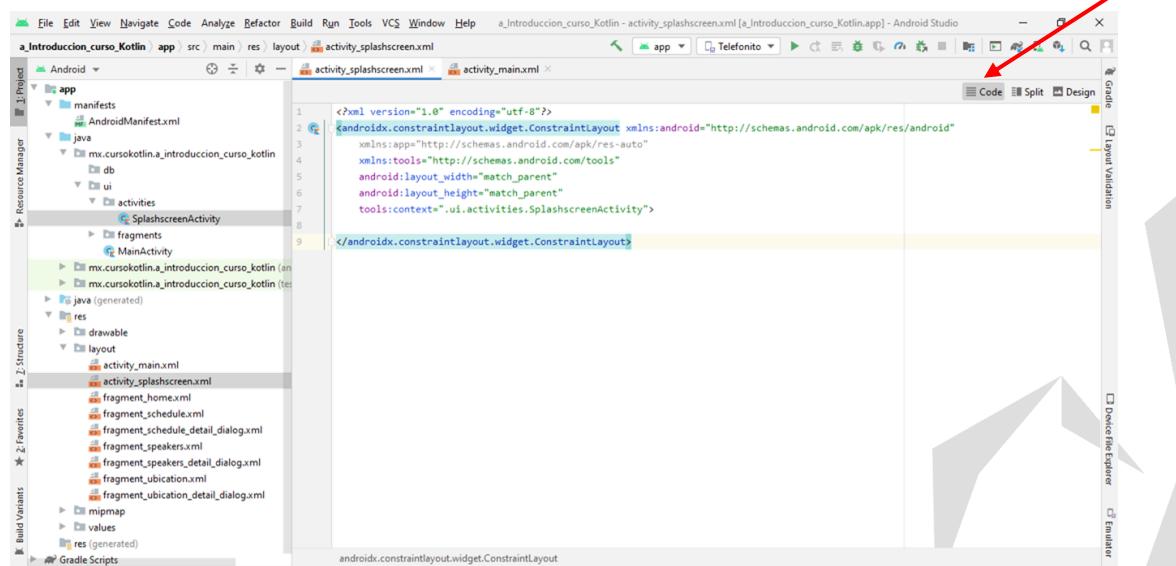
**app → java → res → layout:** En la carpeta **app/java/res/layout** se crean todos los archivos XML que representen la interfaz gráfica de todas las actividades o fragmentos, aquí es donde se editan los aspectos visuales de cada interfaz gráfica del proyecto, todas ellas deben estar ligadas a algún archivo hecho con Kotlin o Java que le dé sus funcionalidades.

Aunque XML es un lenguaje de marcado que principalmente sirve para la trasmisión de datos, en Android se utiliza para colocar elementos gráficos como botones, áreas de texto, etc. llamados **Views**.

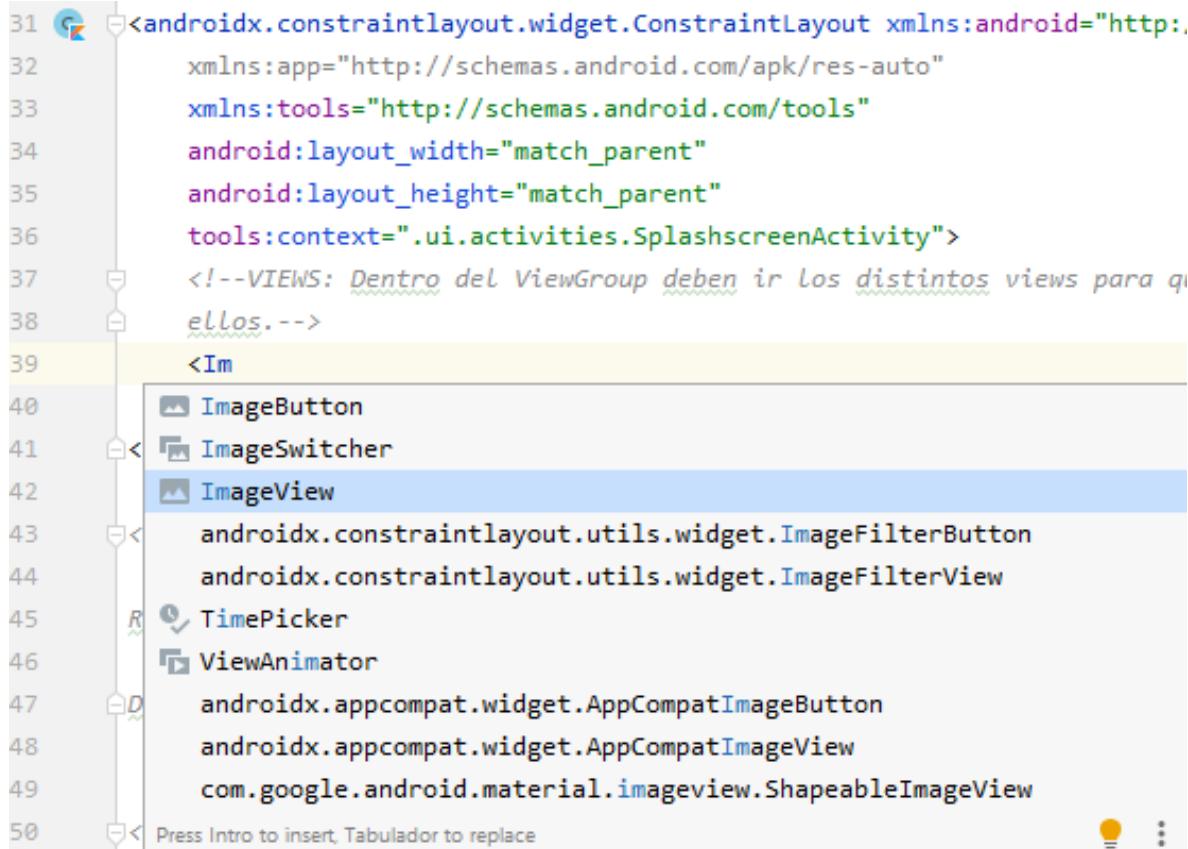


Para ver el código XML de los elementos gráficos que se están creando en la interfaz se selecciona el botón **Code** que se encuentra en la esquina superior derecha, para ver la interfaz de diseño de Android Studio se selecciona el botón **Design** y para ver ambos se da clic en el botón **Split**.

- **Code:** Muestra el código XML de la interfaz:



- De esta manera se deben integrar todos los elementos gráficos por medio de las etiquetas XML de los distintos Views en forma de código.
- Para colocar los Views por medio de código es de gran utilidad hacer uso del apoyo que da Android Studio para completar el código escrito.



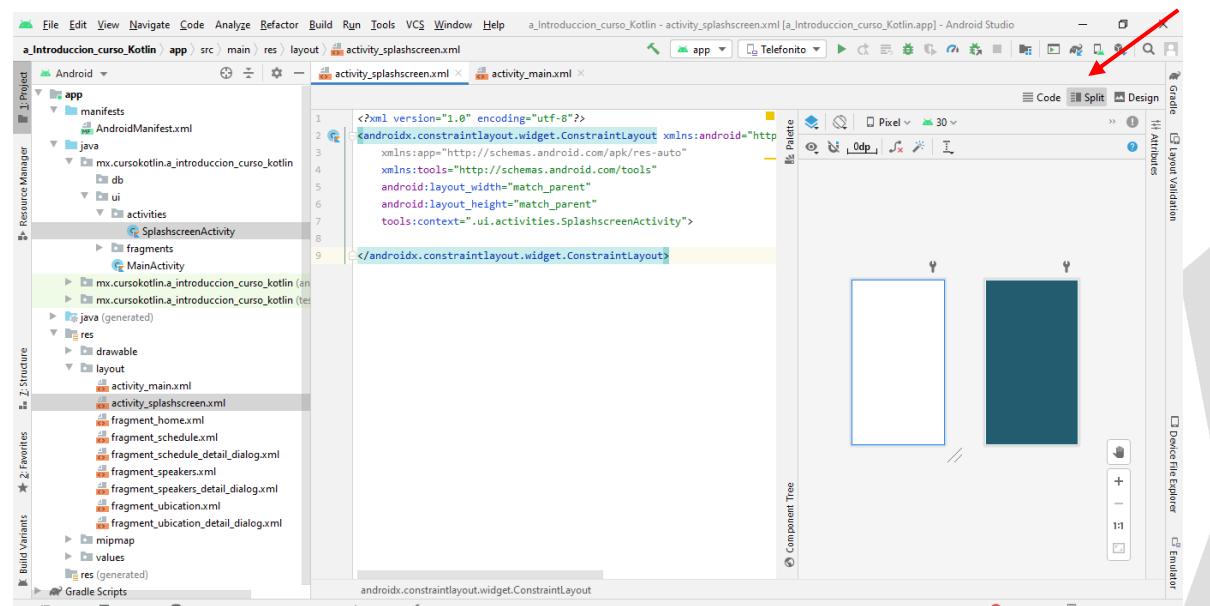
The screenshot shows the Android Studio interface with the code editor open. The cursor is positioned at the start of an `<Im` tag. A code completion dropdown menu is displayed, listing various view types: `ImageButton`, `ImageSwitcher`, `ImageView`, `androidx.constraintlayout.utils.widget.ImageFilterButton`, `androidx.constraintlayout.utils.widget.ImageFilterView`, `TimePicker`, `ViewAnimator`, `androidx.appcompat.widget.AppCompatImageButton`, `androidx.appcompat.widget.AppCompatImageView`, and `com.google.android.material.imageview.ShapeableImageView`. The `ImageView` option is highlighted.

```

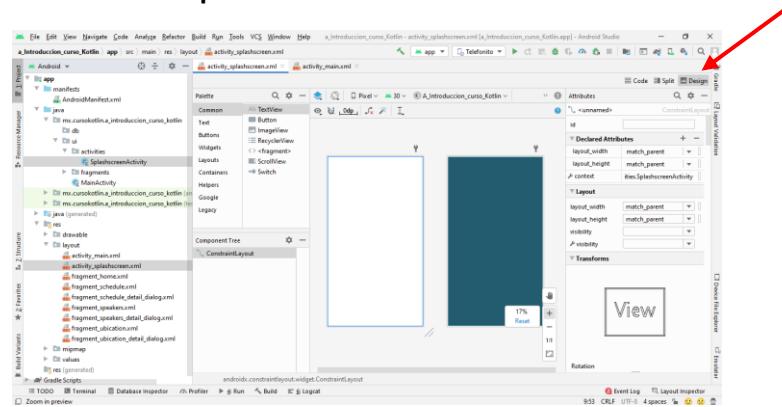
31 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
32     xmlns:app="http://schemas.android.com/apk/res-auto"
33     xmlns:tools="http://schemas.android.com/tools"
34     android:layout_width="match_parent"
35     android:layout_height="match_parent"
36     tools:context=".ui.activities.SplashscreenActivity">
37     <!--VIEWS: Dentro del ViewGroup deben ir los distintos views para que ellos.-->
38     <Im
39         <ImageButton
40         <ImageSwitcher
41         <ImageView
42             <androidx.constraintlayout.utils.widget.ImageFilterButton
43             <androidx.constraintlayout.utils.widget.ImageFilterView
44             <TimePicker
45             <ViewAnimator
46             <androidx.appcompat.widget.AppCompatImageButton
47             <androidx.appcompat.widget.AppCompatImageView
48             <com.google.android.material.imageview.ShapeableImageView
49
50     Press Intro to insert, Tabulator to replace

```

- **Split:** Muestra tanto el código como la interfaz gráfica que se está creando:

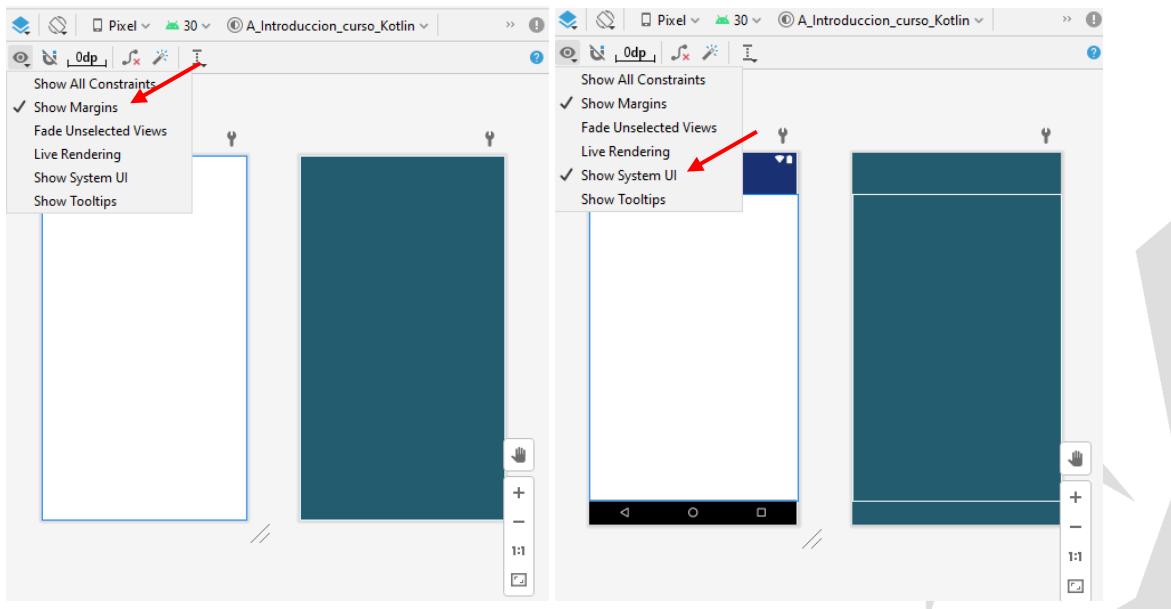


- **Design:** En el área de diseño hay dos ventanas, a la blanca se le llama Design y a la azul se le llama Blueprint y ambas muestran el acomodo de los distintos componentes que conforman la interfaz de la aplicación:
  - **Design:** Muestra los elementos visibles y como se van a ver en la aplicación en todo momento.
  - **Blueprint:** Muestra los componentes visibles e invisibles que conforman la interfaz, ya sea porque se pueden ver en la interfaz de la aplicación en todo momento, porque aparecen momentáneamente o porque son invisibles.
    - Cada uno de los componentes colocados en la interfaz de la aplicación es llamado **View** y al contenedor que puede incluir varios Views se le llama **ViewGroup**.

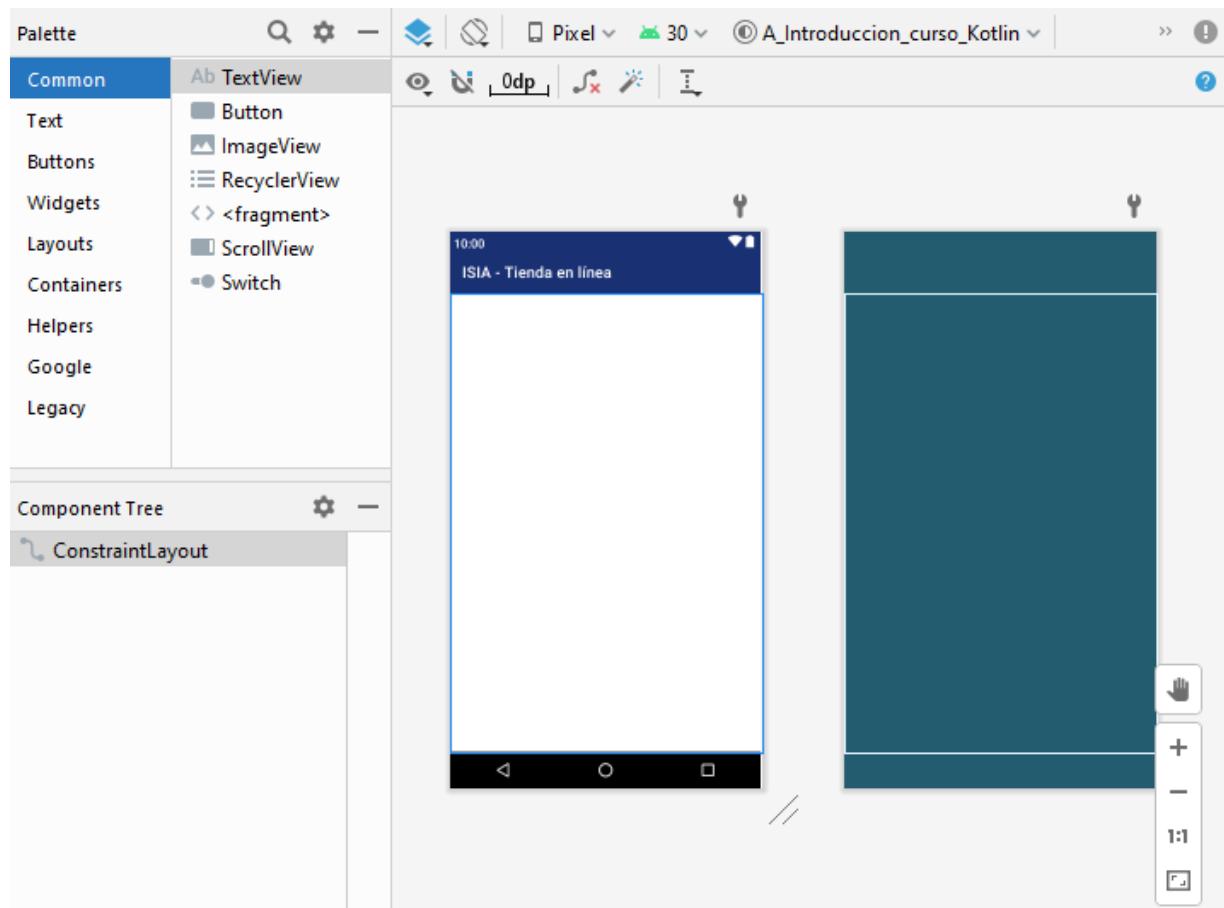


**La ventana de Design se compone de varias partes que muestran distintos elementos y características estéticas o funcionales de los Views en la interfaz como:**

- El ojo que se encuentra encima de la interfaz permite ver distintos aspectos de la interfaz:
  - **Show System UI:** Seleccionar esta opción permite observar elementos que se verán en el celular real o en un AVD ya que sea corrida la interfaz, como la barra superior de la aplicación con su título, y los botones inferiores.



