TEMA 1. Interfaz gráfica de usuario

Práctica 1 ConstraintLayout

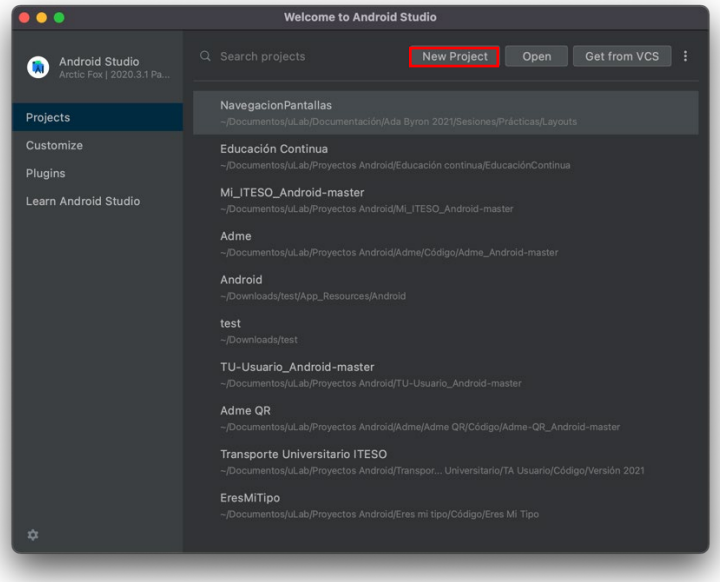
Objetivo específico	Conocer en qué casos podemos usar constraintLayout
Recursos	Android Studio XML
Actividades	Crearemos un activity que contenga un constraintLayout
Producto(s)	Al final de la práctica deberá mostrar una vista en la que mostramos varios objetos posicionados y que se pueda mostrar en modo portrait (vertical) y modo landscape (horizontal)

PASO 1. Abrir Android Studio.

Abrir Android Studio

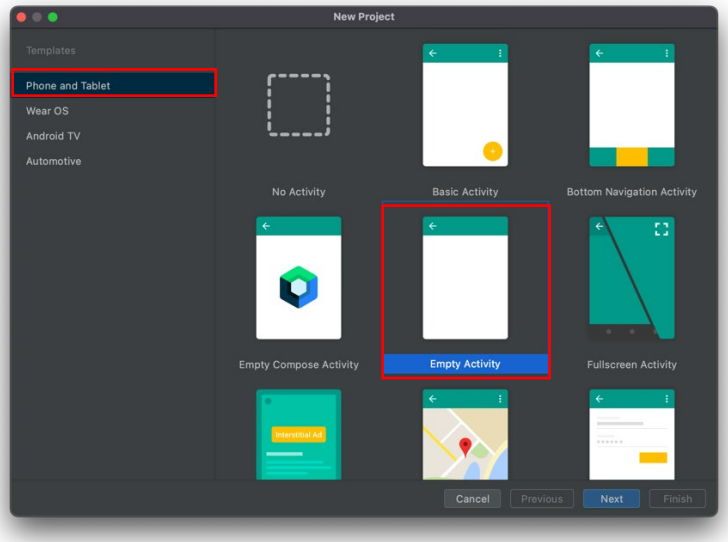
PASO 2. Elegir nuevo proyecto

Seleccionar New project



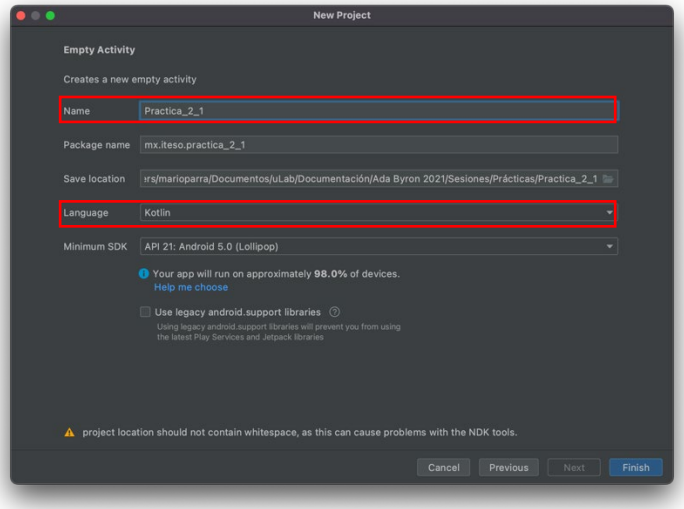
PASO 3. Elegir Empty Activity

Seleccionar Phone and Tablet, luego seleccionar Empty Activity, y dar click en Next



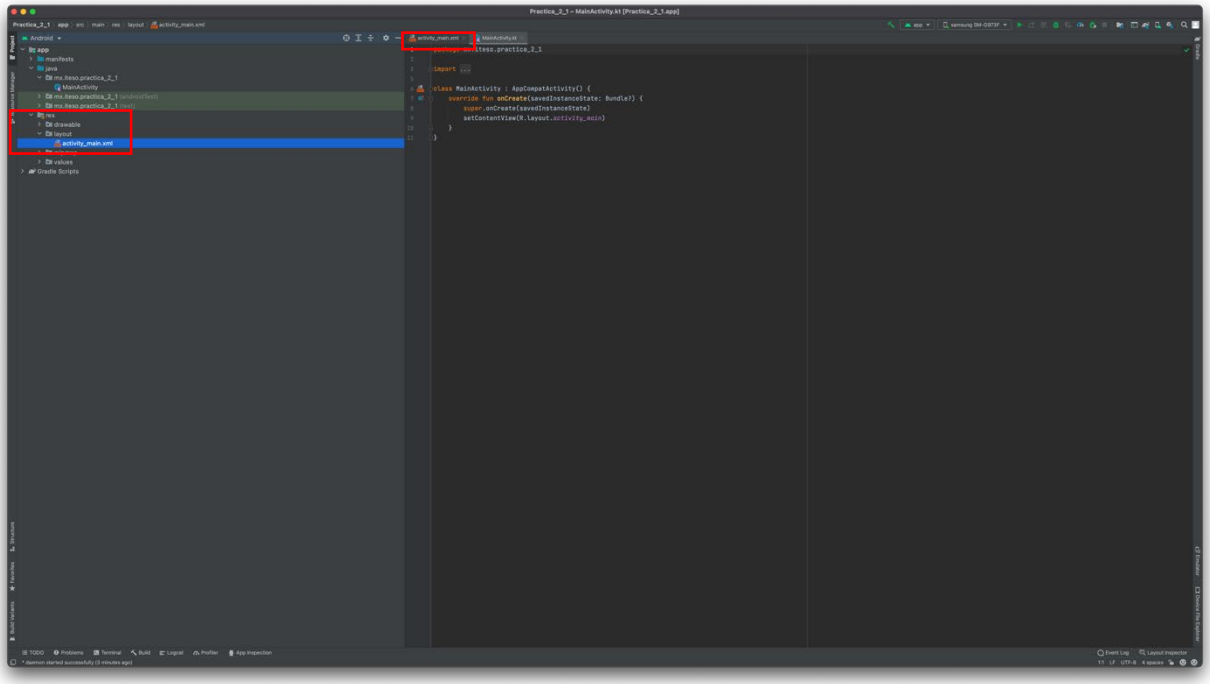
PASO 4. Configuración del nuevo proyecto

En nombre escribir Practica_2_1 y seleccionar el lenguaje Kotlin, después dar click en Finish



PASO 5. Abrir la clase activity_main.xml

Abrir la clase activity_main.xml, en caso de que no esté abierto ir a la estructura de Android, abrir la carpeta res, luego layout y ahí vendrán todas nuestras activities



PASO 6. Seleccionar Split

Seleccionar Split

PASO 7. Crearemos nuestro ConstraintLayout

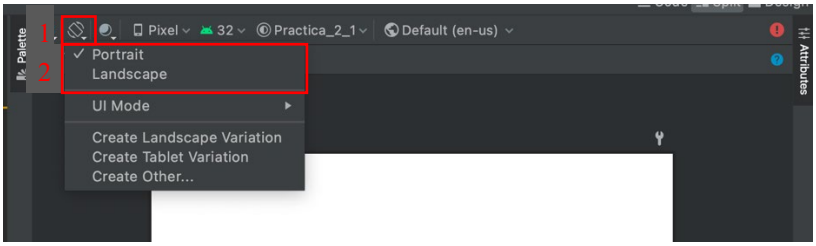
Con el siguiente código agregaremos dos ImageView y un button.

```
<androidx.constraintlayout.widget.ConstraintLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:layout_editor_absoluteX="64dp"
        tools:layout_editor_absoluteY="123dp"
        tools:srcCompat="@tools:sample/avatars" />

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:layout_editor_absoluteX="216dp"
        tools:layout_editor_absoluteY="123dp"
        tools:srcCompat="@tools:sample/avatars" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        tools:layout_editor_absoluteX="158dp"
        tools:layout_editor_absoluteY="306dp" />

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

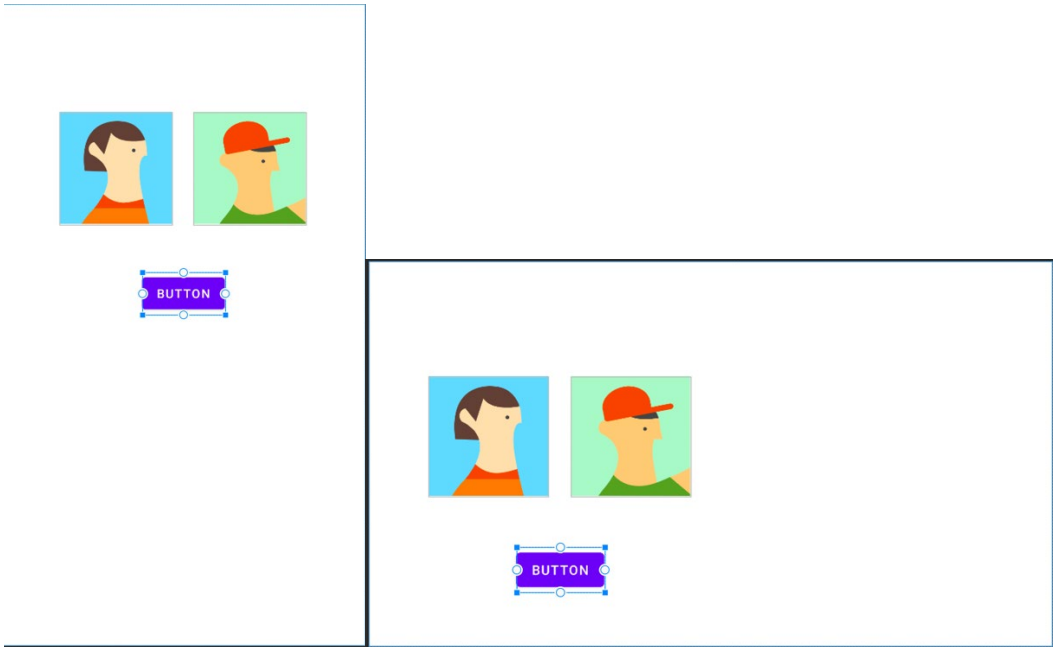
Como podemos ver en el código los tres objetos tienen el atributo ***layout_editor_absoluteX*** y ***layout_editor_absoluteY*** que es la posición donde se posicionará el objeto en la vista. Si arrastramos el objeto por la vista podremos ver que estos valores cambiarán.

PASO 7. Landscape

Para cambiar la posición de la vista debemos dar click aquí.



Resultado.



Práctica 2 FrameLayout

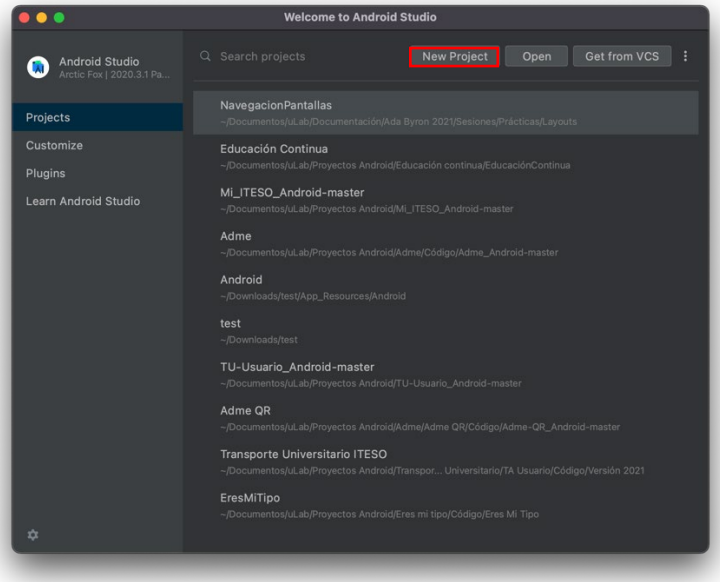
Objetivo específico	Conocer en qué casos podemos usar frameLayout
Recursos	Android Studio XML
Actividades	Crearemos un activity que contenga un frameLayout
Producto(s)	Al final de la práctica deberá mostrar una vista en la que mostramos varios objetos posicionados en un punto específico.

PASO 1. Abrir Android Studio.

Abrir Android Studio

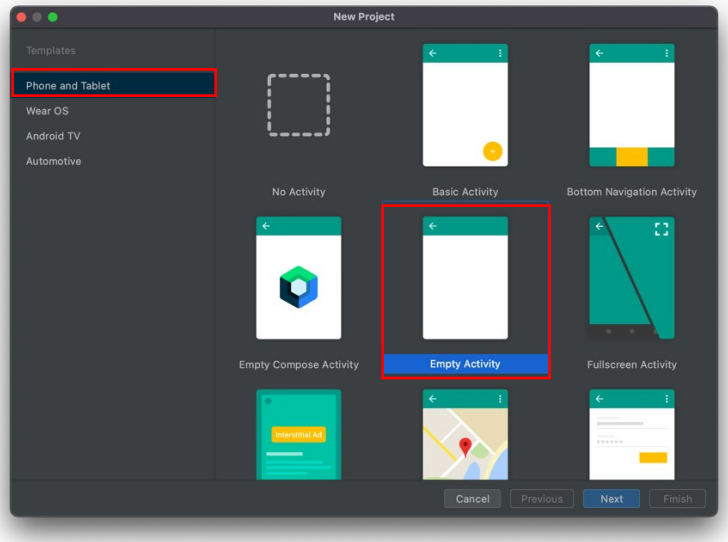
PASO 2. Elegir nuevo proyecto

Seleccionar New project



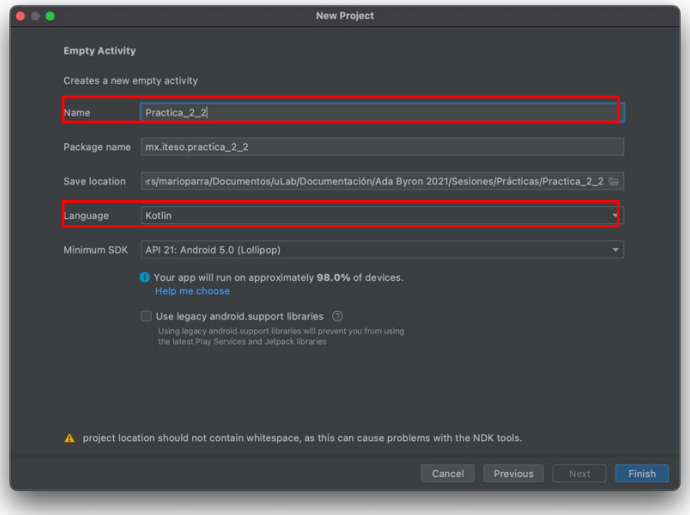
PASO 3. Elegir Empty Activity

Seleccionar Phone and Tablet, luego seleccionar Empty Activity, y dar click en Next



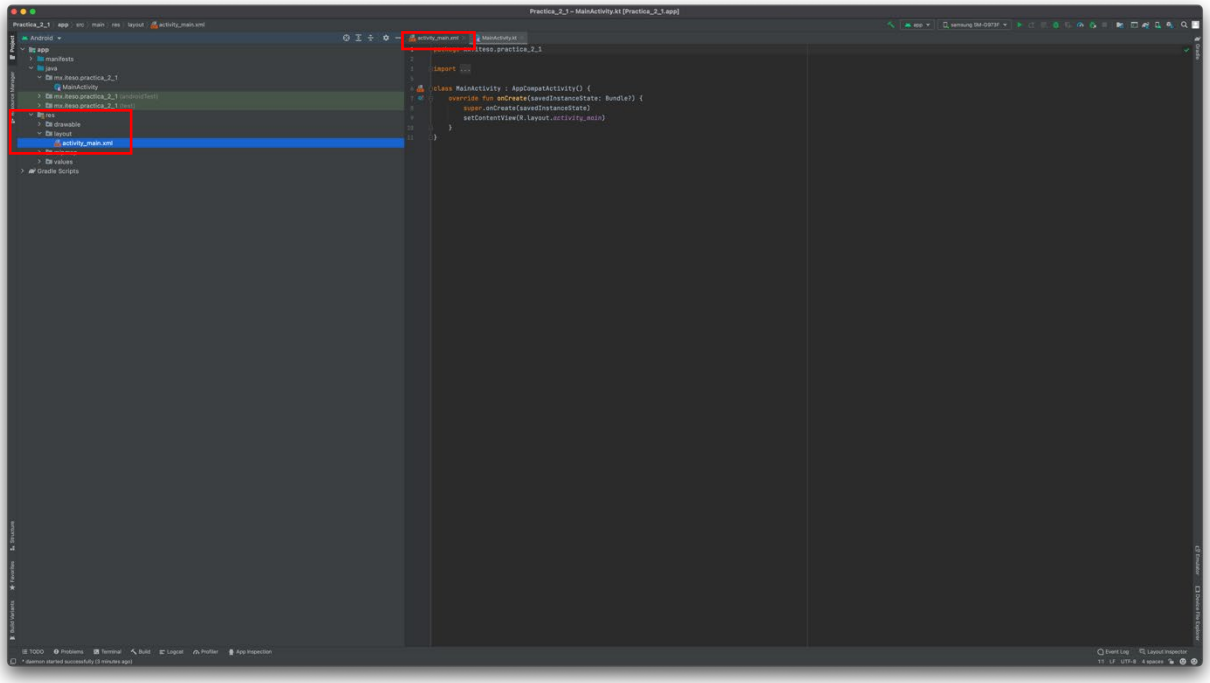
PASO 4. Configuración del nuevo proyecto

En nombre escribir Practica_2_2 y seleccionar el lenguaje Kotlin, después dar click en Finish



PASO 5. Abrir la clase activity_main.xml

Abrir la clase activity_main.xml, en caso de que no esté abierto ir a la estructura de Android, abrir la carpeta res, luego layout y ahí vendrán todas nuestras activities



PASO 6. Seleccionar Split

Seleccionar Split

PASO 7. Crearemos nuestro FrameLayout

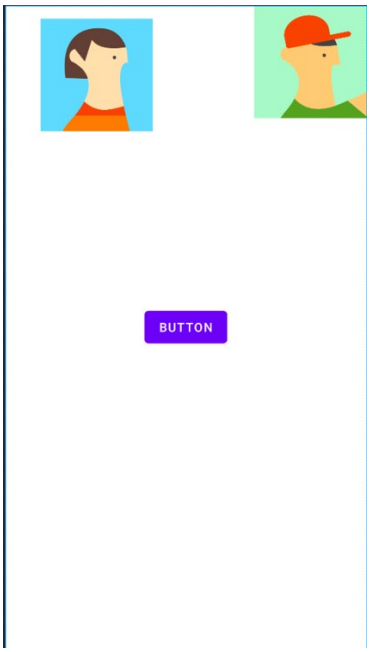
Agregaremos una imagen con margen a la izquierda de 40dp y un margen de arriba de 15dp. La segunda imagen la pondremos en la esquina superior derecha y un botón en el centro de la vista.

