

UNIVERSIDAD DE EL SALVADOR - EDUCACIÓN A DISTANCIA
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACIÓN PARA DISPOSITIVOS MÓVILES
PDM-115



Guía de Laboratorio 4 - Kotlin Fragments, menú de navegación y View Models

Introducción

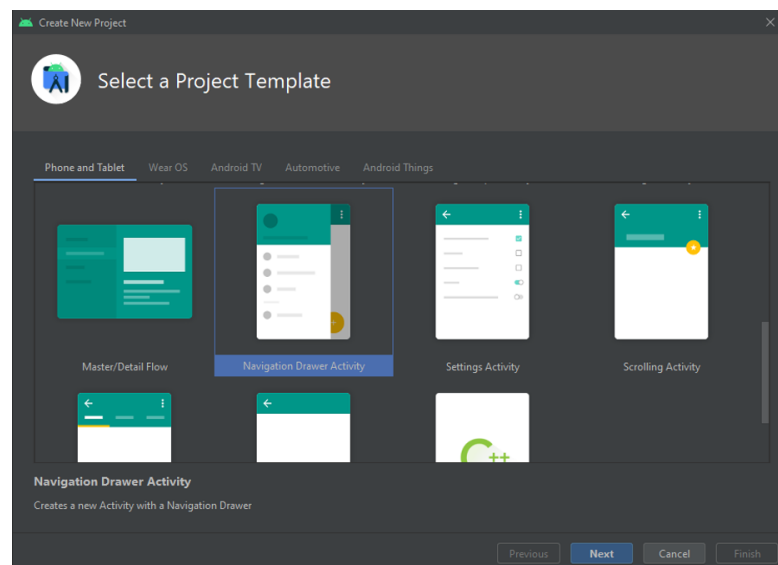
Esta práctica consistirá en crear un programa que contendrá una serie de Fragments que mostrarán el uso de cada uno de los controles básicos de Android (Button, TextView, EditText, CheckBox, etc) utilizando Kotlin

Objetivos

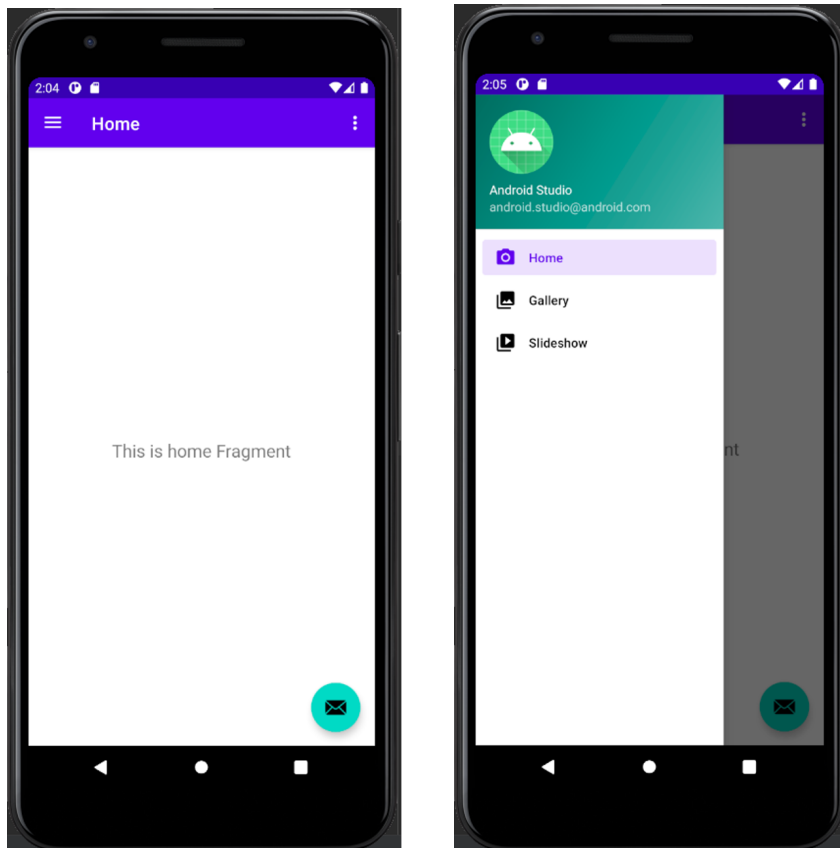
- Aprender la declaración e implementación de los controles básicos en la programación Android haciendo uso de layouts para realizar el diseño de la interfaz (en XML) y el uso de Fragments (en kotlin) para definir el manejo de estos.
- Incorporar nuevos controles de forma intuitiva siguiendo la lógica de controles de similar comportamiento.

Creación de proyecto

Crear un nuevo proyecto seleccionando la plantilla **Navigation Drawer Activity**, dar clic en Next, al presentar la siguiente pantalla colocar el nombre del proyecto y seleccionar el lenguaje Kotlin.



Lo ejecutamos para ver cómo esta configurado por default.



Con la configuración original se puede activar el menú default con presionar el botón de menú el cual contiene fragments para las opciones: **Home, Galery, Slideshow.**

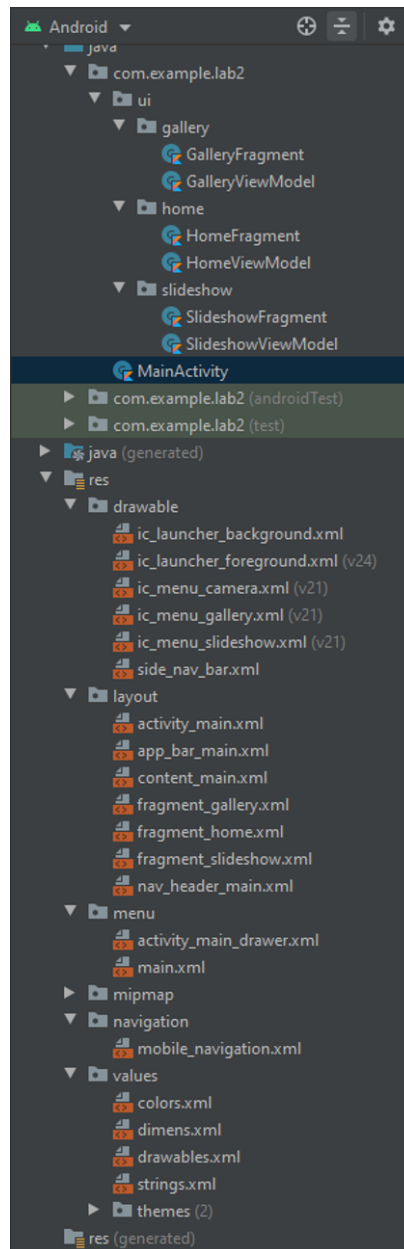
Nota:

Durante el desarrollo de esta guía se le proporciona código que hace referencia a *com.example.lab2*

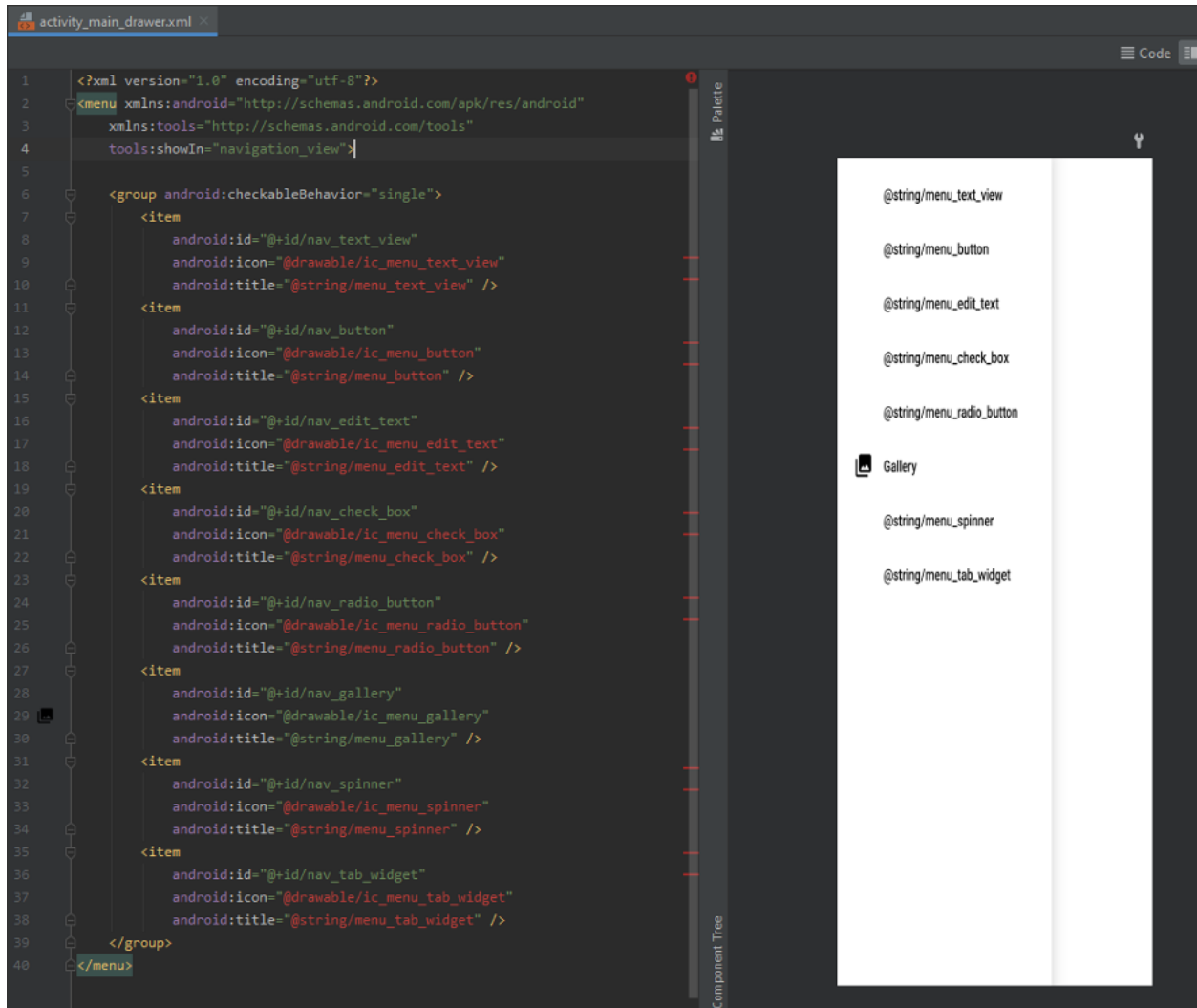
Asegúrese de cambiar esas referencias para que utilice el nombre de paquete que usted tenga en su proyecto.

