

Construir aplicativo según las características de la arquitectura Android

Breve descripción:

En este componente formativo se abordan los conceptos claves para el diseño y desarrollo de aplicaciones móviles nativas, junto con sus elementos, vistas, maquetación, entornos de desarrollo, lenguajes de programación.

Julio 2024

Tabla de contenido

Introducción	4
1. Estructura de un proyecto en Android Studio	5
Manifest	9
Java.....	10
Res.....	10
Gradle	13
2. Interfaz de usuario en Android.....	14
2.1. Layouts	16
LinearLayout.....	17
TableLayout	20
ConstraintLayout	24
2.2. Controles básicos.....	25
Botones	28
Texto e imágenes	29
Checkbox y Radiobutton.....	36
Controles de selección.....	40
ListView	43
3. Eventos Listeners	46

Crear el Button en el Layout	46
Declarar el tipo de view	48
Implementar el evento Listeners	49
4. Navegación (Intents)	51
Intent explícito.....	51
Intent implícito	52
5. Tareas asincrónicas	54
6. Persistencia de datos	56
6.1. Bases de datos local SQLite.....	56
6.2. Bases de datos en tiempo real.....	57
7. Multimedia	63
7.1. MediaPlayer	63
7.2. VideoView	65
Síntesis	68
Material complementario.....	69
Glosario	70
Referencias bibliográficas	71
Créditos.....	72

Introducción

Actualmente, los sistemas operativos para dispositivos móviles permiten la construcción de aplicaciones mediante el uso de un software llamado SDK. Tanto Android como iOS ofrecen sus respectivas herramientas para crear aplicaciones nativas. Las características de los dispositivos móviles, como GPS, cámara y navegación por internet, se aprovechan en el desarrollo de aplicaciones, permitiendo ideas interesantes y útiles para los usuarios.

El sistema operativo Android es compatible con la mayoría de los dispositivos y es, hoy en día, uno de los más utilizados en teléfonos móviles. El mercado de aplicaciones para Android está en constante crecimiento, y la presencia de empresas y sus aplicaciones en esta plataforma es prácticamente un requisito indispensable. Por lo tanto, el desarrollo de aplicaciones para este sistema puede convertirse en una excelente oportunidad laboral para muchos ingenieros.

Este componente formativo se enfoca en el desarrollo de aplicaciones nativas para la plataforma Android, utilizando su entorno de desarrollo oficial, Android Studio, y el lenguaje Java.

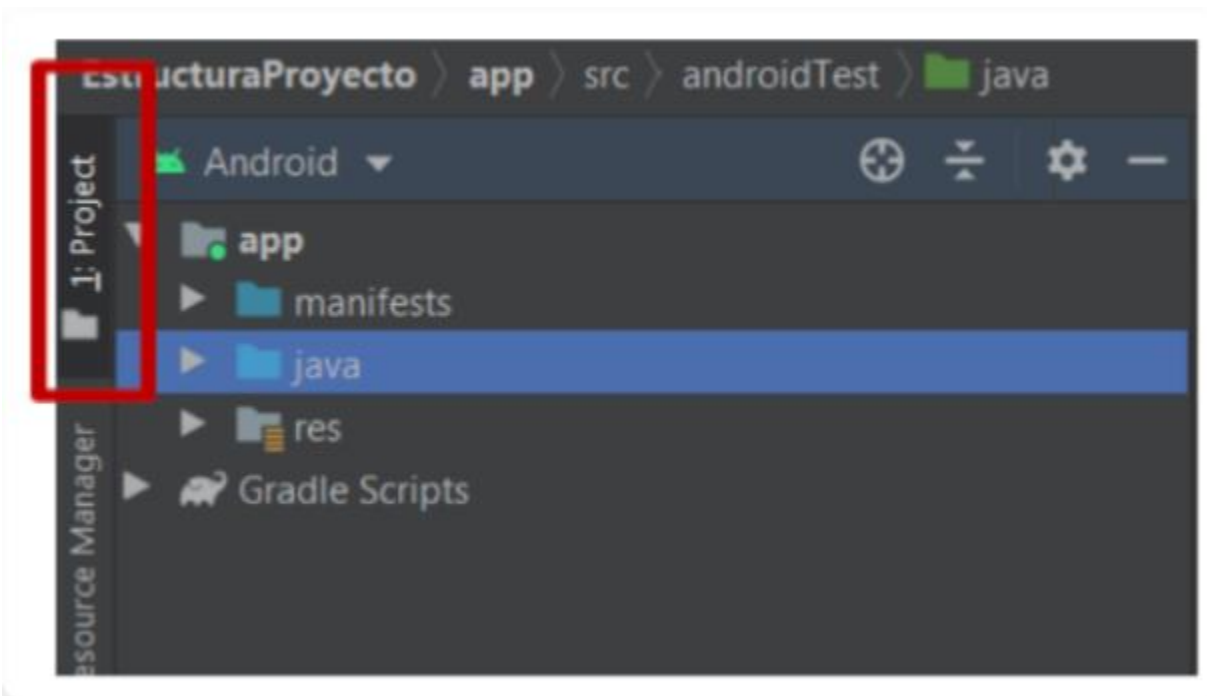
1. Estructura de un proyecto en Android Studio

Las actualizaciones en Android Studio avanzan rápidamente. Para este componente, los ejemplos se presentan para la versión 4.1.3.

En Android Studio, un proyecto contiene todos los elementos necesarios para el desarrollo de una aplicación: desde código fuente y recursos hasta código de prueba y configuraciones de compilación. Al iniciar un nuevo proyecto, Android Studio crea la estructura necesaria para todos los archivos y los organiza en la ventana Project, ubicada al lado izquierdo del IDE (accede a ella mediante View > Tool Windows > Project), o utilizando el acceso directo Alt + 1.

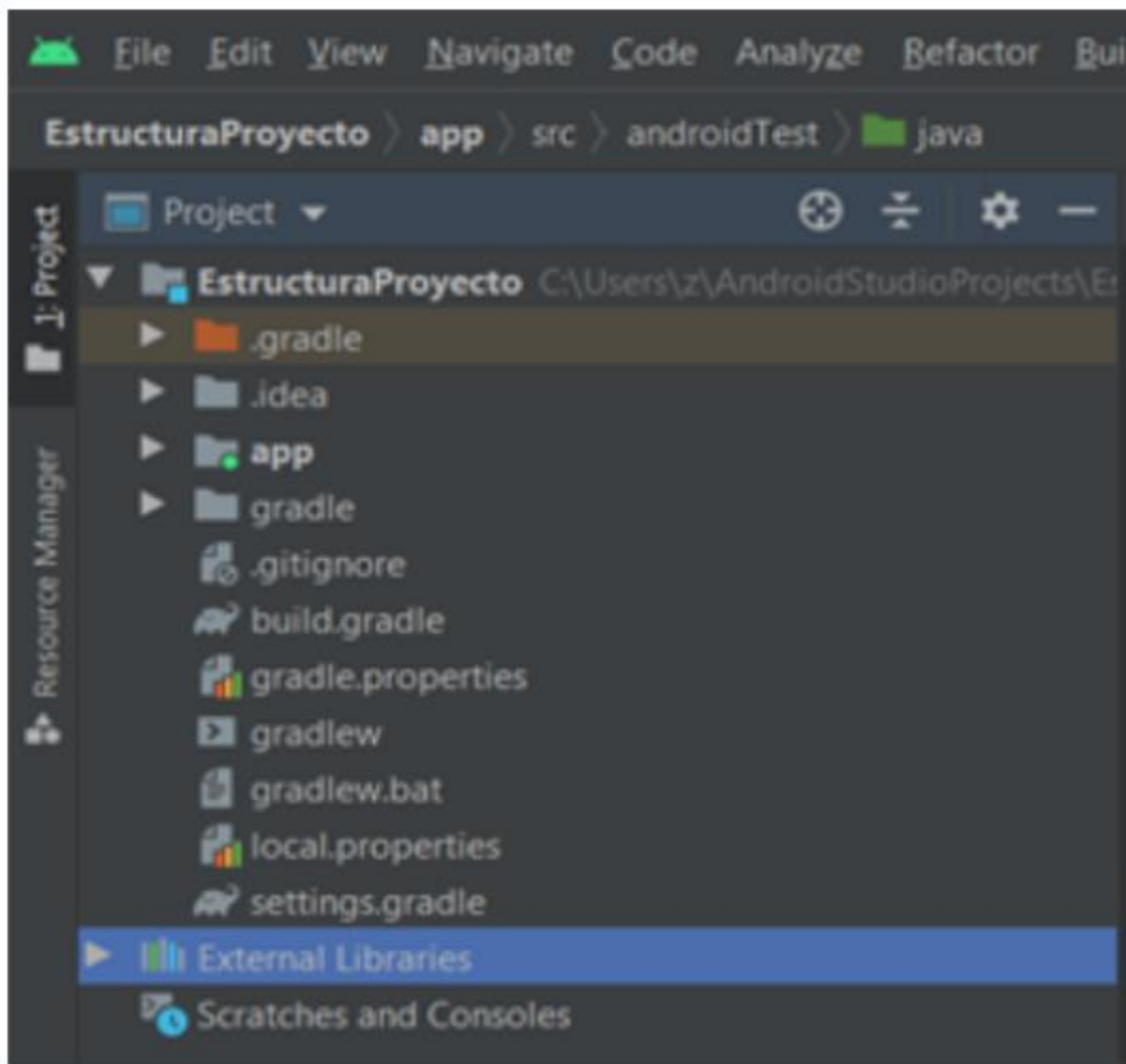
A continuación, se presenta la estructura:

Figura 1. Imagen estructura de proyecto en Android Studio



Ahora bien, las vistas de proyectos en Android, o la visualización completa de la estructura de archivos del proyecto, se realizan mediante la opción "Project" del menú desplegable en la parte superior de la ventana, como se ilustra a continuación.