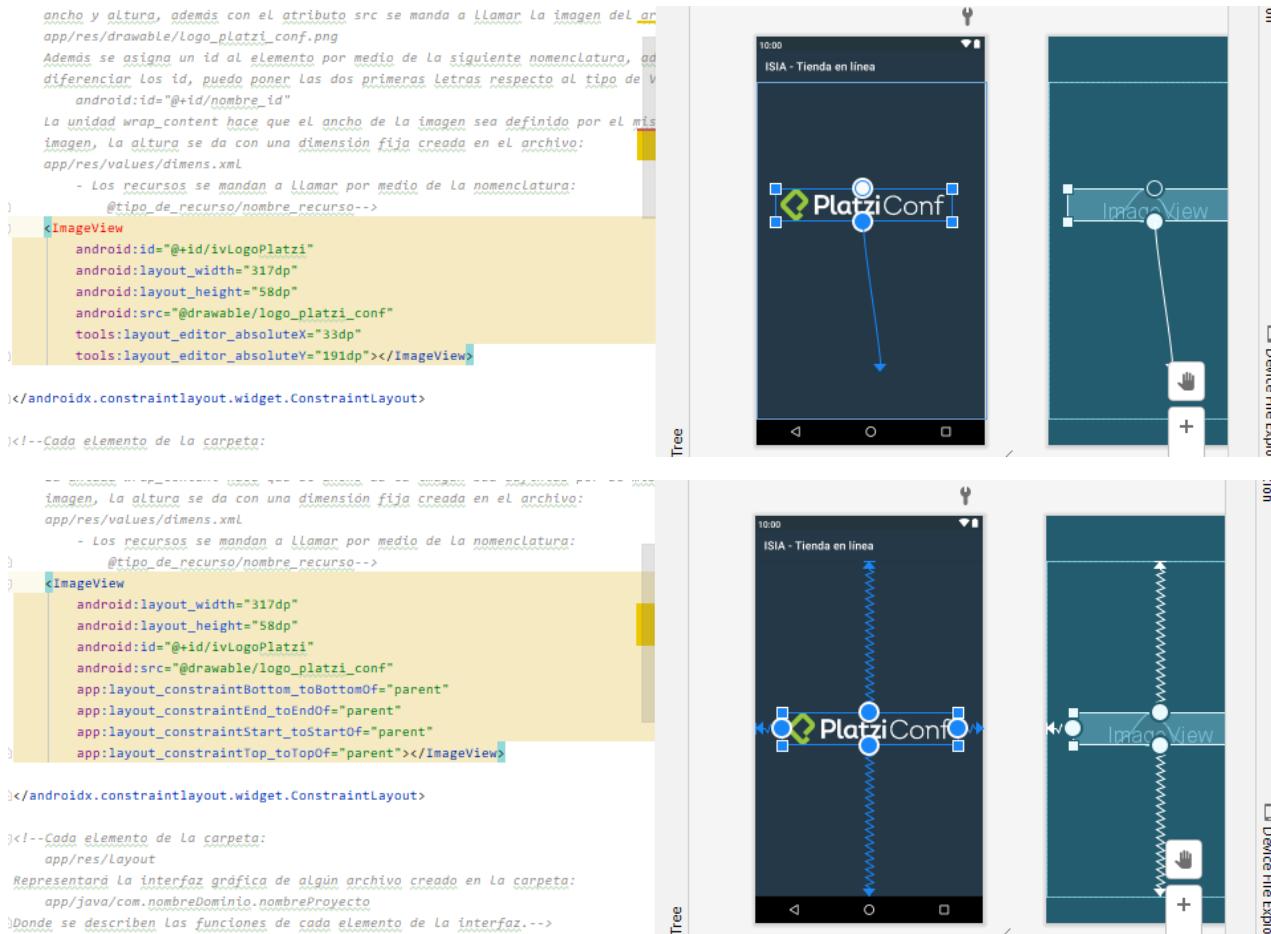
- En la esquina superior izquierda donde dice Palette se muestran las categorías de todos los views que podemos arrastrar a la interfaz y así agregarla:
  - **TextView:** Es una caja de texto, su texto se introduce como atributo y se jala del archivo values en donde es declarado como string.
  - **Button:** Es un botón.
  - **ImageView:** Es una imagen.
  - **RecyclerView:** Es un View que permite crear listas o rejillas de los elementos obtenidos de la base de datos.
  - **Fragment:** Es un contenedor que puede cambiar su contenido de forma dinámica.
  - **ScrollView:** Es un contenedor que permite al usuario ver su contenido haciendo scroll (subiendo o bajando en la pantalla de la aplicación), se usa cuando el contenido de la pantalla es mucho.
  - **Switch:** Interruptor digital que da como respuesta un booleano true o false.



- Ya habiendo agregado un View al ViewGroup que por defecto que se crea en las actividades llamado **Constraint Layout**, se puede colocar donde sea el elemento referenciándolo respecto a algún otro para que se mantenga fijo y el diseño sea responsivo, esto se hace a través de 4 nodos y respecto a ellos se crearán distintos atributos de código en el View dependiendo de la forma en la que se fije el elemento y respecto a qué se estén fijando:
  - **Constraint Layout:** Esta etiqueta XML se utiliza para crear diseños planos, no permitiendo que exista mucha anidación dentro del contenedor, es el contenedor que

se crea por default en las Actividades y su peculiaridad es que permite aplicar una serie de atributos a los View que tenga dentro para hacerlo responsive, enlistados a continuación:

- **layout\_constraintBottom\_toBottomOf:** Hace que el **nodo inferior del View** se coloque sobre **la parte inferior de otro elemento**, ya sea su elemento padre o el elemento que tenga un id en específico.
- **layout\_constraintTop\_toTopOf:** Hace que el **nodo superior del View** se coloque en **la parte superior de otro elemento**, ya sea su elemento padre o el elemento que tenga un id en específico.
- **layout\_constraintEnd\_toEndOf:** Hace que el **nodo derecho del View** se coloque sobre **la parte derecha de otro elemento**, ya sea su elemento padre o el elemento que tenga un id en específico.
- **layout\_constraintStart\_toStartOf:** Hace que el **nodo izquierdo del View** se coloque en **la parte izquierda de otro elemento**, ya sea su elemento padre o el elemento que tenga un id en específico.
- **Nota:** Los atributos se crearán por sí solos cuando fijemos el View por medio de la ventana de Design.



```

ancho y altura, además con el atributo src se manda a llamar la imagen del archivo
app/res/drawable/logo_platzi.conf.png
Además se asigna un id al elemento por medio de la siguiente nomenclatura, ad
diferenciar los id, puedo poner las dos primeras letras respecto al tipo de V
        android:id="@+id/nombre_id"
La unidad wrap_content hace que el ancho de la imagen sea definido por el mis
imagen, la altura se da con una dimensión fija creada en el archivo:
app/res/values/dimens.xml
    - Los recursos se mandan a llamar por medio de la nomenclatura:
        @tipo_de_recurso/nombre_recurso-->
)
<ImageView
    android:id="@+id/ivLogoPlatzi"
    android:layout_width="317dp"
    android:layout_height="58dp"
    android:src="@drawable/logo_platzi_conf"
    tools:layout_editor_absoluteX="33dp"
    tools:layout_editor_absoluteY="191dp"></ImageView>
)
</androidx.constraintlayout.widget.ConstraintLayout>

<!-- Cada elemento de la carpeta:
    El ancho y la altura se da con la medida wrap_content, esto es porque la imagen
    la altura se da con una dimensión fija creada en el archivo:
    app/res/values/dimens.xml
    - Los recursos se mandan a llamar por medio de la nomenclatura:
        @tipo_de_recurso/nombre_recurso--&gt;
)
&lt;ImageView
    android:layout_width="317dp"
    android:layout_height="58dp"
    android:id="@+id/ivLogoPlatzi"
    android:src="@drawable/logo_platzi_conf"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"&gt;&lt;/ImageView&gt;
)
&lt;/androidx.constraintlayout.widget.ConstraintLayout&gt;

<!-- Cada elemento de la carpeta:
    app/res/Layout
Representará la interfaz gráfica de algún archivo creado en la carpeta:
app/java/com.nombreDominio.nombreProyecto
Donde se describen las funciones de cada elemento de la interfaz.--&gt;
</pre>

```

Las 4 flechas que se muestran en la ventana de Design son las que fijan el View.

Con eso se termina el diseño de la primera Actividad.

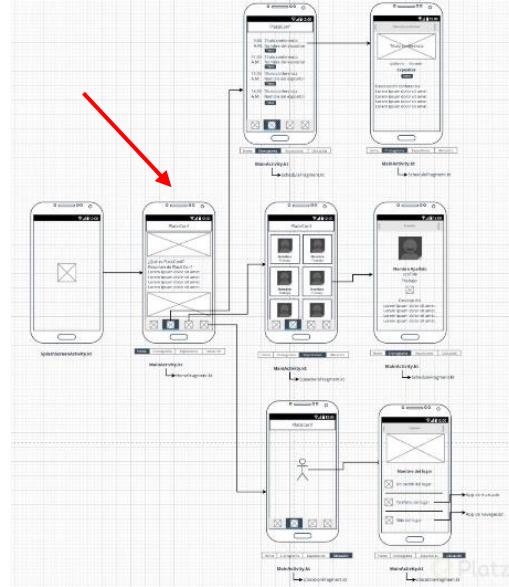
```

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
    xmlns:app="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ui.activities.SplashscreenActivity"
    android:background="@color/dark_1">

    <!--VIEWS: Dentro del ViewGroup deben ir los distintos views para que el usuario los vea-->
    <!--ImageView: Muestra una imagen, sus atributos de width y height se utilizan tanto ancho como altura, además con el atributo src se manda a llamar la imagen del archivo app/res/drawable/logo_platzi_conf.png-->
    Además se asigna un id al elemento por medio de la siguiente nomenclatura, diferenciar los id, puede poner las dos primeras Letras respecto al tipo de recurso android:id="@+id/nombre_id"
    La unidad wrap_content hace que el ancho de la imagen sea definido por el mismo archivo, la altura se da con una dimensión fija creada en el archivo: app/res/values/dimens.xml
    - Los recursos se mandan a llamar por medio de la nomenclatura: @tipo_de_recurso/nombre_recurso-->
<ImageView
    android:layout_width="317dp"
    android:layout_height="58dp"
    android:id="@+id/vlLogoPlatzi"
    android:src="@drawable/logo_platzi_conf"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

2.- Luego se creará la otra activity que contendrá los DialogFragments llamada **MainActivity.kt** con su layout correspondiente que es **activity\_main.xml**, en ella se creará el contenedor de los DialogFragments, la barra superior (toolbar) y el menú inferior:



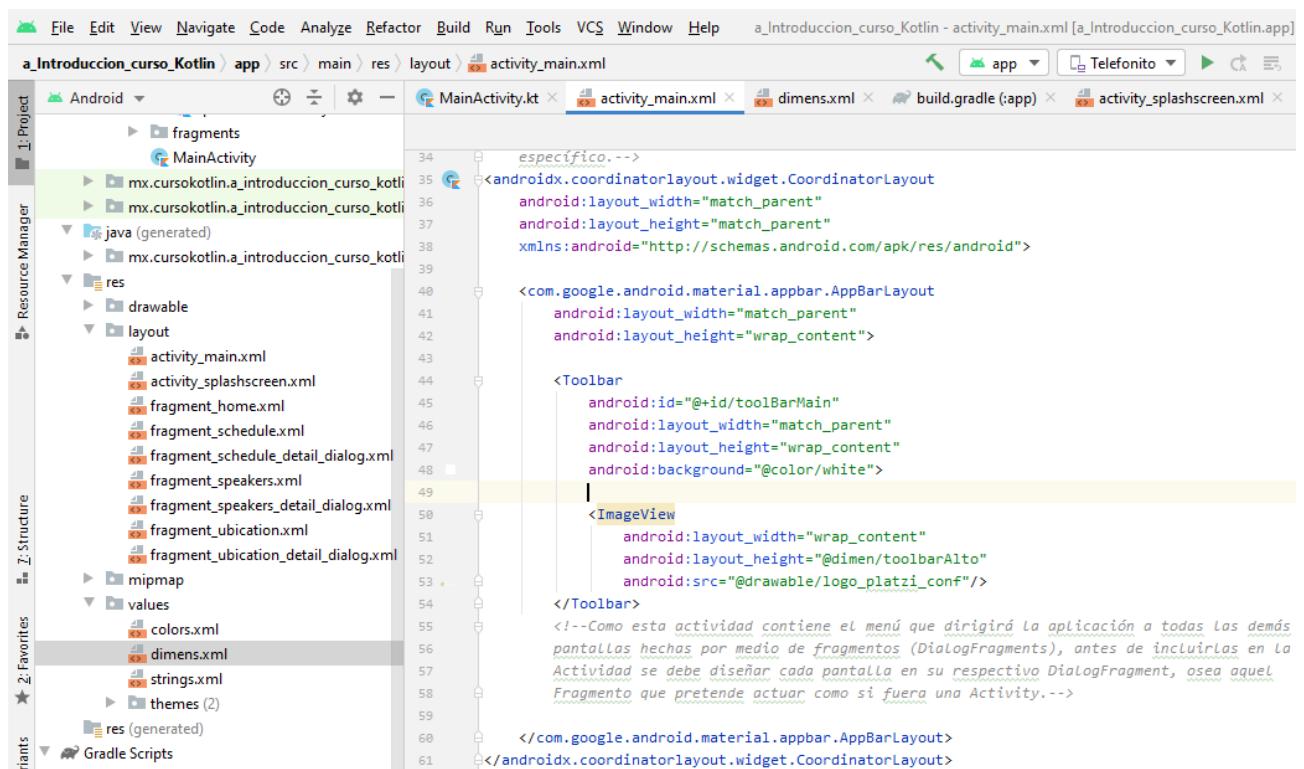
En la carpeta layout es donde podemos diseñar la interfaz de cada pantalla (Activity).

- **Barra superior (Toolbar):** Para poder modificar la barra superior de la aplicación, que en un inicio tiene el nombre de la aplicación, debo descargar una dependencia en el archivo de build.gradle(Module: NombreProyecto.app), además de la ya incluida por default dentro del gradle , esto para poder utilizar la librería de **Material Design** para colocar la barra superior, se hace por medio del siguiente código:

- Implementation 'com.google.android.material:material:1.2.0-alpha04'

*Nota: Cuando se agregue el código al archivo Graddle se debe dar clic en el botón Sync Now que aparece en la esquina superior derecha para agregar la librería.*

- **Coordinator Layout:** El ViewGroup que utilice esta etiqueta se considerará como el principal, osea el que debería estar al principio del diseño, se utiliza cuando se quiera incluir animaciones, anidar muchos elementos o añadir dinamismo a la pantalla porque da funciones extra a algunos Views y contenedores.
  - **En este caso se utilizará el contenedor Coordinator Layout porque el contenedor Constraint Layout no permite anidar muchos elementos dentro de él y al ser esta la pantalla principal que contendrá muchos elementos es necesario cambiar de contenedor.**



```

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help a_Introduccion_curso_Kotlin - activity_main.xml [a_Introduccion_curso_Kotlin.app]
a_Introduccion_curso_Kotlin / app / src / main / res / layout / activity_main.xml
MainActivity.kt activity_main.xml dimens.xml build.gradle (app) activity_splashscreen.xml
Project Android + MainActivity
Resource Manager Java (generated)
res drawable layout
activity_main.xml activity_splashscreen.xml fragment_home.xml fragment_schedule.xml
fragment_schedule_detail_dialog.xml fragment_speakers.xml fragment_speakers_detail_dialog.xml
fragment_ubicacion.xml fragment_ubicacion_detail_dialog.xml
mipmap values
colors.xml dimens.xml strings.xml themes (2)
res (generated)
Gradle Scripts

```

```

34     específico.-->
35     <androidx.coordinatorlayout.widget.CoordinatorLayout
36         android:layout_width="match_parent"
37         android:layout_height="match_parent"
38         xmlns:android="http://schemas.android.com/apk/res/android">
39
40         <com.google.android.material.appbar.AppBarLayout
41             android:layout_width="match_parent"
42             android:layout_height="wrap_content">
43
44             <Toolbar
45                 android:id="@+id/toolBarMain"
46                 android:layout_width="match_parent"
47                 android:layout_height="wrap_content"
48                 android:background="@color/white">
49
50                 <ImageView
51                     android:layout_width="wrap_content"
52                     android:layout_height="@dimen/toolbarAlto"
53                     android:src="@drawable/logo_platzi_conf"/>
54             </Toolbar>
55             <!--Como esta actividad contiene el menú que dirigirá la aplicación a todas las demás
56             pantallas hechas por medio de fragmentos (DialogFragments), antes de incluirlas en la
57             Actividad se debe diseñar cada pantalla en su respectivo DialogFragment, osea aquel
58             Fragmento que pretende actuar como si fuera una Activity.--&gt;
59
60         &lt;/com.google.android.material.appbar.AppBarLayout&gt;
61     &lt;/androidx.coordinatorlayout.widget.CoordinatorLayout&gt;
</pre>