Acondicionamiento del Menú (Principal)

La estructura del proyecto esta compuesta por los diferentes archivos para los 3 opciones del menú que se presentan en el proyecto inicial, se debe de observar de la siguiente manera.



Modificar el activity_main_drawer.xml como se muestra a continuación



Agregar los siguientes valores al archivo strings.xml

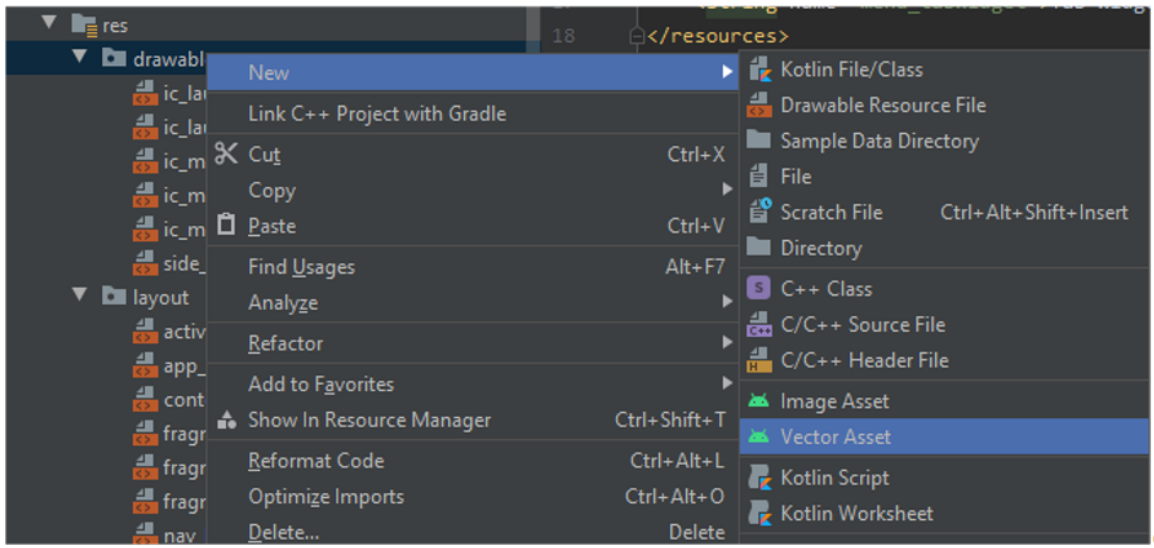
```

<!-- Nav menu strings -->
<string name="menu_home">Home</string>
<string name="menu_text_view">Text View</string>
<string name="menu_button">Button</string>
<string name="menu_edit_text">Edit Text</string>
<string name="menu_check_box">Check Box</string>
<string name="menu_radio_button">Radio Button</string>
<string name="menu_gallery">Gallery</string>
<string name="menu_spinner">Spinner</string>
<string name="menu_tab_widget">Tab Widget</string>

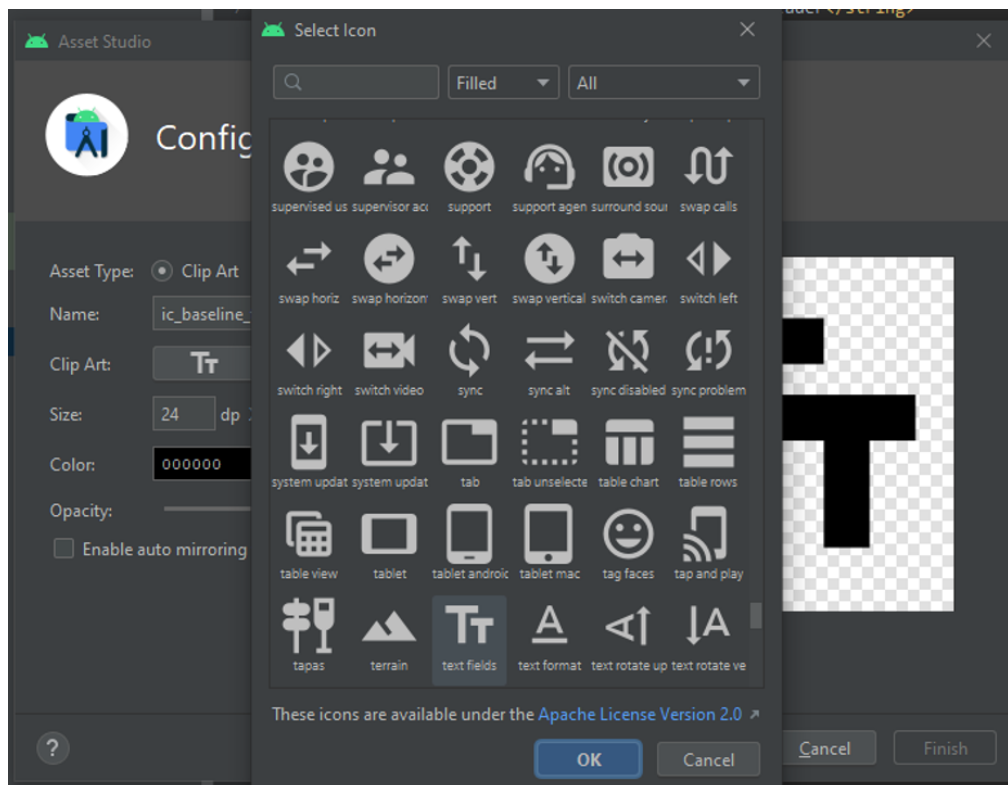
<!-- Fragments strings -->
<string name="fragment_label">%s Fragment</string>
  
```

Cambiar la iconografía de la app

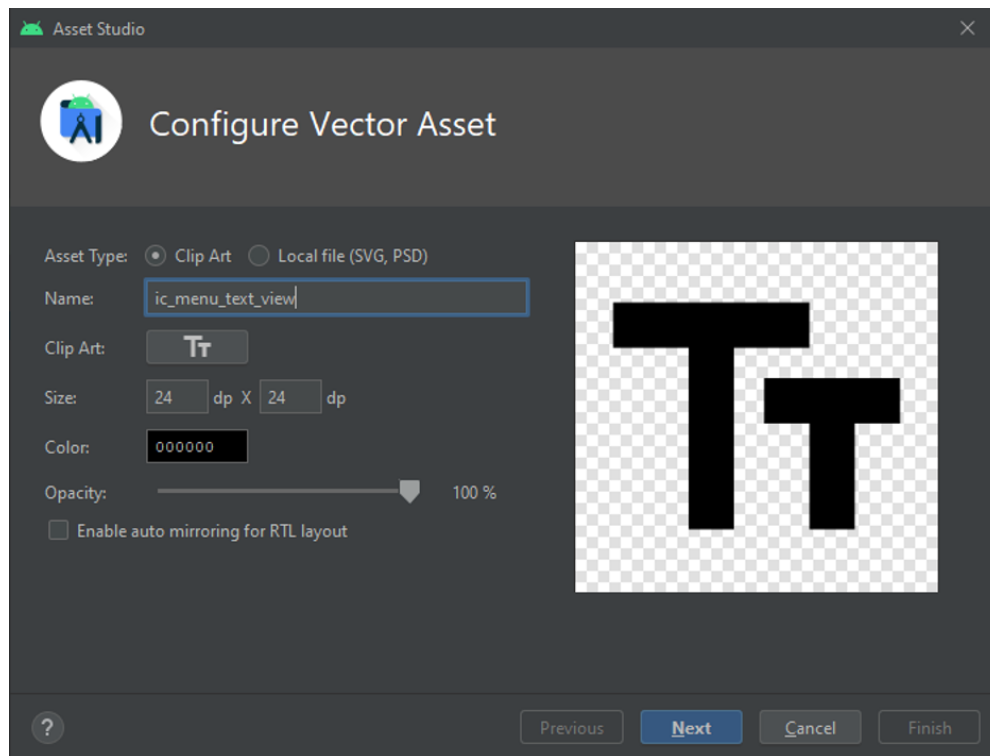
Para Agregar iconos vaya a la carpeta de drawable y presione clic derecho, New, Vector Asset



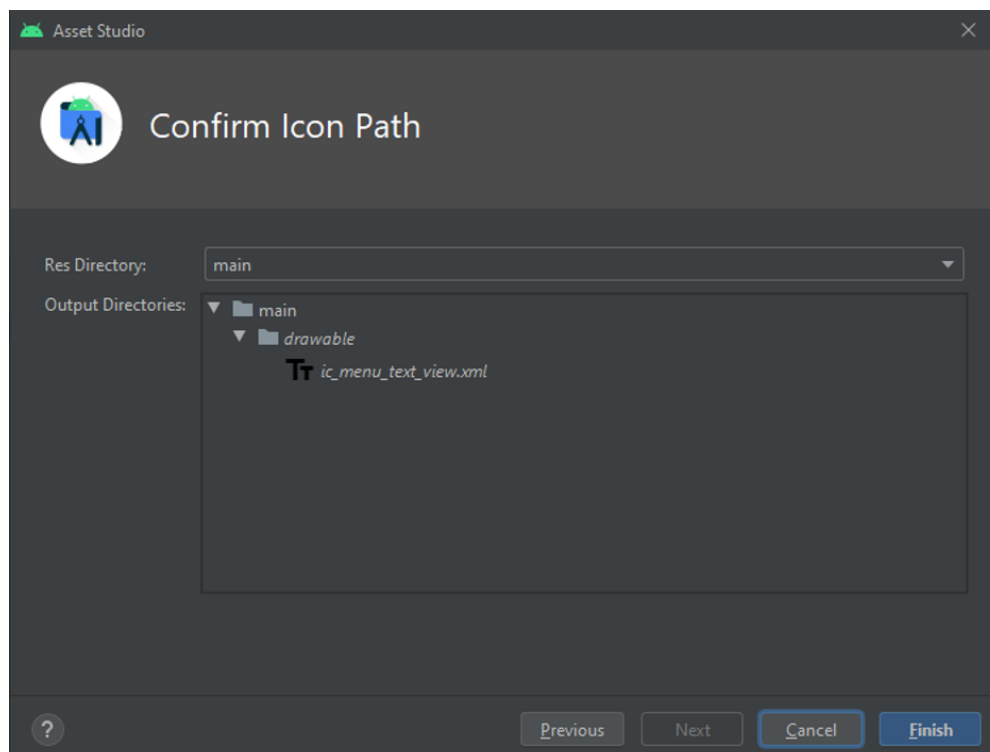
Presionamos doble click en el clip art...seleccionar la imagen...