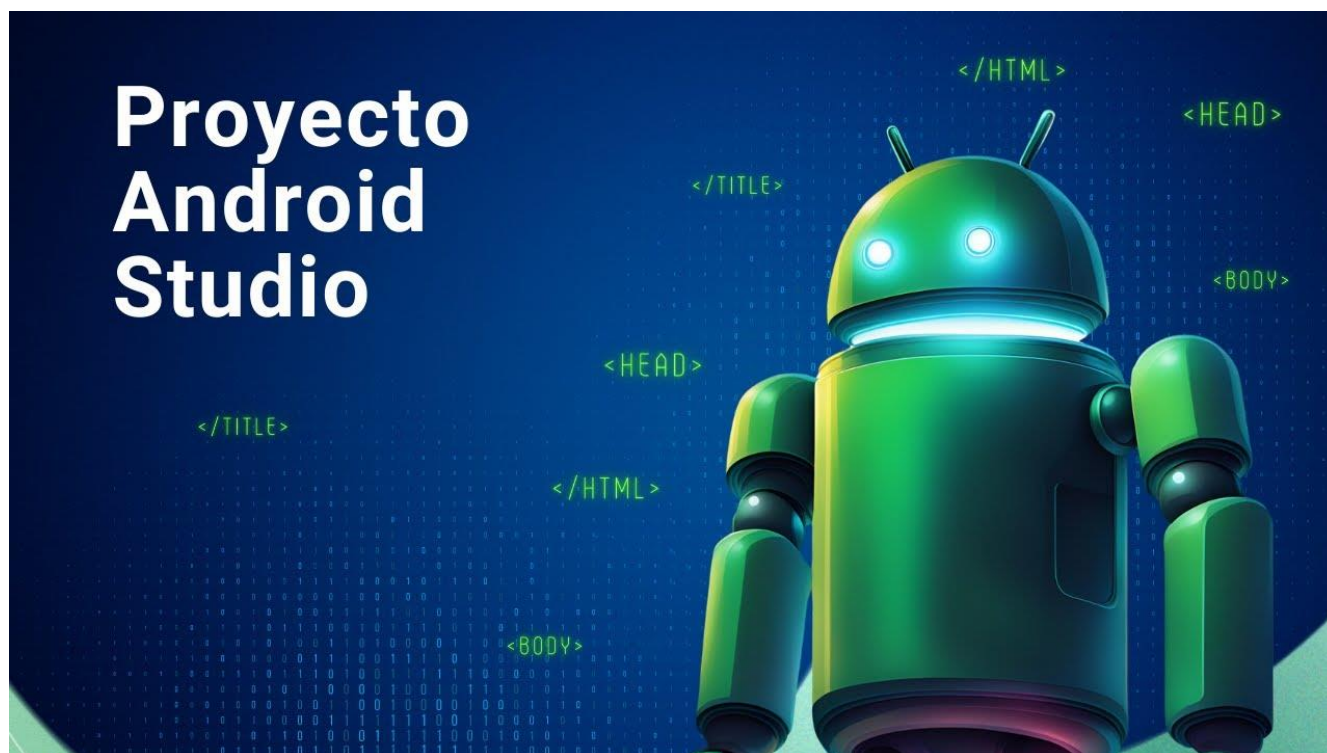
Figura 2. Vista estructura del proyecto



Los proyectos de Android Studio tienen unos componentes principales que se explican en el siguiente video, los cuales se distribuyen entre:

- Módulos
- Archivos de proyecto

Video 1. Proyecto Android Studio



[Enlace de reproducción del video](#)

Síntesis del video: Proyecto Android Studio

Estructura de un proyecto en Android Studio

Un proyecto en Android Studio está compuesto principalmente por módulos y por archivos del proyecto, en donde cada uno de estos cumple unas funciones principales, las cuales se describen brevemente a continuación.

Los Módulos: Contienen el grupo de archivos que permiten realizar la división del proyecto. Un proyecto puede contener uno o varios módulos. Estos se pueden trabajar de forma independiente para compilar, depurar o probar. El uso de los módulos se realiza principalmente para crear bibliotecas de código o conjuntos de código para diferentes dispositivos, como teléfonos o relojes inteligentes.

Para la creación de un módulo, lo hacemos a través del menú *File > New > New Module. Android ofrece diferentes tipos de módulos, como módulos de función o módulos de bibliotecas. Por ejemplo, seleccionaremos el módulo para relojes inteligentes, presionamos "Next", colocamos el nombre del módulo, presionamos "Next" nuevamente, seleccionamos el tipo de actividad y, por último, damos clic en "Finalizar". Así, se crea el módulo.

Archivos del proyecto: Dentro de cada uno de estos módulos, Android organiza los siguientes grupos principales de archivos:

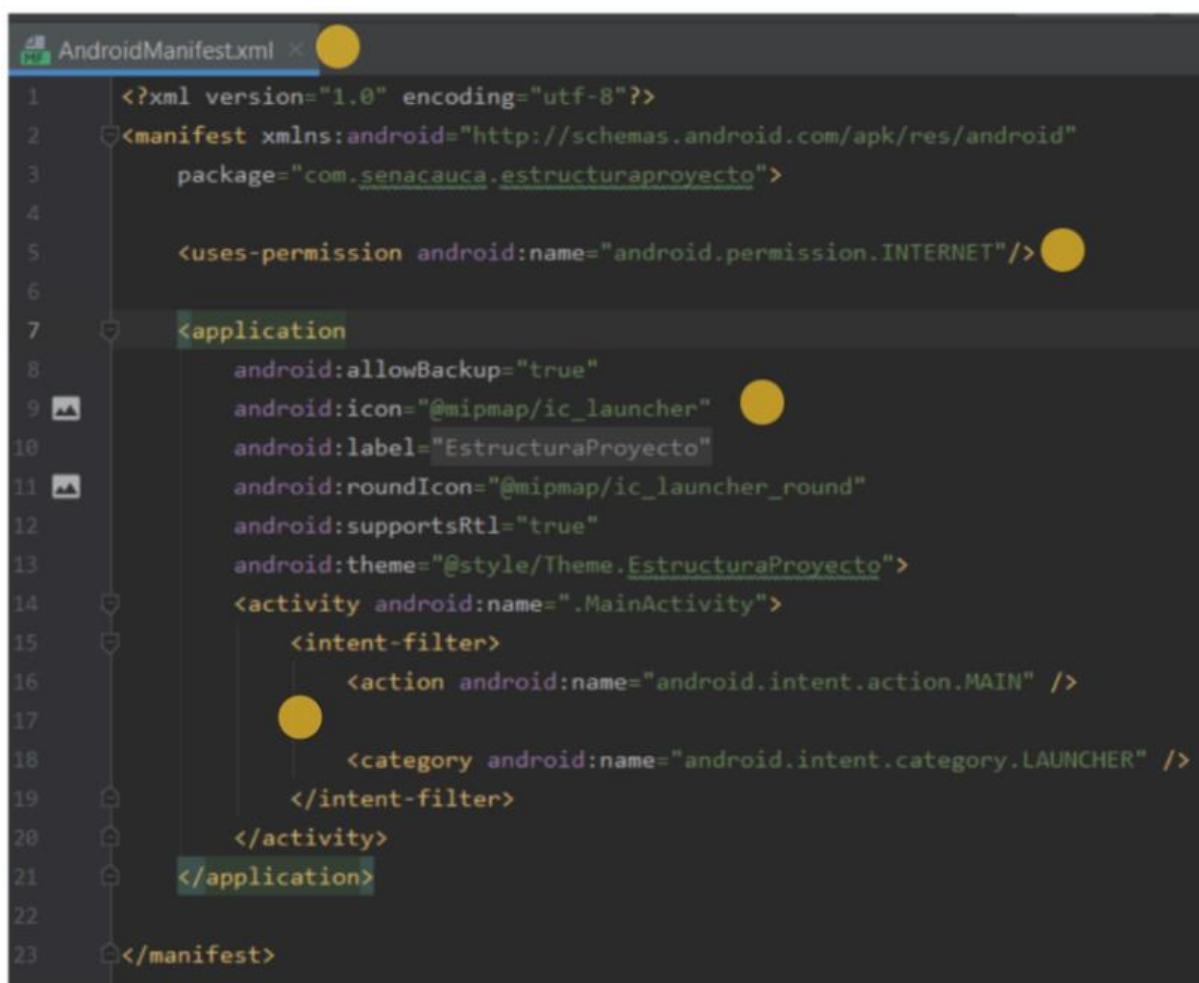
- Manifest: Contiene el archivo "AndroidManifest.xml", que se encarga de la configuración de íconos, etiquetas, temas y permisos de la aplicación.
- Java: Subdividido en directorios, contiene todos los archivos de código fuente Java.
- Res: En él se encuentran todos los recursos de la aplicación, como diseños XML, cadenas de texto, imágenes, entre otros, que se encuentran divididos en subdirectorios.

Manifest

En el manifest se encuentra la información esencial de una aplicación para las herramientas de desarrollo de Android, el sistema operativo Android y Google Play. El manifest es un archivo en lenguaje XML donde se definen aspectos principales de la aplicación, como su identificación (nombre, ícono, estilos), sus componentes (pantallas, servicios) y los permisos necesarios para su ejecución.