```
<FrameLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="40dp"
        android:layout_marginTop="15dp"
        tools:srcCompat="@tools:sample/avatars" />

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end|end"
        tools:srcCompat="@tools:sample/avatars"/>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        android:layout_gravity="center"/>

</FrameLayout>
```

Como veremos el atributo que usamos para dar márgenes es ***android:layout_margin*** y solo cambiamos el final para saber a donde le pondremos el margen por ejemplo

android:layout_marginLeft para poner el margen izquierdo seguido del valor númerico y el prefijo dp (densidad de pixeles) ejemplo 40dp significa que el margen izquierdo será de 40.

Resultado.



Práctica 3 LinearLayout

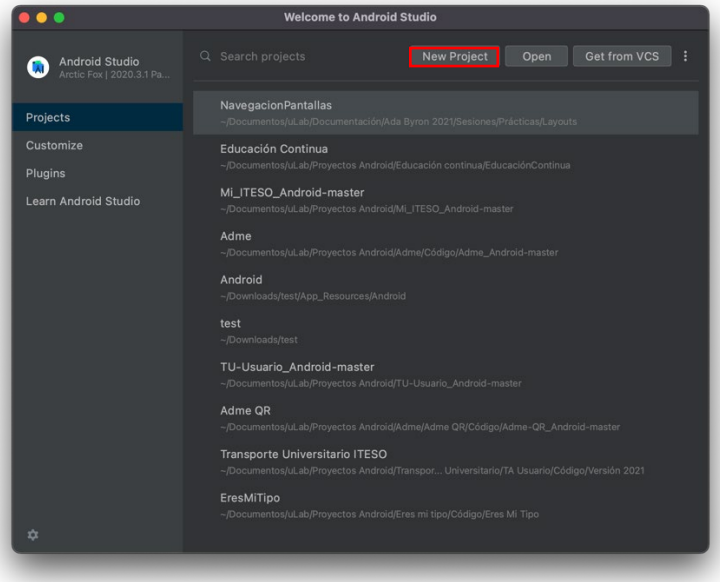
Objetivo específico	Conocer en qué casos podemos usar linearLayout
Recursos	Android Studio XML
Actividades	Crearemos un activity que contenga un linearLayout
Producto(s)	Al final de la práctica deberá mostrar una vista en la que mostramos varios objetos orientados de forma horizontal u vertical.

PASO 1. Abrir Android Studio.

Abrir Android Studio

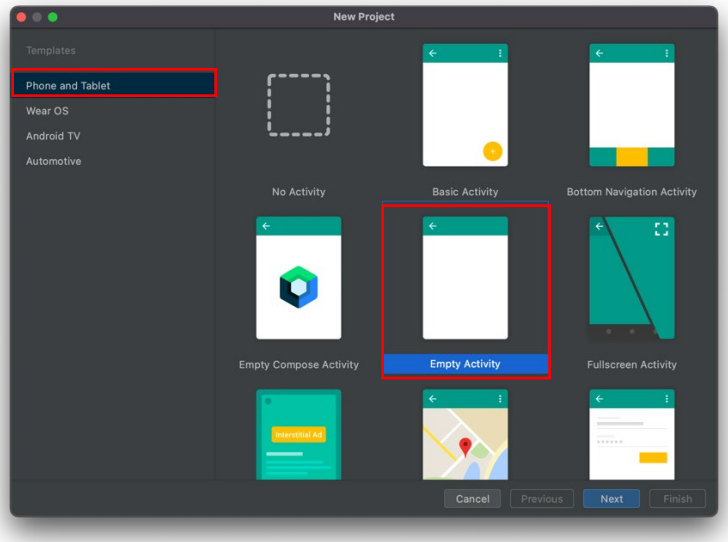
PASO 2. Elegir nuevo proyecto

Seleccionar New project



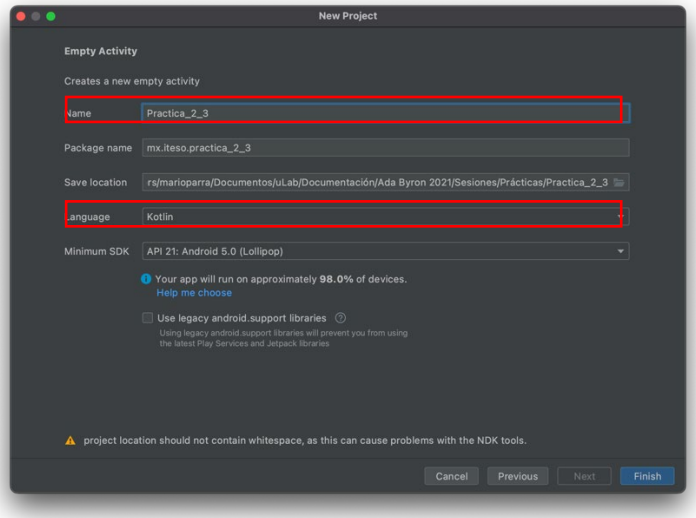
PASO 3. Elegir Empty Activity

Seleccionar Phone and Tablet, luego seleccionar Empty Activity, y dar click en Next



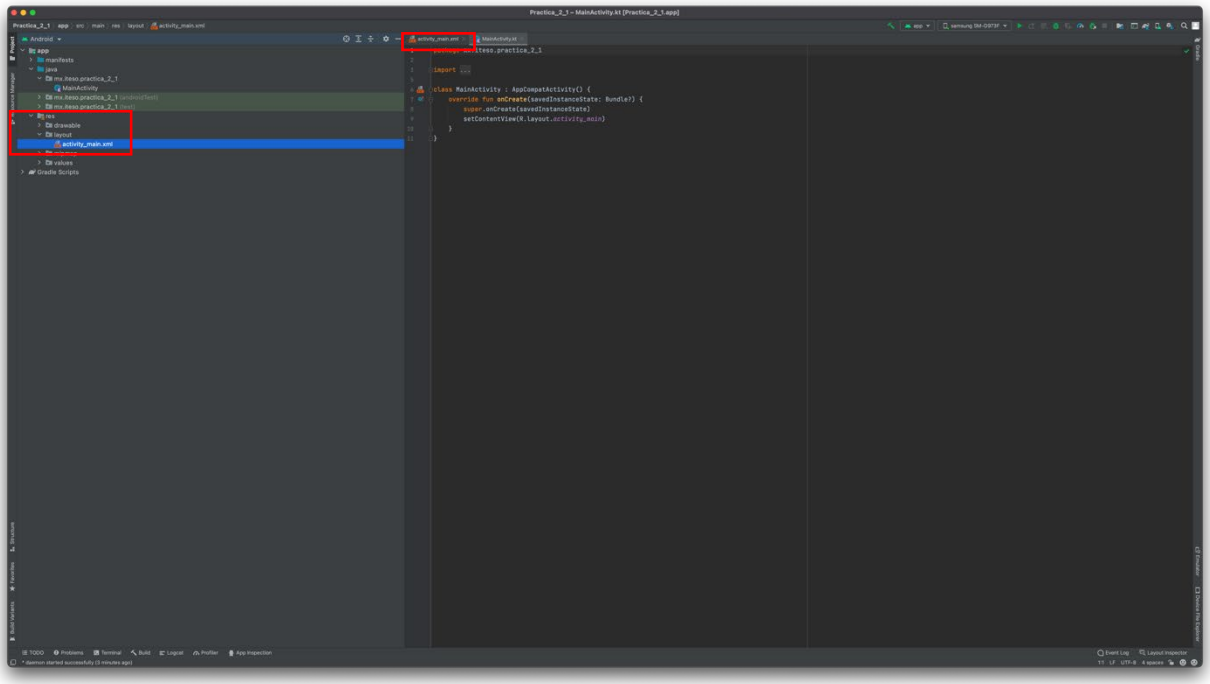
PASO 4. Configuración del nuevo proyecto

En nombre escribir Practica_2_3 y seleccionar el lenguaje Kotlin, después dar click en Finish



PASO 5. Abrir la clase activity_main.xml

Abrir la clase activity_main.xml, en caso de que no esté abierto ir a la estructura de Android, abrir la carpeta res, luego layout y ahí vendrán todas nuestras activities



PASO 6. Seleccionar Split

Seleccionar Split

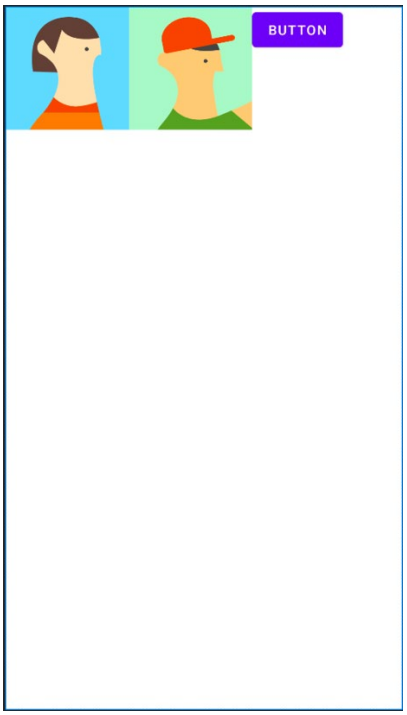
PASO 7. Crearemos nuestro LinearLayout

Agregaremos dos imágenes y un botón con orientación horizontal.