Definir el nombre de acuerdo a un estándar `ic_menu_xxxxx` donde `xxxxx` será el nombre de la opción

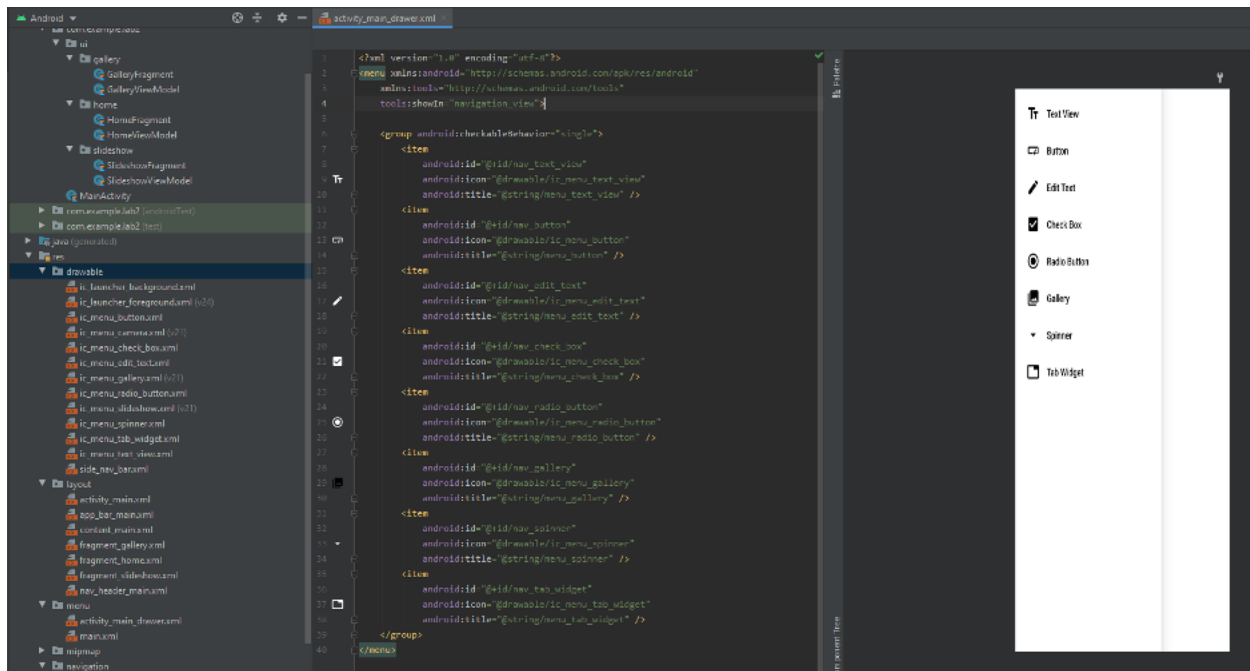


Presionamos next y elegimos la ruta donde se guardará el icono creado:



Presionamos **Finish** y realizamos esta tarea hasta completar los 4 iconos.

Ahora modificaremos el archivo `activity_main_drawer.xml` para agregar todas las opciones que estaremos utilizando, haciendo referencia a los iconos previamente creados y colocar los diferentes fragments a los cuales haremos referencias.



Enlazar Fragments correctos con la navegación

Luego se harán modificaciones para que los fragments se encuentren entre las opciones de navegación de la aplicación, para lo cual se modificara el archivo **mobile_navigation.xml**, para esto se modificara el archivo de la siguiente manera, agregando todos los fragments que hemos creado con sus respectivos, ubicaciones de archivos y fragments.

```
<?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/mobile_navigation"
    app:startDestination="@+id/nav_text_view">

    <fragment
        android:id="@+id/nav_text_view"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_text_view"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_button"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_button"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_edit_text"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_edit_text"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_check_box"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_check_box"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_radio_button"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_radio_button"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_gallery"
        android:name="com.example.lab2.ui.gallery.GalleryFragment"
        android:label="@string/menu_gallery"
        tools:layout="@layout/fragment_gallery" />

    <fragment
        android:id="@+id/nav_spinner"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_spinner"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_tab_widget"
        android:name="com.example.lab2.ui.home.HomeFragment"
        android:label="@string/menu_tab_widget"
        tools:layout="@layout/fragment_home" />

</navigation>
```

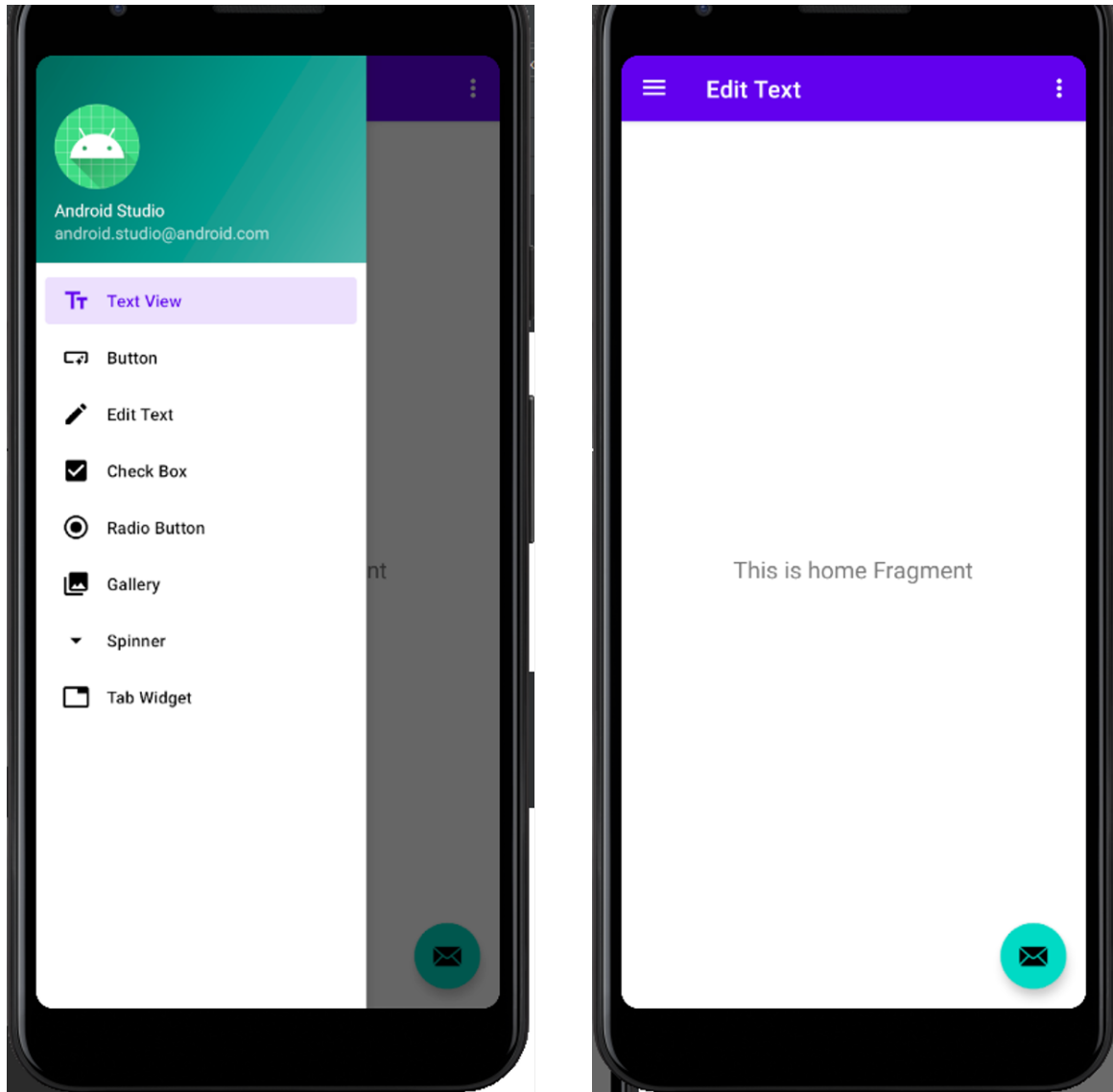

Crear un arreglo que contenga las referencias para todos los fragments necesarios, correspondientes a las opciones de menú.

```
private lateinit var appBarConfiguration: AppBarConfiguration
private val topLvlDest: Set<Int> = setOf(
    R.id.nav_text_view,
    R.id.nav_button,
    R.id.nav_edit_text,
    R.id.nav_check_box,
    R.id.nav_radio_button,
    R.id.nav_gallery,
    R.id.nav_spinner,
    R.id.nav_tab_widget
)
```

En la siguiente instrucción se hará referencia al arreglo antes creado, permitiendo así el cambio de pantallas por parte del menú de navegación

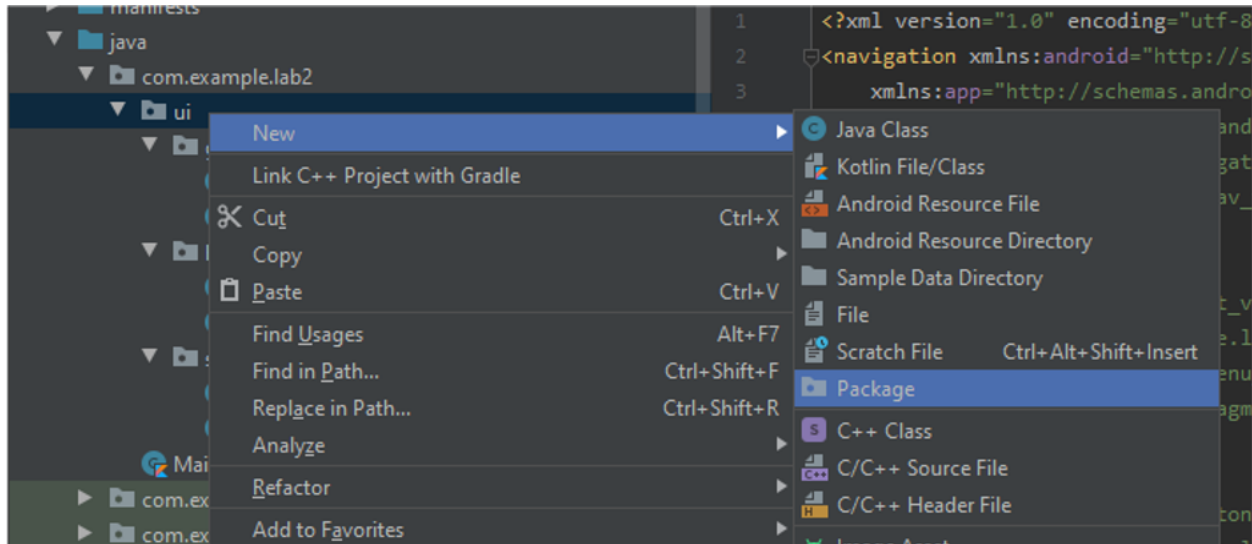
```
// Passing each menu ID as a set of Ids because each
// menu should be considered as top level destinations.
appBarConfiguration = AppBarConfiguration(topLvlDest, drawerLayout)
```