A continuación, se presentan algunos de los elementos más importantes que se deben declarar en el archivo manifest:

Figura 3. Archivo manifest



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.senacauca.estructuraproyecto">

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

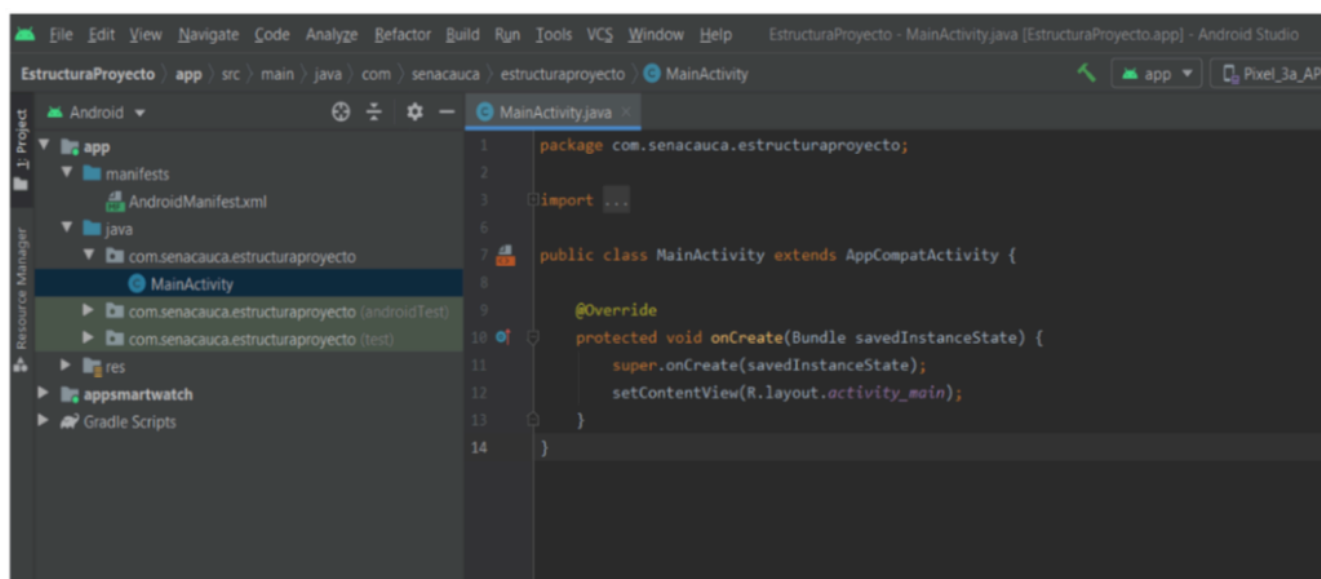
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="EstructuraProyecto"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.EstructuraProyecto">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Java

La carpeta java contiene todo el código fuente de la aplicación, organizado en paquetes, clases, interfaces, POJOs (Plain Old Java Objects), entre otros.

Inicialmente, Android Studio crea el código básico de la pantalla principal de la aplicación (actividad o activity), que por defecto se llamará MainActivity, siempre bajo la estructura del paquete java definido durante la creación del proyecto.

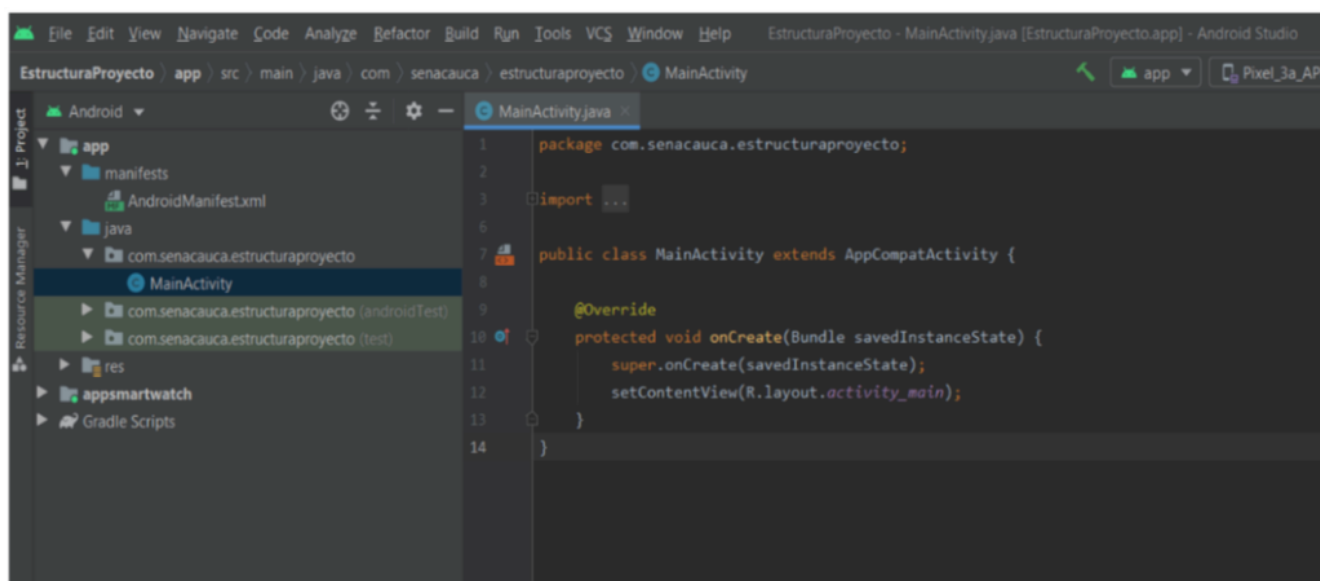
Figura 4. Clase MainActivity de la carpeta java



Res

En esta carpeta se encuentran todos los recursos del proyecto que no contienen código, como diseños XML, cadenas de texto de la interfaz de usuario y también imágenes de mapa de bits, organizados en subdirectorios.

Figura 5. Imagen estructura carpeta res



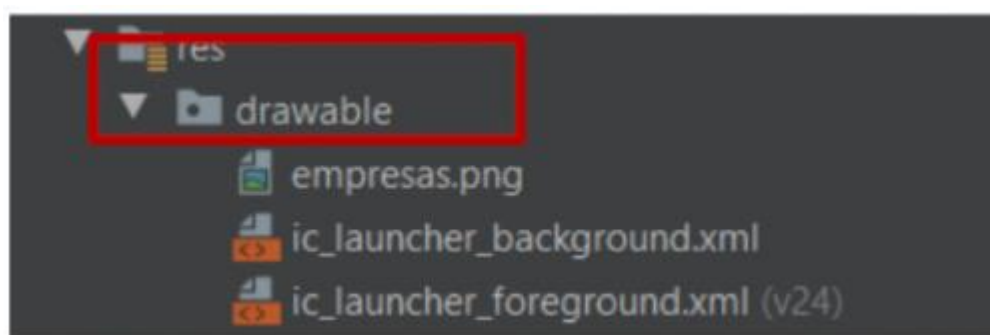
A continuación, se explican las subcarpetas:

- **Subcarpeta drawable**

/res/drawable: Contiene las imágenes que se van a usar en la aplicación.

Los formatos que soporta son (png, jpg, gif).

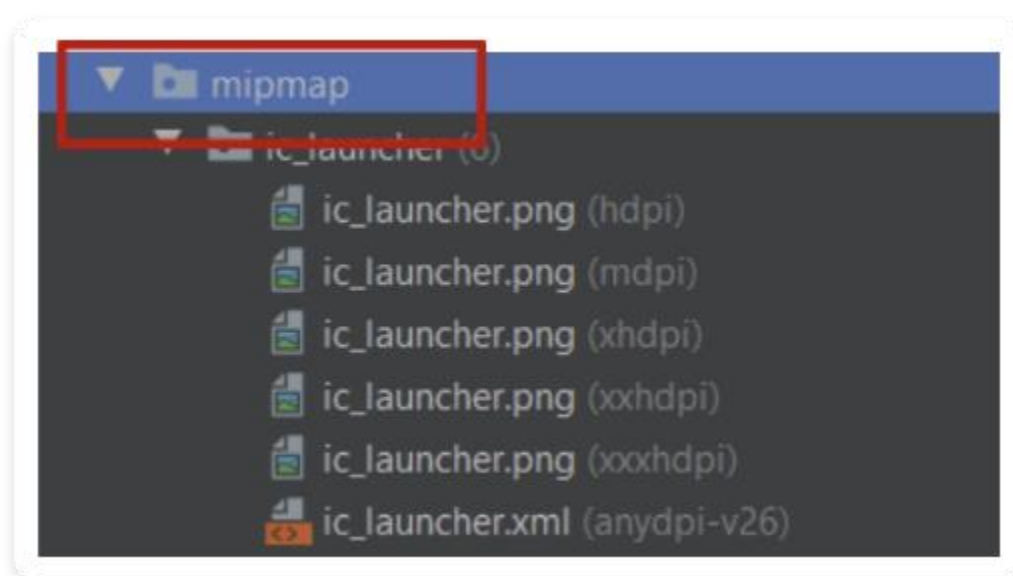
Figura 6. Explorador de archivos de Android Studio



- **Subcarpeta mipmap**

/res/mipmap: Contiene los íconos que se van a utilizar en la aplicación para las diferentes densidades de las pantallas existentes. Al igual que la carpeta drawable, se divide en subcarpetas dependiendo de la densidad de la pantalla.