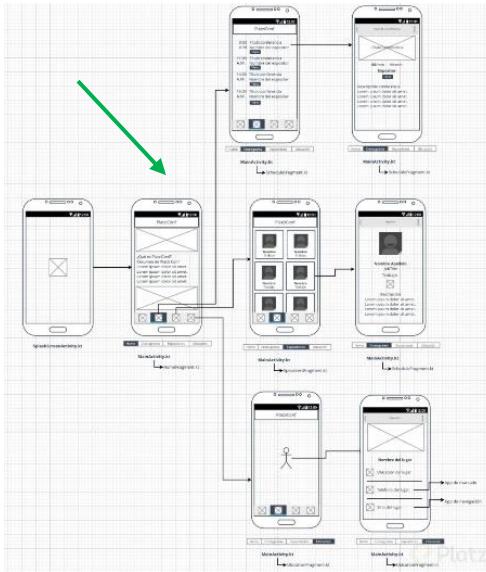
```

Como esta actividad contiene el menú que dirigirá la aplicación a todas las demás pantallas hechas por medio de fragmentos (DialogFragments), antes de incluirlas en la Actividad se debe diseñar cada pantalla en su respectivo DialogFragment, osea aquel Fragmento que pretende actuar como si fuera una Activity. Primero se diseñará el fragmento del Home y al terminar de diseñar todos los dialogFragments se pasará a incluir el bottomNavigationMenu.

## Diseño de DialogFragments:

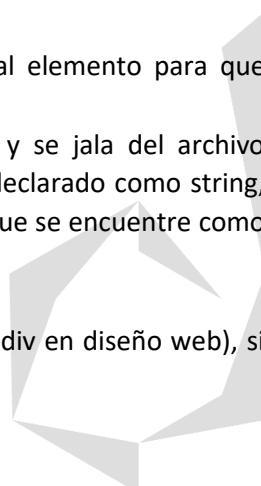
2.1.- Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se empieza con el fragmento llamado **fragment\_home.xml** en donde se utilizará el contenedor ScrollView para mostrar mucho contenido

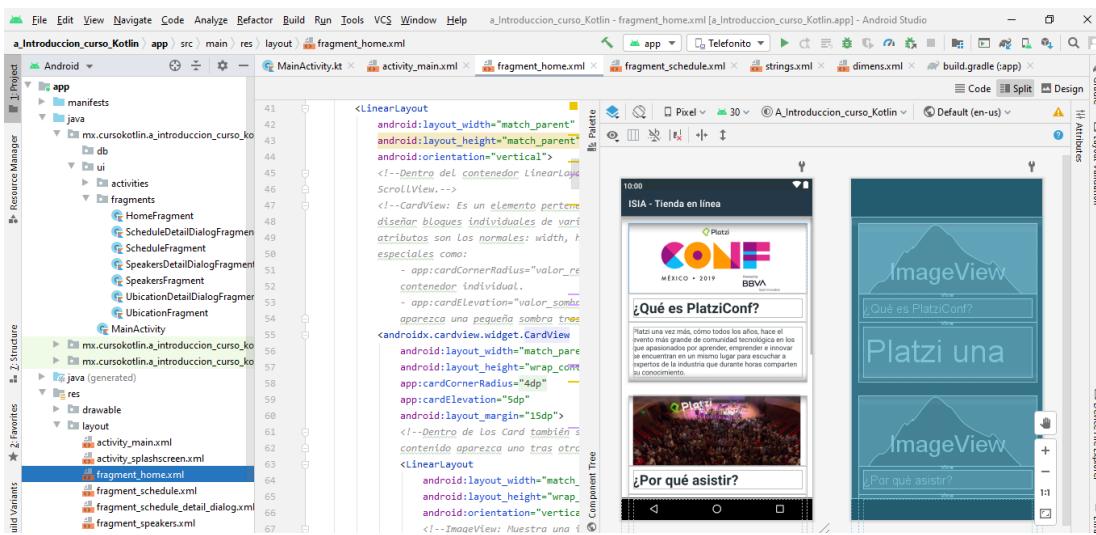
pudiendo hacer scroll en él, LinearLayout para acomodar los elementos linealmente en forma vertical u horizontal y CardView para agrupar distintos Views y mostrar información de la aplicación en pantalla:



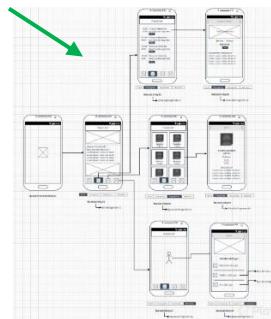
En la carpeta layout es donde podemos diseñar el fragmento que representa cada pantalla de la interfaz (DialogFragment) a la que conduce cada botón del menú.

- **Fragments:** La etiqueta principal para la creación y manejo de fragmentos que se utilizan como DialogFragments es **FrameLayout**.
  - **ScrollView:** Es un contenedor que permite al usuario ver su contenido haciendo scroll (subiendo o bajando en la pantalla de la aplicación), se usa cuando el contenido de la pantalla es mucho, pero como el ScrollView no puede contener más de un componente, se opta por poner un contenedor (ViewGroup) en su interior que contenga otros contenedores para mostrar su contenido.
    - **Linear Layout:** Contenedor que ordena sus componentes de forma lineal, vertical u horizontalmente, uno arriba del otro o uno alado del otro y es muy usado dentro de las etiquetas ScrollView ya que permite anidación y usar más de un elemento.
  - **CardView:** Es un elemento perteneciente a la librería Material Design que sirve para colocar elementos individuales dentro de un contenedor, sus atributos son los normales: width, height, background, pero tienen algunos especiales como:
    - **app:cardCornerRadius="valor\_redondeo":** Se utiliza para indicar el redondeo del contenedor individual.
    - **app:cardElevation="valor\_elevación":** Da una elevación al elemento para que aparezca una pequeña sombra tras él.
  - **TextView:** Es una caja de texto, su texto se introduce como atributo y se jala del archivo strings.xml que se encuentra en la carpeta app/res/values en donde es declarado como string, esto para que pueda ser traducido automáticamente a cualquier idioma que se encuentre como principal en el celular donde se abra la app.
  - **ImageView:** Es una imagen obtenida de la carpeta app/res/drawable.
  - **View:** Es un contenedor donde se puede poner lo que sea (semejante al div en diseño web), si está vacío puede servir para crear una línea de separación.





2.2.1.- Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se sigue con el fragmento llamado **fragment\_schedule.xml** en donde se utilizará el contenedor Relative Layout para mostrar una pantalla de carga (loader o progress bar) y el elemento RecyclerView para mostrar una lista de elementos, cuyo diseño se creará en un archivo XML aparte que representará el aspecto visual de todos los ítems:



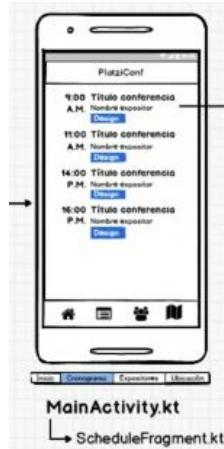
**Todas las pantallas de la serie de elementos que contienen un listado o rejilla hecha con RecyclerView necesitan 3 distintos layout:**

- El principal (2.2.1), donde es declarada la etiqueta **RecyclerView** y se puede poner una pantalla de carga con el elemento **Progress bar**, en este caso el archivo es: **fragment\_schedule.xml**
- Uno que indique **el estilo de los ítems para un listado o rejilla (grid)** (2.2.2), en este caso el archivo es: **item\_schedule.xml**
- El último que **muestra texto con los detalles de cada ítem cuando se dé clic en él** (2.2.3), en este caso el archivo es: **fragment\_schedule\_detail\_dialog.xml**

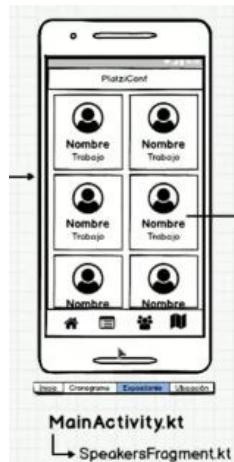
En la carpeta layout es donde podemos diseñar el fragmento de la interfaz de cada pantalla (DialogFragment).

- **Fragments:** La etiqueta principal para la creación y manejo de fragmentos que se utilizan como DialogFragments es **FrameLayout**.
- **Relative Layout:** Ordena los elementos que contenga sobreponiendo uno encima de otro.

- **View:** Es un contenedor donde se puede poner lo que sea (semejante al div en diseño web), si está vacío puede servir para crear una línea de separación.
- **Progress bar:** Es un view que sirve para colocar una pantalla que muestre un loader o progress bar, esto se refiere a una animación o símbolo que aparece mientras está cargando el contenido del fragmento.
- **Listas:** Es un View donde su contenido y el formato en el que se muestra se va a repetir, esto sirve para crear un enlistado de elementos como horarios y descripción de conferencias, precio y descripción de productos, etc.
  - **ListView:** Es un tipo de View muy antiguo que se utilizaba para crear listas en Android, pero en la actualidad se opta por mejor utilizar RecyclerView que también sirve para crear listas.
  - **RecyclerView:** Es un View que permite colocar elementos en forma de lista o rejilla:
    - **Lista (List):** View que coloca un elemento tras otro en forma de párrafo.

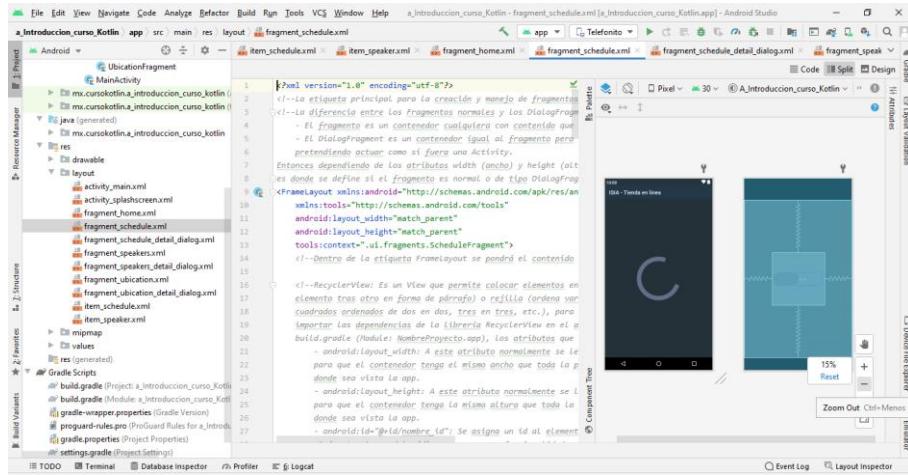


- **Rejilla o grilla (Grid):** View que ordena varios elementos en contenedores cuadrados de dos en dos, tres en tres, etc.



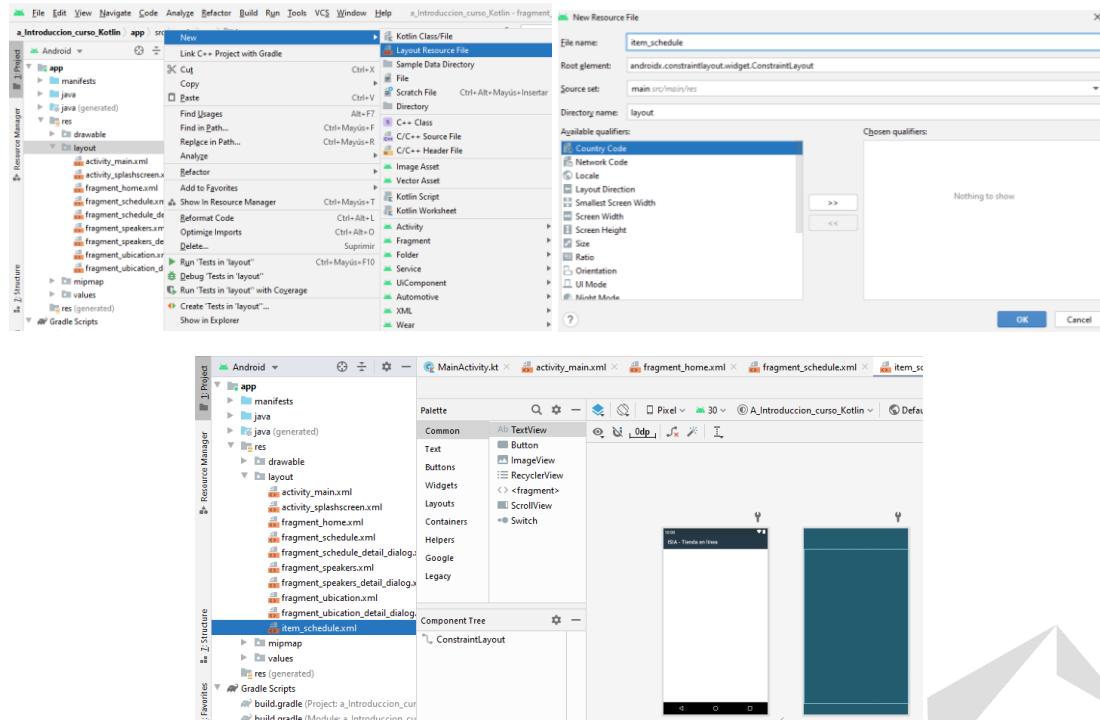
- Para poder utilizar el RecyclerView, debo descargar una dependencia en el archivo de build.gradle(Module: NombreProyecto.app), esto para poder utilizar la librería de **Recycler View** por medio del siguiente código:
  - `implementation 'androidx.recyclerview:recyclerview:1.0.0'`

*Nota: Cuando se agregue el código al archivo Graddle se debe dar clic en el botón Sync Now que aparece en la esquina superior derecha para que se agregue la librería.*

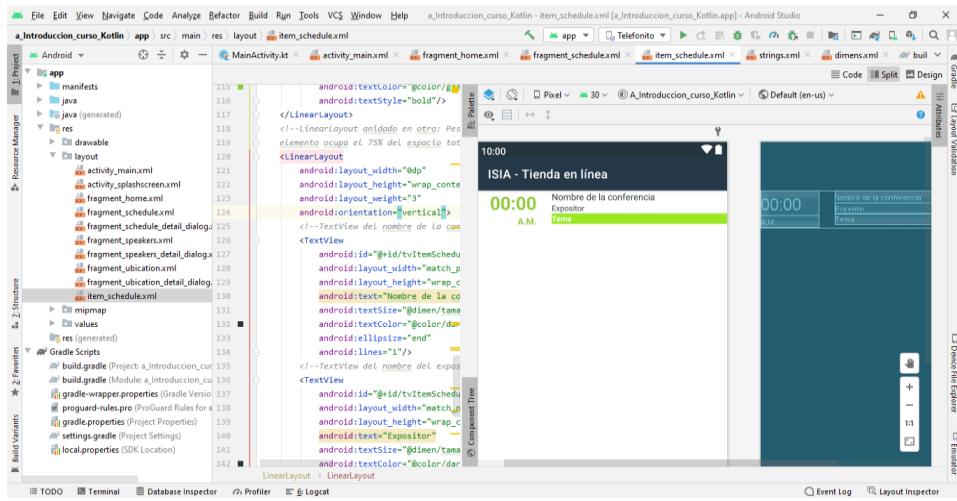


2.2.2.- Ahora se creará el ítem de cada elemento del RecyclerView declarado en el fragmento llamado **fragment\_schedule.xml**, el ítem es un archivo llamado **item\_schedule.xml** donde se describe el diseño personalizado de cada uno de los elementos de la lista:

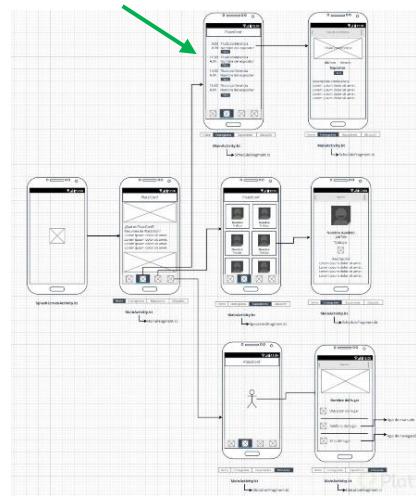
- **Item:** Para crear un elemento personalizado de un RecyclerView es necesario crear un diseño XML especial llamado item, este se crea dando clic derecho en la carpeta layout → New → Layout resource file → County Code → File name: nombre con la palabra item en un inicio y sin caracteres especiales ni mayúsculas → Ok.



El diseño creado en el item se aplicará a todos los elementos de la lista o rejilla creada con el View RecyclerView en el archivo **fragment\_schedule.xml** que tiene la pantalla de carga.



2.2.3.- Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se sigue con el fragmento llamado **fragment\_schedule\_detail\_dialog.xml** que aparece cuando se haga clic en cada ítem del fragmento **fragment\_schedule.xml** y donde se mostrará una descripción detallada de cada elemento:

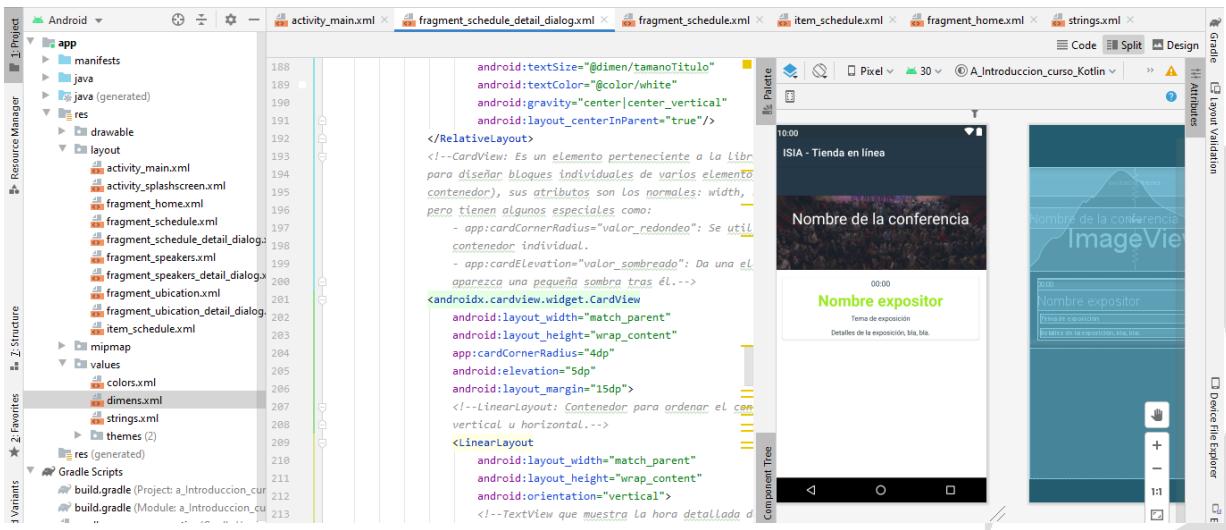


**En este caso el diseño no se hará en un fragmento, solo se hará en un contenedor cualquiera para dentro de él poner el fragmento que cambiará dependiendo de la información que muestre, esto no afecta a la navegación de la aplicación porque el layout es solamente la interfaz gráfica, pero el archivo XML sigue conectado a un archivo Kotlin que se maneja como Fragmento.**

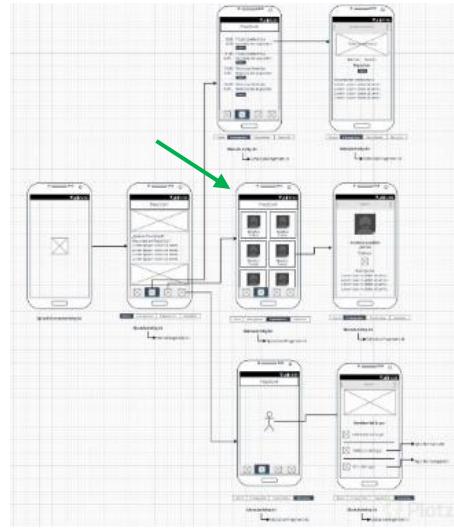