Así es como se verá al ejecutar la aplicación:

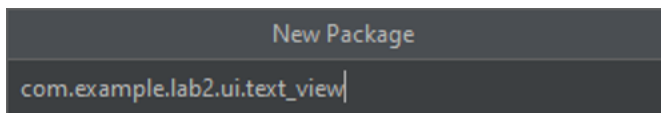


Editar el primer layout (text_view)

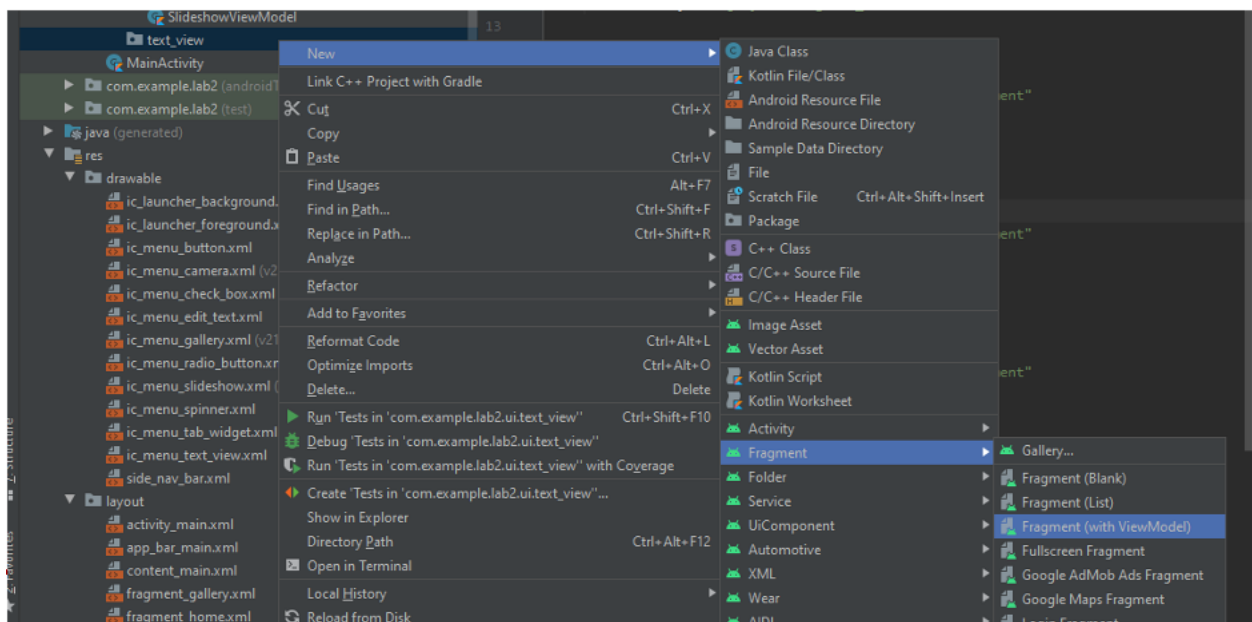
Para iniciar la creación de los diferentes layouts, se debe de crear un paquete en la carpeta de **ui** ubicada en **java** y dentro del paquete del proyecto. Seleccione la carpeta y al hacer click derecho ubicarse en nuevo (**New**) y crear un paquete (**Package**)



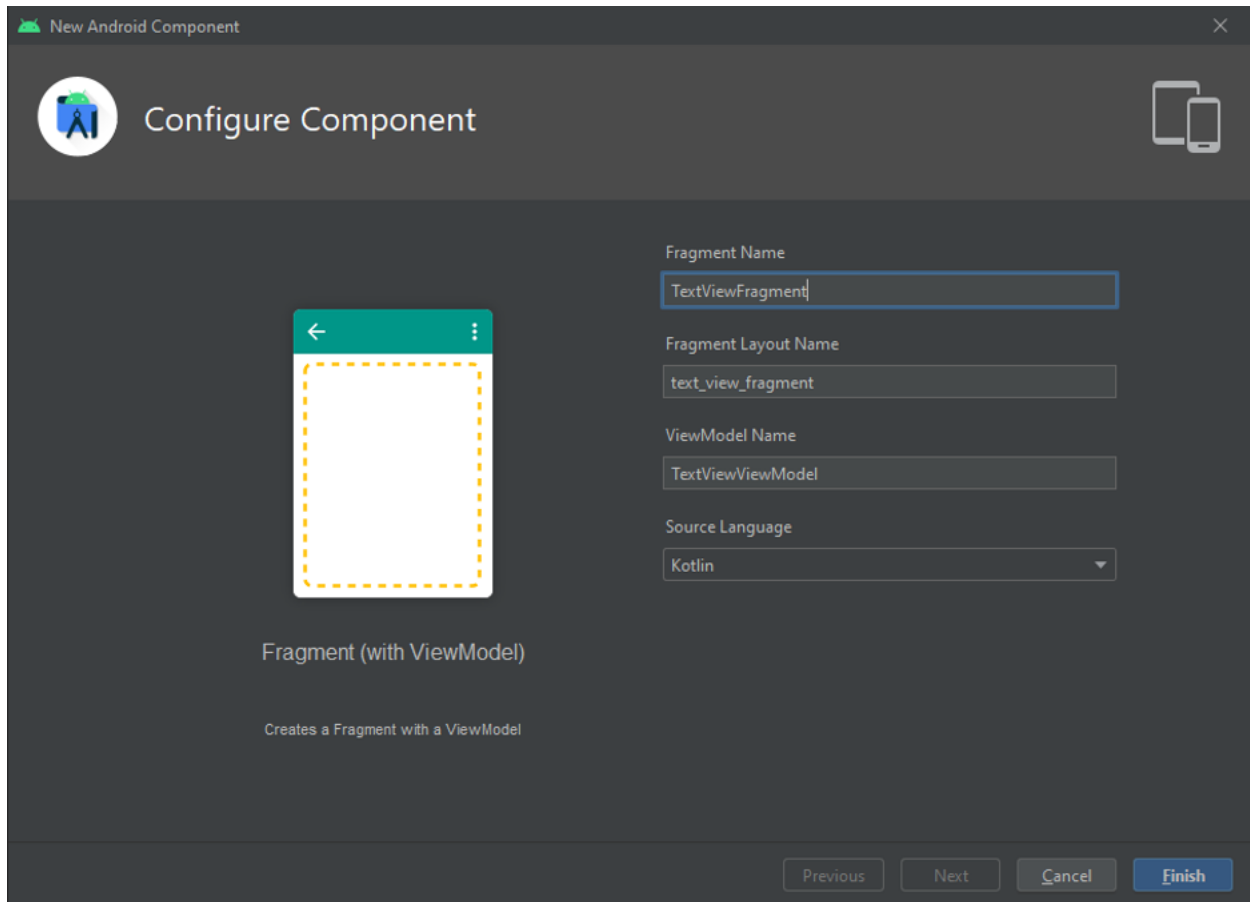
Al seleccionar la opción se presentara una ventana en la cual se debe de ingresar el nombre del paquete, en este caso le dejamos el prefijo del proyecto y agregamos el nombre del paquete (text_view).



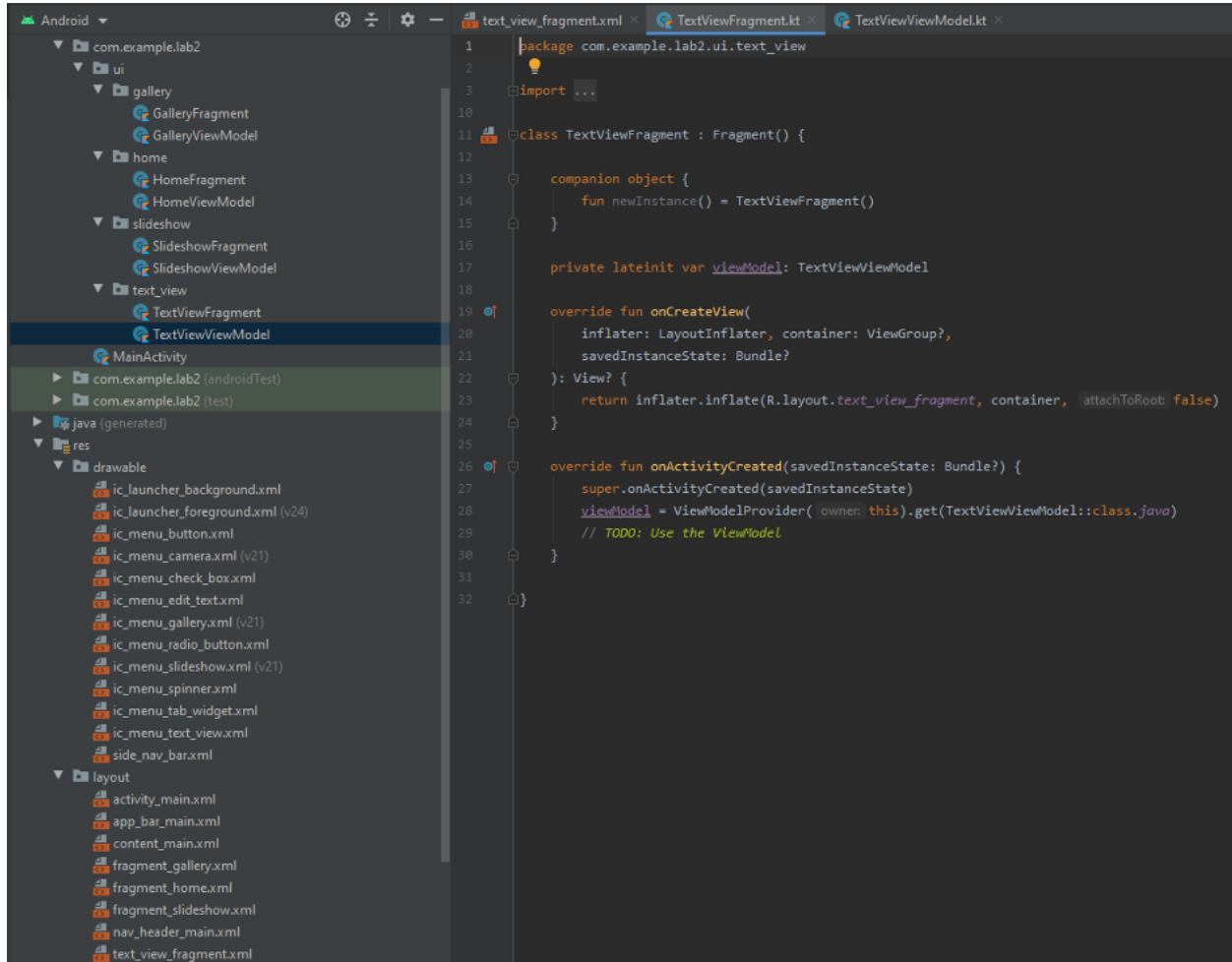
Al tener el paquete creado, ubicarse en el paquete y crear un nuevo **fragment**, este tiene que ser con el formato de **ViewModel**.