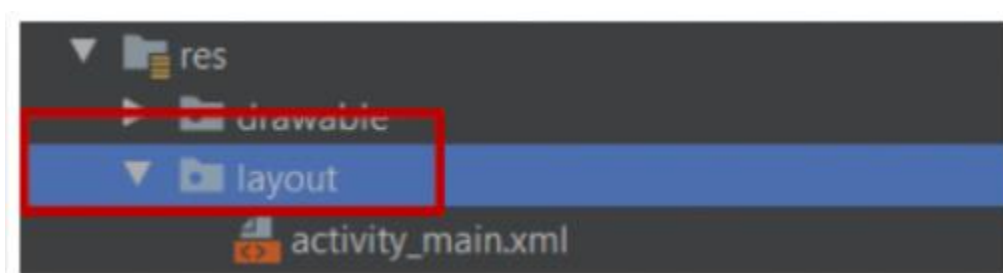
Figura 7. Subcarpeta mipmap



- **Subcarpeta layout**

/res/layout: En esta carpeta se define el diseño de la interfaz de usuario. El formato del archivo es en XML.

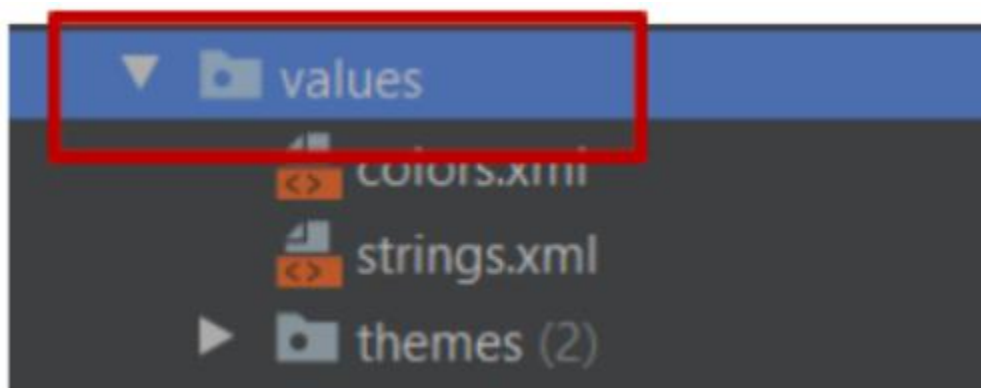
Figura 8. Subcarpeta layout



- **Subcarpeta values**

/res/values: Esta carpeta contiene archivos XML que contienen valores simples, como strings, temas y colores.

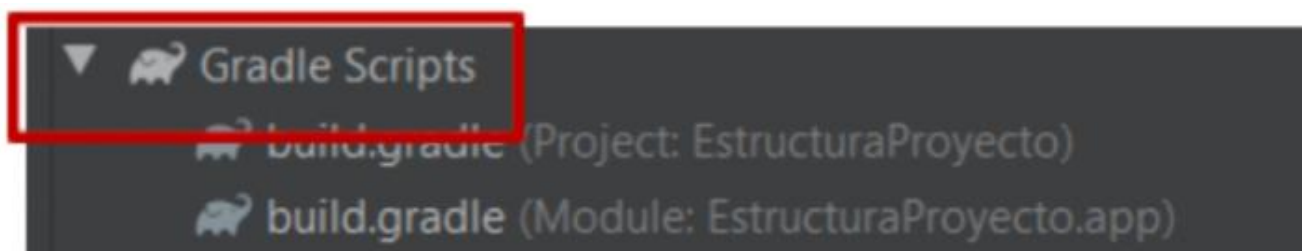
Figura 9. Subcarpeta values



Gradle

Gradle permite la compilación del proyecto. En este, es posible configurar la versión del SDK que Android utilizará para compilar, la versión mínima de Android que soportará la aplicación, referencias a las bibliotecas externas utilizadas, entre otros aspectos.

Figura 10. Gradle Scripts

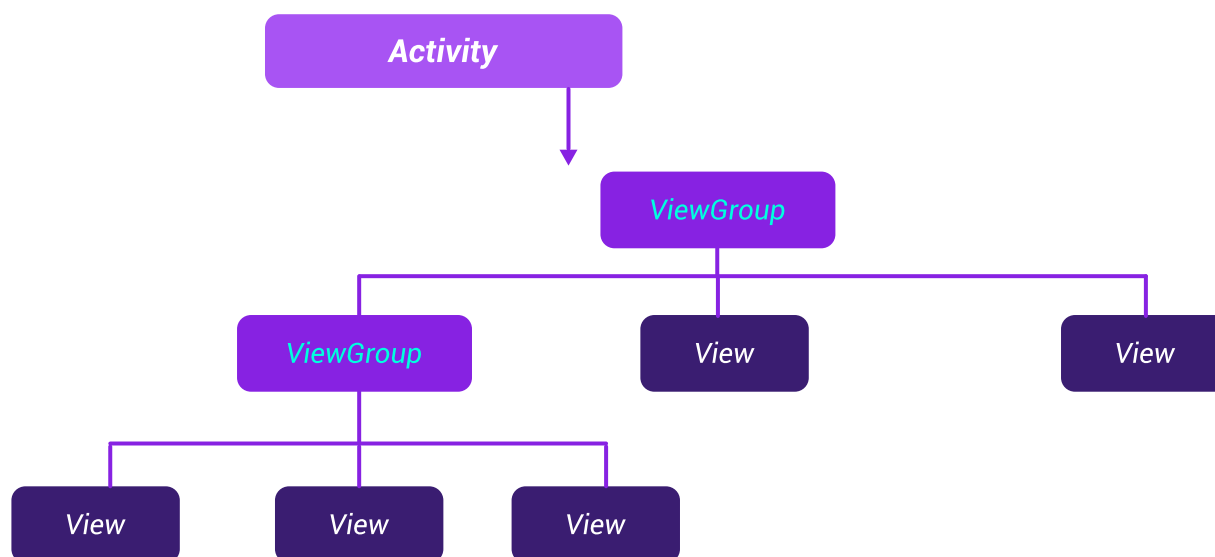


2. Interfaz de usuario en Android

La interfaz de usuario se define a través de la estructura de un diseño para la aplicación, como en el caso de una actividad en Android. La creación de los elementos del diseño se realiza utilizando una jerarquía de objetos View y ViewGroup. Los View son los elementos que el usuario puede interactuar con, como botones, textos e imágenes, mientras que un ViewGroup es un contenedor no visible que define la estructura del diseño de los View y otros objetos.

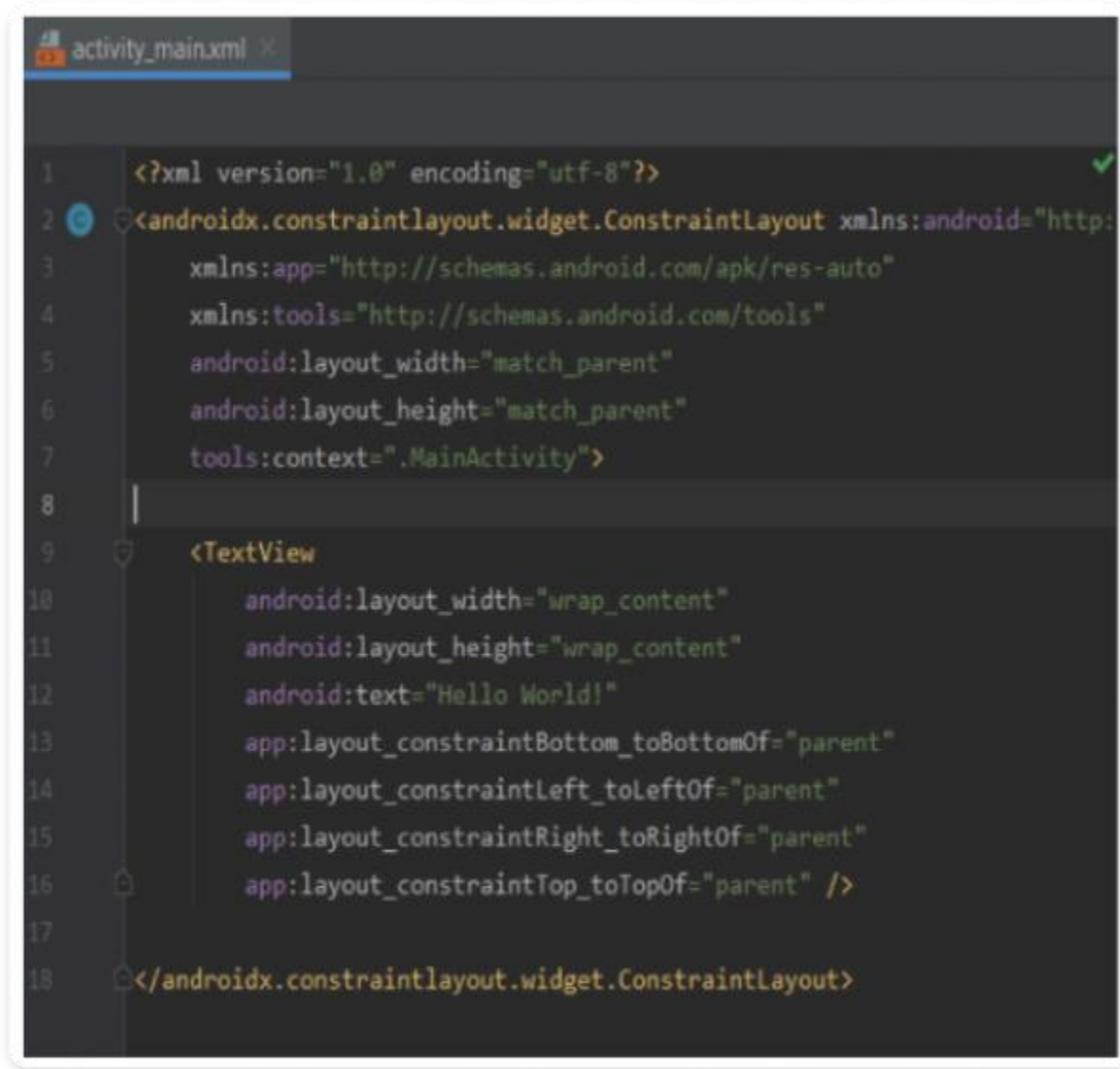
Los objetos ViewGroup se denominan Layout (diseños) y existen en varios tipos, dependiendo de la distribución y la posición de sus View. Ejemplos de estos son LinearLayout, RelativeLayout, entre otros.