```
<LinearLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    tools:context=".MainActivity">


    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:srcCompat="@tools:sample/avatars" />

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:srcCompat="@tools:sample/avatars"/>

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"/>

</LinearLayout>
```

Este layout es más sencillo y con el atributo android:orientation="horizontal" le damos la orientación horizontal y con esto los objetos se mostrarán en la vista juntos y se acomodarán de izquierda a derecha.

Resultado.


BUTTON

Práctica 4 RelativeLayout

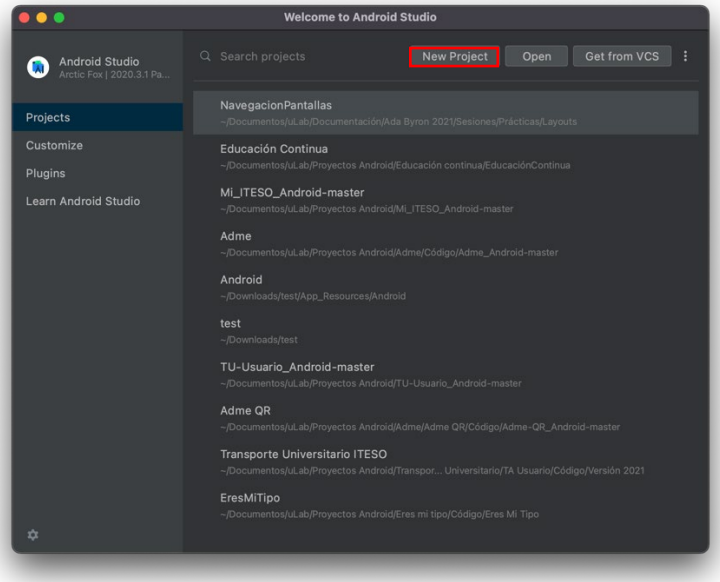
Objetivo específico	Conocer en qué casos podemos usar relativeLayout
Recursos	Android Studio XML
Actividades	Crearemos un activity que contenga un relativeLayout
Producto(s)	Al final de la práctica deberá mostrar una vista en la que mostramos varios objetos donde podemos acomodarlos donde nos guste.

PASO 1. Abrir Android Studio.

Abrir Android Studio

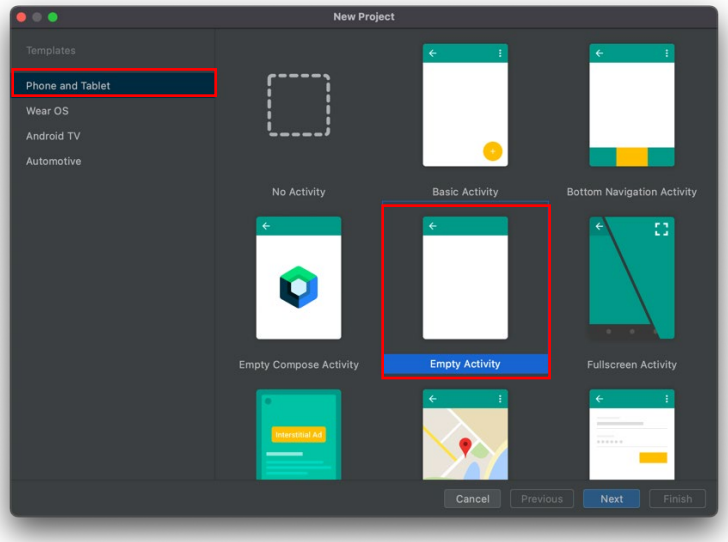
PASO 2. Elegir nuevo proyecto

Seleccionar New project



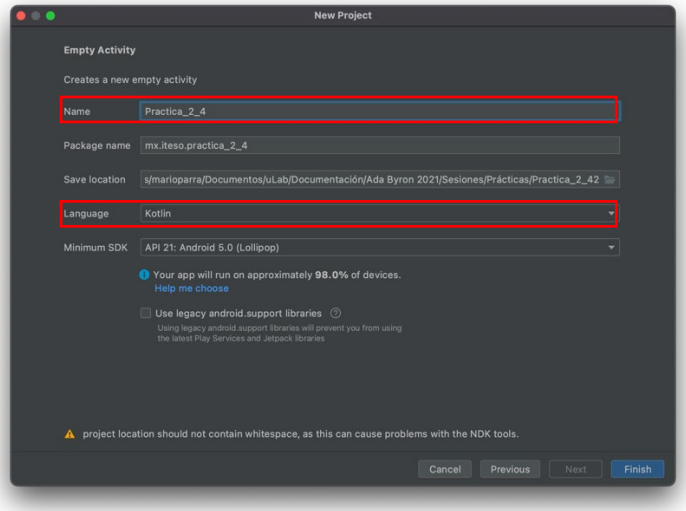
PASO 3. Elegir Empty Activity

Seleccionar Phone and Tablet, luego seleccionar Empty Activity, y dar click en Next



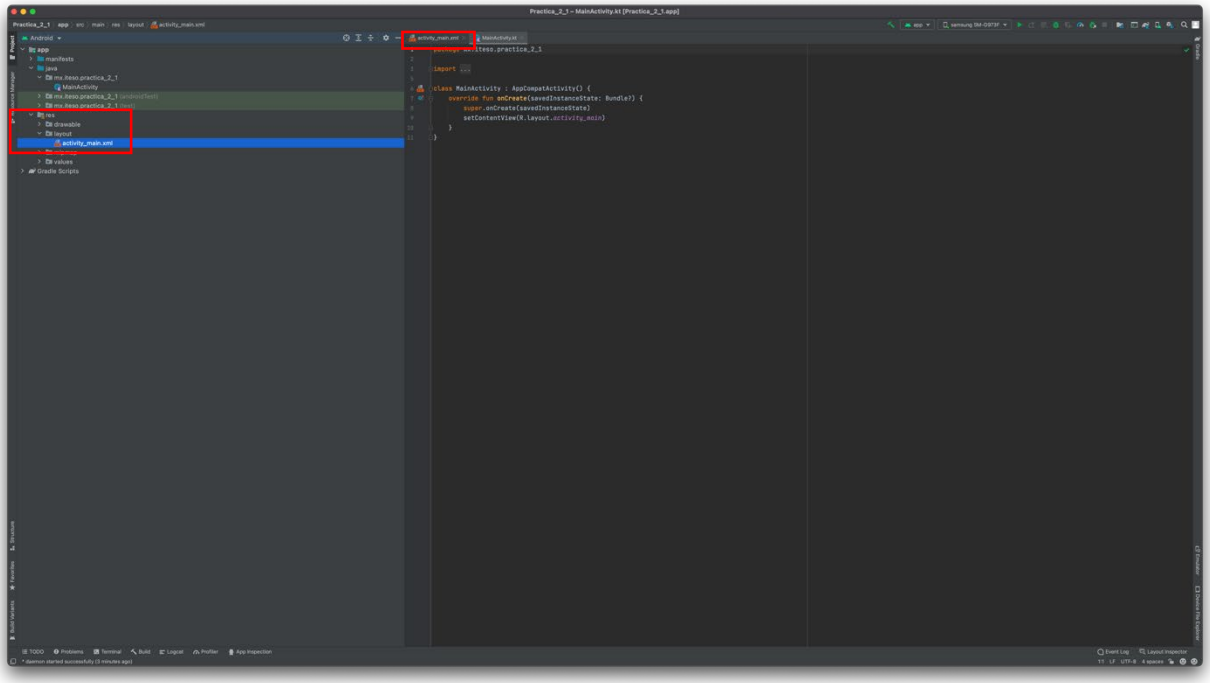
PASO 4. Configuración del nuevo proyecto

En nombre escribir Practica_2_4 y seleccionar el lenguaje Kotlin, después dar click en Finish



PASO 5. Abrir la clase activity_main.xml

Abrir la clase activity_main.xml, en caso de que no esté abierto ir a la estructura de Android, abrir la carpeta res, luego layout y ahí vendrán todas nuestras activities



PASO 6. Seleccionar Split

Seleccionar Split

PASO 7. Crearemos nuestro RelativeLayout

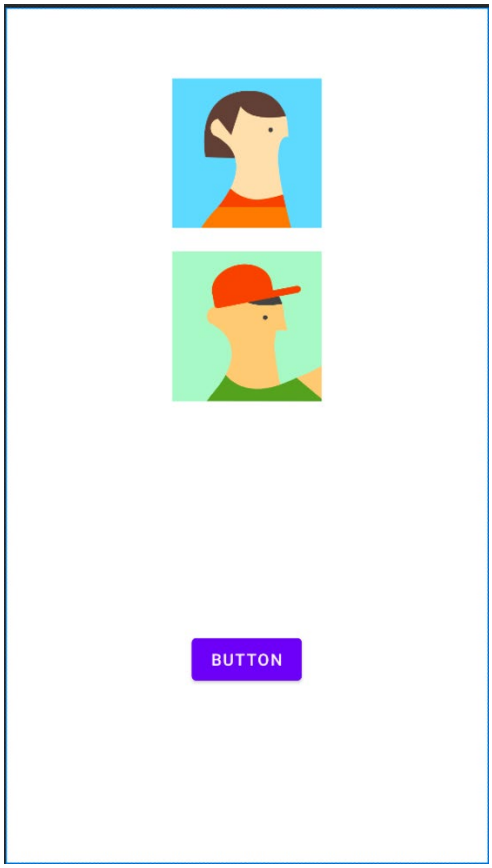
Agregaremos dos imágenes y un botón, la primera imagen tendrá un margen superior anclado al tope de la vista con 60 dp, la segunda imagen estará anclada a la parte inferior de la primera imagen con 20 dp. Para finalizar el botón estará anclado al margen inferior de la vista con 150 dp.

```
<RelativeLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="60dp"
        tools:srcCompat="@tools:sample/avatars" />

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        tools:srcCompat="@tools:sample/avatars" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="150dp"
        android:layout_centerHorizontal="true"
        android:text="Button"
        android:layout_gravity="center"/>
```

```
</RelativeLayout>
```

Como vemos en el código el atributo android:layout_alignParentTop="true" nos dice que estamos anclando la primera imagen al tope de la vista y podemos darle un margen para dar el diseño que se desea con el atributo marginTop. Con la segunda imagen tenemos el atributo android:layout_below="@+id/imageView" donde decimos que queremos anclarnos a la parte inferior de la primera imagen. Aquí vemos algo nuevo @+id/imageView con esto nos referimos que estamos llamando al objeto con el id imageView, para esto necesitamos agregarle un id al objeto que deseamos buscar como lo vemos en los tres objetos que tenemos con el atributo android:id="@+id/nombre_objeto". En el boton vemos android:layout_alignParentBottom="true" que igual que en la primera imagen esta vez anclamos el boton a la parte inferior de la vista con 150dp.

Resultado.



TEMA 2. Navegación entre pantallas

Práctica 5 Navegación entre pantallas

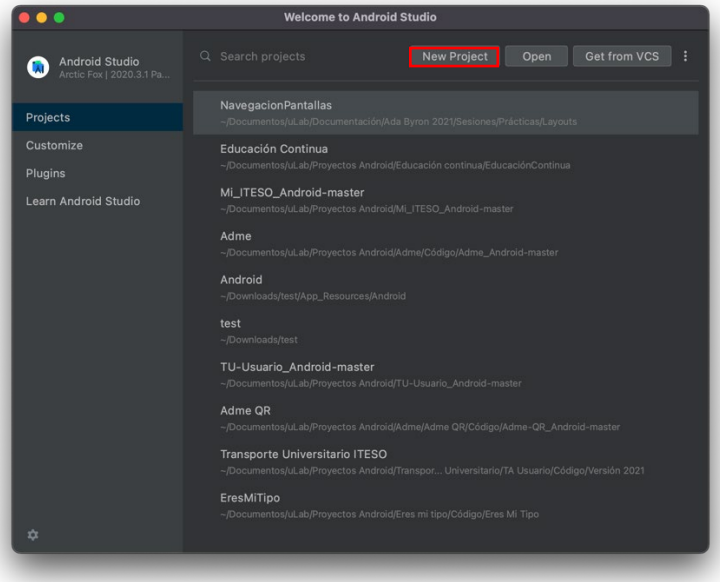
Objetivo específico	Navegar de la vista 1 a la vista 2
Recursos	Android Studio XML Kotlin
Actividades	Crearemos un segundo activity donde reciba información de main activity
Producto(s)	Al final de la práctica deberá enviar información del main activity al segundo activity y mostrar el resultado.

PASO 1. Abrir Android Studio.

Abrir Android Studio

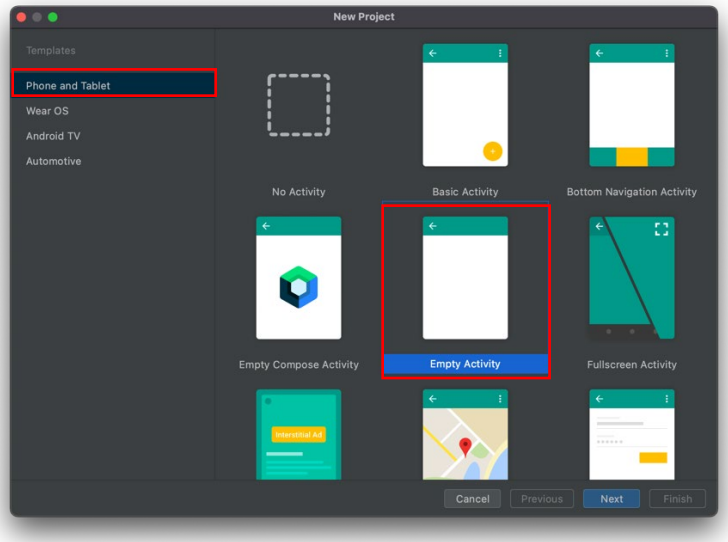
PASO 2. Elegir nuevo proyecto

Seleccionar New project



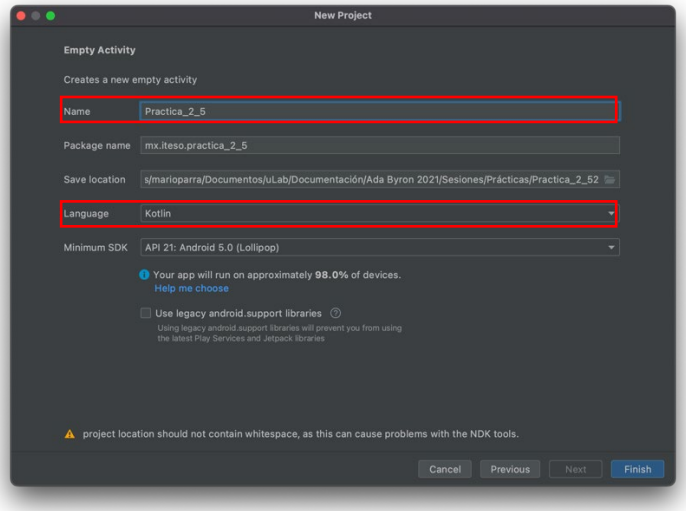
PASO 3. Elegir Empty Activity

Seleccionar Phone and Tablet, luego seleccionar Empty Activity, y dar click en Next



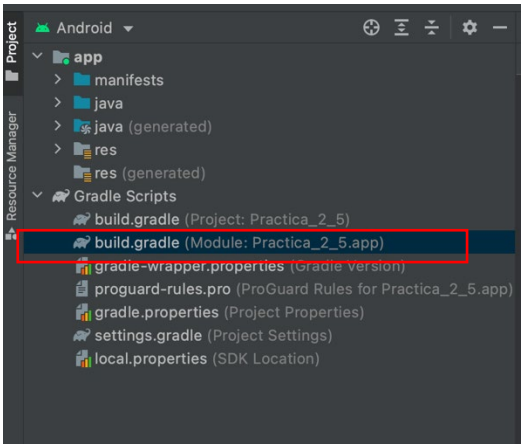
PASO 4. Configuración del nuevo proyecto

En nombre escribir Practica_2_5 y seleccionar el lenguaje Kotlin, después dar click en Finish



PASO 5. Configurar ViewBinding

View Binding es un nuevo Sistema de vinculación de vistas para acceder a los objetos que agregamos en nuestro xml desde nuestra clase activity. Para esto vamos a ir a *build.gradle (module)*.

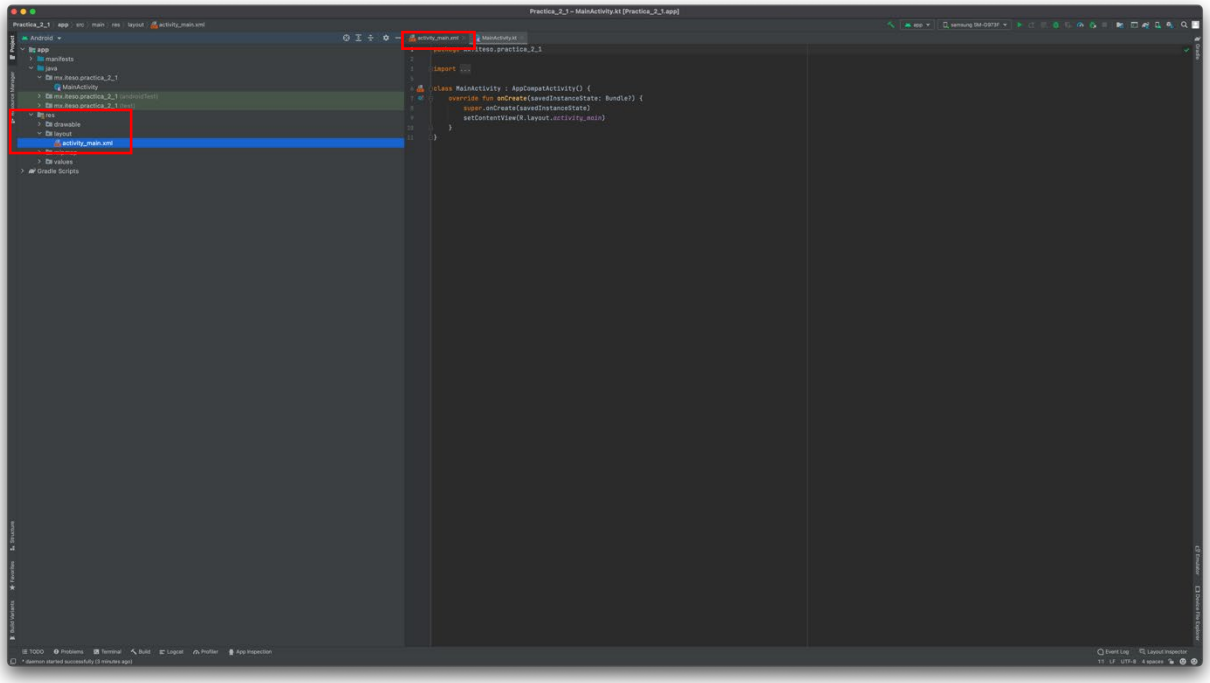


Una vez aquí, buscamos la sección android{}, normalmente es de las primeras secciones y agregamos lo siguiente.