Al seleccionar la opción de creación de **Fragment (with ViewModel)**, se presentara una ventana para elegir la configuración inicial del fragment, en la cual colocaremos el nombre del fragment (**TextViewFragment**) y elegiremos el lenguaje de programación (**Kotlin**).

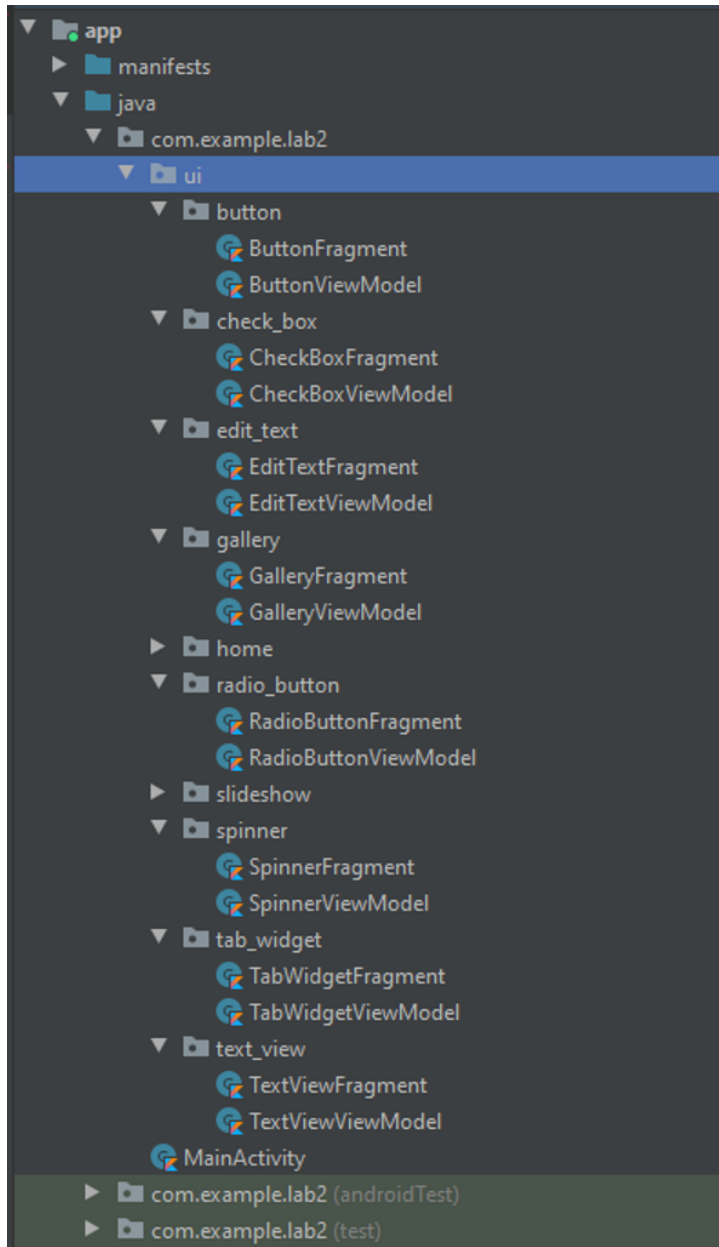


Al crear el fragment se puede observar que se crean 2 archivos .kt adentro del paquete creado previamente, de los cuales uno corresponde al código relacionado con la vista del fragment, adonde se crea dicha vista, y el otro es el archivo ViewModel, con el cual estaremos trabajando posteriormente, en el primer archivo se puede observar una referencia hacia el archivo ViewModel correspondiente.

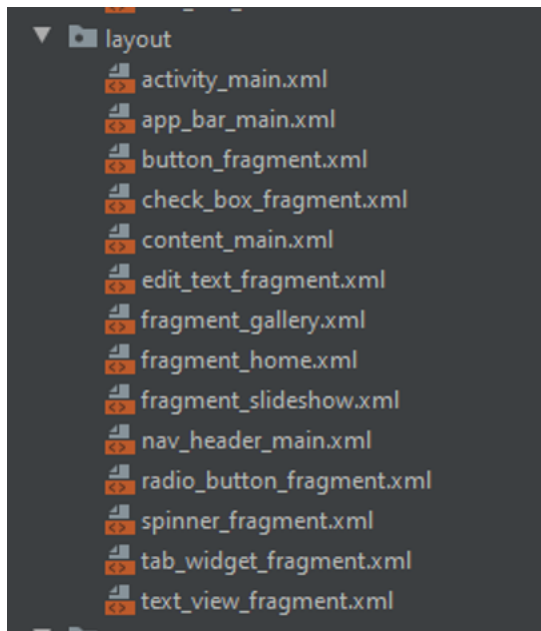


Luego de haber creado el primer fragment, proceder a crear todos los fragments faltantes que estaremos utilizando, siempre realizando el mismo proceso de crear paquete y luego crear el fragment con ViewModel.

Crear los siguientes fragmentes y paquetes.



Observar que al momento de crear los archivos Kotlin en los paquetes correspondientes, se crean de igual manera los archivos de la vista, archivos XML relacionadas a los archivos Kotlin que se crearon previamente. La carpeta de Layout debe de lucir como la siguiente.



Luego de crear todos estos fragments, asegurarse de que el archivo **mobile_navigation.xml** este correcto, que cada referencia a las clases estén en **android:name** y los layouts están correctamente relacionados en **tools:layout**, como se muestra a continuación.

```
<?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/mobile_navigation"
    app:startDestination="@+id/nav_text_view">

    <fragment
        android:id="@+id/nav_text_view"
        android:name="com.example.lab2.ui.text_view.TextViewFragment"
        android:label="@string/menu_text_view"
        tools:layout="@layout/text_view_fragment" />

    <fragment
        android:id="@+id/nav_button"
        android:name="com.example.lab2.ui.button.ButtonFragment"
        android:label="@string/menu_button"
        tools:layout="@layout/button_fragment" />

    <fragment
        android:id="@+id/nav_edit_text"
        android:name="com.example.lab2.ui.edit_text.EditTextFragment"
        android:label="@string/menu_edit_text"
        tools:layout="@layout/edit_text_fragment" />
```

```

<fragment
    android:id="@+id/nav_check_box"
    android:name="com.example.lab2.ui.check_box.CheckBoxFragment"
    android:label="@string/menu_check_box"
    tools:layout="@layout/check_box_fragment" />

<fragment
    android:id="@+id/nav_radio_button"
    android:name="com.example.lab2.ui.radio_button.RadioButtonFragment"
    android:label="@string/menu_radio_button"
    tools:layout="@layout/radio_button_fragment" />

<fragment
    android:id="@+id/nav_gallery"
    android:name="com.example.lab2.ui.gallery.GalleryFragment"
    android:label="@string/menu_gallery"
    tools:layout="@layout/fragment_gallery" />

<fragment
    android:id="@+id/nav_spinner"
    android:name="com.example.lab2.ui.spinner.SpinnerFragment"
    android:label="@string/menu_spinner"
    tools:layout="@layout/spinner_fragment" />

<fragment
    android:id="@+id/nav_tab_widget"
    android:name="com.example.lab2.ui.tab_widget.TabWidgetFragment"
    android:label="@string/menu_tab_widget"
    tools:layout="@layout/tab_widget_fragment" />
</navigation>

```

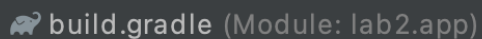
Al utilizar el patrón de diseño MVVM (ModelViewViewModel), la información que la vista muestra esta alojada en el ViewModel, por lo que necesitamos una manera de saber cuándo ha ocurrido algún cambio en esa información para poder actualizar nuestra vista, o una manera de poder vincular directamente una propiedad del view model a la vista; esto es lo que podríamos conocer como DataBinding.

Para poder ocupar databinding en nuestra aplicación necesitamos activarlo, agregando lo siguiente en nuestra configuración de nuestro modulo de gradle, dentro de la etiqueta android. (**build.gradle(Module:)**)

```

buildFeatures {
    dataBinding true
}

```



```

build.gradle (Module: lab2.app)

```

Activando el data binding, el gradle generara automáticamente las clases de vinculo (binding) y así no tenemos que preocuparnos por crear nuestras propias formas de estar escuchando los cambios que suceden dentro de los view models.

Ahora proceda a modificar los archivos relacionados con el layout de text_view. Modifique el archivo XML **text_view_fragment.xml** con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <variable
            name="vm"
            type="com.example.lab2.ui.text_view.TextViewViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.text_view.TextViewFragment">

        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:textAlignment="center"
            android:textSize="20sp"
            android:text="@{String.format(@string/fragment_label, @string/
menu_text_view)}"
            android:textColor="@{vm.textColor}"
            android:onClick="@{vm::changeTextColor}"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</layout>
```

En este código se puede observar que se utiliza la variable **vm** para determinar qué es el componente **ViewModel** del fragment **TextViewFragment**, con este logramos vincular variables del ViewModel (**vm.textColor**), como también funciones (**vm::changeTextColor**) que se encargaran de actualizar la vista de igual manera desde ViewModel.

Ahora se debe de modificar el archivo correspondiente al componente ViewModel del TextViewFragment, llamado **TextViewViewModel**, en este archivo ingresaremos el siguiente código.