- **Coordinator Layout:** El ViewGroup que utilice esta etiqueta se considerará como el principal, osea el que debería estar al principio del diseño, se utiliza cuando se quiera incluir animaciones, anidar muchos elementos o colocar mucho dinamismo a la pantalla.
  - En este caso se utilizará el contenedor Coordinator Layout porque permite anidar muchos elementos dentro de él.
- **Barra superior (Toolbar):** Para poder modificar la barra superior de la aplicación, que en un inicio tiene el nombre de la aplicación, debo descargar la librería de **Material Design** en el archivo de build.gradle(Module: NombreProyecto.app), para ello se usan las etiquetas AppBarLayout y Toolbar.

*Nota: En este punto del código ya la dependencia ha sido incluida, por lo que no es necesario descargarla de nuevo.*

- **Fragments:** La etiqueta principal para la creación y manejo de fragmentos que se utilizan como DialogFragments o como un fragmento cualquiera es **FrameLayout**, en este caso el fragmento está contenido dentro del contenedor del diseño.
  - El contenido de la descripción será dinámico, esto quiere decir que cambiará dependiendo del item que sea elegido del archivo **fragment\_schedule.xml** y los datos que jale de la base de datos, para ello es necesario usar un fragmento (Fragment) por medio de la etiqueta FrameLayout dentro del contenedor y después de la etiqueta Toolbar.
- **ScrollView:** Es un contenedor que permite al usuario ver su contenido haciendo scroll (subiendo o bajando en la pantalla de la aplicación), se usa cuando el contenido de la pantalla es mucho, pero como el ScrollView no puede contener más de un componente, se opta por poner un contenedor (ViewGroup) en su interior que contenga otros contenedores para mostrar su contenido.
  - **Linear Layout:** Contenedor que ordena sus componentes de forma lineal vertical u horizontalmente, uno arriba del otro o uno alado del otro y es muy usado dentro de las etiquetas ScrollView ya que permite anidación y usar más de un elemento.
- **Relative Layout:** Ordena los elementos que contenga sobreponiendo uno encima de otro.
- **Linear Layout:** Contenedor que ordena sus componentes de forma lineal vertical u horizontalmente, uno arriba del otro o uno alado del otro. Permite anidación.
- **View:** Es un contenedor donde se puede poner lo que sea (semejante al div en diseño web), si está vacío puede servir para crear una línea de separación.
- **CardView:** Es un elemento perteneciente a la librería Material Design que sirve para colocar elementos individuales dentro de un contenedor, sus atributos son los normales: width, height, background, pero tienen algunos especiales como:
  - **app:cardCornerRadius="valor\_redondeo":** Se utiliza para indicar el redondeo del contenedor individual.
  - **app:cardElevation="valor\_redondeo":** Da una elevación al elemento para que aparezca una pequeña sombra tras él.

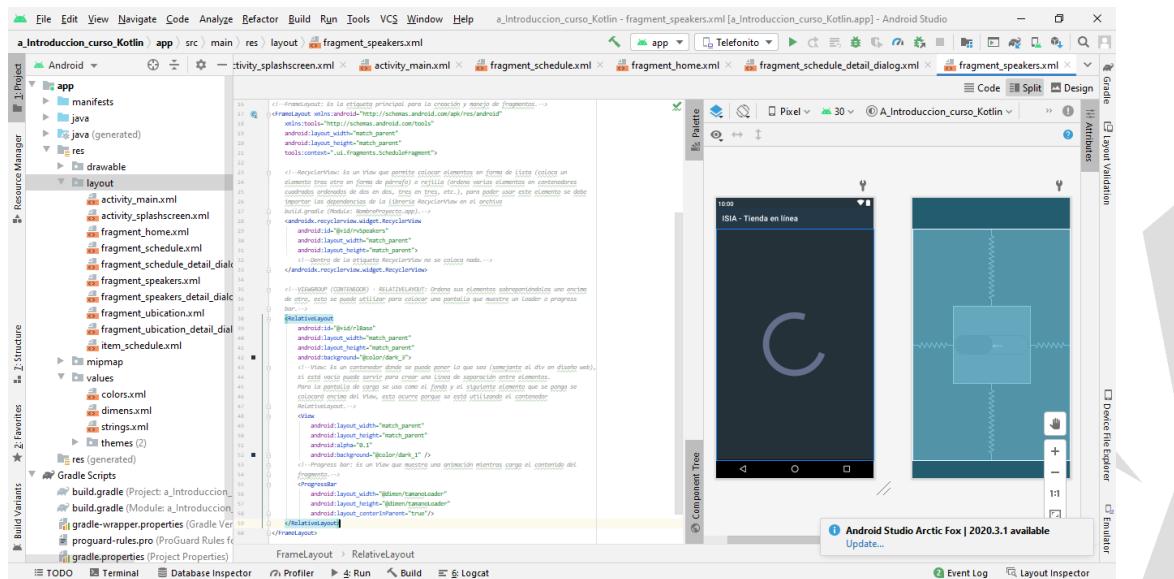


2.3.1.- Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se sigue con el fragmento llamado **fragment\_speakers.xml** en donde se utilizará el contenedor **RelativeLayout** para mostrar una pantalla de carga (loader o progress bar) y el elemento **RecyclerView** para mostrar una lista de elementos, cuyo diseño se creará en un archivo XML aparte que representará el aspecto visual de todos los ítems:

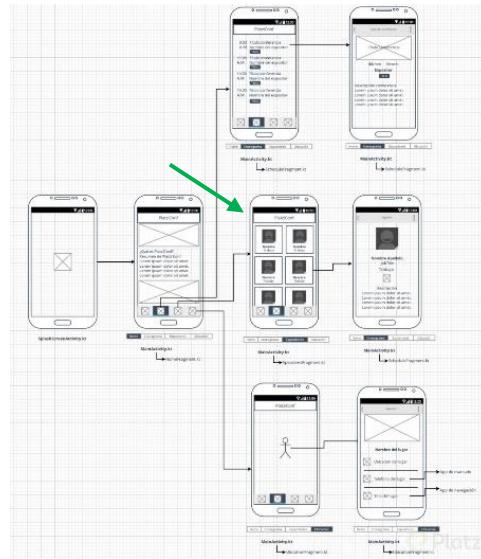


Así como en el ejemplo pasado se utilizó un **RecyclerView** para mostrar una lista de elementos, ahora se utilizará para crear una rejilla. Todas las pantallas de la serie de elementos que contienen un listado o rejilla hecha con **RecyclerView** necesitan 3 distintos layout:

- El principal (2.3.1), donde es declarada la etiqueta **RecyclerView** y se puede poner una pantalla de carga con el elemento **Progress bar**, en este caso el archivo es: **fragment\_speakers.xml**
- Uno que indique el estilo de los ítems (2.3.2), ya sea para un listado o rejilla (grid), en este caso el archivo es: **item\_speakers.xml**
- El último que indica los detalles de cada ítem cuando se dé clic en él (2.3.3), en este caso el archivo es: **fragment\_speakers\_detail.xml**

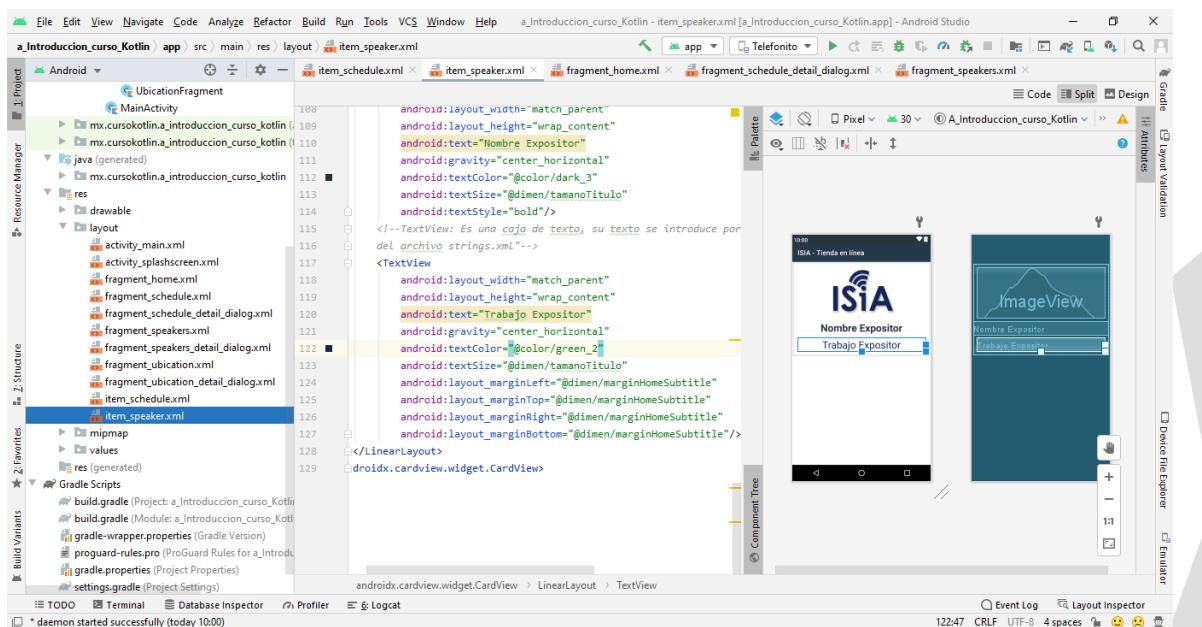


2.3.2.- Ahora se creará el ítem de cada elemento del RecyclerView declarado en el fragmento llamado **fragment\_speakers.xml**, el ítem es un archivo llamado **item\_speakers.xml** donde se describe el diseño personalizado de cada uno de los elementos de la lista o rejilla:

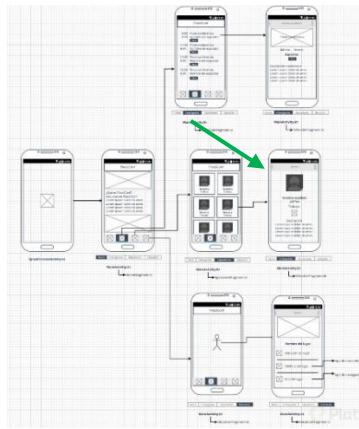


Así como en el ejemplo pasado donde se utilizó un RecyclerView para mostrar una lista de elementos, ahora se utilizará para crear una rejilla, Para las pantallas que contienen un listado o rejilla hecha con RecyclerView se necesitan 3 archivos:

- El principal (2.3.1), donde es declarada la etiqueta RecyclerView y se puede poner una pantalla de carga con el elemento Progress bar, en este caso el archivo es: **fragment\_speakers.xml**
- Uno que indique el estilo de ítem (2.3.2), ya sea para un listado o rejilla (grid), en este caso el archivo es: **item\_speakers.xml**, este archivo debe ser creado.
- El último que indica los detalles de cada ítem cuando se dé clic en él (2.3.3), en este caso el archivo es: **fragment\_speakers\_detail.xml**



2.2.3.- Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se sigue con el fragmento llamado **fragment\_speakers\_detail\_dialog.xml** que aparece cuando se haga clic en cada ítem del fragmento **fragment\_speakers.xml** y donde se mostrará una descripción detallada de cada elemento:

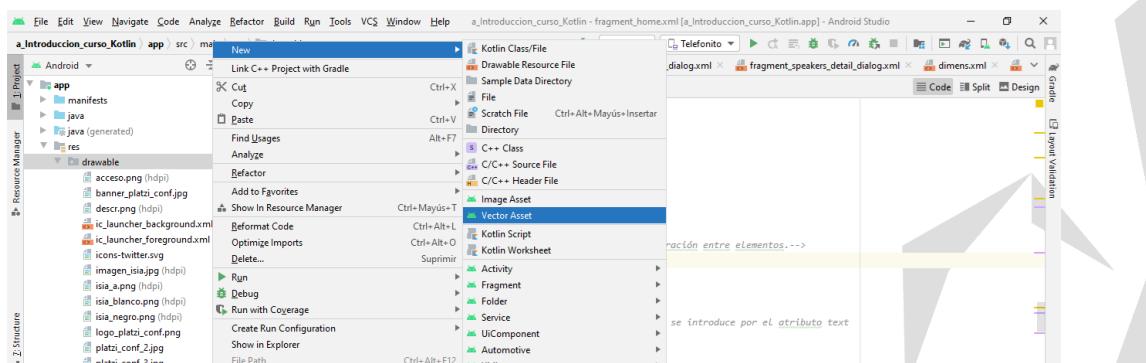


Así como en el ejemplo pasado donde se utilizó un **RecyclerView** para mostrar una lista de elementos, ahora se utilizará para crear una rejilla, Para las pantallas que contienen un listado o rejilla hecha con **RecyclerView** se necesitan 3 archivos:

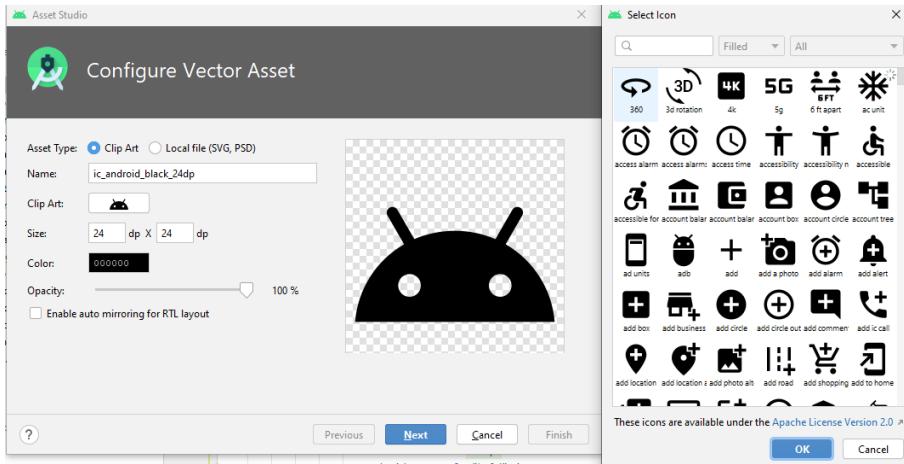
- El principal (2.3.1), donde se declara la etiqueta **RecyclerView** y se puede poner una pantalla de carga con el elemento **Progress bar**, en este caso el archivo es: **fragment\_speakers.xml**
- Uno que indique el estilo de ítem (2.3.2), ya sea para un listado o rejilla (grid), en este caso el archivo es: **item\_speakers.xml**, este archivo debe ser creado.
- El último que indica los detalles de cada ítem cuando se dé clic en él (2.3.3), en este caso el archivo es: **fragment\_speakers\_detail.xml**

*Nota: Cuando quiera incluir una imagen svg pero que esta pese lo menos posible, lo que puedo hacer es convertirla a xml por medio del siguiente procedimiento:*

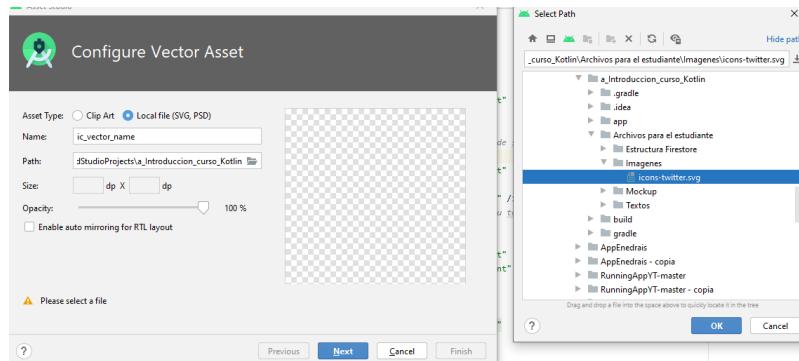
- **Imagen de svg a xml (Vector):** Cuando una imagen svg se convierte a formato xml esta se convierte en un vector, la ventaja de esto es que la imagen pesa lo menos posible que puede pesar una imagen dentro de la app, lo cual la hará más rápida de cargar y no se pierde calidad. Android Studio solo acepta imágenes en formato xml, jpg o png. Para crear un vector se debe dar clic derecho en la carpeta **app/res/drawable** (donde se encuentran todas las imágenes) → Vector Asset → Vector Asset.



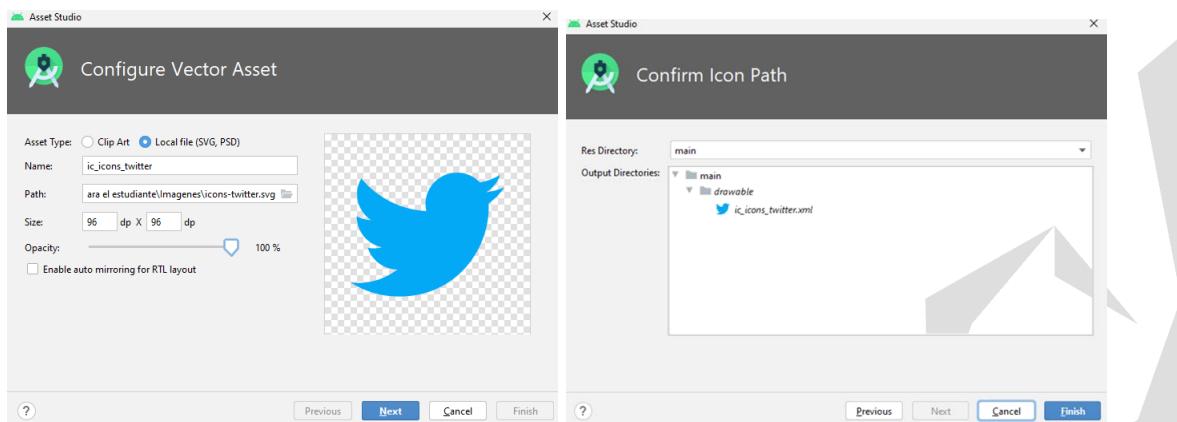
De esta manera no solamente se puede convertir un archivo svg a un vector xml, sino que se puede crear un ícono cualquiera de los que se muestran en Android Studio para poderlo utilizar en el proyecto por medio del radio button de Clip Art, dando clic sobre el ícono de Android aparecerá la ventana de Select Icon donde se puede elegir de una librería de íconos.

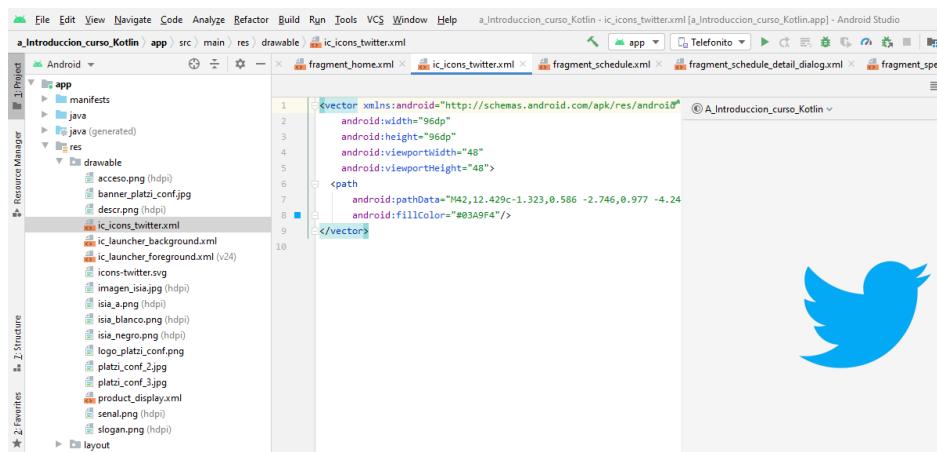


Si lo que se quiere hacer es convertir un archivo externo svg a un vector, lo que se debe hacer es seleccionar el radio button Local file (SVG, PSD) y elegir un archivo svg después de dar clic en la parte donde dice Path.

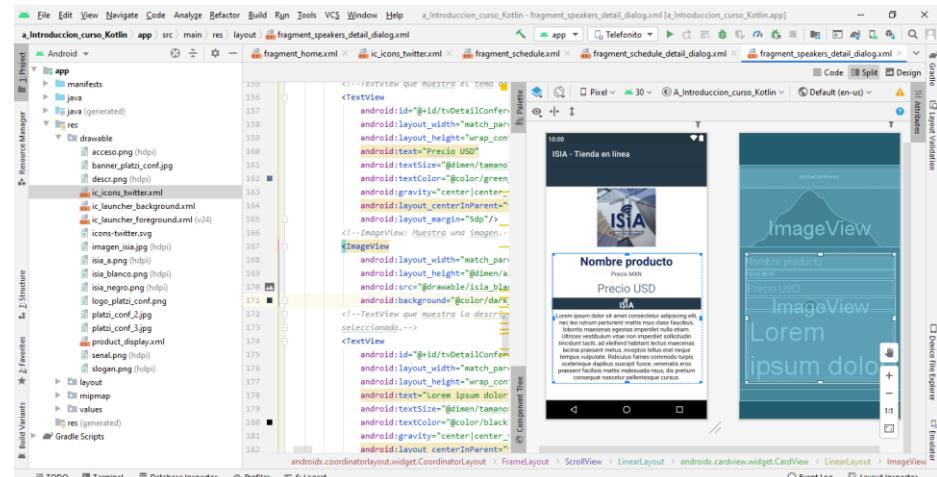


Aquí además se puede indicar el nombre del vector, su tamaño en dp y transparencia, al dar clic en el botón de Next y luego Finish, se habrá creado el vector con extensión xml en la carpeta app/res/drawable.