```
buildFeatures{
    viewBinding = true
}

viewBinding {
    enabled = true
}
```

PASO 5. Abrir la clase activity_main.xml

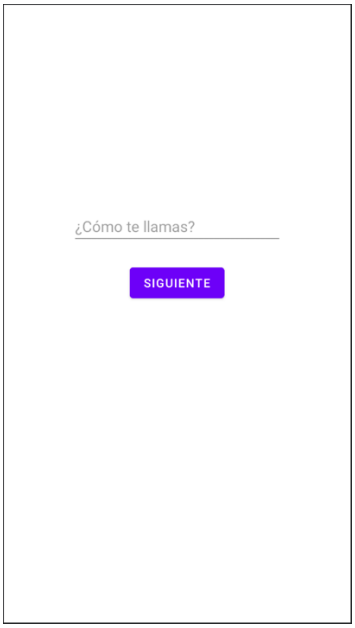
Abrir la clase activity_main.xml, en caso de que no esté abierto ir a la estructura de Android, abrir la carpeta res, luego layout y ahí vendrán todas nuestras activities



PASO 6. Crear la primera vista

Crearemos un editText y un botón en la primera vista donde lo que se escriba en el editText que al hacer tap en el botón será enviado a la segunda vista.

```
<RelativeLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:hint="¿Cómo te llamas?"
        android:layout_marginTop="240dp"
        android:layout_centerHorizontal="true"
        />

    <Button
        android:id="@+id/btnSiguiente"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/editText"
        android:layout_marginTop="20dp"
        android:layout_centerHorizontal="true"
        android:text="Siguiente"/>

</RelativeLayout>
```



PASO 6. Abrir la clase MainActivity.kt

Primero debemos crear la estancia del viewBinding

```
private lateinit var binding: ActivityMainBinding
```

Crearemos las estancias de los objetos que acabamos de crear en nuestra clase activity_main.xml.

```
private lateinit var editText : EditText
private lateinit var button : Button
```

Lo primero que vemos es que será una constante y que se llamará button, findViewById nos ayuda a encontrar el objecto en nuestra clase xml con ayuda del id que le agregamos, después entra <> escribimos el tipo de objeto que en este caso es Button, si en dado caso hay un error aquí podría ser el principal problema. Entre parentesis escribimos R.id enseguida de un punto (.) y después el id del objeto que buscamos.

En nuestra función onCreate le daremos la acción al hacer tap al botón con setOnClickListener.

```
button.setOnClickListener {
    checkValue()
}
```

En seguida crearemos la función checkValue donde checaremos si nuestro editText viene vacío o no, con esto queremos avisar al usuario que es un requisito escribir en el editText antes de avanzar a la segunda vista.

```
private fun checkValue() {
    if(editText.text.toString().isEmpty()) {
        Toast.makeText(this, "El nombre no puede estar vacío",
Toast.LENGTH_SHORT).show()
    } else {
        val intent = Intent(this, ShowNameActivity::class.java)
        intent.putExtra("name", editText.text.toString())
        startActivity(intent)
    }
}
```

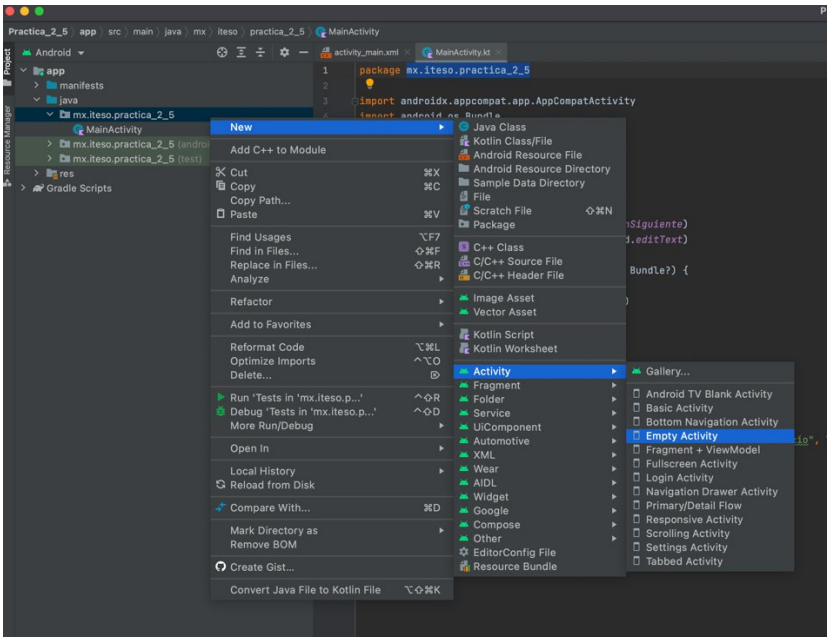
Nuestra clase quedaría de la siguiente manera.

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    private lateinit var editText : EditText
    private lateinit var button : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        button = binding.btnSiguiente
        editText = binding.editText

        button.setOnClickListener {
            checkValue()
        }
```

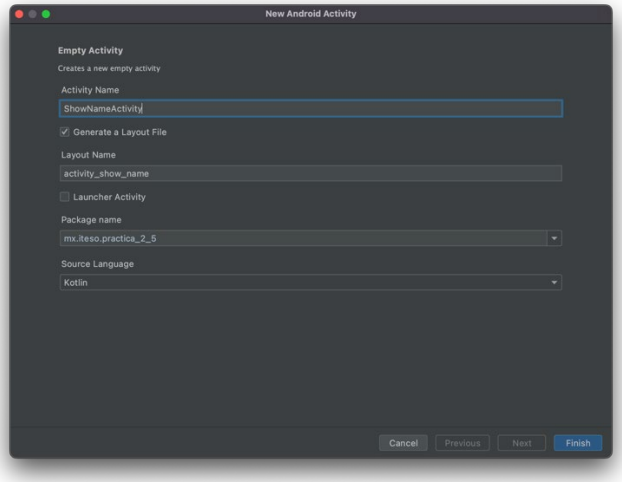
```
    }

    private fun checkValue() {
        if(editText.text.toString().isEmpty()) {
            Toast.makeText(this, "El nombre no puede estar vacío",
Toast.LENGTH_SHORT).show()
        } else {
            val intent = Intent(this, ShowNameActivity::class.java)
            intent.putExtra("name", editText.text.toString())
            startActivity(intent)
        }
    }
}
```

PASO 7. Crearemos nuestra segunda vista

Ahora crearemos nuestra segunda activity que se abrirá cunado hayamos escrito algo en el editExt y dando tap al botón. Para ello iremos a la ruta app/mx.iteso.practica_2_5 y hacemos click derecho sobre él. Después vamos a New/Activity/Empty Activity



Reemplazamos el nombre en Activity Name por ShowNameActivity y damos Finish.



PASO 8. Hacemos el diseño de la segunda vista

Vamos a crear el diseño con un textView donde mostraremos el nombre que traeremos de la primera vista y un botón para regresar a la vista anterior. Abriremos activity_show_name.

```
<RelativeLayout
    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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/purple_700"
    tools:context=".ShowNameActivity">

    <TextView
        android:id="@+id/tvNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/white"
        android:text="Nombre"
        android:layout_centerInParent="true"
        android:textSize="30sp"/>

    <Button
        android:id="@+id/btnVolver"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Volver"
        android:layout_centerHorizontal="true"
        android:layout_alignParentBottom="true"/>

</RelativeLayout>
```

Como vemos en los atributos de RelativeLayout tenemos android:background="@color/purple_700" que con esto cambiaremos el fondo de la vista a color purpura, el textView lo centraremos a la vista y cambiaremos el color del texto a blanco con un tamaño de letra 30sp. El botón se anclará a la parte inferior de la vista.

Resultado.

PASO 8. Estableceremos la comunicación entre las dos vistas.

Regresamos a MainActivity.kt para establecer la comunicación entre las dos vistas, en el if else que dejamos pendientes, en el else codificaremos la comunicación. Empezamos con un intent que es un objeto que contiene instrucciones con las cuales se comunica entre activities.

```
val intent = Intent(this, ShowNameActivity::class.java)
intent.putExtra("name", editText.text.toString())
startActivity(intent)
```

Al escribir Intent nos arrojará en rojo un error, esto lo solucionamos oprimiendo las teclas alt y enter. En la segunda línea tenemos intent con el atributo putExtra, esto nos sirve para enviar datos entre las activities. “name” es el nombre de nuestra variable y de esta forma lo podemos encontrar en la segunda activity, el valor que le enviamos será lo que encuentre en editText para sacar el texto tenemos el atributo text que con esto accedemos el valor que tenga nuestr editText y solo falta castearlo a string con .toString(). Finalmente tenemos la función startActivity que enviamos como parametro el intent que hemos creado.

Código.

```
fun checkValue() {
    if(editText.text.toString().isEmpty()) {
        Toast.makeText(this, "El nombre no puede estar vacío",
Toast.LENGTH_SHORT).show()
    } else {
        val intent = Intent(this, ShowNameActivity::class.java)
        intent.putExtra("name", editText.text.toString())
        startActivity(intent)
    }
}
```

PASO 9. Recuperamos los valores del intent.

En nuestra clase ShowNameActivity.kt recuperamos los datos del intent, crearemos una función donde obtendremos el valor del intent y lo mostraremos en el textView que creamos.

```
fun getAndShowName() {
    val bundle = intent.extras
    val name = bundle?.get("name")
    tvNombre.text = name.toString()
}
```

Con intent.extras creamos la estancia del intent. Para poder obtener el valor que guardamos en la variable “name” tenemos el bundle?.get(“name”), usamos el signo ? para poder cachar un valor nulo y que no de error de ejecución al correr nuestra aplicación y usamos el atributo .toString() para castaerlo a string. Después en nuestro textView agregamos el valor obtenido de nuestro intent, para poder escribir ese valor usamos el atributo .text por eso es que antes se castea el valor obtenido en el intent.

Ahora solo nos queda configurar el botón de volver para que cuando hagamos tap vuelva a la vista anterior.

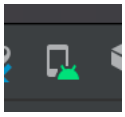
```
btnVolver.setOnClickListener {
    onBackPressed()
}
```

Código.

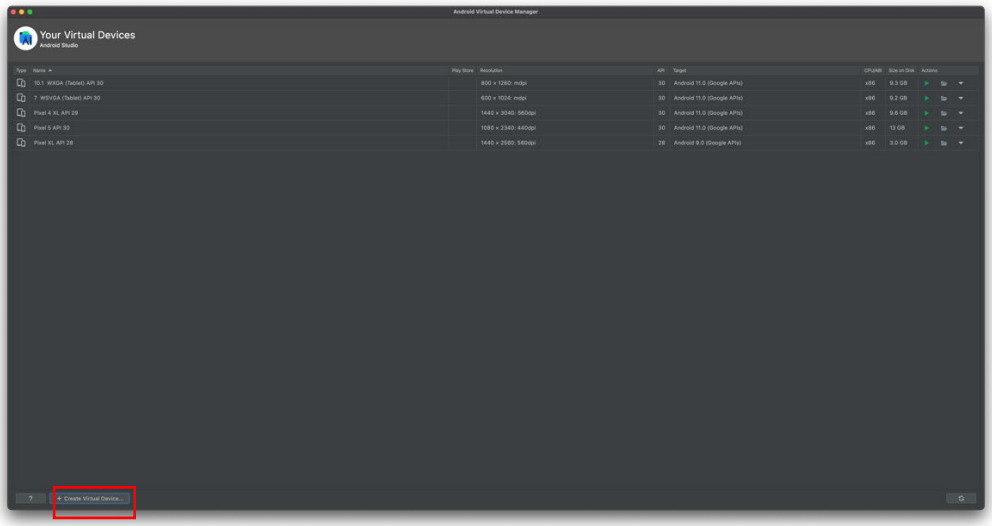
```
class ShowNameActivity : AppCompatActivity() {

    private lateinit var binding: ActivityShowNameBinding

    private lateinit var tvNombre : TextView
    private lateinit var btnVolver : Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityShowNameBinding.inflate(layoutInflater)
        setContentView(binding.root)

        tvNombre = binding.tvNombre
        btnVolver = binding.btnVolver

        getAndShowName()

        btnVolver.setOnClickListener {
            onBackPressed()
        }
    }

    fun getAndShowName() {
        val bundle = intent.extras
        val name = bundle?.get("name")
        tvNombre.text = name.toString()
    }
}
```

PASO 10. Configurar un dispositivo virtual

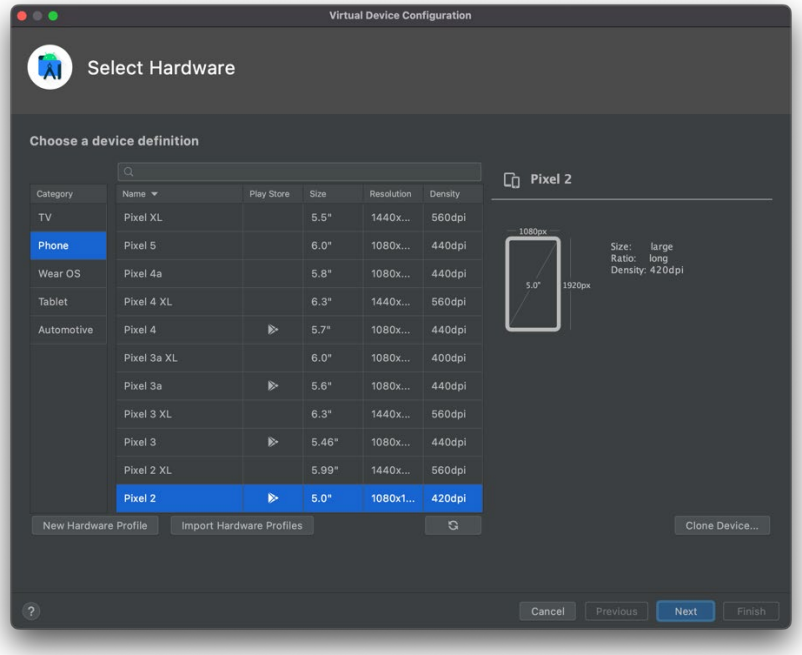
Vamos a configurar un dispositivo virtual antes de poder correr nuestra aplicación. Damos click en