Figura 11. Ilustración jerarquía de diseño



El diseño en Android se puede trabajar de dos maneras: en XML, o en el editor de diseño (mediante una interfaz de arrastrar y soltar).

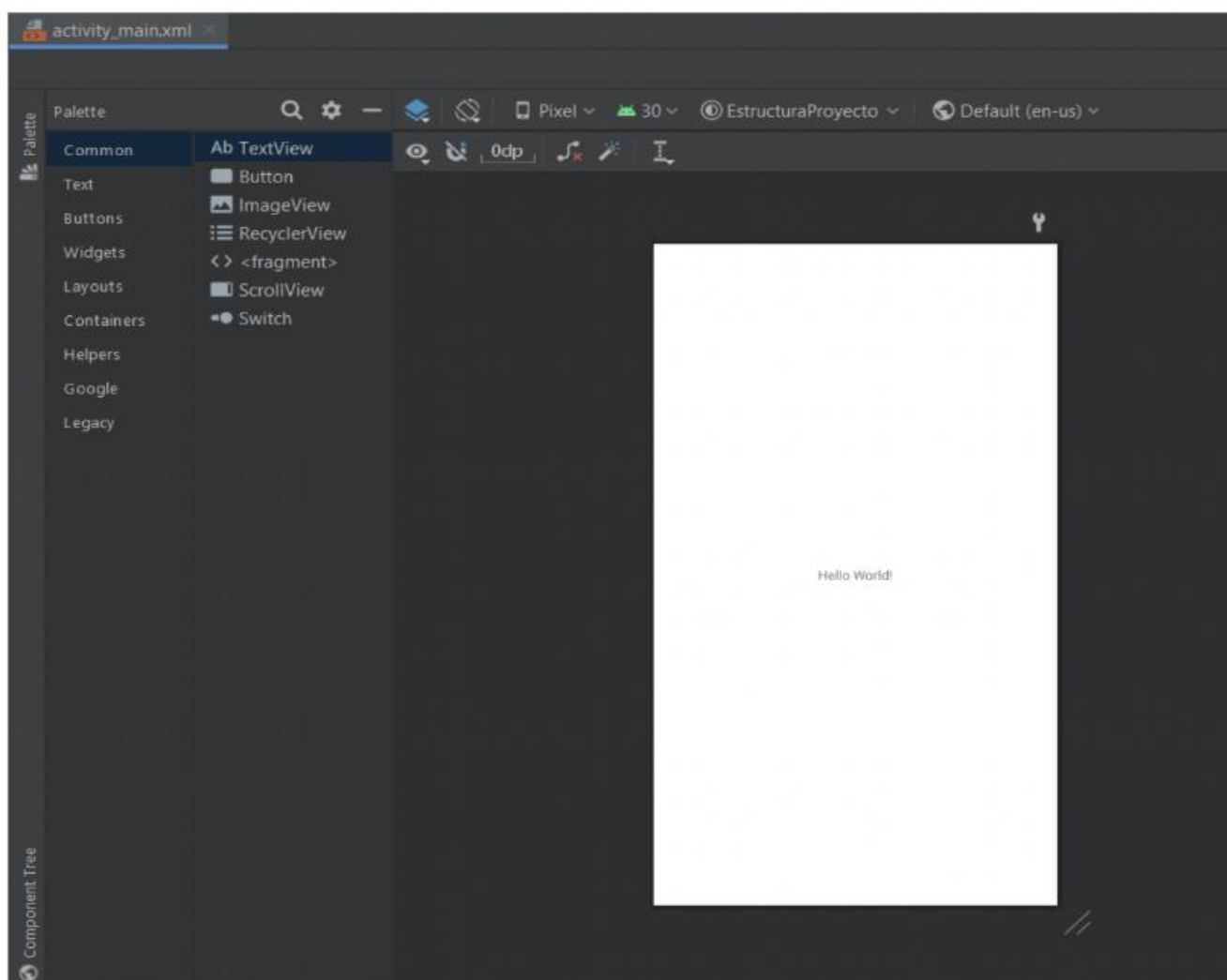
Figura 12. Vista diseño XML



The image shows a screenshot of an IDE window titled 'activity_main.xml'. The XML code is as follows:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res-auto"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
```

Figura 13. Vista diseño XML



2.1. Layouts

En Android Studio, los componentes visuales (como Button, EditText, TextView, etc.) se organizan mediante contenedores llamados Layout. Los Layouts son elementos no visuales que permiten controlar la organización, posición y dimensiones de los View que se crean dentro de ellos. Al igual que muchos otros componentes contenedores, son capaces de contener otros controles, es decir, se puede tener un contenedor dentro de otro contenedor.

Existen diferentes tipos de Layout, entre los más comunes están: LinearLayout, TableLayout y ConstraintLayout. A continuación, se profundiza en cada uno:

LinearLayout

Un LinearLayout dispone las vistas de la interfaz, una debajo de otra (orientación vertical) o una al lado de la otra (orientación horizontal). Sus propiedades principales son:

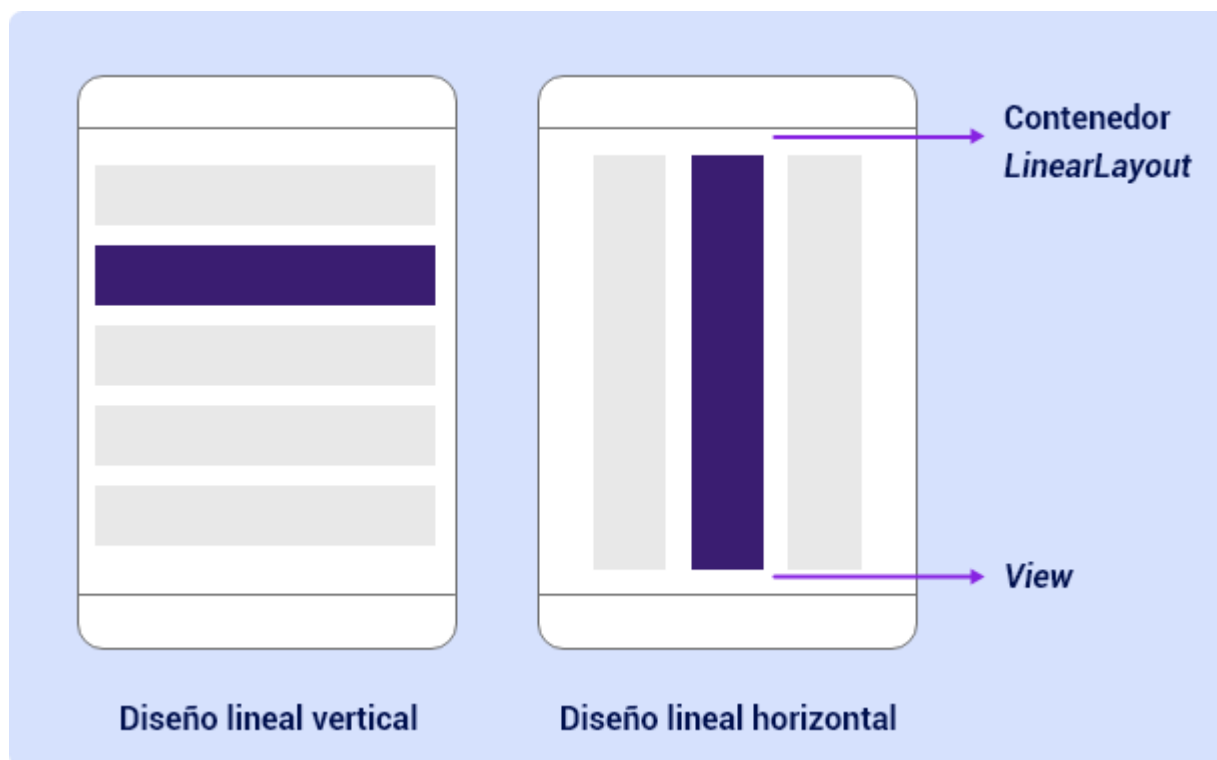
- a) Orientation: es la propiedad que define la disposición de sus vistas, ya sea horizontal o vertical.
- b) Layout_height: esta propiedad especifica el alto del ViewGroup (contenedor).
- c) Layout_width: esta propiedad especifica el ancho del ViewGroup.

Los valores que pueden tomar las propiedades height y width son:

- match_parent: para que ocupe el mismo tamaño de la pantalla.
- wrap_content: para que se adapte al tamaño de su contenido.
- Valores personalizados que se miden con dp (píxeles de densidad independiente).

Estas propiedades de height y width se aplican a todos los contenedores y vistas.

Figura 14. Diseño lineal vertical y diseño lineal horizontal en Android



```
<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:orientation="vertical"
```

```
android:layout_width="match_parent"
```

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