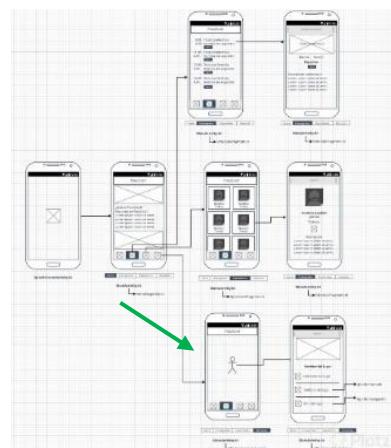




Ya que se haya agregado el vector a la carpeta drawable, se podrá obtener con una etiqueta ImageView como cualquier otra imagen, además el nombre de todos los íconos por buenas prácticas debe empezar con las letras **ic\_nombre\_vector.xml**



**2.4.1.-** Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se sigue con el fragmento llamado **fragment\_ubication.xml** en donde se mostrará el mapa de Google Maps que contiene la ubicación de una locación y al dar clic sobre la ubicación, posteriormente se mostrará la descripción del lugar:



En la carpeta layout es donde podemos diseñar el fragmento de la interfaz de cada pantalla (DialogFragment).

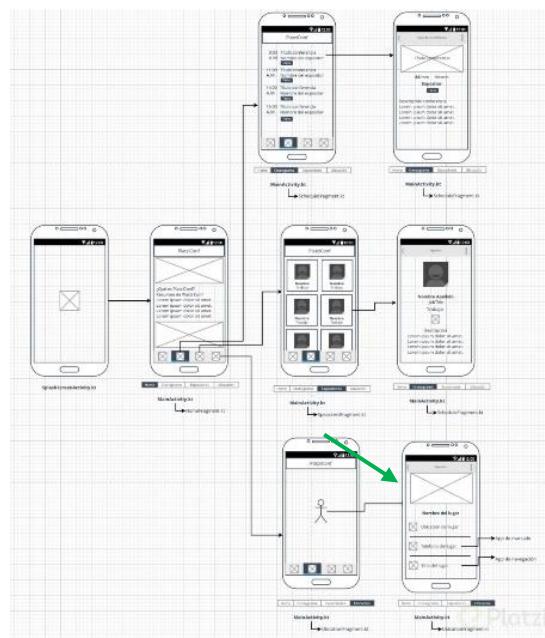
**En este caso el diseño no se hará en un fragmento, solo se hará en un contenedor cualquiera.**

- **Coordinator Layout:** El ViewGroup que utilice esta etiqueta se considerará como el principal, osea el que debería estar al principio del diseño, se utiliza cuando se quiera incluir animaciones, anidar muchos elementos o colocar mucho dinamismo a la pantalla.
  - En este caso se utilizará el contenedor Coordinator Layout porque permite anidar muchos elementos dentro de él.
- **Barra superior (Toolbar):** Para poder modificar la barra superior de la aplicación, que en un inicio tiene el nombre de la aplicación, debo descargar la librería de **Material Design** en el archivo de build.gradle(Module: NombreProyecto.app), para ello se usan las etiquetas AppBarLayout y Toolbar.

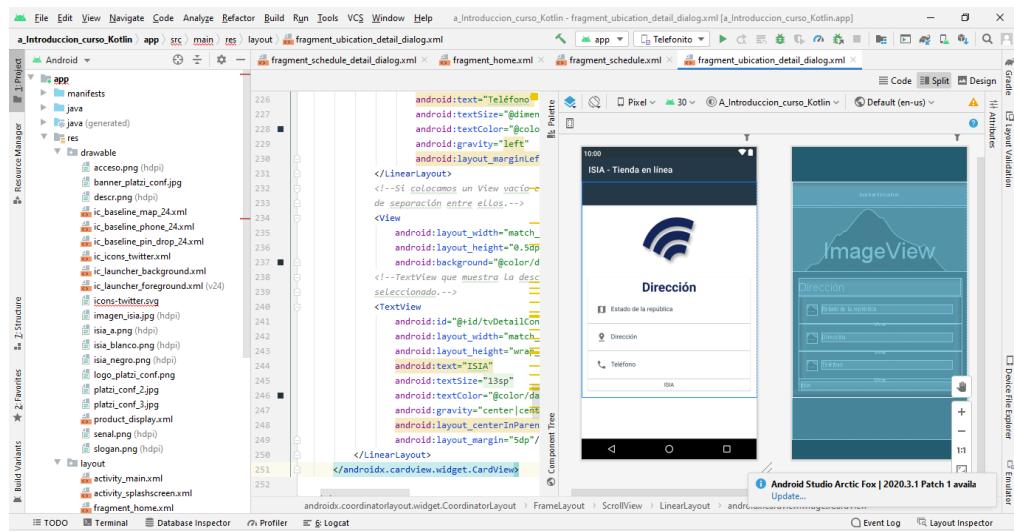
*Nota: En este punto del código ya la dependencia ha sido incluida, por lo que no es necesario descargarla de nuevo.*

- **Fragments:** La etiqueta principal para la creación y manejo de fragmentos que se utilizan como DialogFragments o como un fragmento cualquiera es **FrameLayout**, en este caso el fragmento está contenido dentro del contenedor del diseño.
  - El contenido de la descripción será dinámico, esto quiere decir que cambiará dependiendo del item que sea elegido del archivo **fragment\_schedule.xml**, para ello es necesario usar un fragmento (Fragment) por medio de la etiqueta FrameLayout dentro del contenedor y después de la etiqueta Toolbar.

2.4.2.- Ya que se haya creado la interfaz donde aparece el menú, se deben diseñar individualmente los **DialogFragments** que representen cada una de sus pantallas, se sigue con el fragmento llamado **fragment\_ubication\_detail\_dialog.xml** que aparece cuando se haga clic en la ubicación mostrara en el mapa del fragmento **fragment\_ubication.xml** y donde se mostrará una descripción detallada:



La pantalla después de diseñar el detalle de la ubicación del mapa es la siguiente:

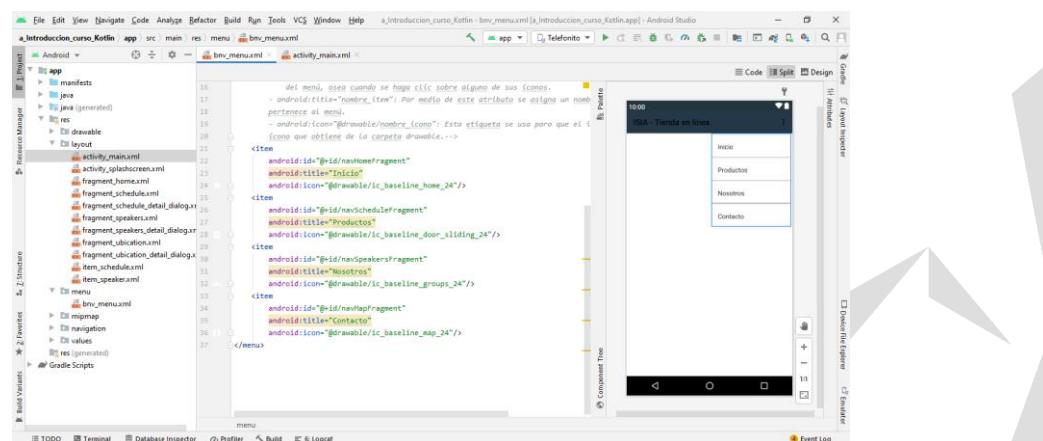


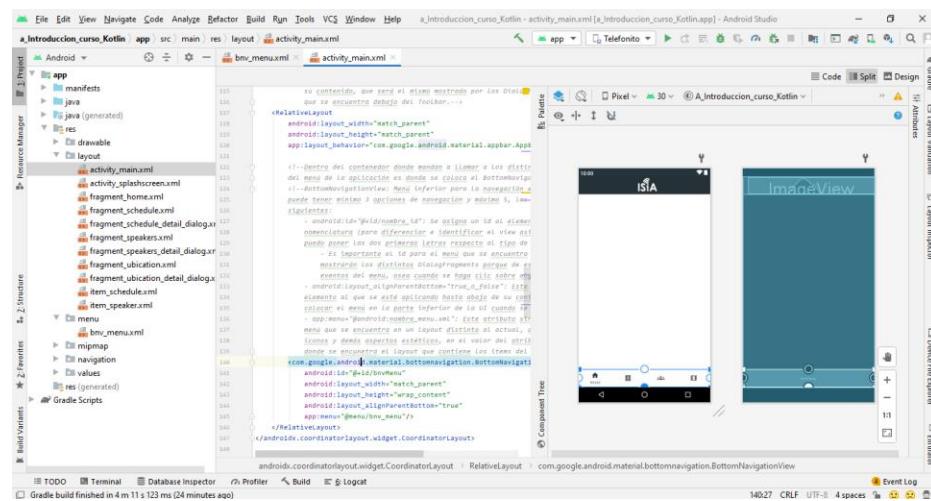
## Diseño del menú:

• **Menú:** Es un tipo de elemento visual muy especial, ya que su uso es exclusivo para crear las opciones del menú inferior de navegación que se coloca dentro del layout principal, que pertenece a la actividad principal normalmente llamada `activity_main.xml`.

- Recordemos que el menú se creó en una carpeta (paquete) separado de los demás dentro de la carpeta de recursos (res), este sirve para indicar por medio de elementos item las opciones que se pueden seleccionar del menú de navegación creado por medio de la etiqueta **BottomNavigationView** en el archivo `activity_main.xml`. En los ítems se indican los siguientes aspectos de cada botón:

- `android:id="@+id/nombre_id"`: Un id que haga alusión de a dónde queremos que nos dirija cada botón del menú inferior, ya que este mismo id se tendrá que utilizar en el navgraph cuando realice las transiciones entre pantallas para habilitar las acciones del menú.
- `android:title="@string/nombre_boton"`: El texto de cada botón.
- `android:icon="@drawable/nombre_icono"`: El ícono de cada opción.



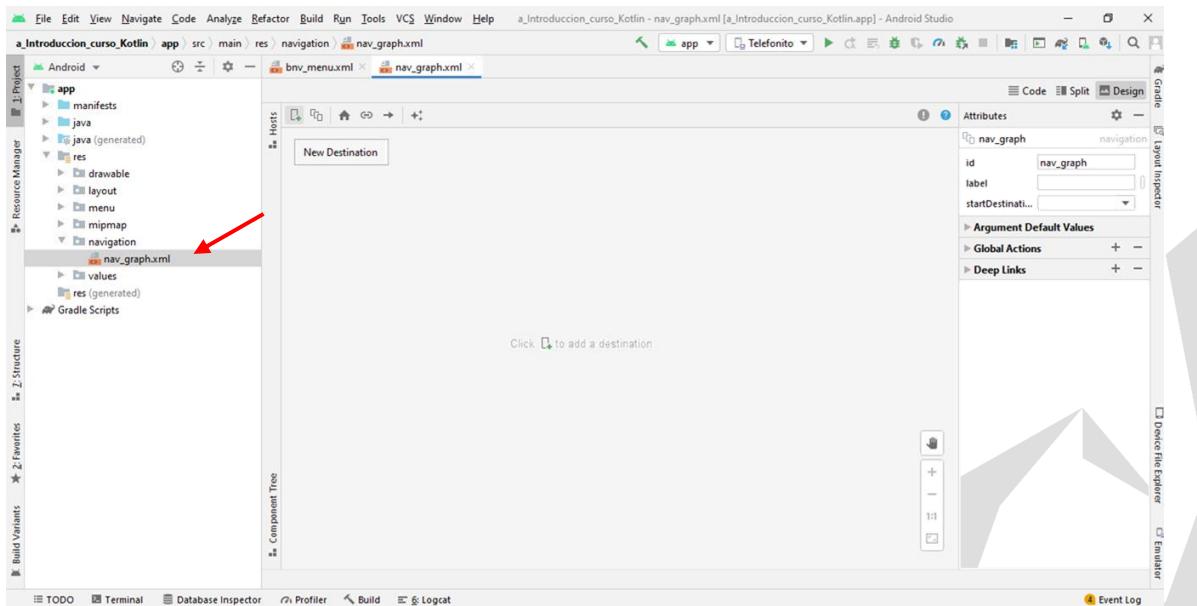


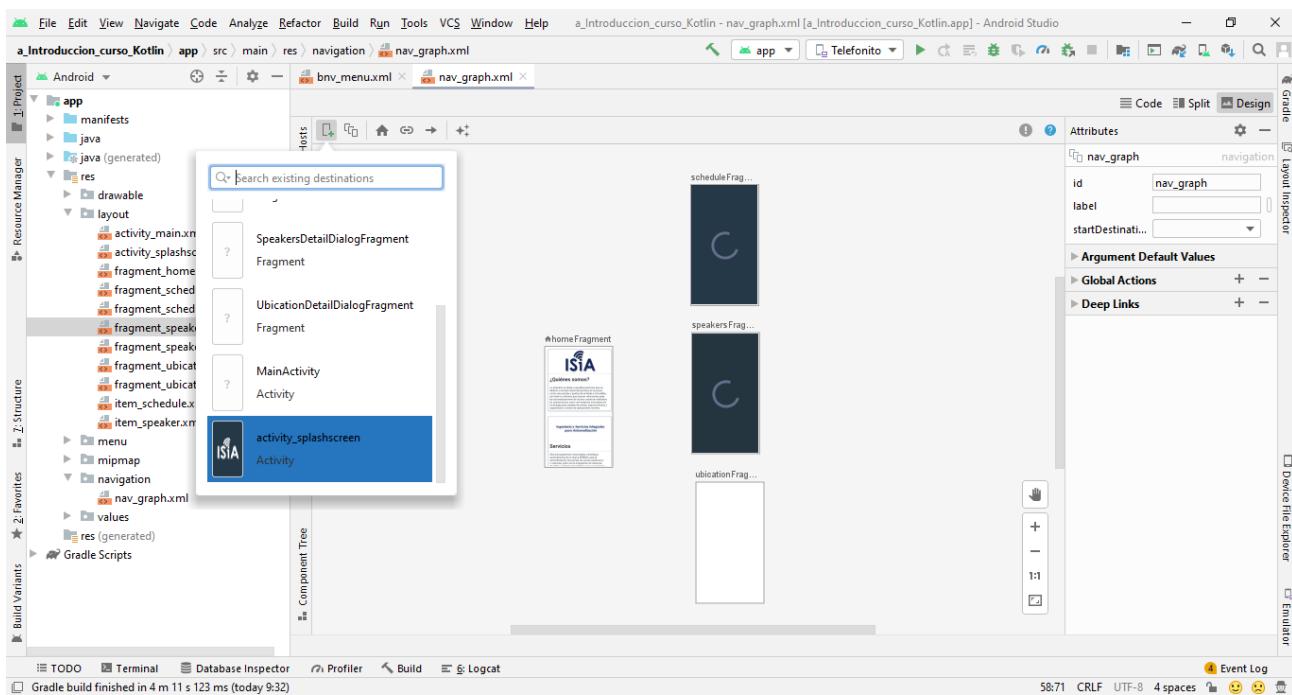
## Diseño del NavGraph:

### 1.- Creación del navgraph:

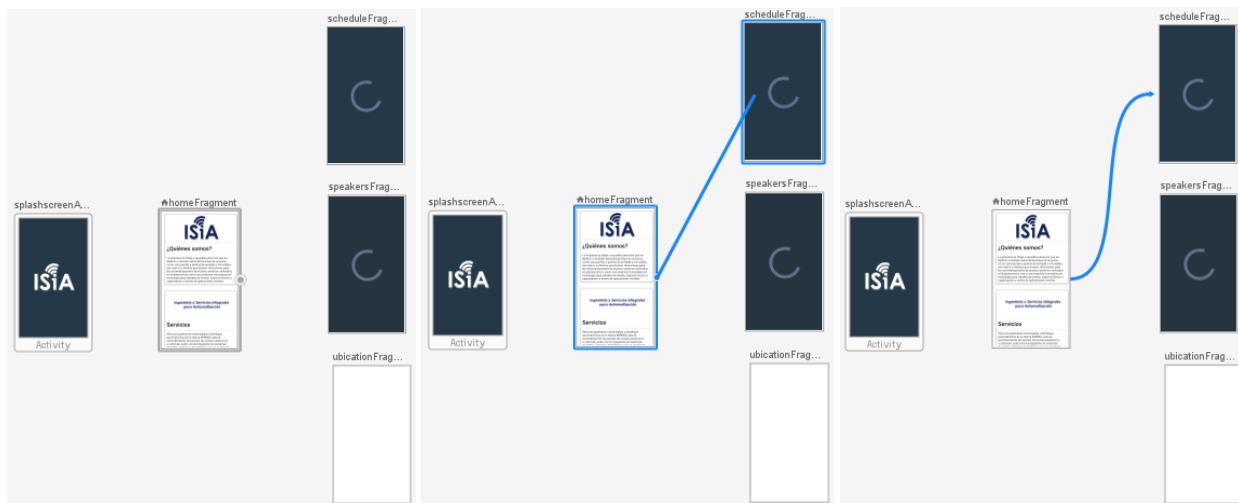
- **Navigation:** Es un componente de la librería Android Jetpack que sirve para poder indicar al menú como va a pasar de una pantalla a otra de la aplicación Android.
- **NavGraph:** Es un gráfico de navegación entre fragmentos y/o actividades para poder observar sus conexiones de mejor manera y crear líneas de código que representen el evento de navegar de un DialogFragment a otro, de una Activity a otra o de una Activity a un DialogFragment y viceversa, recordemos que el archivo de navigation se creó en una carpeta (paquete) separada de los demás, dentro de la carpeta de recursos (res), en este archivo es donde se crea el gráfico de navegación o navgraph.

En la esquina superior izquierda se encuentra un botón donde se pueden añadir los distintos fragmentos o actividades de la aplicación Android hechos hasta ahora.

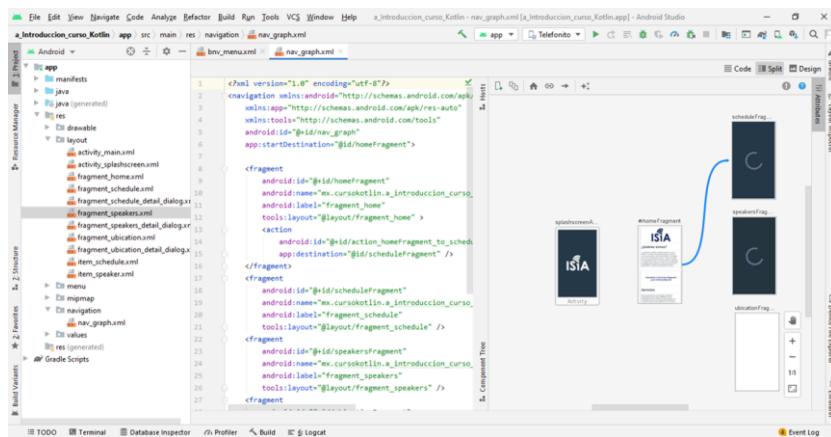




Para crear las conexiones se debe arrastrar el punto que tienen las pantallas del NavGraph.

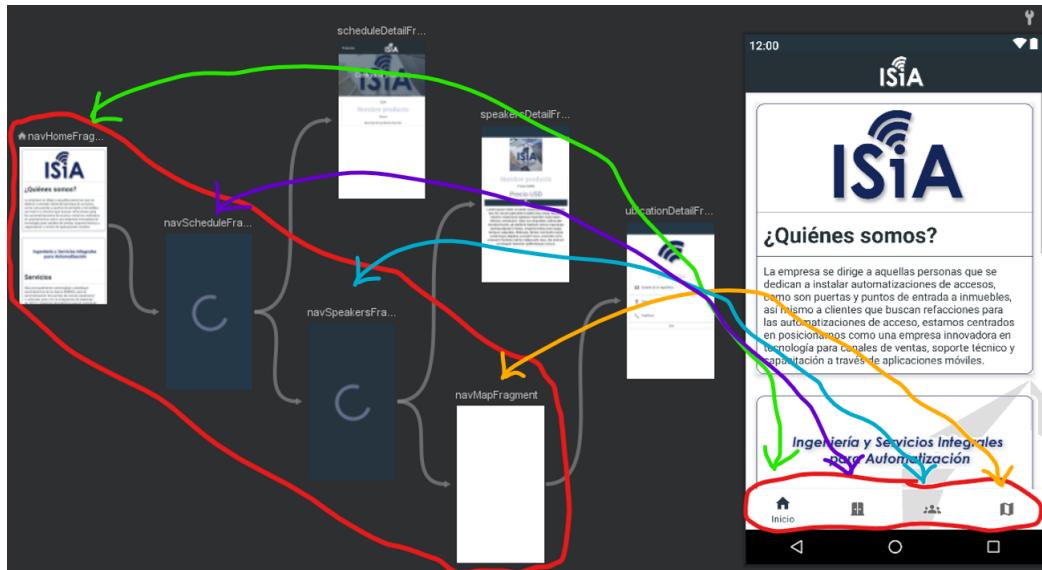


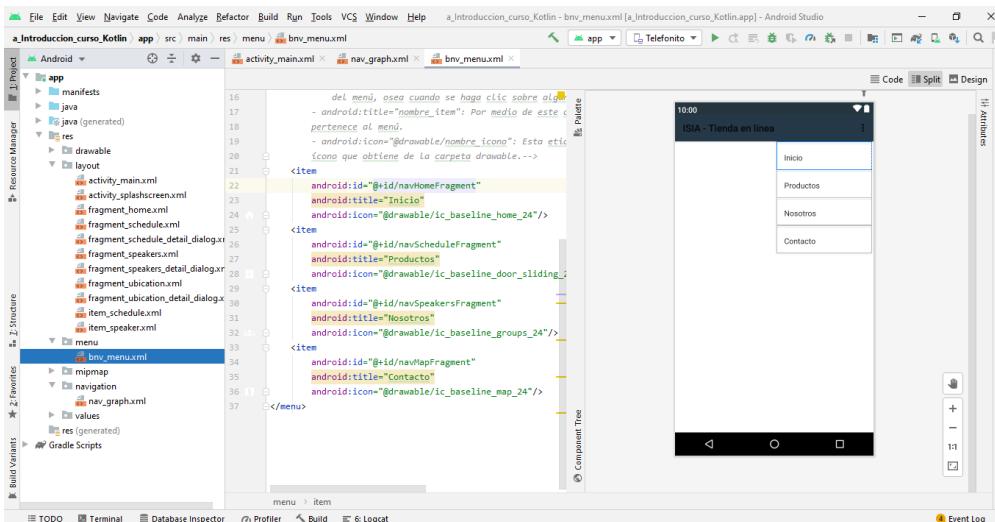
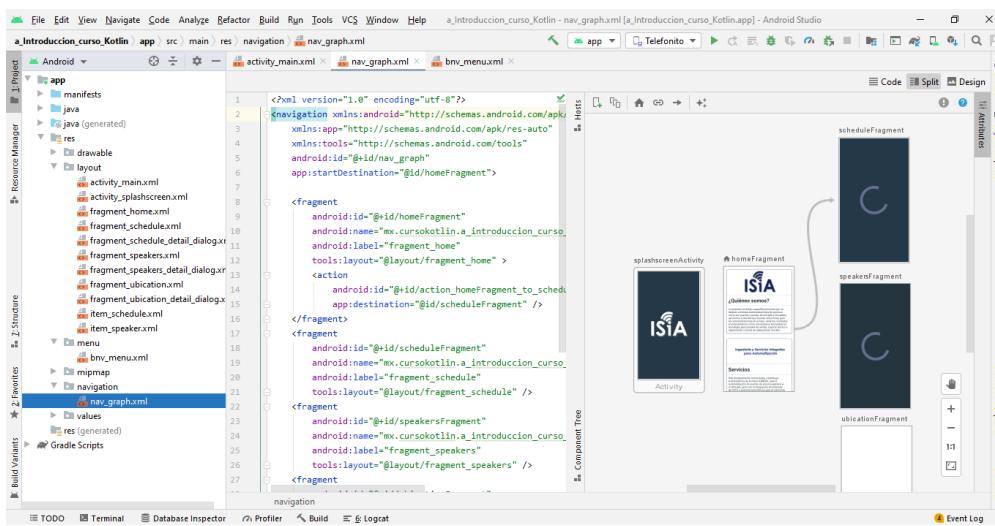
Esto lo que hará es crear un código con una etiqueta global llamada `<navigation>` que indicará la pantalla inicial y dentro de la cual cada fragmento del navgraph tendrá su etiqueta `<fragment>` y cada actividad tendrá su etiqueta `<activity>`, dentro de ellas se creará una o varias etiquetas `<action>` cuando se cree una conexión nueva, representando así la acción de brincar de una pantalla a otra. La forma de conectar las pantallas entre ellas en el navgraph no es importante, lo que importa es que existan las conexiones necesarias para poder movernos de una pantalla a otra como se pretende en el flujo de la app.



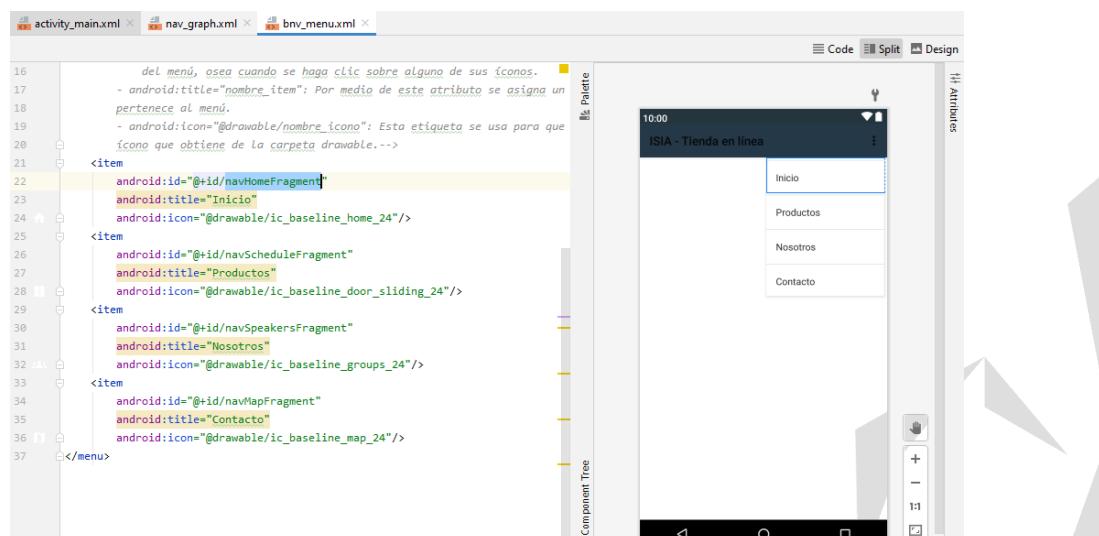
## 2.- Conexión del navgraph con el bottomNavigationMenu:

- Un punto muy importante para poder **enlazar** el diagrama de navegación (**NavGraph**) con el menú inferior (**BottomNavigationView**) que se encuentra en el archivo **activity\_main.xml** es:
  - Que se debe **hacer que los id de las etiquetas fragment o activity del NavGraph a donde queremos que nos dirija cada uno de los botones del menú inferior tengan el mismo id que los ítems del menú** que conducen a cada una.
    - **Esto solo se debe hacer con las pantallas principales a las cuales se quiera navegar a través del menú**, no con las que se deba navegar a través de otra manera, como las pantallas que muestran los detalles de los elementos descritos en listas o rejillas, donde se navega a ellas a través de dar clic en un elemento.
    - Además, para hacer esta conexión y transición entre pantallas por medio del BottomNavigationView **no son importantes las conexiones que tengan las etiquetas action de las etiquetas fragment o activity del NavGraph**.
  - Las transiciones del navgraph realizadas por las etiquetas **<action/>** pertenecientes a las etiquetas **<fragment/>** y **<activity/>** sirven más que nada para realizar transiciones entre pantallas por medio de eventos como dar clic en un botón, realizar correctamente una conexión Bluetooth, entre otros.

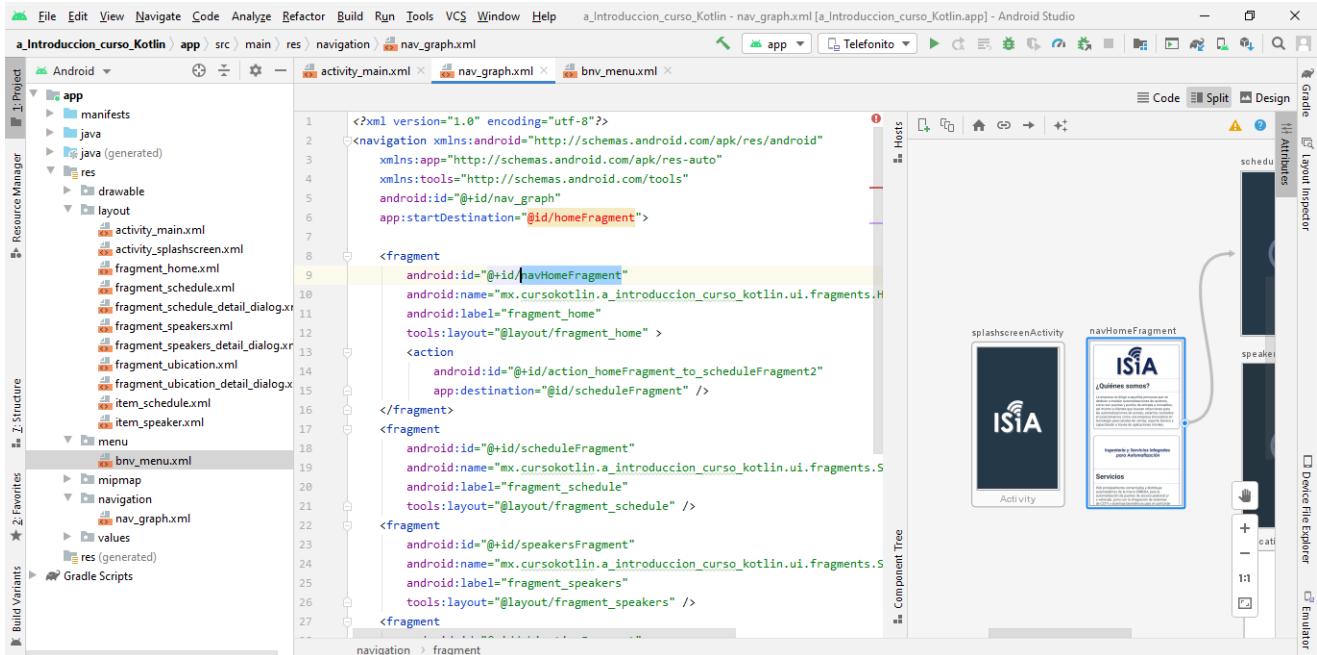




Por lo tanto, copiaremos y pegaremos en el archivo **nav\_graph.xml** los id de los ítems del menú que se encuentran en el archivo **bnv\_menu.xml** del menú inferior, se empezará con el id del botón de home.



Al hacerlo puede aparecer un error en el atributo startDestination de la etiqueta `<navigation>` ya que se ha cambiado el id de la pantalla principal, por lo que se debe poner el mismo id que hay en el archivo de los ítems del menú dentro de esta etiqueta, ya que indica el punto de partida de la aplicación.

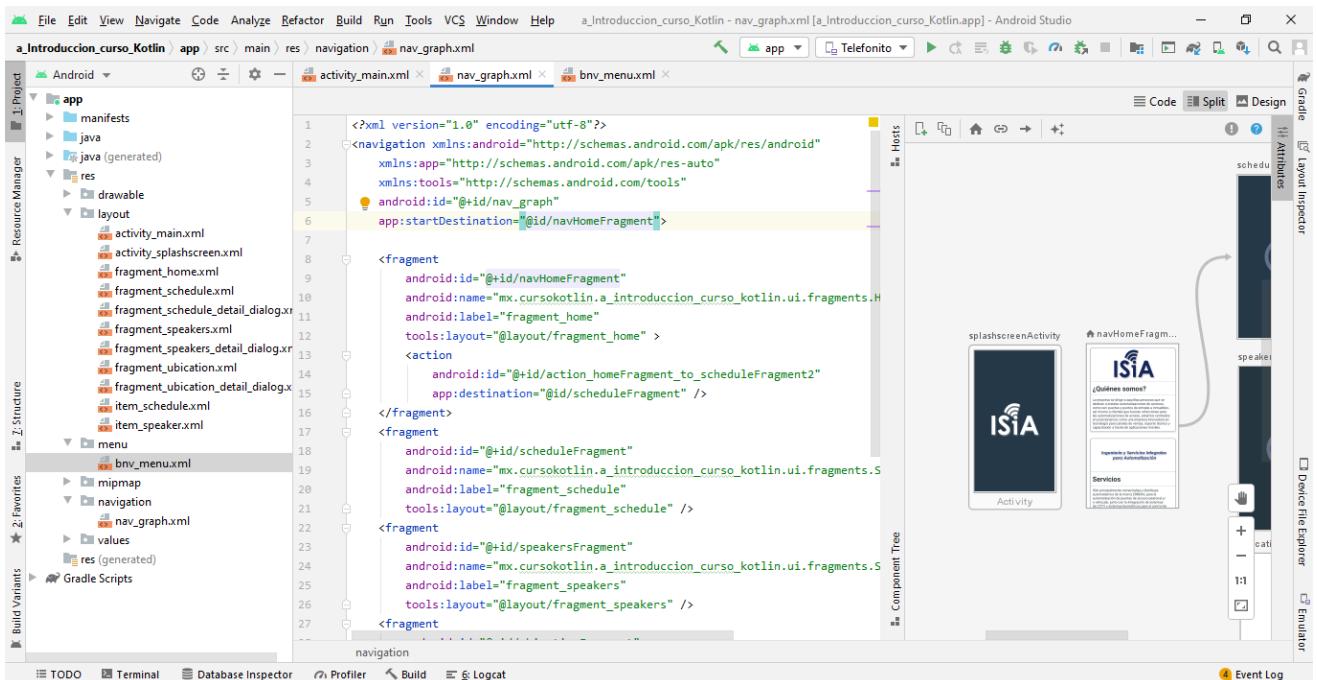