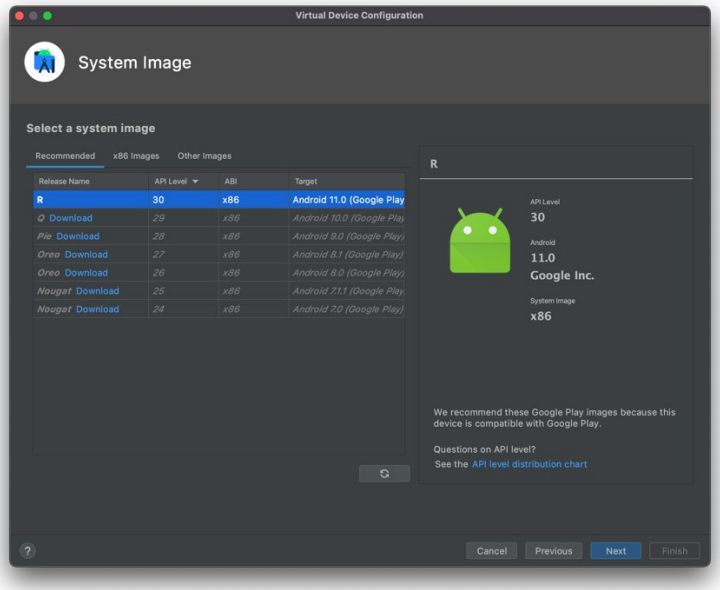
Se encuentra en la parte superior derecha.
Le damos click en create virtual device



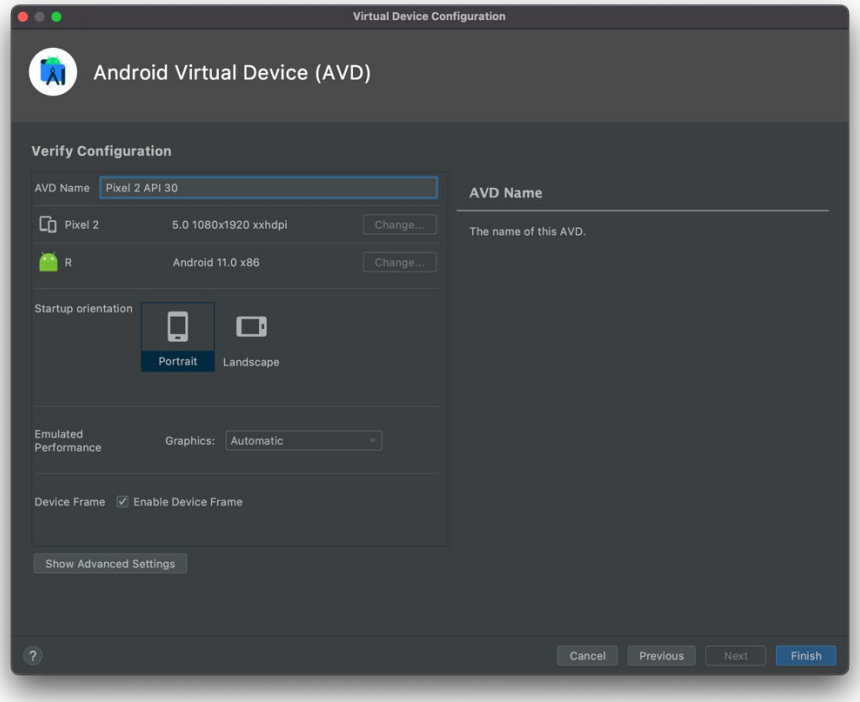
Seleccionamos el dispositivo que queramos y damos click en Next



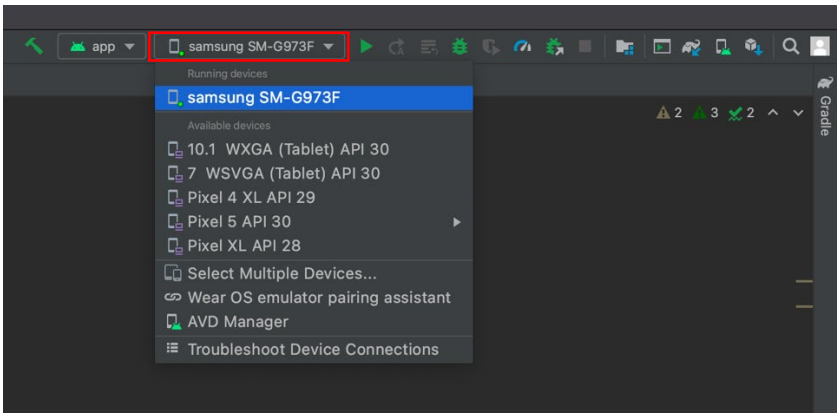
Seleccionamos el sistema operativo más reciente y damos click en Next. En caso de no tener ninguno descargado le daremos click en descargar, puede tardar algunos minutos.



Aquí podemos editar el nombre del dispositivo y configurar su orientación. Nosotros solo le daremos click en Finish.

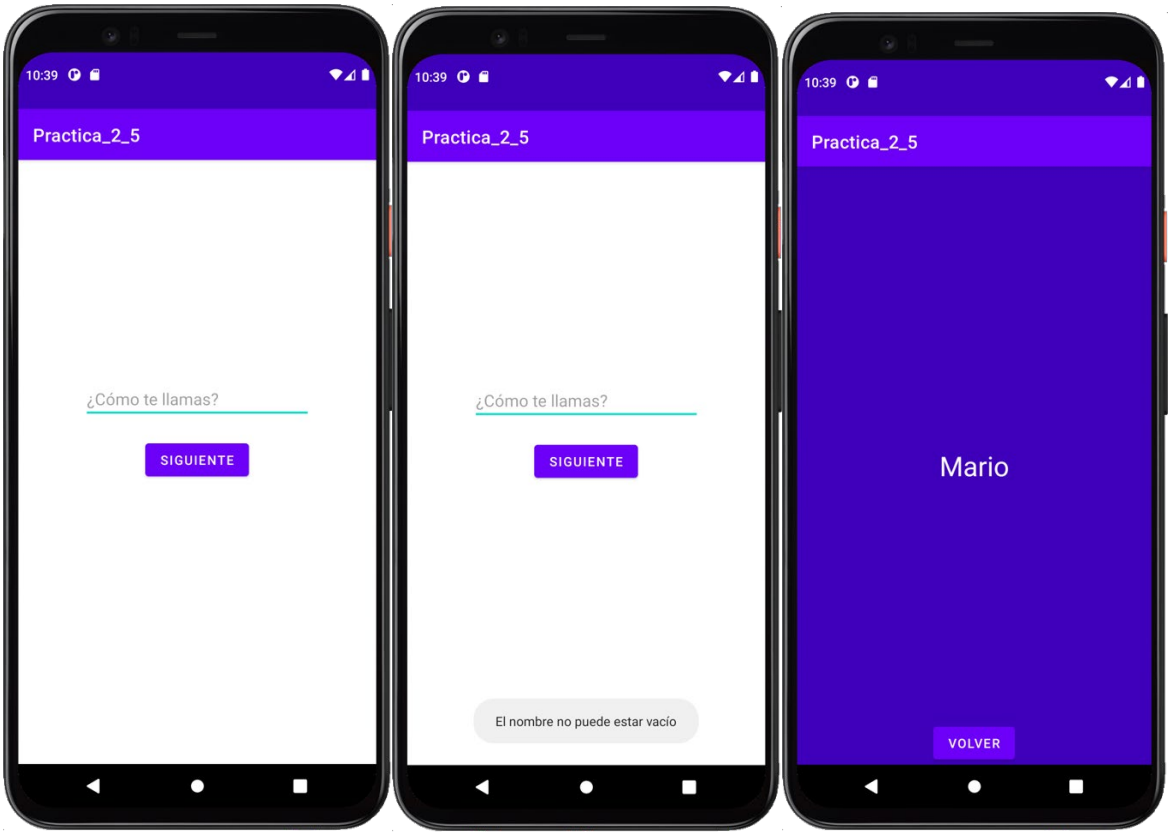


Una vez creado en la parte superior de nuestro IDE podemos acceder a nuestros dispositivos creados o conectados vía usb, seleccionamos el dispositivo recien creado.



PASO 11. Corremos nuestra aplicación

Al final nuestra aplicación se verá así.



TEMA 3. Material Design

Práctica 6 Material Design

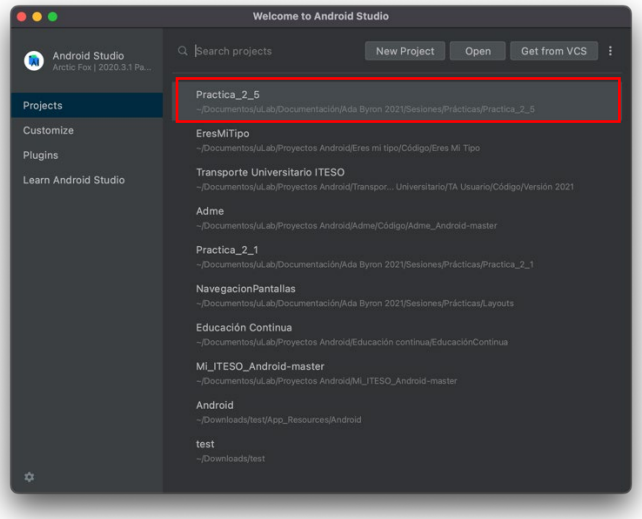
Objetivo específico	Modificar estilos de la aplicación
Recursos	Android Studio XML Kotlin
Actividades	Modificaremos el color adaptando los colores para los modos claro y obscuro
Producto(s)	Al final de la práctica deberá adaptar los colores para cunado se cambie el modo obscuro del dispositivo.

PASO 1. Abrir Android Studio.

Abrir Android Studio

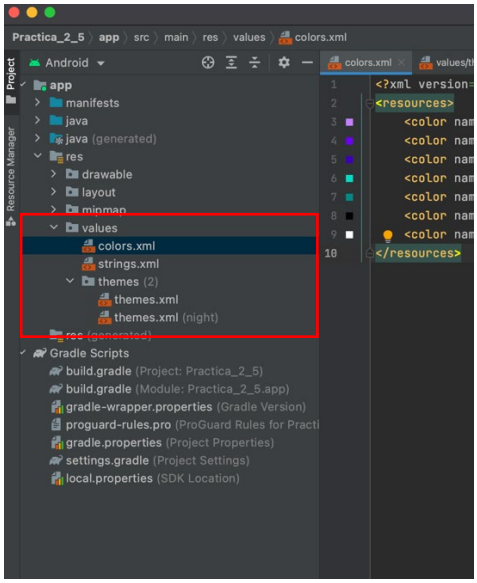
PASO 2. Elegir la Practica_2_5

Seleccionar Practica_2_5



PASO 3. Abrir colors.xml

Abriremos colors.xml en la ruta app/res/values



PASO 4. Agregar nuevos colores

Agregamremos nuevos colores, agragaremos un color azul y para esto necesitaremos su código hexadecimal.

```
<color name="azul">#3498db</color>
```

Podemos agregar los colores que deseamos. Agregue algún otro color de su preferencia.
Su código quedaría así

```
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <!-- Nuevos colores -->
    <color name="azul">#3498db</color>
    <color name="naranja">#d35400</color>
</resources>
```

PASO 5. Agregaremos el color a nuestro tema

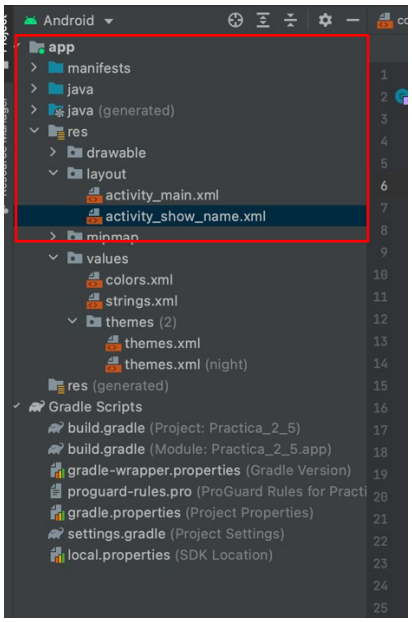
Para cambiar el color de nuestra vista debemos abrir themes.xml que se encuentra en la carpeta themes. Abriremos solo el que dice themes para modificar el modo claro.

Modificaremos el colorPrimary con el color azul que acabamos de agregar. En la línea 5 quedaría así.

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

PASO 6. Cambiaremos el fondo de color de nuestra primera activity

Abrimos nuestra activity_main



En nuestro RelativeLayout agregamos el atributo background y agregamos #edbb99, con esto veremos en automático que se cambia en nuestro visor. El código quedará de la siguiente manera.

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:background="#edbb99"
    tools:context=".MainActivity">
```

Resultado.

PASO 7. Modo dark

Para configurar la aplicación para el modo agregaremos un color nuevo en theme y otro color nuevo en theme (night)

Modo Light

```
<color name="background">#cd6155</color>
```

Así quedaría el código

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Practica_2_5"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/azul</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor"
tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
```

```
    </style>
    <color name="background">#cd6155</color>
</resources>
```

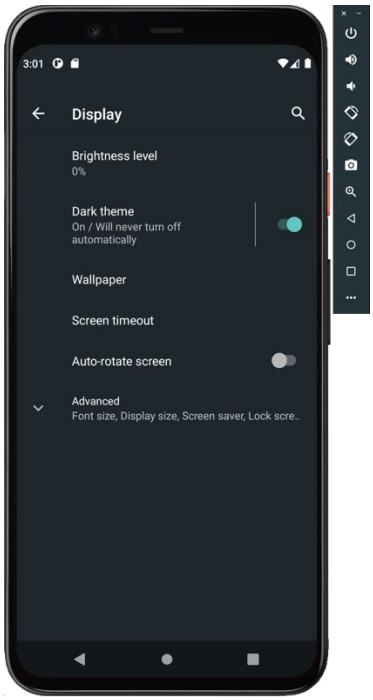
Modo dark

```
<color name="background">#85929e</color>
```

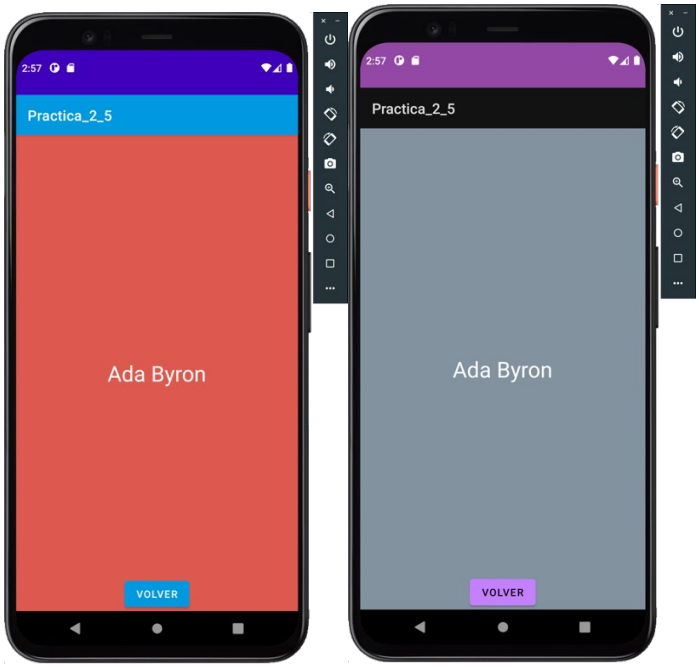
Así quedaría el código.