```
package com.example.lab2.ui.text_view

import android.graphics.Color
import android.view.View
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class TextViewViewModel : ViewModel() {

    private val _textColor = MutableLiveData<Int>().apply {
        value = Color.rgb(0,0,0);
    }

    val textColor: LiveData<Int> = _textColor

    fun changeTextColor(view: View) {
        _textColor.value = Color.rgb(
            (0..255).random(),
            (0..255).random(),
            (0..255).random()
        )
    }
}
```

En el código previo, se puede observar el uso del LiveData, el cual es un tipo de objeto el cual se encarga de notificar los cambios a la vista que esta vinculada, y así realizar todas las actualizaciones necesarias, lo cual nos facilita el estar escuchando todos los cambios que sucedan en el ViewModel y gracias a eso modificar la vista como se desea o sea conveniente.

Posteriormente se modificara el archivo de **TextViewFragment** para enlazar la variable de **vm** que se encuentra en la vista con la clase ViewModel con la que se estaba trabajando.

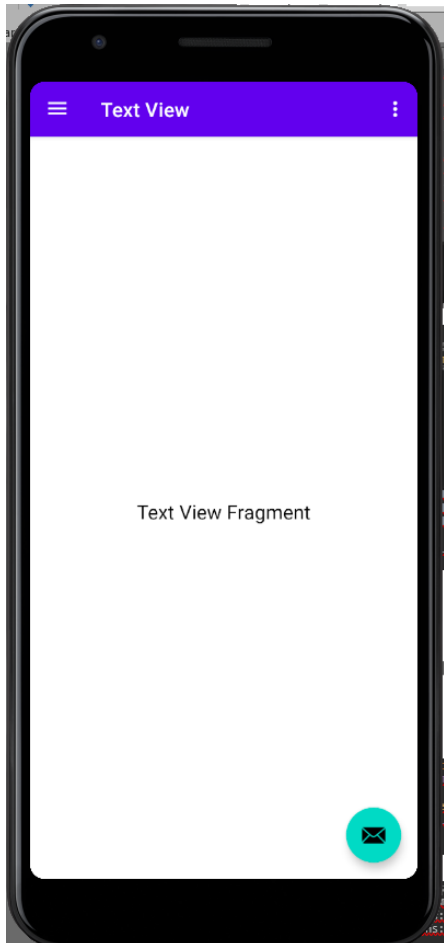
```
package com.example.lab2.ui.text_view
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import com.example.lab2.databinding.TextViewFragmentBinding

class TextViewFragment : Fragment() {

    private lateinit var textViewViewModel: TextViewViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        textViewViewModel = ViewModelProvider(activity ?: this).get(TextViewViewModel::class.java)
        val binding = TextViewFragmentBinding.inflate(inflater, container, false)
        binding.lifecycleOwner = this
        binding.vm = textViewViewModel
        return binding.root
    }
}
```

Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente. Si hay algún inconveniente intenta hacer Sync Now desde el archivo Gradle para generar los archivos de data binding.



Modificar todos los Fragments restantes

Editar componente Button (button_fragment, ButtonFragment, ButtonViewModel)

Para trabajar con el siguiente componente, se modificara el archivo de **button_fragment** para modificar el layout con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">
    <data>
        <import type="android.view.View"/>
        <variable
            name="vm"
            type="com.example.lab2.ui.button.ButtonViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.button.ButtonFragment">

        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:text="@{String.format(@string/fragment_label, @string/menu_button)}"
            android:textAlignment="center"
            android:textSize="20sp"
            android:visibility="@{vm.isOn ? View.VISIBLE : View.GONE}"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ToggleButton
            android:id="@+id/toggleButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="ToggleButton"
            android:checked="@{vm.isOn}"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>
```

Se observa en el código anterior, que se relacionan diferentes características de la vista (**View.VISIBLE**) y también variables de el view model correspondiente al ButtonFragment (**vm.isOn**).

Luego procesa a modificar el archivo **ButtonViewModel** con el siguiente código.

```
package com.example.lab2.ui.button

import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class ButtonViewModel : ViewModel() {

    private val _isOn = MutableLiveData<Boolean>().apply {
        value = true
    }
    val isOn: LiveData<Boolean> = _isOn

    fun toggle(value: Boolean) {
        _isOn.value = value
    }
}
```

Se observa en el código anterior como la variable **_isOn** es MutableLiveData y **isOn** es de tipo LiveData, estas nos sirven para controlar el estado del toggle del botón, para ver si esta activo o no el botón, mostrando u ocultando así el **TextView**.

Luego modifique el archivo de **ButtonFragment**, agregue el siguiente código, en este se puede observar que se hace el enlace entre el view model a la clase **ButtonFragment**, como también se establece el listener de cuándo cambie el toggle button.

```
package com.example.lab2.ui.button
import ...

class ButtonFragment : Fragment() {
    private lateinit var viewModel: ButtonViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        viewModel = ViewModelProvider(activity ?: this).get(ButtonViewModel::class.java)
        val binding = ButtonFragmentBinding.inflate(inflater, container, false)
        binding.lifecycleOwner = this
        binding.vm = viewModel
        val toggle: ToggleButton = binding.root.findViewById(R.id.toggleButton)
        toggle.setOnCheckedChangeListener { _, isChecked -> viewModel.toggle(isChecked)}
        return binding.root
    }
}
```

Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente.



Editar componente EditText (edit_text_fragment, EditTextFragment, EditTextViewModel)

Para seguir con la pantalla de EditText proceda a agregar al archivo de **string.xml** las siguientes cadenas de caracteres.

```
<!-- Misc -->
<string name="message_init">He pensado un número entre %d y %d, cuál crees que sea?</string>
<string name="message_greater">El número que pensé es mayor que %d</string>
<string name="message_lower">El número que pensé es menor que %d</string>
<string name="message_congrats">Felicitades! el número que había pensado era %d</string>
<string name="message_try_again">Estoy seguro que regresas porque quieres volver a intentar</string>
<string name="message_restart">Ya ganaste! Quieres que piense en otro número?</string>
<string name="message_restart_2">Si es asi toca el TextView</string>
<string name="message_error">Tienes que digitar un número válido!</string>
<string name="edit_text_hint">Ingresa un número</string>
<string name="try_btn">Probar número</string>
```

Para trabajar con el siguiente componente, se modificara el archivo de **edit_text_fragment** para modificar el layout con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">
    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.edit_text.EditTextFragment">
        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:text="@{String.format(@string/fragment_label, @string/menu_edit_text)}"
            android:textAlignment="center"
            android:textSize="20sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
        <EditText
            android:id="@+id/editText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="@string/edit_text_hint"
            android:importantForAutofill="no"
            android:inputType="number"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
```

```

        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:text="@string/try_btn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />
</androidx.constraintlayout.widget.ConstraintLayout>
</layout>

```

Ahora proceda a modificar el archivo viewModel relacionado con el EditText, llamado EditTextViewModel, este estará compuesto de la siguiente manera:

```

package com.example.lab2.ui.edit_text

import android.content.Context
import android.widget.Toast
import androidx.lifecycle.ViewModel
import com.example.lab2.R

class EditTextViewModel : ViewModel() {

    private val range = (1..100)
    private var desired_num = range.random()
    private var last_try = -1
    private lateinit var context: Context

    fun init(ctxt: Context?) {
        if (ctxt != null){
            this.context = ctxt
            when {
                last_try < 0 -> {
                    Toast.makeText(context, context.getString(R.string.message_init, range.first, range.last),
                        Toast.LENGTH_LONG).show()
                }
                last_try == desired_num -> {
                    Toast.makeText(context, R.string.message_restart, Toast.LENGTH_SHORT).show()
                    Toast.makeText(context, R.string.message_restart_2, Toast.LENGTH_LONG).show()
                }
                else -> {
                    Toast.makeText(context, R.string.message_try_again, Toast.LENGTH_LONG).show()
                }
            }
        }
    }

    fun tryNum(num: Int) {
        if (context != null) {
            when (last_try) {
                desired_num -> {
                    Toast.makeText(context, R.string.message_restart, Toast.LENGTH_SHORT).show()
                    Toast.makeText(context, R.string.message_restart_2, Toast.LENGTH_LONG).show()
                }
            }
        }
    }
}

```



```

        else -> {
            when {
                num < range.first || num > range.last -> {
                    Toast.makeText(context, R.string.message_error, Toast.LENGTH_SHORT).show()
                    return
                }
                num == desired_num -> {
                    Toast.makeText(context, context.getString(R.string.message_congrats, num),
Toast.LENGTH_LONG).show()
                }
                num < desired_num -> {
                    Toast.makeText(context, context.getString(R.string.message_greater, num),
Toast.LENGTH_SHORT).show()
                }
                num > desired_num -> {
                    Toast.makeText(context, context.getString(R.string.message_lower, num),
Toast.LENGTH_SHORT).show()
                }
            }
            last_try = num
        }
    }
}

fun reset() {
    desired_num = range.random()
    last_try = -1
    Toast.makeText(context, context.getString(R.string.message_init, range.first, range.last),
Toast.LENGTH_LONG).show()
}
}

```

En el código anterior se pueden observar diferentes funciones para lograr jugar en la aplicación, adivinando un numero que es generado al azar (**private var desired_num = range.random()**) y al momento que el usuario ingrese un dato, este le dará pistas por medio de toast informando si el numero es mayor o menor que el que se ha ingresado.

Ahora proceda a modificar el archivo **EditTextFragment**, con el siguiente código que muestra cómo se enlazan los diferentes componentes con el funciones del archivo viewModel (**viewModel.tryNum**).