```

<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@+id/homeFragment">

    <fragment
        android:id="@+id/navHomeFragment"
        android:name="mx.curso kotlin.a_introduccion_curso_kotlin.ui.fragments.HomeFragment"
        android:label="fragment_home"
        tools:layout="@layout/fragment_home" >
        <action
            android:id="@+id/action_homeFragment_to_scheduleFragment2"
            app:destination="@+id/scheduleFragment" />
    </fragment>
    <fragment
        android:id="@+id/scheduleFragment"
        android:name="mx.curso kotlin.a_introduccion_curso_kotlin.ui.fragments.ScheduleFragment"
        android:label="fragment_schedule"
        tools:layout="@layout/fragment_schedule" />
    <fragment
        android:id="@+id/speakersFragment"
        android:name="mx.curso kotlin.a_introduccion_curso_kotlin.ui.fragments.SpeakersFragment"
        android:label="fragment_speakers"
        tools:layout="@layout/fragment_speakers" />
</navigation>

```



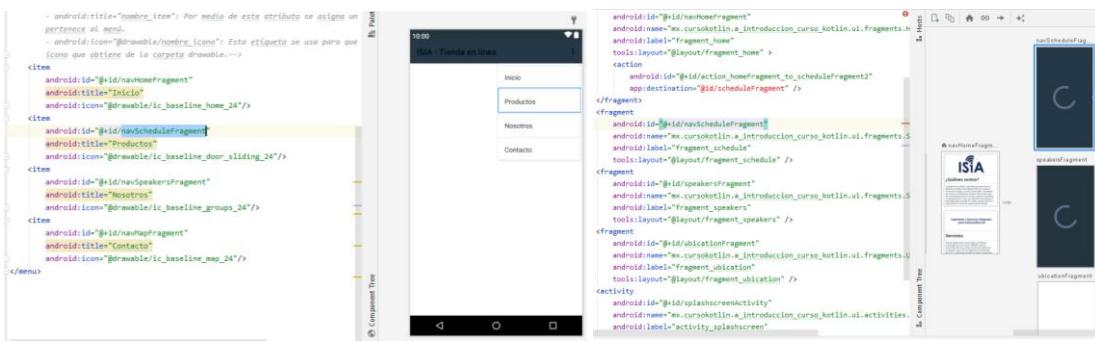
```