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Practica_2_5"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_200</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/black</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_200</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l"> #884ea0
</item>
        <!-- Customize your theme here. -->
    </style>
    <color name="background">#85929e</color>
</resources>
```

Para configurar el modo dark en el simulador hay que ir a ajustes, el simulador se comporta igual que un smartphone con sistema android, luego vamos a display y en Dark theme lo activamos.



Resultado



Modo light Modo dark

TEMA 4. Programación avanzada

Práctica 7 Data classes

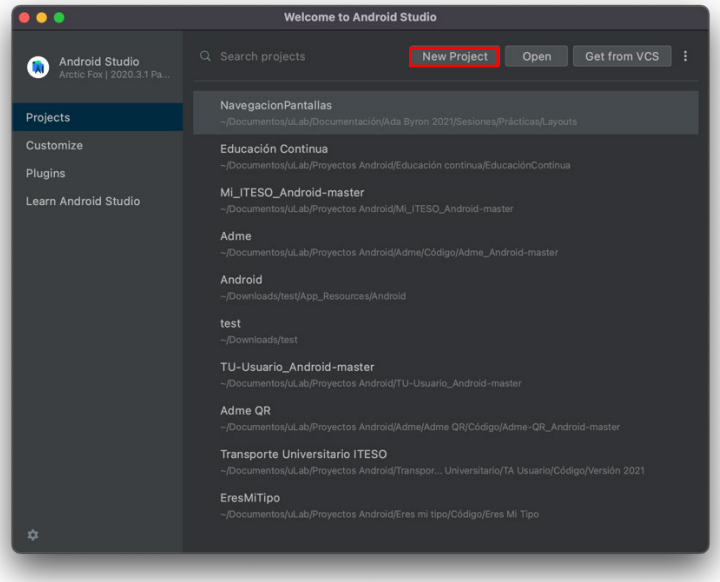
Objetivo específico	Manejo de data classes
Recursos	Android Studio Kotlin
Actividades	Crear una data class para recibir objetos de web service
Producto(s)	Al final de la práctica deberá saber como usar un data class y preparar la aplicación para recibir información de una base de datos.

PASO 1. Abrir Android Studio.

Abrir Android Studio

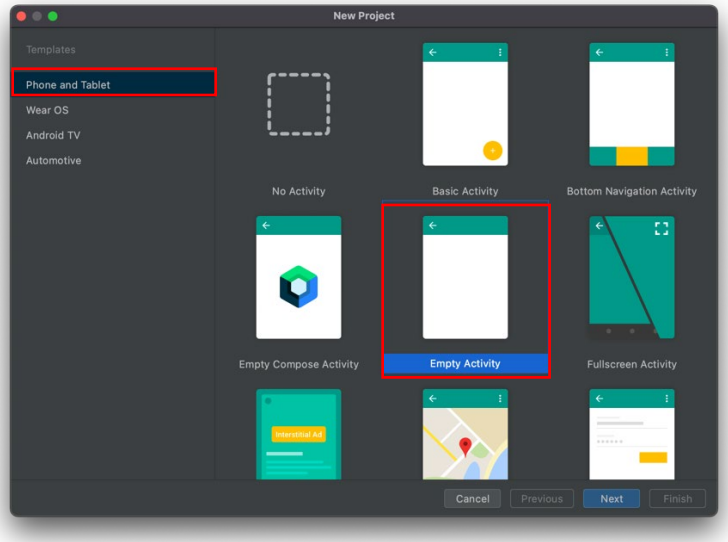
PASO 2. Elegir nuevo proyecto

Seleccionar New project



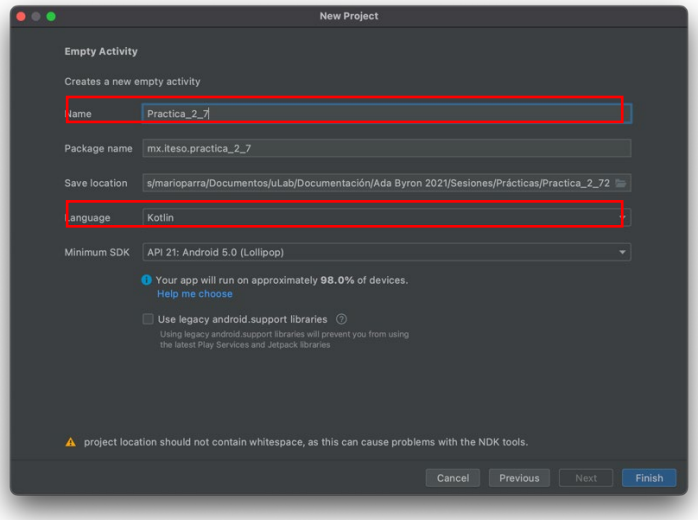
PASO 3. Elegir Empty Activity

Seleccionar Phone and Tablet, luego seleccionar Empty Activity, y dar click en Next


New Project
Templates
Phone and Tablet
Wear OS
Android TV
Automotive
No Activity
Basic Activity
Bottom Navigation Activity
Empty Compose Activity
Empty Activity
Fullscreen Activity
Interstitial Ad
Cancel
Previous
Next
Finish

PASO 4. Configuración del nuevo proyecto

En nombre escribir Practica_2_7 y seleccionar el lenguaje Kotlin, después dar click en Finish



PASO 5. Abrir MainActivity.kt

Crearemos nuestra data class llamada movie que tendrá 5 atributos Title, Year, imdbID, Type y Poster.

```
data class movie(
    var Title: String,
    var Year: String,
    var imdbID: String,
    var Type: String,
    var Poster: String
)
```

Ahora crearemos una lista mutable de tipo movie.

```
var movies : MutableList<Movie> = mutableListOf()
```

Una vez creado ahora agregaremos algunas películas a nuestra lista.

```
movies.add(Movie("Captain Marvel", "2019", "tt4154664", "movie",
"https://m.media-
amazon.com/images/M/MV5BZDIwZTM4M2QtMWFhYy00N2VmLWFlMjItMzI3NjBjYTc0OTMxXkEyX
kFqcGdeQXVyNTE1NjY5Mg@@._V1_SX300.jpg"))
movies.add(Movie("Marvel Studios: Assembled", "2021", "tt14094206", "series",
"https://m.media-
amazon.com/images/M/MV5BNGQ0MjI2ZjctNzk3Zi00ZWYyLThhMDAtMWIwNmJkYWVlYjk4XkEyX
kFqcGdeQXVyMTU5OTc2NTk@._V1_SX300.jpg"))
```

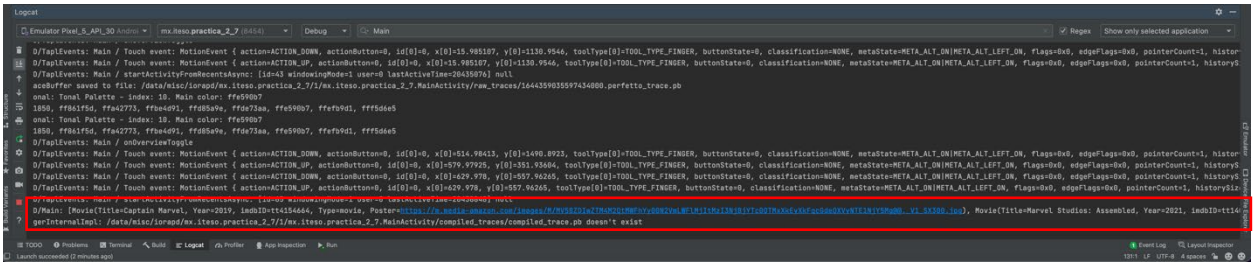
Imprimiremos en el logcat

```
Log.d("Main", movies.toString())
```

El Código quedaría de la siguiente forma.

```
class MainActivity : AppCompatActivity() {

    var movies : MutableList<Movie> = mutableListOf()

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

        movies.add(Movie("Captain Marvel", "2019", "tt4154664", "movie",
"https://m.media-
amazon.com/images/M/MV5BZDIwZTM4M2QtMWFhYy00N2VmLWFlMjItMzI3NjBjYTc0OTMxXkEyX
kFqcGdeQXVyNTE1NjY5Mg@@._V1_SX300.jpg"))
        movies.add(Movie("Marvel Studios: Assembled", "2021", "tt14094206",
"series", "https://m.media-
amazon.com/images/M/MV5BNGQ0MjI2ZjctNzk3Zi00ZWYyLThhMDAtMWIwNmJkYWVlYjk4XkEyX
kFqcGdeQXVyMTU5OTc2NTk@._V1_SX300.jpg"))

        Log.d("Main", movies.toString())
    }
}

data class Movie(
    var Title: String,
    var Year: String,
    var imdbID: String,
    var Type: String,
    var Poster: String
)
```

Resultado



Práctica 8 RecyclerView

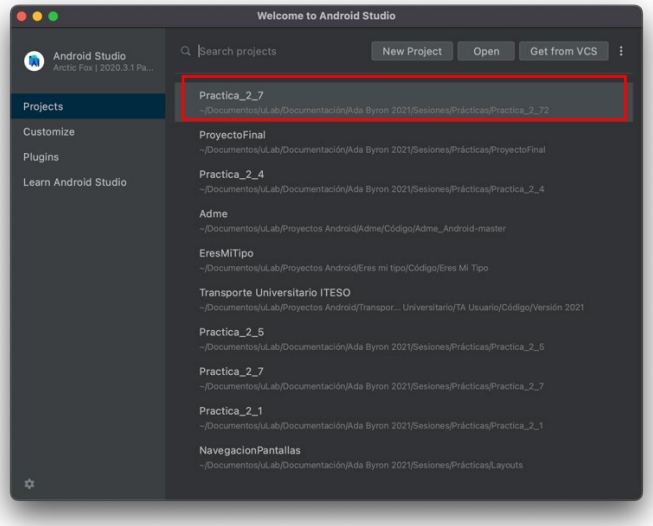
Objetivo específico	Aprender a utilizar un recyclerView
Recursos	Android Studio XML Kotlin
Actividades	Agregaremos un recyclerView para visualizar una lista, no solo contendrá texto sino también imágenes.
Producto(s)	Al final de la práctica deberá agregar un recyclerView donde se verá una lista que obtuvimos como resultado de la práctica anterior.

PASO 1. Abrir Android Studio.

Abrir Android Studio

PASO 2. Elegir la Practica_2_7

Seleccionar Practica_2_7



PASO 3. Configuración del proyecto

Abriremos build.gradle (module) y configuramos nuestro viewBinding en la sección android.

```
buildFeatures{
    viewBinding = true
}

viewBinding {
    enabled = true
}
```

Ahora implementaremos 2 dependencias. Una dependencia es básicamente código externo del proyecto el cual podemos implementar para aprovecharlo, en este caso implementaremos el RecyclerView y la librería llamada Picasso que la usaremos para la gestión de imágenes.

```
//Recycler view
implementation 'androidx.recyclerview:recyclerview:1.1.0'
//picasso
implementation 'com.squareup.picasso:picasso:2.71828'
```

Ahora sincronizamos el proyecto, al modificar este archivo en automático nos aparecerá un aviso de sincronizar el proyecto. El código quedaría de la siguiente manera.

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
}

android {
    compileSdk 32

    defaultConfig {
        applicationId "mx.iteso.practica_2_7"
        minSdk 21
        targetSdk 32
        versionCode 1
```

```
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }

    buildFeatures{
        viewBinding = true
    }

    viewBinding {
        enabled = true
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'

    //Recycler view
    implementation 'androidx.recyclerview:recyclerview:1.1.0'
    //picasso
    implementation 'com.squareup.picasso:picasso:2.71828'
}
```

PASO 4. Creamos el recyclerView

Abrimos activity_main.xml, cambiaremos el constraintLayout por RelativeLayout. Ahora eliminamos el textView que aparece y lo reemplazaremos con nuestro recyclerView. Así quedaría el código.

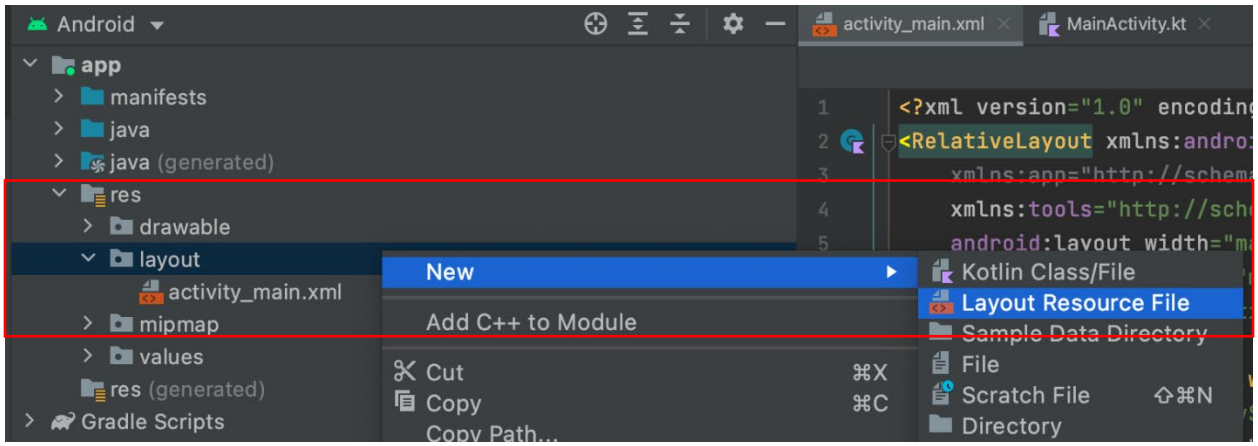
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvMovieList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Ya creamos nuestra lista, pero ahora falta crear las celdas de la lista que son donde se mostrará la información deseada en pantalla, a estas celdas es a las que se les dan el diseño.

PASO 5. Creamos la celda

Vamos a res/layout, damos click derecho New/Layout Resource File.



Le damos el nombre item_movie y damos click en OK. Una vez creado vamos a cambiar la vista a split. Aquí cambiaremos el ConstraintLayout por RelativeLayout. La celda tendra orientación vertical y un margen general de 10dp, esto quiere decir que de todos lados tendrá el mismo margen. La altura la cambiaremos por wrap_content, esto significa que crecerá dependiendo del contenido que contenga.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_margin="10dp">
```

Agregamos una imagen y 2 textView.