```
tools:context=".MainActivity">
```

```
<!-- Dentro del contenedor se agregan los view en este caso p Button -->
```

```
<Button  
  
    android:layout_width="match_parent"  
  
    android:layout_height="wrap_content"  
  
    android:layout_marginTop="45dp"  
  
    android:text="@string/boton1" />  
  
<Button  
  
    android:layout_width="match_parent"  
  
    android:layout_height="wrap_content"  
  
    android:text="@string/boton2" />  
  
</LinearLayout>  
  
<!-- Cierre del contenedor LinearLayout -->
```

El código XML para la creación de un layout de tipo `LinearLayout`: con orientación vertical, el cual contiene dos vistas de tipo `Button`.

- `<LinearLayout`: tipo de contenedor lineal.
- `orientation=vertical`: propiedad que define la ubicación de los views de forma vertical.
- `layout_width`: propiedad que define el ancho del contenedor.
- `layout_height`: propiedad que define el alto del contenedor >

`<Button`: etiqueta para crear un view de tipo `Button` (Botón).

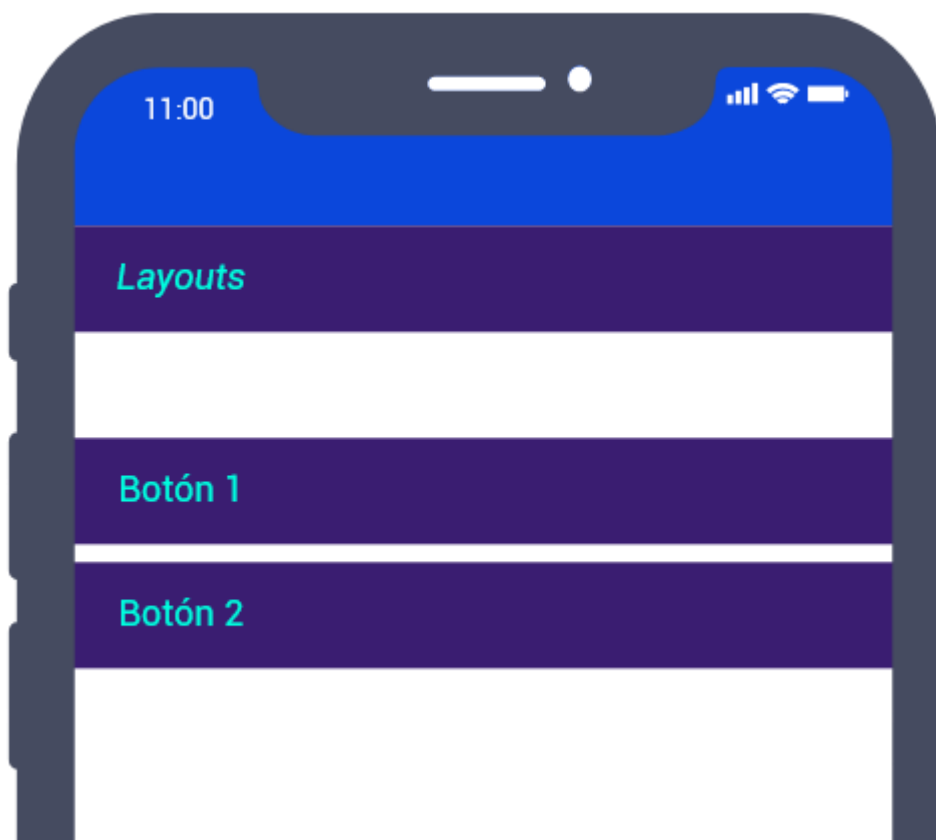
`layout_width`: propiedad que define el ancho del view.

`layout_height`: propiedad que define el alto del view.

`text`: propiedad que define el título del button />

El código anterior mostraría el LinearLayout de la siguiente forma:

Figura 15. LinearLayout



TableLayout

Es un contenedor ViewGroup que permite mostrar los elementos View en columnas y filas. Para crear las filas se utiliza el objeto TableRow. Cada fila puede tener una o más celdas, o ninguna, y cada celda se crea mediante un tipo de objeto View. Es

decir, las celdas de una fila pueden formarse por una diversidad de objetos View, como ImageView o TextView.

El siguiente código XML de diseño muestra dos filas con tres celdas en cada una, y una tercera fila con dos celdas:

```
<!-- Crea la tabla -->

<TableLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="match_parent"

android:layout_height="match_parent"

android:stretchColumns="0,1,2">

<!-- Crea la fila -->

<TableRow>

<TextView android:text="Celda 1.1" /> <!-- Crea la Columna -->

<TextView android:text="Celda 1.2" /> <!-- Crea la Columna -->

<TextView android:text="Celda 1.3" /> <!-- Crea la Columna -->

</TableRow>

<!-- Crea la fila -->

<TableRow>

<TextView android:text="Celda 2.1" /> <!-- Crea la Columna -->
```

```
<TextView android:text="Celda 2.2" /> <!-- Crea la Columna -->
```

```
<TextView android:text="Celda 2.3" /> <!-- Crea la Columna -->
```

```
</TableRow>
```

```
<!-- Crea la fila -->
```

```
<TableRow>
```

```
<TextView android:text="Celda doble de ancho 3.1"
```

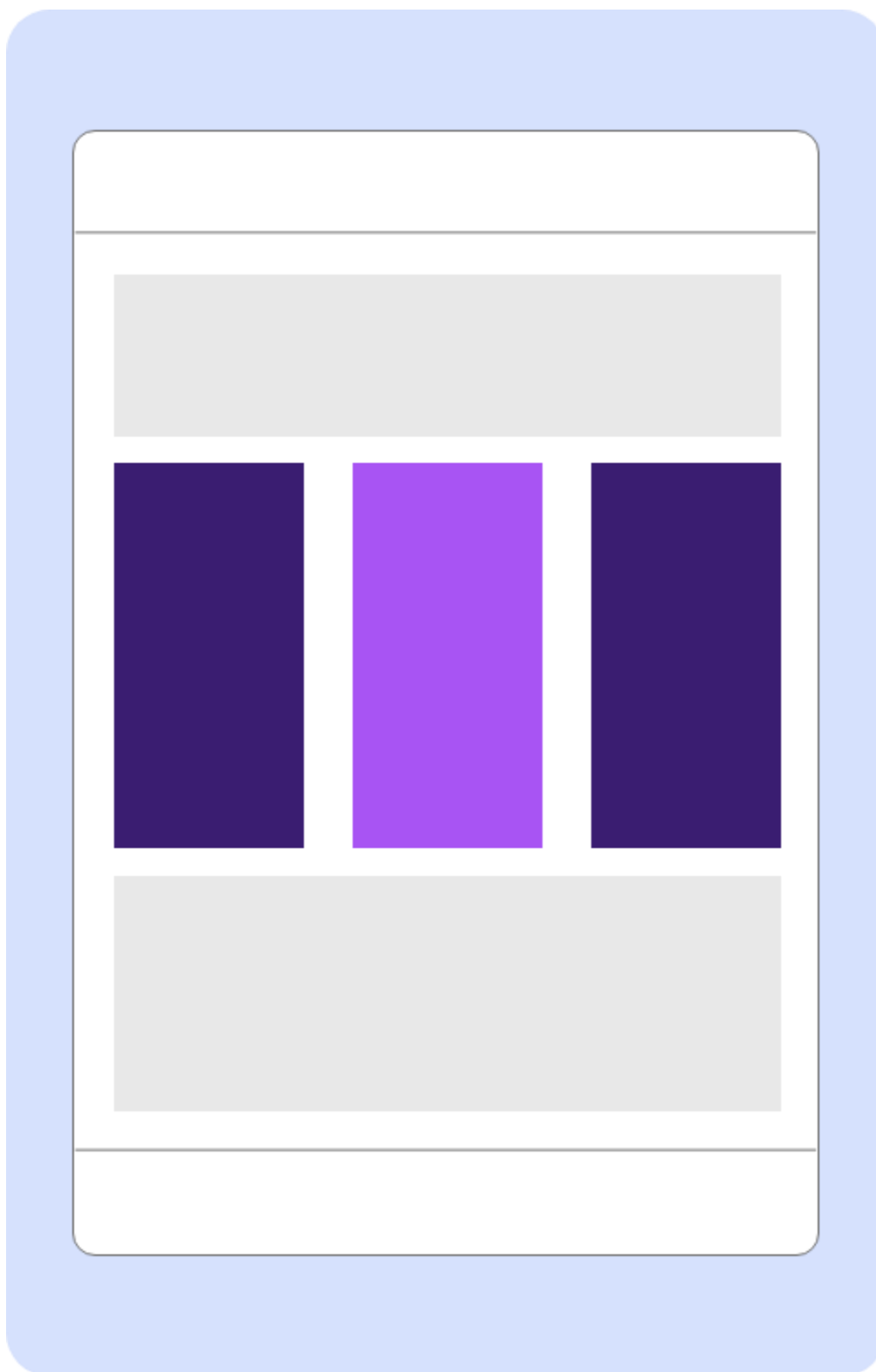
```
android:layout_span="2" />
```