<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@+id/navHomeFragment">

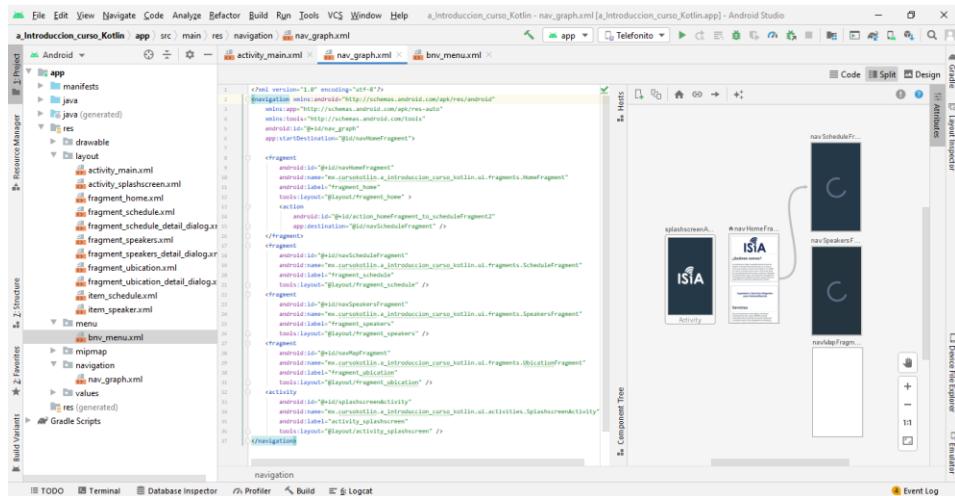
    <fragment
        android:id="@+id/navHomeFragment"
        android:name="mx.curso kotlin.a_introduccion_curso_kotlin.ui.fragments.HomeFragment"
        android:label="fragment_home"
        tools:layout="@layout/fragment_home" >
        <action
            android:id="@+id/action_homeFragment_to_scheduleFragment2"
            app:destination="@+id/scheduleFragment" />
    </fragment>
    <fragment
        android:id="@+id/scheduleFragment"
        android:name="mx.curso kotlin.a_introduccion_curso_kotlin.ui.fragments.ScheduleFragment"
        android:label="fragment_schedule"
        tools:layout="@layout/fragment_schedule" />
    <fragment
        android:id="@+id/speakersFragment"
        android:name="mx.curso kotlin.a_introduccion_curso_kotlin.ui.fragments.SpeakersFragment"
        android:label="fragment_speakers"
        tools:layout="@layout/fragment_speakers" />
</navigation>

```

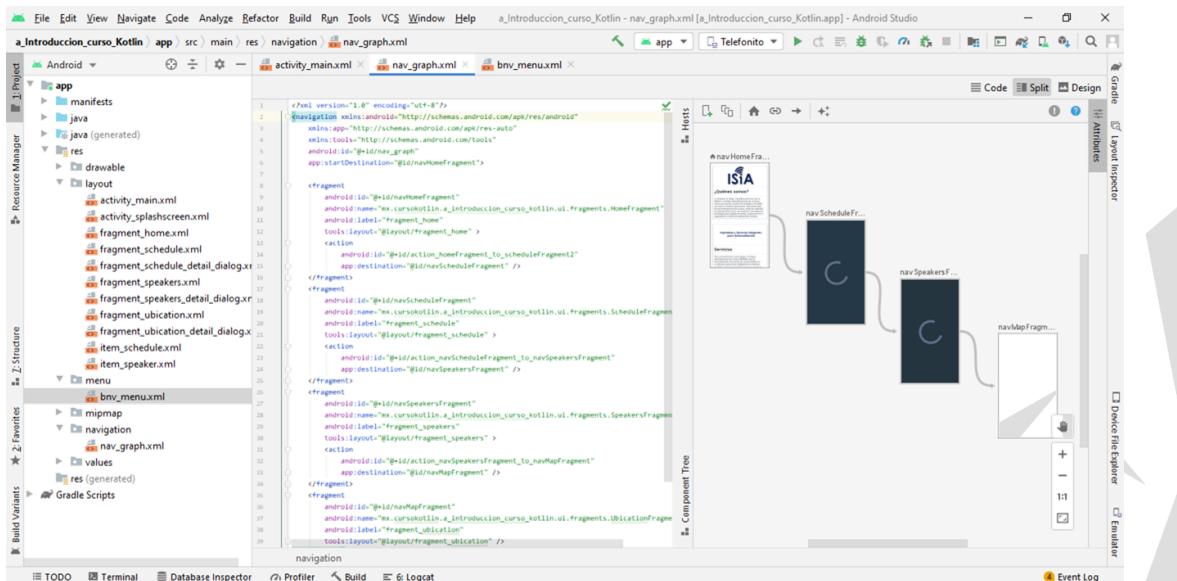
Se hace lo mismo con los id de los demás ítems y los fragmentos a donde llevan, además cuando se haga esto si es que ya existe alguna transición entre pantallas con la etiqueta `<action>`, se tendrá que cambiar ese id también, ya que en ellas se indica el id de la pantalla a donde dirige cada transición del navgraph en su atributo `destination`.



Se hace lo mismo con los demás elementos hasta que los id del archivo **nav\_graph.xml** del NavGraph se hayan puesto igual a los id de los ítems que se encuentran en el archivo **bnv\_menu.xml** del BottomNavigationView.



Si las transiciones se reacomodan después de haber asignado los id de los ítems del menú, no es necesario cambiar manualmente cambiar el id de la transición, por sí solo Android Studio los coloca de forma correcta.



3.- Conexión de los DialogFragments que muestran listas o rejillas con sus ítems de diseño y pantalla donde se muestran los detalles de cada elemento del listado o rejilla:

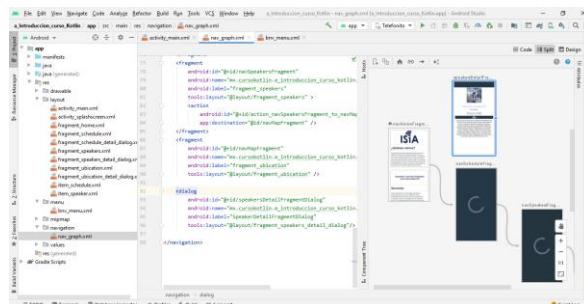
Ahora falta hacer los diálogos, representados por una etiqueta llamada `<dialog/>` que realiza la conexión entre los DialogFragments, sus ítems de diseño personalizado creados para los elementos de los listados o rejillas y la pantalla donde se muestran los detalles de cada elemento del listado o rejilla, esto se hace a través de etiquetas `dialog` dentro del código del navgraph, en específico dentro de la etiqueta `<navigation/>`. Ya que recordemos que para los DialogFragments que contienen un listado o rejilla hechas con la etiqueta `RecyclerView` se necesitan 3 layouts:

- **El principal:** Donde es declarada la etiqueta `RecyclerView` que crea el listado o rejilla y encima se puede poner una pantalla de carga con el elemento `Progress bar`.
- **Item de diseño:** Layout donde se indica el diseño personalizado con el que se mostrarán todos los ítems del listado o rejilla.
- **Pantalla de detalles de cada elemento del listado:** El último archivo indica los detalles de cada ítem cuando se dé clic en él.

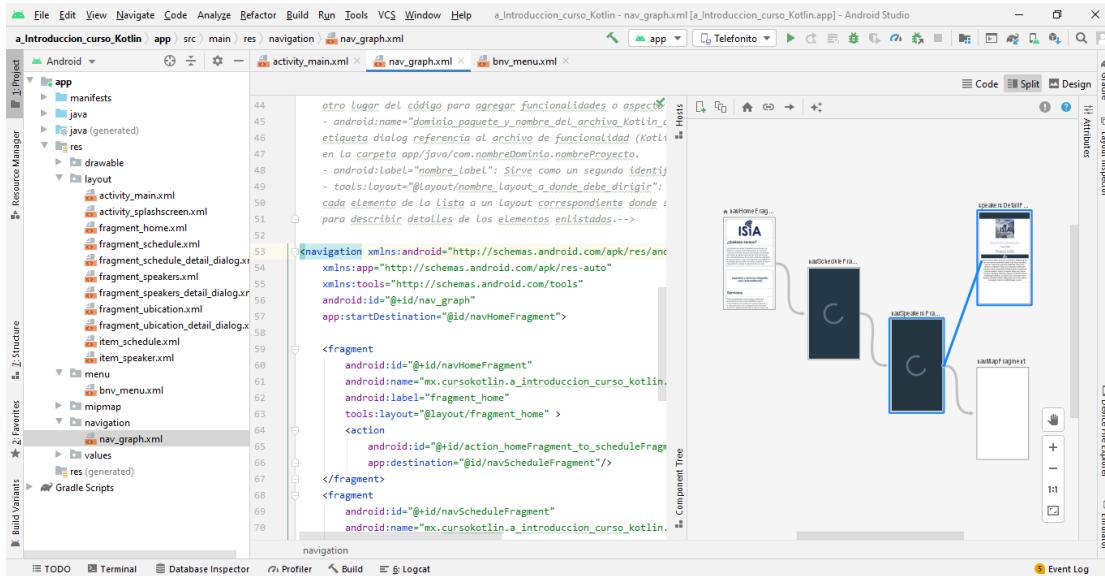
Por lo tanto, se usan los diálogos para hacer la conexión entre el DialogFragments que muestra un listado o rejilla, sus ítems y su pantalla de detalles:

- **Dialog:** Es una etiqueta que sirve para hacer conexiones entre los fragmentos que contienen listas o rejillas, en sus atributos se colocan distintos elementos para poder realizar la conexión entre los 3 layouts necesarios:
  - `android:id="@+id/nombre_id"`: En su id se debe poner el mismo id que se encuentre dentro de la etiqueta `<action/>` que indique la transición entre el DialogFragment que muestra el listado de elementos y la pantalla que muestra los detalles de los elementos enlistados.
  - `android:name="paquete_del_fragment_o_activity"`: En el atributo name se indica paquete del archivo tipo Fragment o Activity hecho con el lenguaje Kotlin de donde obtendrán los datos para mostrarse en el listado o rejilla del fragmento principal y los detalles de cada uno.
  - `tools:layout="@drawable/nombre_icono"`: En el atributo layout se indica la pantalla donde se muestran los detalles de cada elemento del listado.

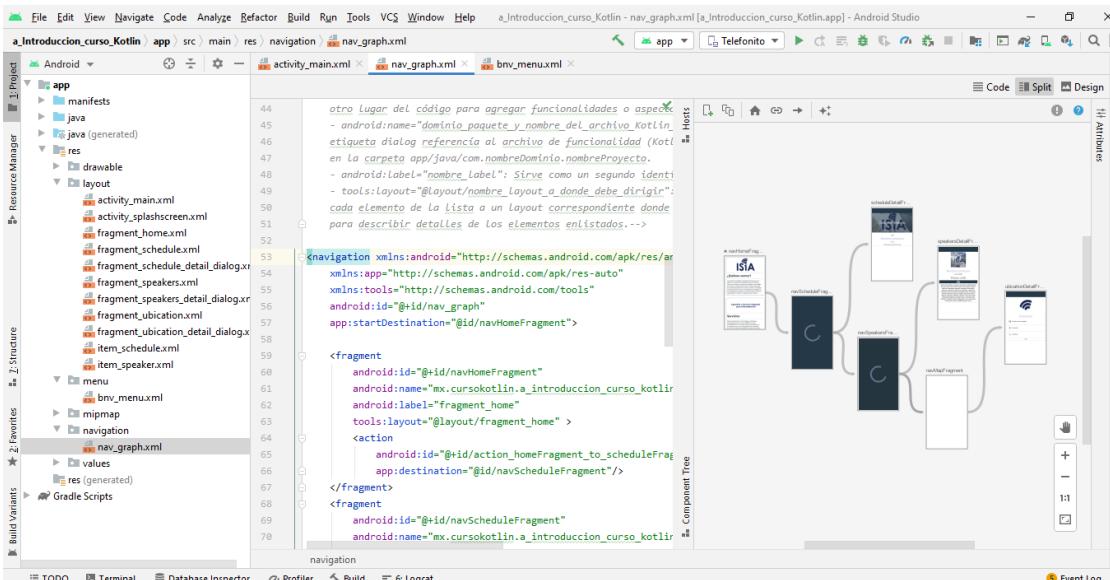
Ya que se haya creado la etiqueta `Dialog` indicando su layout, este aparecerá en el navgraph, posteriormente se deberá hacer la conexión con la pantalla donde se muestra el listado de elementos que después se describirán en la pantalla de detalles.



La pantalla simplemente se conectará desde donde está el listado de elementos hasta donde se encuentra el layout que muestra la descripción detallada.



Se debe repetir lo mismo con todas las pantallas que dirigan a otra donde se da a conocer detalles del elemento contenido en el DialogFragment.



Finalmente, para que funcione la navegación y ver su resultado debemos ir a la Activity principal que está en el archivo activity\_main.xml y crear una etiqueta **FragmentContainerView**, donde se mostrarán todos los DialogFragments que indica la navegación.

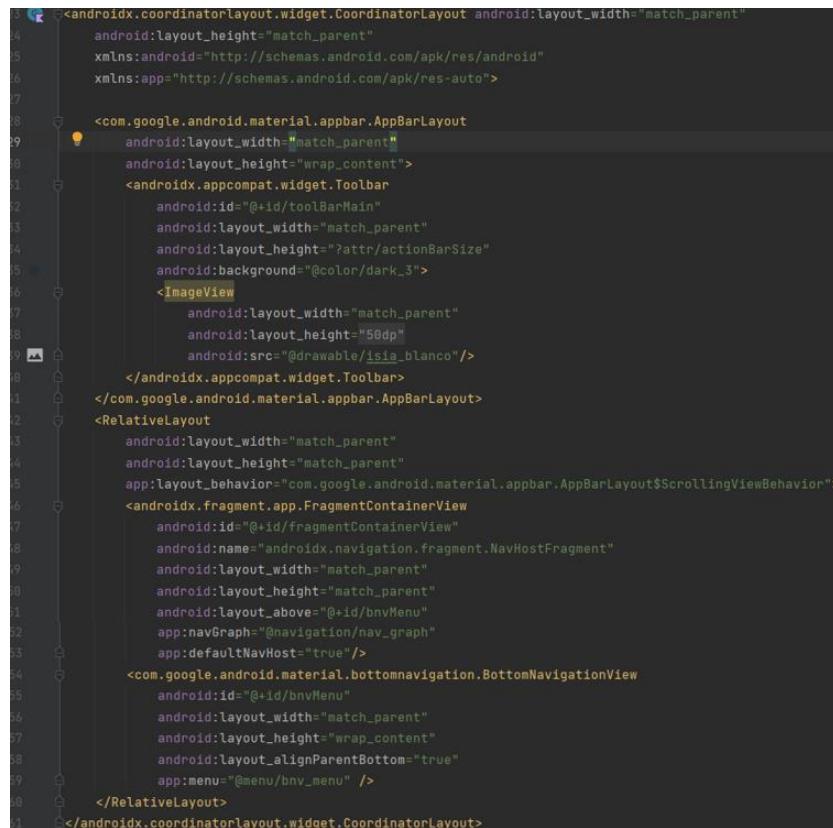
## Activación del menú en el archivo Kotlin:

Recordemos que dentro del layout principal, normalmente llamado activity\_main.xml, donde después de haber puesto la **barra de navegación dentro de un contenedor CoordinatorLayout con la etiqueta**

**AppBarLayout** y **Toolbar**, se coloca un contenedor **RelativeLayout** para colocar elementos uno encima de otro (dentro del mismo CoordinatorLayout) que tendrá el contenido de los distintos **DialogFragments** de la aplicación en una etiqueta **FragmentContainerView** y el mismo menú de navegación creado por medio de la etiqueta **BottomNavigationView**:

```
<CoordinatorLayout  
    <AppBarLayout  
        <Toolbar/>  
    />  
    <RelativeLayout  
        <FragmentContainerView/>  
        <BottomNavigationView/>  
    />  
/>
```

Ya teniendo el código del layout completo, el navgraph unido a los ítems del BottomNavigationView y los dialog conectados a las pantallas de las listas o rejillas en el navgraph solo queda añadir un código extra en el archivo `MainActivity.kt` para poder activar la función del bottomNavigationView.

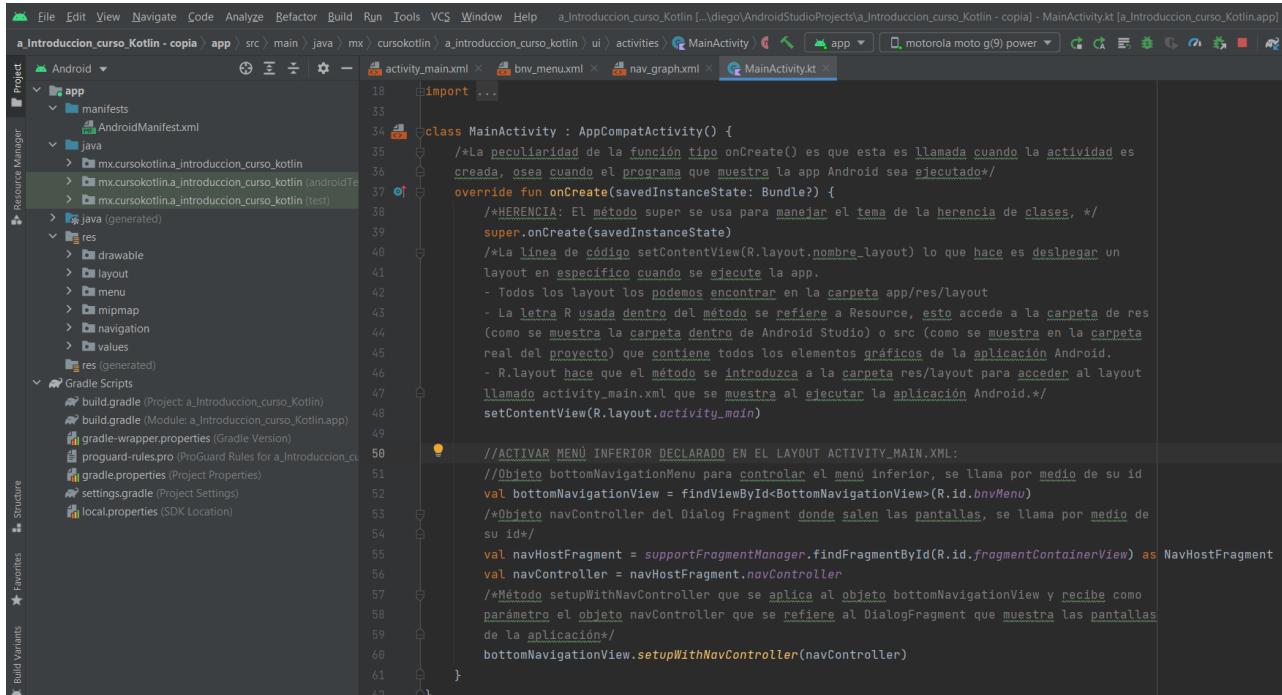