```

package com.example.lab2.ui.edit_text
import ...

class EditTextFragment : Fragment() {

    private lateinit var viewModel: EditTextViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        viewModel = ViewModelProvider(activity ?: this).get(EditTextViewModel::class.java)
        val binding = EditTextFragmentBinding.inflate(inflater, container, false)
        binding.lifecycleOwner = this

        val tryBtn: Button = binding.root.findViewById(R.id.button)
    }
}

```

```

    val editText: EditText = binding.root.findViewById(R.id.editText)
    tryBtn.setOnClickListener() {
        viewModel.tryNum(editText.text.toString().toIntOrNull() ?: -1)
    }
    val textView: TextView = binding.root.findViewById(R.id.text_view)
    textView.setOnClickListener() {
        viewModel.reset()
    }
    return binding.root
}
override fun onStart() {
    viewModel.init(activity)
    super.onStart()
}
}

```

Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente.



Editar componente CheckBox (check_box_fragment, CheckBoxFragment, CheckBoxViewModel)

Para trabajar con el siguiente componente, agregamos las siguientes cadenas de caracteres al archivo **string.xml**

```
<string name="red">Rojo</string>
<string name="green">Verde</string>
<string name="blue">Azul</string>
```

Luego se modificara el archivo de **check_box_fragment** para modificar el layout con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <variable
            name="vm"
            type="com.example.lab2.ui.check_box.CheckBoxViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.check_box.CheckBoxFragment">

        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:text="@{String.format(@string/fragment_label, @string/menu_check_box)}"
            android:textAlignment="center"
            android:textSize="20sp"
            android:textColor="@{vm.color}"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <CheckBox
            android:id="@+id/checkBox"
            android:layout_width="100dp"
            android:layout_height="wrap_content"
            android:layout_marginBottom="32dp"
            android:text="@string/red"
```

```

    android:checked="@{vm.cb1State}"
    app:layout_constraintBottom_toTopOf="@+id/checkbox2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.2"
    app:layout_constraintStart_toStartOf="parent" />

```

<CheckBox

```

    android:id="@+id/checkbox2"
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:text="@string/green"
    android:checked="@{vm.cb2State}"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.2"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

<CheckBox

```

    android:id="@+id/checkbox3"
    android:layout_width="100dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:text="@string/blue"
    android:checked="@{vm.cb3State}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.2"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/checkbox2" />

```

<EditText

```

    android:id="@+id/editTextNumber"
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="32dp"
    android:ems="10"
    android:hint="@string/edit_text_hint"
    android:inputType="number"
    android:enabled="@{vm.cb1State}"
    android:text="@{vm.redValue.toString()}"
    android:importantForAutofill="no"
    app:layout_constraintBottom_toTopOf="@+id/editTextNumber2"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/checkbox" />

```

<EditText

```

    android:id="@+id/editTextNumber2"
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="@string/edit_text_hint"
    android:inputType="number"
    android:enabled="@{vm.cb2State}"
    android:text="@{vm.greenValue.toString()}"

```

```

    android:importantForAutofill="no"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/checkBox2"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<EditText
    android:id="@+id/editTextNumber3"
    android:layout_width="160dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:ems="10"
    android:hint="@string/edit_text_hint"
    android:inputType="number"
    android:enabled="@{vm.cb3State}"
    android:text="@{vm.blueValue.toString()}"
    android:importantForAutofill="no"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/checkBox3"
    app:layout_constraintTop_toBottomOf="@+id/editTextNumber2" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

```

</layout>

```

En este XML se puede observar que el ViewModel (**vm**) se encargara de los estados de los diferentes check box(**vm.cb1State**), como tambien el color que se le asignara al textview (**vm.color**).

Luego proceda a modificar el archivo view model correspondiente al check box, CheckBoxViewModel.

```

package com.example.lab2.ui.check_box

```

```

import android.graphics.Color
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

```

```

class CheckBoxViewModel : ViewModel() {
    private val _checkBoxStates: MutableLiveData<MutableLiveData<Boolean>> = MutableLiveDataOf(
        MutableLiveData<Boolean>().apply { value = false },
        MutableLiveData<Boolean>().apply { value = false },
        MutableLiveData<Boolean>().apply { value = false }
    )
    val cb1State: LiveData<Boolean> = _checkBoxStates[0]
    val cb2State: LiveData<Boolean> = _checkBoxStates[1]
    val cb3State: LiveData<Boolean> = _checkBoxStates[2]

    private val _rgb: MutableLiveData<MutableLiveData<Int>> = MutableLiveDataOf(
        MutableLiveData<Int>().apply { value = 0 },
        MutableLiveData<Int>().apply { value = 0 },
        MutableLiveData<Int>().apply { value = 0 }
    )
}

```

```

val redValue: LiveData<Int> = _rgb[0]
val greenValue: LiveData<Int> = _rgb[1]
val blueValue: LiveData<Int> = _rgb[2]

private val _color = MutableLiveData<Int>().apply {
    value = Color.rgb(0, 0, 0)
}
val color: LiveData<Int> = _color

fun toggleCheckBox(index: Int, value: Boolean) {
    _checkBoxStates[index].value = value
    updateColor()
}

fun setColorChannel(index: Int, value: Int?) {
    _rgb[index].value = value
    updateColor()
}

private fun updateColor() {
    val red = if (_checkBoxStates[0].value == true) _rgb[0].value ?: 0 else 0
    val green = if (_checkBoxStates[1].value == true) _rgb[1].value ?: 0 else 0
    val blue = if (_checkBoxStates[2].value == true) _rgb[2].value ?: 0 else 0
    _color.value = Color.rgb(red, green, blue)
}
}

```

Se observa que en el view model se declaran los diferentes estados de los checkbox (activo o no activo) como también el valor de cada uno de los textfields relacionados a dichos checkbox, relacionando así los 3 checkbox con los colores RGB (**redValue**, **greenValue**, **blueValue**).

Para finalizar el checkbox, modifique el archivo de CheckBoxFragment con el siguiente código.

```

package com.example.lab2.ui.check_box

import androidx.lifecycle.ViewModelProvider
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.widget.doAfterTextChanged
import com.example.lab2.databinding.CheckBoxFragmentBinding

class CheckBoxFragment : Fragment() {

    private lateinit var viewModel: CheckBoxViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {

```

```

viewModel = ViewModelProvider(activity ?: this).get(CheckBoxViewModel::class.java)
val binding = CheckBoxFragmentBinding.inflate(inflater, container, false)
binding.lifecycleOwner = this
binding.vm = viewModel

binding.checkBox.setOnCheckedChangeListener { _, isChecked -> viewModel.toggleCheckBox(0,
isChecked)}
binding.checkBox2.setOnCheckedChangeListener { _, isChecked -> viewModel.toggleCheckBox(1,
isChecked)}
binding.checkBox3.setOnCheckedChangeListener { _, isChecked -> viewModel.toggleCheckBox(2,
isChecked)}

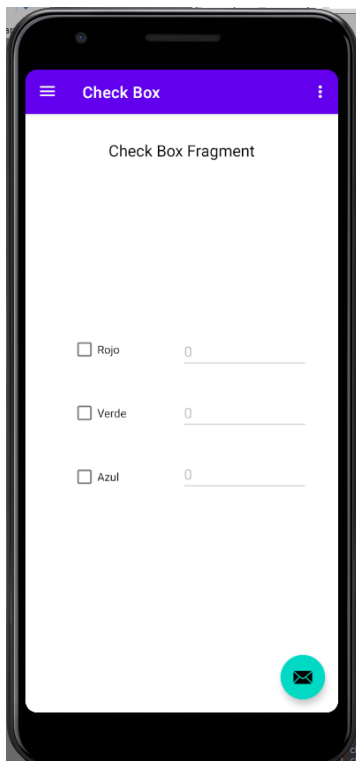
binding.editTextNumber.doAfterTextChanged { viewModel.setColorChannel(0,
it.toString().toIntOrNull()) }
binding.editTextNumber2.doAfterTextChanged { viewModel.setColorChannel(1,
it.toString().toIntOrNull()) }
binding.editTextNumber3.doAfterTextChanged { viewModel.setColorChannel(2,
it.toString().toIntOrNull()) }

return binding.root
}
}

```

En el código anterior, se observa que cada uno de los edittext se le declara un listener el cual al recibir un cambio del valor, llama a las funciones del viewModel que cambian el estado de la vista.

Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente.



Editar componente RadioButton (radio_button_fragment, RadioButtonFragment, RadioButtonViewModel)

Para trabajar con el siguiente componente, agregamos las siguientes cadenas de caracteres al archivo **string.xml**

```
<string name="victories">Victorias %d</string>
<string name="losses">Derrotas %d</string>
<string name="draws">Empates %d</string>
<string name="rock">Piedra</string>
<string name="paper">Papel</string>
<string name="scissor">Tijera</string>
<string name="won">Ganaste! mi decisión fue %s</string>
<string name="lost">Jaja te gané! mi decisión fue %s</string>
<string name="draw">Empatamos! decidimos lo mismo</string>
```

Para trabajar con el siguiente componente, se modificara el archivo de **radio_button_fragment** para modificar el layout con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <import type="android.view.View"/>
        <variable
            name="vm"
            type="com.example.lab2.ui.radio_button.RadioButtonViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.radio_button.RadioButtonFragment">

        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:text="@{String.format(@string/fragment_label, @string/menu_radio_button)}"
            android:textAlignment="center"
            android:textSize="20sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <TextView
```



```

        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:text="@{String.format(@string/victories, vm.victories)}"
        app:layout_constraintEnd_toStartOf="@+id/textView3"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:text="@{String.format(@string/losses, vm.losses)}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:text="@{String.format(@string/draws, vm.draws)}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/textView3"
    app:layout_constraintTop_toTopOf="parent" />
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">
    <RadioButton
        android:id="@+id/radioButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/rock" />

    <RadioButton
        android:id="@+id/radioButton2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/paper" />

    <RadioButton
        android:id="@+id/radioButton3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/scissor" />
</RadioGroup>
</androidx.constraintlayout.widget.ConstraintLayout>
</layout>

```

Para este componente crearemos un juego sencillo de piedra papel o tijera utilizando radio buttons. Proceder a modificar el archivo **RadioButtonViewModel** con el siguiente código, este esta compuesto por arreglos y validaciones para verificar los casos de victoria del jugador. Llevo el estado de los resultados (Victoria, derrota, empate) por medio de variables LiveData.