```
<ImageView
    android:id="@+id/ivPoster"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_marginEnd="10dp"
    android:layout_marginRight="10dp"
    android:scaleType="centerCrop" />

<TextView
    android:id="@+id/tvTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:textStyle="bold"
    android:textSize="20sp"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/ivPoster"
    android:layout_toEndOf="@+id/ivPoster"
    android:text="" />

<TextView
    android:id="@+id/tvType"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="15sp"
    android:layout_below="@+id/tvTitle"
    android:layout_toRightOf="@+id/ivPoster"
    android:layout_toEndOf="@+id/ivPoster"
    android:text=""/>
```

El código quedaría de la siguiente forma.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_margin="10dp">

    <ImageView
        android:id="@+id/ivPoster"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_marginEnd="10dp"
        android:layout_marginRight="10dp"
        android:scaleType="centerCrop" />

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@+id/ivPoster"
        android:layout_toEndOf="@+id/ivPoster"
        android:text="" />

    <TextView
        android:id="@+id/tvType"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="15sp"
        android:layout_below="@+id/tvTitle"
        android:layout_toRightOf="@+id/ivPoster"
        android:layout_toEndOf="@+id/ivPoster"
        android:text=""/>

</RelativeLayout>
```

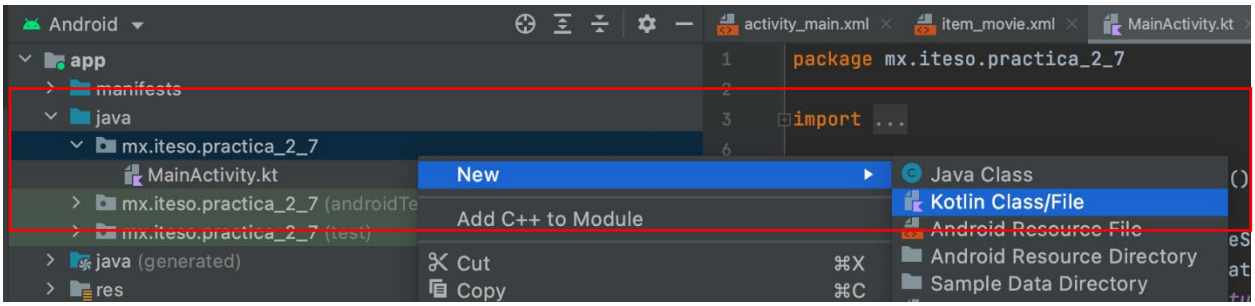
PASO 6. Creamos los objetos movies

Volvemos a MainActivity.kt. Vamos a crear dos funciones, una donde configuremos nuestro recyclerview con un adapter y la otra la que generará la lista de objetos de movies.

```
class MainActivity : AppCompatActivity() {

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

        setUpRecyclerView()

    }

    fun setUpRecyclerView() {

    }

    fun getMovie(): MutableList<Movie> {
        var movies : MutableList<Movie> = mutableListOf()
        movies.add(Movie("Captain Marvel", "2019", "tt4154664", "movie",
"https://m.media-
amazon.com/images/M/MV5BZDIwZTM4M2QtMWFhYy00N2VmLWFlMjItMzI3NjBjYTc0OTMxXkEyX
kFqcGdeQXVyNTE1NjY5Mg@@._V1_SX300.jpg"))
        movies.add(Movie("Marvel Studios: Assembled", "2021", "tt14094206",
"movie", "https://m.media-
amazon.com/images/M/MV5BNGQ0MjI2ZjctNzk3Zi00ZWYyLThhMDAtMWIwNmJkYWVlYjk4XkEyX
kFqcGdeQXVyMTU5OTc2NTk@._V1_SX300.jpg"))
        movies.add(Movie("Game of Thrones", "2012", "tt2972984", "series",
"https://m.media-
amazon.com/images/M/MV5BMDM0ZDlkOTktODkwNi00OGE4LTg3ZDEtODc3OGIyNmNiMWUyXkEyX
kFqcGdeQXVyNzgxNDk0NTI@._V1_SX300.jpg"))
        movies.add(Movie("The Wonderful World of Disney", "1997",
"tt0132666", "movie", "https://m.media-
amazon.com/images/M/MV5BNmJjYzNkNjgtZDZiNS00YjdmLTk4MDEtMzJkYWFkZmQ3ZWFiXkEyX
kFqcGdeQXVyMjIzMTQ5NjE@._V1_SX300.jpg"))
        movies.add(Movie("The Lego Batman Movie", "2017", "tt4116284",
"movie", "https://m.media-
amazon.com/images/M/MV5BMTcyNTEyOTY0M15BMl5BanBnXkFtZTgwOTAyNzU3MDI@._V1_SX30
0.jpg"))
        return movies
    }
}

data class Movie(
    var Title: String,
    var Year: String,
    var imdbID: String,
    var Type: String,
    var Poster: String
)
```

PASO 7. Adapter

Ahora vamos a crear el adapter del RecyclerView. Un adapter es la clase que hace de puente entre la vista (el recyclerView) y los datos. Vamos a crear una clase llamada RecyclerAdapter. Esta clase se encargará de recorrer la lista de movies que le pasaremos más tarde. Vamos a la ruta java/practica_2_7, damos click derecho New/kotlin Class/file.



Copiamos y pagamos el siguiente código.

```
class RecyclerAdapter : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>() {

    var movies: MutableList<Movie>  = ArrayList()
    lateinit var context: Context

    fun RecyclerAdapter(movies : MutableList<Movie>, context: Context){
        this.movies = movies
        this.context = context
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = movies.get(position)
        holder.bind(item, context)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        return ViewHolder(layoutInflater.inflate(R.layout.item_movie, parent,
false))
    }

    override fun getItemCount(): Int {
        return movies.size
    }

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val title = view.findViewById(R.id.tvTitle) as TextView
        val type = view.findViewById(R.id.tvType) as TextView
        val poster = view.findViewById(R.id.ivPoster) as ImageView

        fun bind(movie:Movie, context: Context){
            title.text = movie.Title
            type.text = movie.Type
            itemView.setOnClickListener(View.OnClickListener {
Toast.makeText(context, movie.Title, Toast.LENGTH_SHORT).show() })
            Picasso.get().load(movie.Poster).into(poster)
        }
```

```
    }
}
```

La función RecyclerAdapter es un simple constructor que tiene le mismo nombre de la clase y que lo único que hará será recibir la lista y el contexto que le pasamos desde la clase MainActivity.

Los siguientes 3 métodos tienen la palabra reservada override que es porque son métodos obligatorios que se implemetan de la clase RecyclerView. onBindViewHolder() se encarga de recoger cada una de las posiciones de la lista de movies y pasarlas a la clase ViewHolder para que esta pinte todos los valores.

Después tenemos onCreateViewHolder() que como su nombre indica nos devolverá un objeto de ViewHolder al cuál le pasamos la celda que hemos creado.

Y para finalizar el método getItemCount() nos devuelve el tamaño de la lista.

En la clase ViewHolder se hace referencia a los items de la celda, el view.findViewByID() busca los items a través de la id que le ponemos y luego añadimmos as y enseguida el tipo que es el objeto (TextView, ImageView). Dentro del método bind rellenamos los datos que vamos a mostrar.

PASO 8. Agregar Permiso

Ahora necesitamos agregar el permiso de internet. Abrimos AndroidManifest y agregamos el siguiente código.

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

PASO 9. Completamos el método setUpRecyclerView

Volvemos a MainActivity y creamos las dos variables de recyclerView y del adapter.

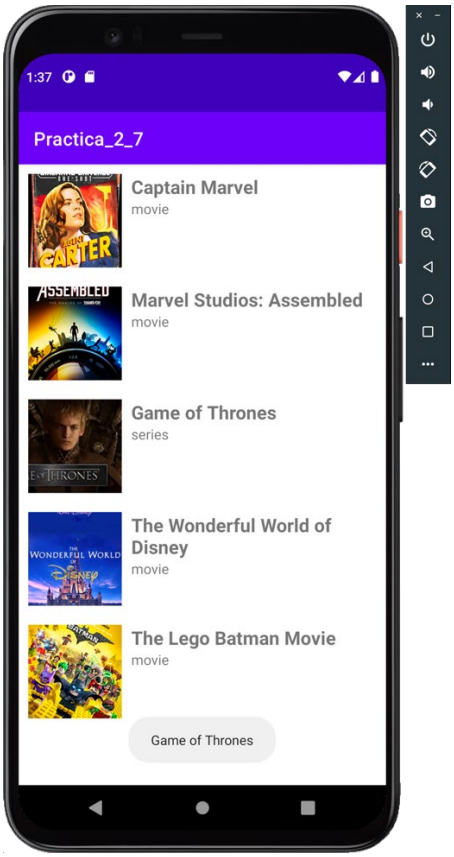
```
lateinit var mRecyclerView : RecyclerView
val mAdapter : RecyclerAdapter = RecyclerAdapter()
```

Después en el método setUpRecyclerView() buscaremos el layout del recycler y lo asiganmos a la variable creada anteriormente y terminaremos de configurar nuestro recyclerView. Configuramos el adapter donde le mandaremos la lista de movies que tenemos para despues asignarlo a nuestro recyclerView.

```
fun setUpRecyclerView() {
    mRecyclerView = findViewById(R.id.rvMovieList) as RecyclerView
    mRecyclerView.setHasFixedSize(true)
    mRecyclerView.layoutManager = LinearLayoutManager(this)
    mAdapter.RecyclerAdapter(getMovies(), this)
    mRecyclerView.adapter = mAdapter
}
```

PASO 10. Corremos la aplicación

Resultado


1:37
Practica_2_7
Captain Marvel
movie
Marvel Studios: Assembled
movie
Game of Thrones
series
The Wonderful World of Disney
movie
The Lego Batman Movie
movie
Game of Thrones

TEMA 4. Proyecto Final

Práctica 9 Proyecto final parte 1

Objetivo específico	Desarrollar una aplicación que sea un buscador de películas donde muestre una lista del resultado de la búsqueda.
Recursos	Android Studio Kotlin
Actividades	Crearemos la vista del buscador donde se despliegue el resultado en un RecyclerView
Producto(s)	Aplicación corriendo en el emulador mostrando el buscador con la lista de resultados de las películas encontradas.

PASO 1. Abrir Android Studio.

Abrir Android Studio

PASO 2. Crearemos un nuevo proyecto

El nombre de la aplicación será ProyectoFinal, no olvidemos configurar el ViewBinding y agregar la librería de RecyclerView y Picasso para mostrar imágenes.

PASO 5. Agregaremos nuevos colores

Agregaremos dos nuevos colores en colors.xml

```
<color name="rosa">#F7CFCF</color>
<color name="morado">#3B3BE5</color>
```

En themes cambiaremos el colorPrimary y colorPrimaryVariant por el color rosa

```
<item name="colorPrimary">@color/rosa</item>
<item name="colorPrimaryVariant">@color/rosa</item>
```

PASO 6. Crearemos un fondo degradado

Iremos a res/drawable y crearemos un nuevo drawable con el nombre gradient

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="@color/morado"
        android:endColor="@color/rosa"
        android:angle="90" />
</shape>
```

PASO 7. Modificamos nuestro activity_main.xml

Agregamos el searchView donde haremos nuestra búsqueda y el recyclerView donde mostraremos el resultado.

```
<RelativeLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/gradient"
    tools:context=".MainActivity">

    <SearchView
        android:id="@+id/searchView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:iconifiedByDefault="false"
        android:queryHint="Películas"/>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvMovieList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/searchView"/>

</RelativeLayout>
```

PASO 8. Creamos la celda item_movie

Primero crearemos el shape con esquinas redondeadas. Vamos a la ruta res/drawable y llamamos al archivo layout_rounded.

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <solid android:color="#FFFFFF"/>
    <corners android:radius="10dp"/>
    <padding android:left="0dp" android:top="0dp" android:right="0dp"
android:bottom="0dp" />
</shape>
```

Ahora crearemos la celda de la película donde tendremos una imagen y un textView del título de la película. Agregamos en el background del RelativeLayout el shape que creamos antes. La imagen tendrá las esquinas redondeadas, para hacer esto debemos poner el imageView dentro de un cardView y será a este mismo a quien le pondremos las esquinas redondeadas.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="150dp"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:background="@drawable/layout_rounded"
    android:layout_margin="20dp">

    <androidx.cardview.widget.CardView
        android:id="@+id/cv_poster"
```

```
        android:layout_width="100dp"
        android:layout_height="150dp"
        android:layout_margin="10dp"
        app:cardCornerRadius="15dp">

        <ImageView
            android:id="@+id/ivPoster"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@color/rosa"
            android:scaleType="centerCrop"/>

    </androidx.cardview.widget.CardView>

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp"
        android:layout_alignTop="@id/cv_poster"
        android:layout_toRightOf="@+id/cv_poster"
        android:layout_toEndOf="@+id/cv_poster"
        android:text="" />

</RelativeLayout>
```

PASO 9. Creamos objetos movies

Primero crearemos el objeto Movie.

```
data class Movie(
    var Title: String,
    var Year: String,
    var imdbID: String,
    var Type: String,
    var Poster: String
)
```

Volvemos a MainActivity y creamos los objetos movies y las instancias para searchView y recyclerView.

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

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

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setUpRecyclerView()
    }
```

```
    fun setUpRecyclerView() {

    }

    fun getMovies(): MutableList<Movie> {
        var movies : MutableList<Movie> = mutableListOf()
        movies.add(Movie("Captain Marvel", "2019", "tt4154664", "movie",
"https://m.media-
amazon.com/images/M/MV5BZDIwZTM4M2QtMWFhYy00N2VmLWFlMjItMzI3NjBjYTc0OTMxXkEyX
kFqcGdeQXVyNTE1NjY5Mg@@._V1_SX300.jpg"))
        movies.add(Movie("Marvel Studios: Assembled", "2021", "tt14094206",
"movie", "https://m.media-
amazon.com/images/M/MV5BNGQ0MjI2ZjctNzk3Zi00ZWYyLThhMDAtMWIwNmJkYWVlYjk4XkEyX
kFqcGdeQXVyMTU5OTc2NTk@._V1_SX300.jpg"))
        movies.add(Movie("Game of Thrones", "2012", "tt2972984", "series",
"https://m.media-
amazon.com/images/M/MV5BMDM0ZDlkOTktODkwNi00OGE4LTg3ZDEtODc3OGIyNmNiMWUyXkEyX
kFqcGdeQXVyNzgxNDk0NTI@._V1_SX300.jpg"))
        movies.add(Movie("The Wonderful World of Disney", "1997",
"tt0132666", "movie", "https://m.media-
amazon.com/images/M/MV5BNmJjYzNkNjgtZDZiNS00YjdmLTk4MDEtMzJkYWFkZmQ3ZWFiXkEyX
kFqcGdeQXVyMjIzMTQ5NjE@._V1_SX300.jpg"))
        movies.add(Movie("The Lego Batman Movie", "2017", "tt4116284",
"movie", "https://m.media-
amazon.com/images/M/MV5BMTcyNTEyOTY0M15BMl5BanBnXkFtZTgwOTAyNzU3MDI@._V1_SX30
0.jpg"))
        return movies
    }
}
```

PASO 10. Creamos el RecyclerAdapter

Crearemos el adapter para poder mostrar la lista en el recyclerView.

```
class RecyclerAdapter : RecyclerView.Adapter<RecyclerAdapter.ViewHolder>() {

    var movies: List<DataMovie>  = listOf()
    lateinit var context: Context

    fun RecyclerAdapter(movies : List<DataMovie>, context: Context){
        this.movies = movies
        this.context = context
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = movies.get(position)
        holder.bind(item, context)
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ViewHolder {
        val layoutInflater = LayoutInflater.from(parent.context)
        return ViewHolder(layoutInflater.inflate(R.layout.item_movie, parent,
false))
    }
```

```
    override fun getItemCount(): Int {
        return movies.size
    }

    class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
        val title = view.findViewById(R.id.tvTitle) as TextView
        val poster = view.findViewById(R.id.ivPoster) as ImageView

        fun bind(movie:DataMovie, context: Context){
            title.text = movie.Title
            itemView.setOnClickListener(View.OnClickListener {
                val intent = Intent(context,
DetalleMovieActivity::class.java)
                intent.putExtra("Title", movie.Title)
                context.startActivity(intent)
            })
            Picasso.get().load(movie.Poster).into(poster)
        }
    }
}
```

PASO 11. Terminamos de configurar nuestro adapter.

Volvemos a nuestro MainActivity y configuramos el adapter con nuestro recyclerView.