```
1 <androidx.coordinatorlayout.widget.CoordinatorLayout android:layout_width="match_parent"  
2     android:layout_height="match_parent"  
3     xmlns:android="http://schemas.android.com/apk/res/android"  
4     xmlns:app="http://schemas.android.com/apk/res-auto">  
5  
6     <com.google.android.material.appbar.AppBarLayout  
7         android:layout_width="match_parent"  
8         android:layout_height="wrap_content">  
9             <androidx.appcompat.widget.Toolbar  
10                 android:id="@+id/toolBarMain"  
11                 android:layout_width="match_parent"  
12                 android:layout_height="?attr/actionBarSize"  
13                 android:background="@color/dark_3">  
14                     <ImageView  
15                         android:layout_width="match_parent"  
16                         android:layout_height="50dp"  
17                         android:src="@drawable/isi_blanco"/>  
18             </androidx.appcompat.widget.Toolbar>  
19         </com.google.android.material.appbar.AppBarLayout>  
20     <RelativeLayout  
21         android:layout_width="match_parent"  
22         android:layout_height="match_parent"  
23         app:layout_behavior="com.google.android.material.appbar.AppBarLayout$ScrollingViewBehavior">  
24             <androidx.fragment.app.FragmentContainerView  
25                 android:id="@+id/fragmentContainerView"  
26                 android:name="androidx.navigation.fragment.NavHostFragment"  
27                 android:layout_width="match_parent"  
28                 android:layout_height="match_parent"  
29                 android:layout_above="@+id/bnvMenu"  
30                 app:navGraph="@navigation/nav_graph"  
31                 app:defaultNavHost="true"/>  
32             <com.google.android.material.bottomnavigation.BottomNavigationView  
33                 android:id="@+id/bnvMenu"  
34                 android:layout_width="match_parent"  
35                 android:layout_height="wrap_content"  
36                 android:layout_alignParentBottom="true"  
37                 app:menu="@menu/bnv_menu" />  
38         </RelativeLayout>  
39     </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Dentro de la MainActivity simplemente debemos introducirnos al método onCreate, que, si recordamos el ciclo de vida de las actividades, se ejecuta cuando se abre una actividad, para realizar los siguientes pasos:

- Crear un objeto que herede de la clase `BottomNavigationView` y asociarlo por medio de su constructor al id de la etiqueta `BottomNavigationView` `<com.google.android.material.bottomnavigation.BottomNavigationView/>` del layout que contiene el menú usando el método `findViewById<>` y la sintaxis `R.id.nombre_id`, donde la R se refiere a la carpeta de recursos (res o src).
- Crear un objeto que herede de la clase `NavController` y asociarlo por medio de su constructor al id de la etiqueta `FragmentContainerView` `<androidx.fragment.app.FragmentContainerView/>` del layout que contiene el menú usando la clase `supportFragmentManager` para aplicar el método `findViewById<>` y la sintaxis `R.id.nombre_id`, donde la R se refiere a la carpeta de recursos (res o src).
  - Hacer que el objeto que instancia la clase `NavController` ahora herede de la clase `navController`.
- Finalmente usar el objeto que instancia de la clase `BottomNavigationView` para utilizar el método `setUpWithNavController()` pasándole como parámetro el objeto que instancia de la clase `navController`.

Al hacer esto, ya se podrá utilizar el menú inferior de navegación.



```

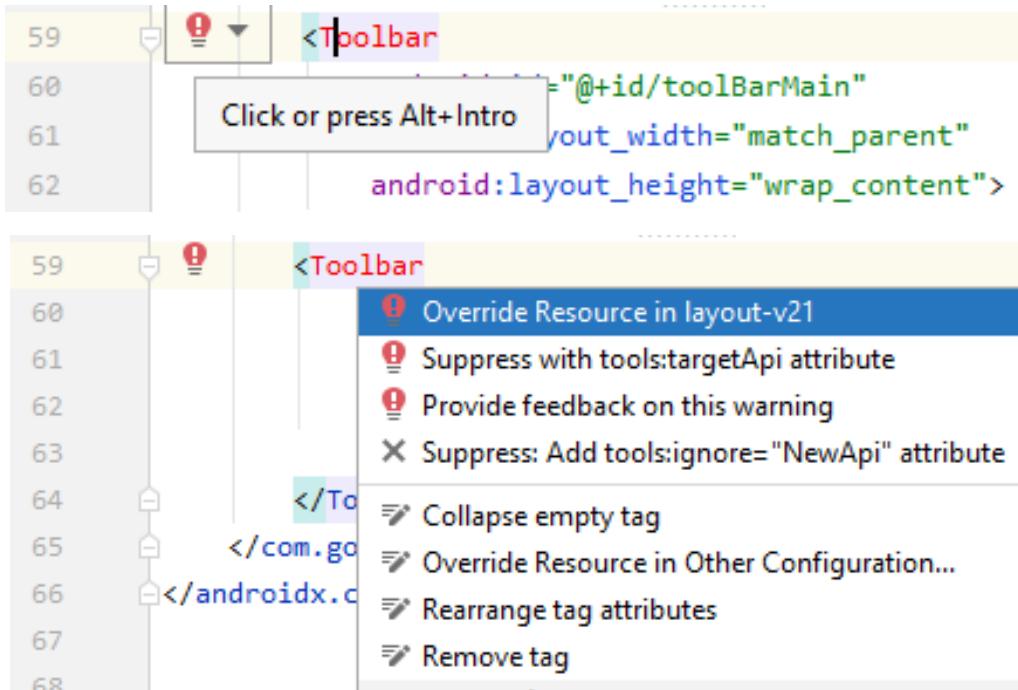
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help a_Introduccion_curso_Kotlin [..]\diego\AndroidStudioProjects\ a_Introduccion_curso_Kotlin - copia - MainActivity.kt [a_Introduccion_curso_Kotlin.app]
a_Introduccion_curso_Kotlin - copia app src main java mx.curso kotlin a_introduccion_curso_kotlin ui activities MainActivity
Project Android manifests AndroidManifest.xml
app Java mx.curso kotlin.a_introduccion_curso_kotlin
res drawable layout menu mipmap values res (generated)
Gradle Scripts build.gradle (Project: a_Introduccion_curso_Kotlin)
build.gradle (Module: a_Introduccion_curso_Kotlin.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for a_Introduccion_cu
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)

import ...
class MainActivity : AppCompatActivity() {
    /*La peculiaridad de la función tipo onCreate() es que esta es llamada cuando la actividad es creada, osea cuando el programa que muestra la app Android sea ejecutado*/
    override fun onCreate(savedInstanceState: Bundle?) {
        /*HERENCIA: El método super se usa para manejar el tema de la herencia de clases, */
        super.onCreate(savedInstanceState)
        /*La linea de código setContentView(R.layout.nombre_layout) lo que hace es desplegar un layout en específico cuando se ejecute la app.
        - Todos los layout los podemos encontrar en la carpeta app/res/layout
        - La letra R usada dentro del método se refiere a Resource, esto accede a la carpeta de res (como se muestra la carpeta dentro de Android Studio) o src (como se muestra en la carpeta real del proyecto) que contiene todos los elementos gráficos de la aplicación Android.
        - R.layout hace que el método se introduzca a la carpeta res/layout para acceder al layout llamado activity_main.xml que se muestra al ejecutar la aplicación Android.*/
        setContentView(R.layout.activity_main)
        //ACTIVAR MENÚ INFERIOR DECLARADO EN EL LAYOUT ACTIVITY_MAIN.XML:
        /*Objeto bottomNavigationMenu para controlar el menú inferior, se llama por medio de su id
        val bottomNavigationView = findViewById<BottomNavigationView>(R.id.bnv_menu)
        /*Objeto navController del Dialog Fragment donde salen las pantallas, se llama por medio de su id/
        val navHostFragment = supportFragmentManager.findFragmentById(R.id.fragmentContainerView) as NavHostFragment
        val navController = navHostFragment.navController
        /*Método setupWithNavController que se aplica al objeto bottomNavigationView y recibe como parámetro el objeto navController que se refiere al DialogFragment que muestra las pantallas de la aplicación*/
        bottomNavigationView.setupWithNavController(navController)
    }
}

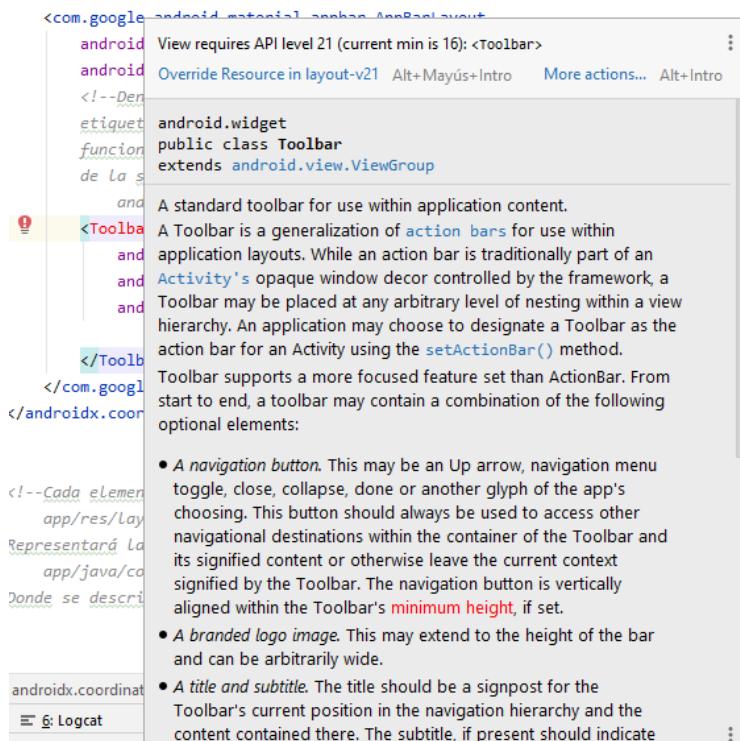
```

# Manejo y visualización de errores

Para obtener consejos de resolución del problema se debe presionar ALT+ENTER sobre el mismo pedazo de código:



Para observar la causa de un error en Android Studio se debe pasar el mouse sobre el pedazo de código que se pinte de rojo:



Siempre es mejor ver primero la causa del error, antes de ejecutar alguna de las soluciones de Android Studio.

- En este caso el error fue causado en una Activity que necesita el nivel de API de Android Studio de nivel 21 para utilizar una etiqueta y el actual es 16.
  - Este problema normalmente es ocasionado cuando se quiere utilizar algún elemento o método de una librería, en este caso fue causado específicamente por la librería de Material Design, cuando se quiso utilizar su etiqueta Toolbar.

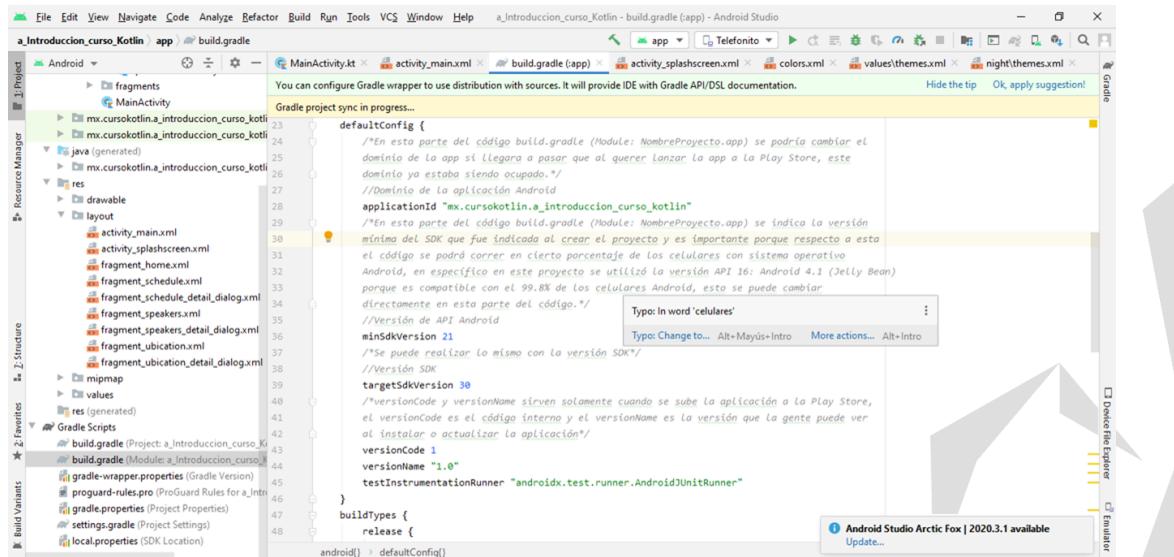
Para resolver el error debemos ir al archivo de configuración build.gradle (Module: NombreProyecto.app) y cambiar el minSdkVersion, que indica la versión mínima del SDK que permite ejecutar el proyecto y cambiarla por la versión 21 en forma manual.

```

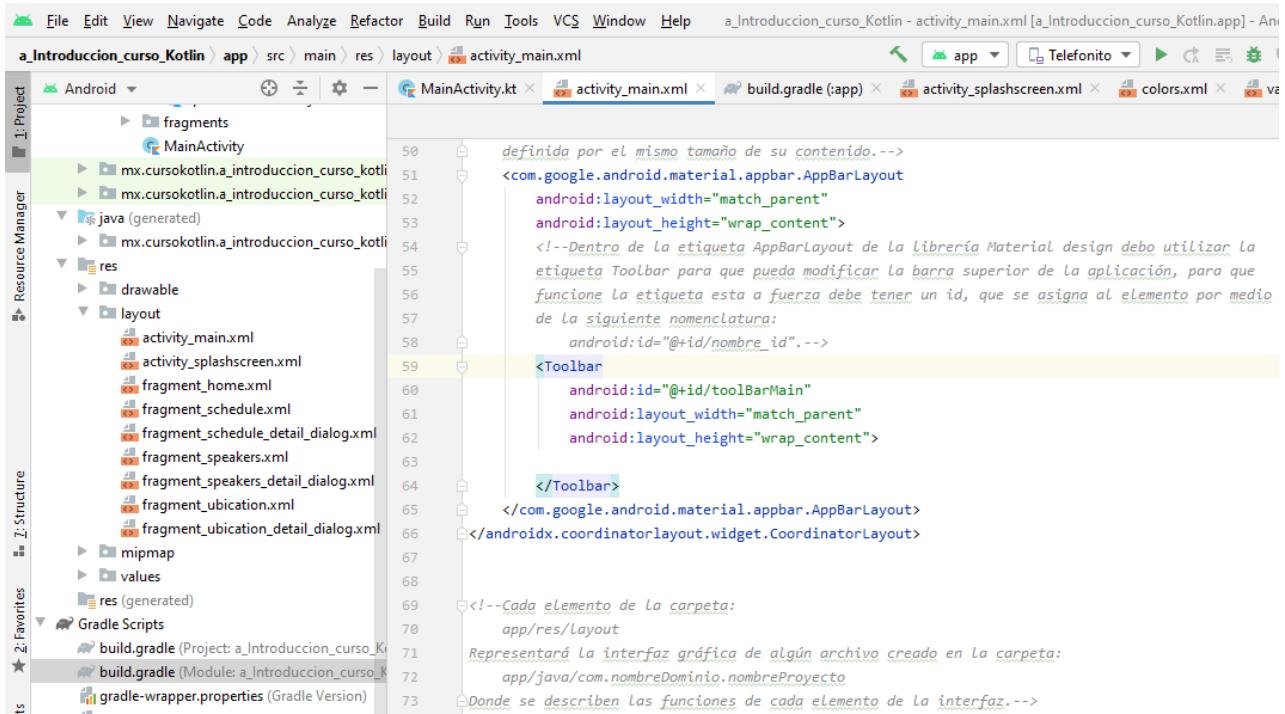
defaultConfig {
    /*En esta parte del código build.gradle (Module: NombreProyecto.app), 
    dominio de la app si llegara a pasar que al querer Lanzar la app a l
    dominio ya estaba siendo ocupado.*/
    //Dominio de la aplicación Android
    applicationId "mx.curso kotlin.a_introduccion_curso_kotlin"
    /*En esta parte del código build.gradle (Module: NombreProyecto.app),
    mínima del SDK que fue indicada al crear el proyecto y es importante
    el código se podrá correr en cierto porcentaje de los celulares con
    Android, en específico en este proyecto se utilizó la versión API 16
    porque es compatible con el 99.8% de los celulares Android, esto se
    directamente en esta parte del código.*/
    //Versión de API Android
    minSdkVersion 16
    /*Se puede realizar lo mismo con la versión SDK*/
    //Versión SDK
    targetSdkVersion 30
    /*versionCode y versionName sirven solamente cuando se sube la aplic
    el versionCode es el código interno y el versionName es la versión c
    al instalar o actualizar la aplicación*/
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
}
buildTypes {

```

Y luego debemos dar clic en el botón que aparece en la esquina superior derecha que dice Sync Now (este botón aparece solo cuando hayamos hecho un cambio en el archivo build.gradle (Module: NombreProyecto.app)).



La actualización del archivo build.gradle (Module: NombreProyecto.app) tarda un poco, pero al finalizar ya se habrá eliminado el error del código en la Actividad.



The screenshot shows the Android Studio interface with the project 'a\_introduccion\_curso\_Kotlin' open. The code editor displays the XML file 'activity\_main.xml'. Handwritten annotations in blue ink are overlaid on the code, explaining the structure of the XML elements:

- Line 50: 'definida por el mismo tamaño de su contenido.-->
- Line 51: '<com.google.android.material.appbar.AppBarLayout'
- Line 52: '    android:layout\_width="match\_parent"
- Line 53: '    android:layout\_height="wrap\_content">
- Line 54: '    <!--Dentro de la etiqueta AppBarLayout de la librería Material design debo utilizar la etiqueta Toolbar para que pueda modificar la barra superior de la aplicación, para que funcione la etiqueta esta a fuerza debe tener un id, que se asigna al elemento por medio de la siguiente nomenclatura:--&gt;</li>- Line 55: '        android:id="@+id/nombre\_id".-->
- Line 56: '        <Toolbar
- Line 57: '            android:id="@+id/toolBarMain"
- Line 58: '            android:layout\_width="match\_parent"
- Line 59: '            android:layout\_height="wrap\_content">
- Line 60: '        </Toolbar>
- Line 61: '    </com.google.android.material.appbar.AppBarLayout>
- Line 62: '  </androidx.coordinatorlayout.widget.CoordinatorLayout>
- Line 63: '  <!--Cada elemento de la carpeta:--&gt;</li>- Line 64: '  app/res/layout
- Line 65: '  Representará la interfaz gráfica de algún archivo creado en la carpeta:
- Line 66: '  app/java/com.nombreDominio.nombreProyecto
- Line 67: '  Donde se describen las funciones de cada elemento de la interfaz.-->

## Referencias:

Platzi, Gustavo Lizárraga, "Curso de Kotlin para Android", [Online], Available: <https://platzi.com/clases/1836-kotlin-android/26986-por-que-desarrollar-para-android-usando-kotlin/>

Platzi, Sinuhé Jaime Valencia, "Curso Básico de Diseño de Interfaces con Android Studio", [Online], Available: <https://platzi.com/clases/1825-interfaces-android/26254-ui-en-android-por-que-como/>