```
package com.example.lab2.ui.radio_button

import android.content.Context
import android.widget.Toast
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.lab2.R

class RadioButtonViewModel : ViewModel() {

    // rps result convention:
    // 0 -> win
    // 1 -> lose
    // 2 -> draw
    //      \_rock\_paper\_scissor_
    // rock   | draw | lose | win
    // paper  | win  | draw | lose
    // scissor| lose | win  | draw

    private val rpsMatrix = listOf(
        listOf(2,1,0),
        listOf(0,2,1),
        listOf(1,0,2)
    )

    private val _victories = MutableLiveData<Int>().apply {
        value = 0
    }
    val victories: LiveData<Int> = _victories

    private val _losses = MutableLiveData<Int>().apply {
        value = 0
    }
    val losses: LiveData<Int> = _losses

    private val _draws = MutableLiveData<Int>().apply {
        value = 0
    }
    val draws: LiveData<Int> = _draws

    fun play(own: Int, context: Context?) {
        if (context != null && own in 0..2) {
            val opponent = (0..2).random()
            val opponentDecision = when (opponent) {
                0 -> context.getString(R.string.rock)
                1 -> context.getString(R.string.paper)
                2 -> context.getString(R.string.scissor)
                else -> ""
            }

            when (rpsMatrix[own][opponent]) {
```

```
0 -> {
    Toast.makeText(context, context.getString(R.string.won, opponentDecision),
Toast.LENGTH_SHORT).show()
    _victories.value = (_victories.value ?: 0) + 1
}
1 -> {
    Toast.makeText(context, context.getString(R.string.lost, opponentDecision),
Toast.LENGTH_SHORT).show()
    _losses.value = (_losses.value ?: 0) + 1
}
2 -> {
    Toast.makeText(context, context.getString(R.string.draw), Toast.LENGTH_SHORT).show()
    _draws.value = (_draws.value ?: 0) + 1
}
}
}
}
}
```

Luego proceda a modificar el archivo de `RadioButtonFragment`, con el siguiente código. En este se observa la asignación de listeners refrendados a funciones del `ViewModel` correspondiente a la clase.

```
package com.example.lab2.ui.radio_button
import androidx.lifecycle.ViewModelProvider
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.example.lab2.databinding.RadioButtonFragmentBinding

class RadioButtonFragment : Fragment() {

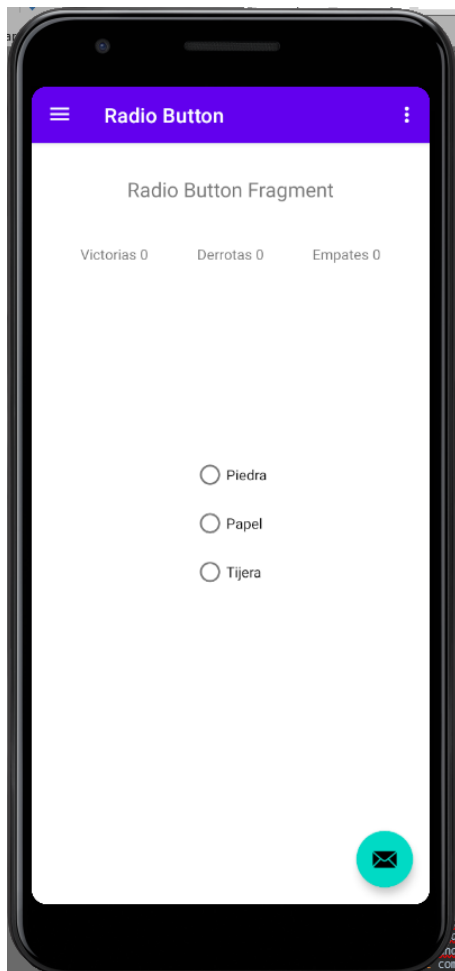
    private lateinit var viewModel: RadioButtonViewModel

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        viewModel = ViewModelProvider(activity ?: this).get(RadioButtonViewModel::class.java)
        val binding = RadioButtonFragmentBinding.inflate(inflater, container, false)
        binding.lifecycleOwner = this
        binding.vm = viewModel

        binding.radioButton.setOnCheckedChangeListener { _, isChecked -> if (isChecked)
            viewModel.play(0, activity) }
        binding.radioButton2.setOnCheckedChangeListener { _, isChecked -> if (isChecked)
            viewModel.play(1, activity) }
        binding.radioButton3.setOnCheckedChangeListener { _, isChecked -> if (isChecked)
            viewModel.play(2, activity) }

        return binding.root
    }
}
```

Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente.



Editar componente Spinner (spinner_fragment, SpinnerFragment, SpinnerViewModel)

Para trabajar con el siguiente componente, agregamos las siguientes cadenas de caracteres al archivo **string.xml**

```
<string-array name="rps_options">
    <item>@string/rock</item>
    <item>@string/paper</item>
    <item>@string/scissor</item>
</string-array>
<string name="rps_prompt">Selecciona tu jugada</string>
```

Se procede a modificar el archivo de **spinner_fragment** para modificar el layout con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <import type="android.view.View"/>
        <variable
            name="vm"
            type="com.example.lab2.ui.spinner.SpinnerViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.spinner.SpinnerFragment">

        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:text="@{String.format(@string/fragment_label, @string/menu_spinner)}"
            android:textAlignment="center"
            android:textSize="20sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
```

```

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:text="@{String.format(@string/victories, vm.victories)}"
    app:layout_constraintEnd_toStartOf="@+id/textView3"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:text="@{String.format(@string/losses, vm.losses)}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="100dp"
    android:text="@{String.format(@string/draws, vm.draws)}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/textView3"
    app:layout_constraintTop_toTopOf="parent" />

```

```

<Spinner
    android:id="@+id/spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="32dp"
    android:layout_marginLeft="32dp"
    android:layout_marginEnd="32dp"
    android:layout_marginRight="32dp"
    android:prompt="@string/rps_prompt"
    android:entries="@array/rps_options"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

```

</layout>

```

Luego proceda a modificar el código de la clase view model correspondiente `SpinnerViewModel`.

```
package com.example.lab2.ui.spinner

import android.content.Context
import android.widget.Toast
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.lab2.R

class SpinnerViewModel : ViewModel() {

    // rps result convention:
    // 0 -> win
    // 1 -> lose
    // 2 -> draw
    //      \_rock\_paper\_scissor\_
    // rock   | draw | lose | win
    // paper  | win  | draw | lose
    // scissor| lose | win  | draw

    private val rpsMatrix = listOf(
        listOf(2,1,0),
        listOf(0,2,1),
        listOf(1,0,2)
    )

    private val _victories = MutableLiveData<Int>().apply {
        value = 0
    }
    val victories: LiveData<Int> = _victories

    private val _losses = MutableLiveData<Int>().apply {
        value = 0
    }
    val losses: LiveData<Int> = _losses
    private val _draws = MutableLiveData<Int>().apply {
        value = 0
    }
    val draws: LiveData<Int> = _draws

    fun play(own: Int, context: Context?) {
        if (context != null && own in 0..2) {
            val opponent = (0..2).random()
            val opponentDecision = when (opponent) {
                0 -> context.getString(R.string.rock)
                1 -> context.getString(R.string.paper)
                2 -> context.getString(R.string.scissor)
                else -> ""
            }

            when (rpsMatrix[own][opponent]) {
```


Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente.



Editar componente Gallery (gallery_fragment, GalleryFragment, GalleryViewModel)

Para trabajar con el siguiente componente, agregamos las siguientes cadenas de caracteres al archivo **string.xml**

```
<string name="select_btn">Seleccionar imagen</string>
<string name="desc_img">Imagen</string>
```

Para permitir el acceso a la galería, se debe de agregar esta linea de código en el **manifest.xml** para poder acceder a memoria externa/interna.

```
<!--Adding Read External Storage Permission-->
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Para trabajar con el siguiente componente, se modificara el archivo de **gallery_fragment** para modificar el layout con el siguiente código.

```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>
        <variable
            name="vm"
            type="com.example.lab2.ui.gallery.GalleryViewModel" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".ui.gallery.GalleryFragment">

        <TextView
            android:id="@+id/text_view"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginStart="8dp"
            android:layout_marginTop="32dp"
            android:layout_marginEnd="8dp"
            android:gravity="center_horizontal"
            android:text="@{String.format(@string/fragment_label, @string/menu_gallery)}"
            android:textAlignment="center"
            android:textSize="20sp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
```

```

    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="100dp"
    android:text="@string/select_btn"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />

```

```

<ImageView
    android:id="@+id/imageView2"
    android:layout_width="250dp"
    android:layout_height="250dp"
    android:src="@{vm.imageUri}"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    android:contentDescription="@string/desc_img" />

```

```

</androidx.constraintlayout.widget.ConstraintLayout>

```

```

</layout>

```

Luego proceda a modificar el código de la clase view model correspondiente GalleryViewModel.

```

package com.example.lab2.ui.gallery

```

```

import android.net.Uri
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

```

```

class GalleryViewModel : ViewModel() {

    private val _imageUri = MutableLiveData<Uri>()
    val imageUri: LiveData<Uri> = _imageUri

    fun setImageUri(uri: Uri?) {
        _imageUri.value = uri
    }
}

```

Para que todo funcione correctamente, modificaremos el código de la clase fragment (GalleryFragment) con el siguiente código.

```
package com.example.lab2.ui.gallery

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProvider
import com.example.lab2.databinding.FragmentGalleryBinding

class GalleryFragment : Fragment() {

    private val REQUEST_CODE = 100
    private lateinit var galleryViewModel: GalleryViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        galleryViewModel = ViewModelProvider(activity ?: this).get(GalleryViewModel::class.java)
        val binding = FragmentGalleryBinding.inflate(inflater, container, false)
        binding.lifecycleOwner = this
        binding.vm = galleryViewModel
        binding.button2.setOnClickListener {
            openGalleryForImage()
        }
        return binding.root
    }

    private fun openGalleryForImage() {
        val intent = Intent(Intent.ACTION_PICK)
        intent.type = "image/*"
        startActivityForResult(intent, REQUEST_CODE)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == Activity.RESULT_OK && requestCode == REQUEST_CODE){
            galleryViewModel.setImageUri(data?.data)
        }
    }
}
```

Luego de las modificaciones, al correr la aplicación y acceder al layout previamente modificado se observa como el siguiente.



Tarea Opcional

- Implementar de acuerdo a tu creatividad el fragment para el uso de un tab widget
- Unificar RadioButtonViewModel y SpinnerViewModel en RpsViewModel, utilizar este nuevo view model en ambos fragments (RadioButtonFragment y SpinnerFragment)
- Agregar efecto de vibración cuando se de click en el botón de ButtonFragment

Nota Recordatorio:

Durante el desarrollo de esta guía se le proporciona código que hace referencia a *com.example.lab2*

Asegúrese de cambiar esas referencias para que utilice el nombre de paquete que usted tenga en su proyecto.