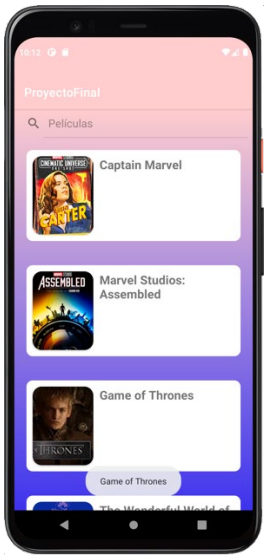
```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

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

    val adapter : RecyclerAdapter = RecyclerAdapter()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        searchView = binding.searchView
        recyclerView = binding.rvMovieList

        setUpRecyclerView()
    }

    fun setUpRecyclerView() {
        recyclerView.setHasFixedSize(true)
        recyclerView.layoutManager = LinearLayoutManager(this)
        adapter.RecyclerAdapter(getMovies(), this)
        recyclerView.adapter = adapter
    }

    fun getMovies(): MutableList<Movie> {
        var movies : MutableList<Movie> = mutableListOf()
        movies.add(Movie("Captain Marvel", "2019", "tt4154664", "movie",
"https://m.media-
```

```
amazon.com/images/M/MV5BZDIwZTM4M2QtMWFhYy00N2VmLWFlMjItMzI3NjBjYTc0OTMxXkEyX
kFqcGdeQXVyNTE1NjY5Mg@@._V1_SX300.jpg"))
        movies.add(Movie("Marvel Studios: Assembled", "2021", "tt14094206",
"movie", "https://m.media-
amazon.com/images/M/MV5BNGQ0MjI2ZjctNzk3Zi00ZWYyLThhMDAtMWIwNmJkYWVlYjk4XkEyX
kFqcGdeQXVyMTU5OTc2NTk@._V1_SX300.jpg"))
        movies.add(Movie("Game of Thrones", "2012", "tt2972984", "series",
"https://m.media-
amazon.com/images/M/MV5BMDM0ZDlkOTktODkwNi00OGE4LTg3ZDEtODc3OGIyNmNiMWUyXkEyX
kFqcGdeQXVyNzgxNDk0NTI@._V1_SX300.jpg"))
        movies.add(Movie("The Wonderful World of Disney", "1997",
"tt0132666", "movie", "https://m.media-
amazon.com/images/M/MV5BNmJjYzNkNjgtZDZiNS00YjdmLTk4MDEtMzJkYWFkZmQ3ZWFiXkEyX
kFqcGdeQXVyMjIzMTQ5NjE@._V1_SX300.jpg"))
        movies.add(Movie("The Lego Batman Movie", "2017", "tt4116284",
"movie", "https://m.media-
amazon.com/images/M/MV5BMTcyNTEyOTY0M15BMl5BanBnXkFtZTgwOTAyNzU3MDI@._V1_SX30
0.jpg"))
        return movies
    }
}
```

Antes de correr nuestra aplicación debemos de dar el permiso para tener acceso a internet. Vamos a AndroidManifest en la ruta app/manifests.

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

Corremos nuestra aplicación y se vería así.



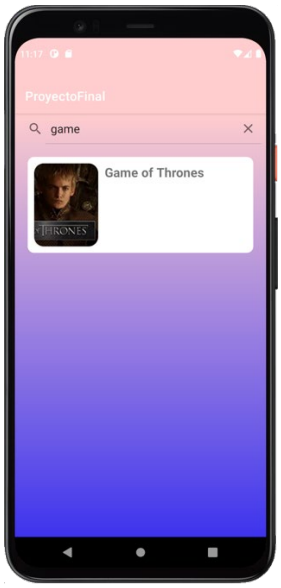
PASO 12. Configuraremos nuestro SearchView

Ahora configuraremos el searchView donde cada que agreguemos un carácter nuevo hará la búsqueda y mostrará el resultado en la lista. Comparará cada carácter de nuestra búsqueda con la lista de películas y guardará el resultado en una nueva lista de resultados que será la información que se mostrará al final.

```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

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

    var adapter : RecyclerAdapter = RecyclerAdapter()

    private var movies = mutableListOf<Movie>()
    private var matchedMovie: ArrayList<Movie> = arrayListOf()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        searchView = binding.searchView
        recyclerView = binding.rvMovieList

        setUpRecyclerView()
    }

    fun setUpRecyclerView() {
        val movies = getMovies()
        recyclerView.setHasFixedSize(true)
        recyclerView.layoutManager = LinearLayoutManager(this)
        adapter.RecyclerAdapter(movies, this)
        recyclerView.adapter = adapter

        searchView.setOnQueryTextListener(object :
SearchView.OnQueryTextListener{
            override fun onQueryTextSubmit(query: String?): Boolean {
                searchView.clearFocus()
                search(query)
                return true
            }

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

    private fun search(text: String?) {
        matchedMovie = arrayListOf()

        text?.let {
            movies.forEach { movie ->
                if (movie.Title.contains(text, true)) {
                    matchedMovie.add(movie)
                }
            }
            updateRecyclerView()
            if (matchedMovie.isEmpty()) {
                Toast.makeText(this, "No match found!",
```

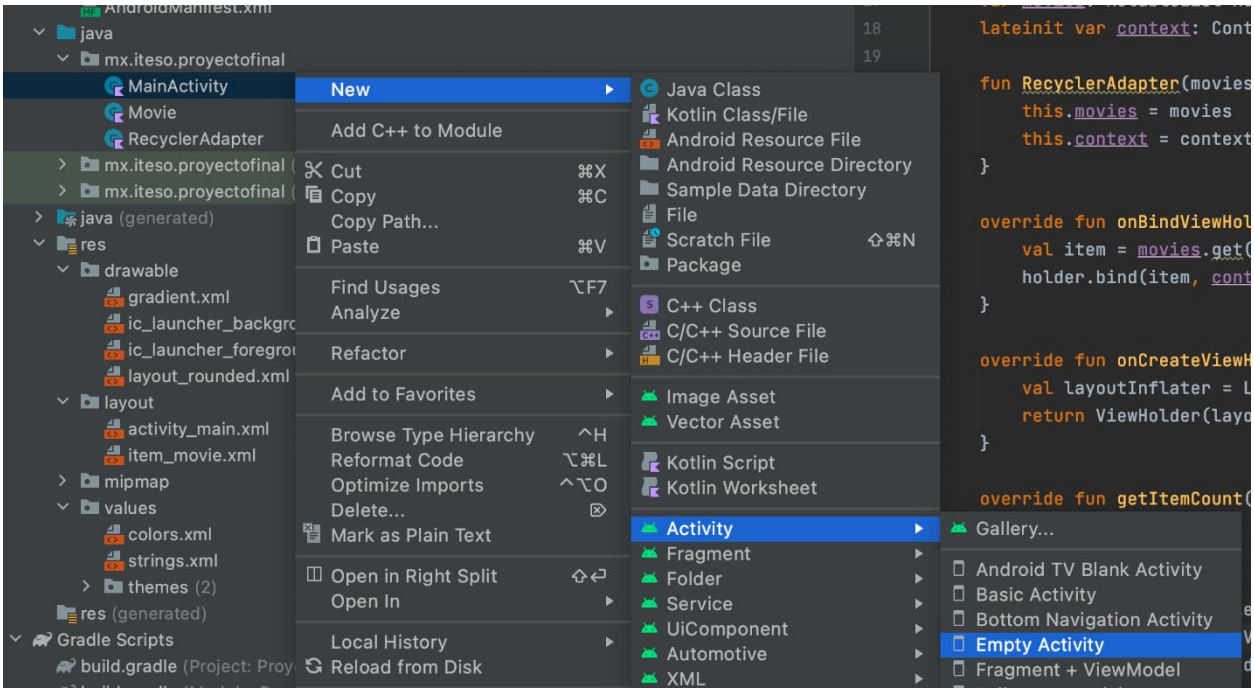
```
Toast.LENGTH_SHORT).show()
            }
            updateRecyclerView()
        }
    }

    private fun updateRecyclerView() {
        adapter.RecyclerAdapter(matchedMovie, this)
        recyclerView.adapter = adapter
        adapter.notifyDataSetChanged()
    }

    fun getMovies(): MutableList<Movie> {
        movies.add(Movie("Captain Marvel", "2019", "tt4154664", "movie",
"https://m.media-
amazon.com/images/M/MV5BZDIwZTM4M2QtMWFhYy00N2VmLWFlMjItMzI3NjBjYTc0OTMxXkEyX
kFqcGdeQXVyNTE1NjY5Mg@@._V1_SX300.jpg"))
        movies.add(Movie("Marvel Studios: Assembled", "2021", "tt14094206",
"movie", "https://m.media-
amazon.com/images/M/MV5BNGQ0MjI2ZjctNzk3Zi00ZWYyLThhMDAtMWIwNmJkYWVlYjk4XkEyX
kFqcGdeQXVyMTU5OTc2NTk@._V1_SX300.jpg"))
        movies.add(Movie("Game of Thrones", "2012", "tt2972984", "series",
"https://m.media-
amazon.com/images/M/MV5BMDM0ZDlkOTktODkwNi00OGE4LTg3ZDEtODc3OGIyNmNiMWUyXkEyX
kFqcGdeQXVyNzgxNDk0NTI@._V1_SX300.jpg"))
        movies.add(Movie("The Wonderful World of Disney", "1997",
"tt0132666", "movie", "https://m.media-
amazon.com/images/M/MV5BNmJjYzNkNjgtZDZiNS00YjdmLTk4MDEtMzJkYWFkZmQ3ZWFiXkEyX
kFqcGdeQXVyMjIzMTQ5NjE@._V1_SX300.jpg"))
        movies.add(Movie("The Lego Batman Movie", "2017", "tt4116284",
"movie", "https://m.media-
amazon.com/images/M/MV5BMTcyNTEyOTY0M15BMl5BanBnXkFtZTgwOTAyNzU3MDI@._V1_SX30
0.jpg"))
        return movies
    }
}
```

Correremos la aplicación y buscaremos Game, veremos como al ir escribiendo carácter por carácter se irá actualizando la lista. Como resultado veremos Game of Thrones.



PASO 13. Detalle de la película

Crearemos una nueva activity para la vista detalle de la película, vamos a la ruta app/proyectofinal damos click derecho New/Activity/Empty Activity y le damos el nombre DetalleMovieActivity.



Veremos como se crea DetalleMovieActivity.kt y activity_detalle_movie.xml. Ahora volvemos a la clase RecyclerAdapter y en nuestro método bind ubicado en la clase ViewHolder haremos un cambio en itemView.setOnClickListener

```
itemView.setOnClickListener(View.OnClickListener {
    val intent = Intent(context, DetalleMovieActivity::class.java)
    intent.putExtra("Title", movie.Title)
    context.startActivity(intent)
})
```

Aquí preparamos la segunda vista que mostraremos al dar tap en la celda que seleccionamos. PutExtra nos ayuda a enviar información entre activities, en este caso estamos mandando el título de la película que seleccionamos.

Vamos a la clase DetalleMovieActivity y recibiremos el título de la película que mandamos previamente.

```
class DetalleMovieActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDetalleMovieBinding

    private lateinit var textView : TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDetalleMovieBinding.inflate(layoutInflater)
        setContentView(binding.root)

        textView = binding.tvPelicula

        val pelicula = intent.getStringExtra("Title")

        textView.text = pelicula
    }
}
```

Con getStringExtra recibimos la información que guardamos anteriormente y como dice el método recibiremos un valor de tipo string y lo mostraremos en nuestr o textView.

Resultado.

Extra. Toolbar (Barra de acción)

La navegación entre pantallas es algo muy común en las aplicaciones y en ocasiones cuando avanzamos a una nueva vista podemos perdernos y no saber en que parte de la aplicación nos quedamos, para esto es que funciona el toolbar entre otras cosas como por ejemplo regresar a la vista anterior, el título de la vista, un menú de opciones, entre otros.

Agregaremos el toolbar en nuestra aplicación con el fin de ayudar al usuario a saber donde se encuentra ubicado y aprovecharemos para poner le flecha para regresar a la vista anterior.

Primero en nuestra paleta de colores agregaremos un nuevo color.

```
<color name="transparente">#00000000</color>
```

Después modificaremos los themes, tanto el light como el night. Solo modificaremos la primera línea donde empieza nuestro style, la supliremos por esta línea.

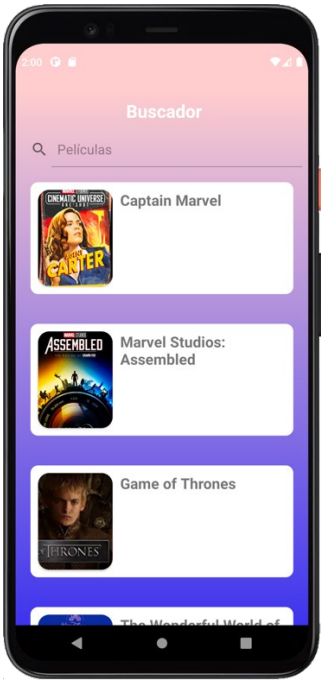
```
<style name="Theme.ProyectoFinal"
parent="Theme.MaterialComponents.Light.NoActionBar">
```

Ahora nos posicionaremos en la clase activity_main.xml donde agregaremos nuestro toolbar entre el RelativeLayout y SearchView quedando de la siguiente manera el código.

```
<RelativeLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/gradient"
    tools:context=".MainActivity">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/activity_main_tb"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/transparente"
        app:layout_constraintTop_toTopOf="parent">

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_margin="10dp">

            <TextView
                android:id="@+id/toolbar_main"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="Buscador"
                android:textColor="@color/white"
                android:textSize="24sp"
                android:textStyle="bold" />

        </RelativeLayout>
```

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

    <SearchView
        android:id="@+id/searchView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/activity_main_tb"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        android:iconifiedByDefault="false"
        android:queryHint="Películas"/>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvMovieList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/searchView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"/>

</RelativeLayout>
```

Al correr el código quedará de la siguiente manera.



Antes de ir a la segunda vista debemos crear la flecha con la que vamos a volver a la primera vista. Vamos a la ruta res/drawable y crearemos un nuevo drawable.

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24.0"
```

```
    android:viewportHeight="24.0">
    <path
        android:fillColor="#FF000000"
        android:pathData="M20,11H7.83l5.59,-5.59L12,4l-8,8 8,8 1.41,-
1.41L7.83,13H20v-2z"/>
</vector>
```

Ahora si lo agregaremos el toolbar en la vista detalle, pero ahora agregaremos la flecha con la que volveremos a la vista anterior.

```
<RelativeLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/gradient"
    tools:context=".DetalleMovieActivity">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/activity_detalle_tb"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@color/transparente"
        app:layout_constraintTop_toTopOf="parent">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent">

            <ImageView
                android:id="@+id/button_back"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:src="@drawable/ic_arrow_back_black_24dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:tint="@color/white"/>

            <TextView
                android:id="@+id/toolbar_detalle"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="Detalle"
                android:textColor="@color/white"
                android:textSize="24sp"
                android:textStyle="bold" />

        </androidx.constraintlayout.widget.ConstraintLayout>

    </androidx.appcompat.widget.Toolbar>

    <TextView
        android:id="@+id/tvPelicula"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:textColor="@color/white"
        android:text="Nombre"
        android:layout_centerInParent="true"
        android:textSize="30sp"/>

</RelativeLayout>
```

Ahora le daremos la acción a la imagen de la flecha, para esto vamos a la clase DetalleMovieActivity e instanciamos la imagen para darle luego la acción de volver a la vista anterior.

```
class DetalleMovieActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDetalleMovieBinding

    private lateinit var textView : TextView
    private lateinit var backButton : ImageView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDetalleMovieBinding.inflate(layoutInflater)
        setContentView(binding.root)

        textView = binding.tvPelicula
        backButton = binding.buttonBack

        val pelicula = intent.getStringExtra("Title")

        textView.text = pelicula

        backButton.setOnClickListener {
            finish()
        }
    }
}
```

Resultado.


2:06
Detalle
Marvel Studios: Assembled