```
<TextView android:text="Celda 3.2" />
```

```
</TableRow>
```

```
</TableLayout>
```

Figura 16. Diseño lineal horizontal



El código anterior mostraría el TableLayout de la siguiente forma:

Figura 17. TableLayout



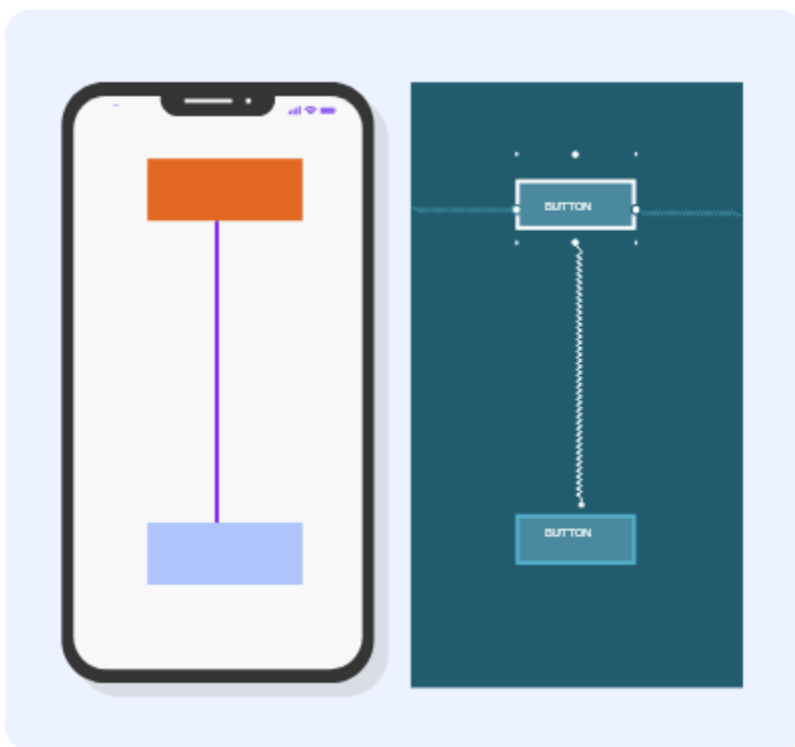
ConstraintLayout

ConstraintLayout es el contenedor más utilizado actualmente y es el que se configura por defecto al crear una actividad, ya que permite diseñar los view directamente desde las herramientas visuales del editor de diseño. Con este contenedor se puede crear todo el diseño arrastrando y soltando elementos, en lugar de editar el XML manualmente.

Cada posición de un view en un ConstraintLayout se define agregando al menos una restricción horizontal y una vertical. Cada restricción representa una conexión o alineación con otra vista. Estas restricciones determinan la posición de la vista a lo largo del eje vertical u horizontal, por lo que cada vista debe tener al menos una restricción para cada eje. Sin embargo, normalmente se requieren más restricciones para lograr un diseño preciso.

Vista de Diseño ConstraintLayout:

Figura 18. Vista de Diseño ConstraintLayout



2.2. Controles básicos

En Android Studio se encuentra una amplia variedad de controles que se pueden utilizar en la interfaz de usuario, tales como botones (Button), campos de texto (EditText), vista de imágenes (ImageView), casillas de verificación (CheckBox), botones de selección (RadioButton), etiquetas de texto (TextView), listas (ListView), entre otros.

Existen tres propiedades indispensables en la creación de los controles mencionados anteriormente:

a) Android: layout_width

Esta propiedad es común a todas las vistas y permite establecer el ancho que se utilizará. Este valor puede ser fijado en relación con el contenido que se visualizará o con la vista que lo contiene.

b) Android: layout_height

Esta propiedad es común a todas las vistas y permite establecer el ancho que se utilizará. Este valor puede ser fijado en relación con el contenido que se visualizará o con la vista que lo contiene.

c) Android: id

Define el identificador asociado a la vista para ser referenciado por código Java. La forma de crear un identificador es la siguiente:

- `android:id="@+id/myText"`
- El símbolo `@+id/` indica que se está generando un nuevo identificador, lo que provocará que la clase `R` del paquete `gen` del proyecto cree una referencia a este objeto, para que luego pueda ser manipulado por código.

Los valores que pueden tomar las propiedades de Android: `layout_width`, y Android: `layout_height` los cuales aplican para todos los controles son:

- a) **MATCH_PARENT:** Esto significa que la vista tendrá el tamaño de su contenedor (en este caso, la pantalla).
- b) **WRAP_CONTENT:** Esto significa que el tamaño de la vista se adaptará al contenido que tenga.

c) **PERSONALIZADO:** También se puede definir el alto y el ancho con valores personalizados usando la unidad de medida dp (píxeles independientes de densidad). Ejemplo:

- `android:layout_width="50dp"`
- `android:layout_height="200dp"`

Estas tres propiedades básicas son las principales en los componentes que se describen a continuación.

Figura 19. Controles básicos



Botones

Un botón (Button) es un elemento de la interfaz en el que el usuario puede tocar o hacer clic para realizar una acción. A continuación, se presenta un ejemplo de un Button con sus propiedades básicas:

Tabla 1. Propiedades básicas Button

Propiedades	Definición
android:layout_width	Define el ancho de la vista.
android:layout_height	Define el alto de la vista.
android:id	Define el identificador asociado a la vista para ser referenciado por código java.
android:text	Define el texto asociado a la vista.

XML Creación Button:

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

Y el botón quedaría como se presenta a continuación:

Figura 20. Vista diseño Button



Texto e imágenes

Uno de los controles más utilizados son los referentes a los textos e imágenes, ya que son con los que más interactúa el usuario. A continuación, se describen tres de los componentes básicos más imprescindibles para el desarrollo de nuestras aplicaciones: los textos o etiquetas (TextView), las imágenes (ImageView) y los cuadros de texto (EditText).

TextView: es un control utilizado para desplegar contenido textual generalmente estático. Tiene la propiedad de acoplarse al control que lo contenga y no captura el foco en ningún momento, por lo que generalmente no se programa ningún evento sobre él. Sus propiedades básicas son las siguientes:

Tabla 2. Propiedades básicas TextView

Propiedades	Definición
android:layout_width	Define el ancho de la vista.
android:layout_height	Define el alto de la vista.
android:id	<p>Establece cuál es el contenido para mostrar por el TextView.</p> <p>Existen dos formas de asociar el texto a una vista, la primera forma y más simple es asociar el texto de forma directa:</p> <p>android:text="Hola Mundo"</p> <p>La segunda forma es por medio de archivo string XML ubicado dentro de la carpeta values de los recursos del proyecto y asociar cada uno de los textos con una llave respectiva.</p> <p>Ya luego se referencia el contenido por medio de este archivo XML de la siguiente forma:</p> <p>android:text="@string/boton2"</p> <p>El color de tinte de la imagen.</p>
android: textColor	Esta propiedad nos permite definir cuál va a ser el color del texto.
android: textSize	<p>Con esta propiedad podemos definir el tamaño de la fuente la unidad es sp.</p> <p>Ejemplo: android:textSize="20sp"</p>
android: textStyle	<p>Define el estilo de la fuente a usar, tiene los siguientes valores:</p> <p>android:textStyle="bold"</p> <p>android:textStyle="italic"</p> <p>android:textStyle="normal"</p>
android: typeface	Esta propiedad me permite cambiar el tipo de fuente dentro de un conjunto de 4 posibles

Propiedades	Definición
	opciones: normal, sans, serif y monospace, se pueden descargar otro tipo de fuentes.

Ejemplo de TextView con sus propiedades básicas:

```
<TextView
    android:id="@+id/txtTexto"
    android:text="@string/control_de_texto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textStyle="bold"
    android:textColor="@color/purple_500"
    android:typeface="monospace"
    android:textSize="40sp" />
```

El TextView quedaría como se presenta a continuación:

Figura 21. Vista diseño TextView



ImageView: es un control de imágenes que permite acceder a los recursos de imágenes (drawable o mipmap) y publicar en la aplicación. Su propiedad principal es `android:src`, que permite indicar la imagen a exponer del proyecto. Lo primero que se debe hacer es copiar la imagen que se quiere presentar en la carpeta drawable del proyecto (`/res/drawable`) y, después, en el XML se debe indicar la ruta de origen así: `android:src="@drawable/imagen"`.

Aparte de las propiedades de `width` y `height` que se encuentran en todos los view, están las siguientes propiedades básicas para el ImageView:

Tabla 3. Propiedades básicas ImageView

Propiedades	Definición
<code>android:src</code>	Establece la imagen del ImageView.

Propiedades	Definición
android:scaleType	Controla cómo se debe cambiar el tamaño o mover la imagen para que coincida con el tamaño de este ImageView.
android:tint	El color de tinte de la imagen.

Ejemplo de ImageView con sus propiedades básicas:

```
<ImageView
    android:id="@+id/txtTexto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/andy"
    android:scaleType="centerInside" />
```

El ImageView quedaría como se presenta a continuación:

Figura 22. Vista diseño ImageView



EditText: es un elemento de la interfaz que permite al usuario ingresar y editar texto. Una de sus propiedades más funcionales es `inputType`, que define el tipo de entrada que tendrá el cuadro de texto. El valor se aplica dependiendo de lo que se quiera ingresar: por ejemplo, un correo electrónico (`textEmailAddress`), edad (`number`), apellido (`text`), entre otros. La elección del tipo de entrada configura el tipo de teclado que se expondrá.

Tabla 4. Propiedades básicas de EditText

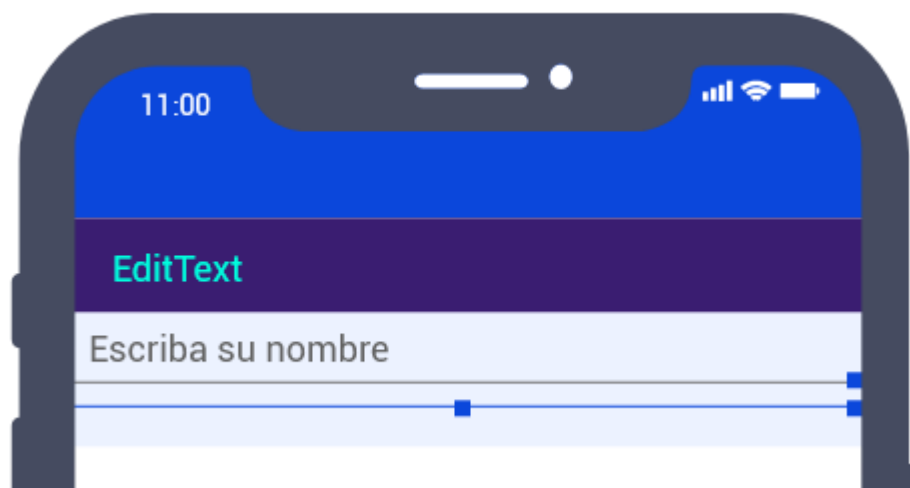
Propiedades	Definición
<code>android:inputType</code>	Establece el tipo de dato que se ingresará en el campo de texto. Para esta propiedad existen valores como: <code>text</code> , <code>number</code> , <code>textUri</code> , <code>textPassword</code> , <code>phone</code> , <code>date</code> , entre otros.
<code>android:hint</code>	Permite desplegar un texto sobre el control a manera de ayuda mientras el usuario aún no ingresa información. Se prefiere sobre la propiedad <code>text</code> .
<code>android:text</code>	Al igual que en <code>TextView</code> , se puede definir cuál es el texto que se mostrará en el control.

Ejemplo de EditText con sus propiedades básicas:

```
<EditText
    android:id="@+id/txtTexto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Escriba su Nombre"
    android:inputType="text" />
```

El EditText quedaría como se presenta a continuación:

Figura 23. Vista diseño EditText



Checkbox y Radiobutton

Otro de los controles más utilizados en las aplicaciones son los que permiten hacer una selección o un chequeo de alguna opción. Android cuenta con dos tipos: checkboxes y radiobuttons.

CheckBox: Las casillas de verificación permiten que el usuario seleccione una o más opciones de un grupo. Aparte de las propiedades de width y height que se encuentran en todas las vistas (view), las siguientes son básicas para el Checkbox:

Tabla 5. Propiedades básicas Checkbox

Propiedades	Definición
android:checked	Establece el estado actual del Checkbox.

Propiedades	Definición
android:text	Define el texto del Checkbox.
android:id	Define el identificador único asociado a la vista para ser referenciado por código Java.

Para definir un control de este tipo, se utiliza el siguiente código, que define dos checkboxes con los textos "español" e "inglés":

```

<CheckBox

android:id="@+id/txtTexto"

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:text="@string/espa_ol"

android:checked="true" />

<CheckBox

android:id="@+id/txtTexto2"

android:layout_width="match_parent"

android:layout_height="wrap_content"

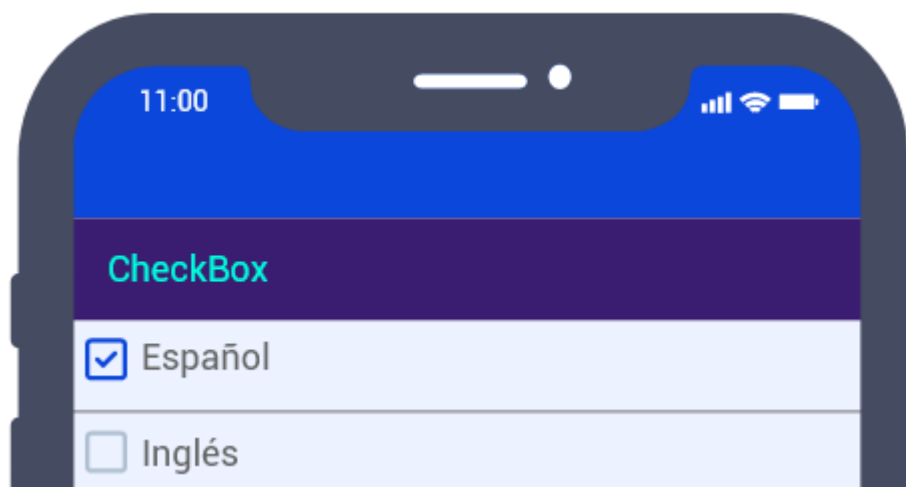
android:text="@string/ingles"

android:checked="false" />

```

Los checkbox quedarían como se presenta a continuación:

Figura 24. Vista diseño Checkbox



RadioButton: a diferencia del CheckBox, el control RadioButton permite realizar una selección única entre una lista de opciones. En Android, un grupo de botones RadioButton se define mediante un elemento RadioGroup, el cual contiene todos los elementos RadioButton necesarios.

Tabla 6. Propiedades básicas de RadioButton

Propiedades	Definición
android:checked	Establece el estado actual de selección por defecto del RadioButton.
android:text	Define el texto que se mostrará en el RadioButton.
android:id	Define el identificador único asociado a la vista para ser referenciado por código Java, al igual que en los otros controles.

Para definir un control de este tipo, se utiliza como componente padre un RadioGroup y como hijos dos RadioButton. El RadioGroup es el que permite que solo

sea seleccionable una sola opción; si se crean RadioButton sin un RadioGroup, todas las opciones serán seleccionables, lo que no es el propósito de este componente.

En las propiedades del RadioGroup se puede definir la orientación de los RadioButton, que puede ser vertical u horizontal.

El siguiente código presenta la creación de dos RadioButton con los textos de "femenino" y "masculino", los cuales están dentro de un RadioGroup con la propiedad de orientación vertical.

```
<RadioGroup
    android:id="@+id/rg1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <RadioButton
        android:id="@+id/rb1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/femenino"
        android:checked="true" />

    <RadioButton
        android:id="@+id/rb2"
```

```

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:text="@string/masculino"

android:checked="false" />

</RadioGroup>

```

Los RadioButton quedarían como se presenta a continuación:

Figura 25. Vista diseño RadioButton



Controles de selección

Los diferentes controles que Android ofrece para seleccionar opciones dentro de una lista de posibilidades se llaman Spinner (para listas desplegadas) y ListView (para listas fijas).

Spinner

Este permite seleccionar un valor de un conjunto. Al hacer clic sobre él, se presenta una especie de lista emergente con todas las opciones disponibles. Su propiedad más importante es `android:entries="@array/lista"`, que permite fijar el contenido de datos que se quiere mostrar en la lista.

Para añadir una lista de este tipo, se deben seguir los pasos que se mencionan a continuación:

- a) En los recursos `/res/values/strings.xml`, se debe crear un arreglo de strings (string-array) con los valores que se quieren presentar en el spinner.

```
<resources>

<string name="app_name">RadioButton</string>

<string-array name="ciudades">

<!-- Se crea el Arreglo, en este caso se coloca el nombre de ciudades -->

<item>Popayán</item>

<item>Cali</item>

<item>Bogotá</item>

<item>Cartagena</item>

<item>Medellín</item>

<item>Pasto</item>

</string-array> <!-- Cierre del arreglo -->
```

`</resources>`

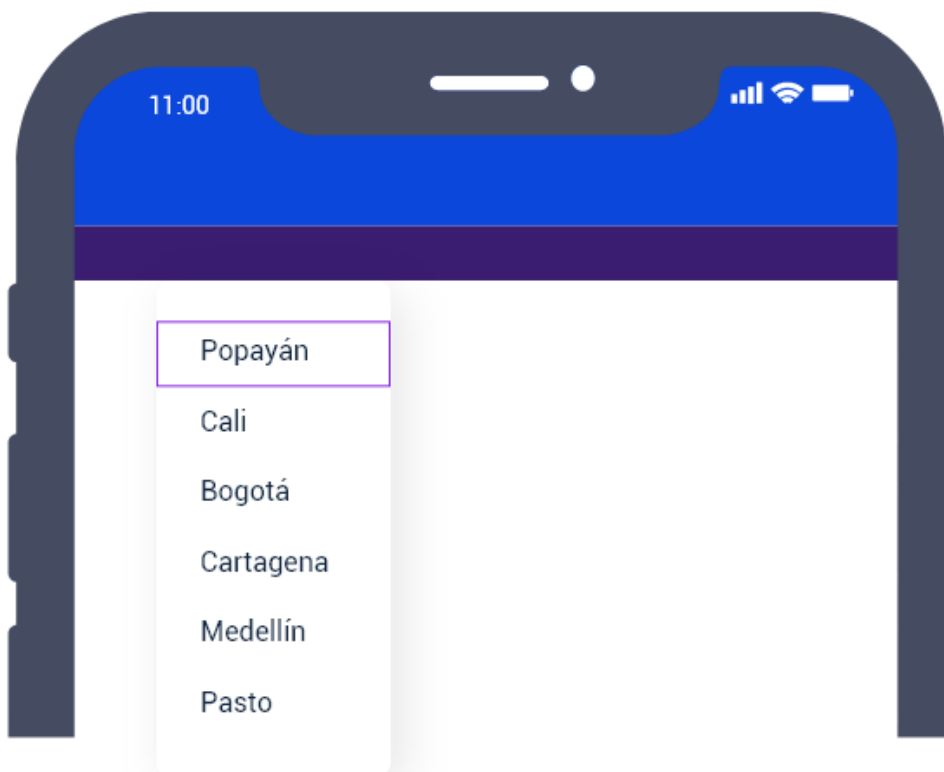
- b) Ahora en el `/res/layout` se crea el siguiente código para crear el Spinner y enlazar el string-array a través de la propiedad `entries` como se presenta:

```
<Spinner  
    android:id="@+id/spCiudades"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:entries="@array/ciudades" />
```

El código enseña la creación de un Spinner con sus propiedades básicas `height`, `width` y `id`, y la propiedad `entries`, donde se relaciona el array "ciudades" creado en los recursos.

El Spinner se presentaría de la siguiente manera:

Figura 26. Vista Diseño Spinner ciudades



ListView

El ListView permite mostrar al usuario una lista de opciones seleccionables directamente sobre el propio control, sin listas emergentes, como en el caso del control Spinner. Si existen más opciones de las que se pueden mostrar en el control, se podrá hacer scroll sobre la lista para acceder al resto de los elementos.

Para crear el ListView, al igual que en el Spinner, también se debe crear el arreglo de elementos que se quiere mostrar en la lista. Para este ejemplo, se utilizará el mismo arreglo de ciudades que se usó para el Spinner.

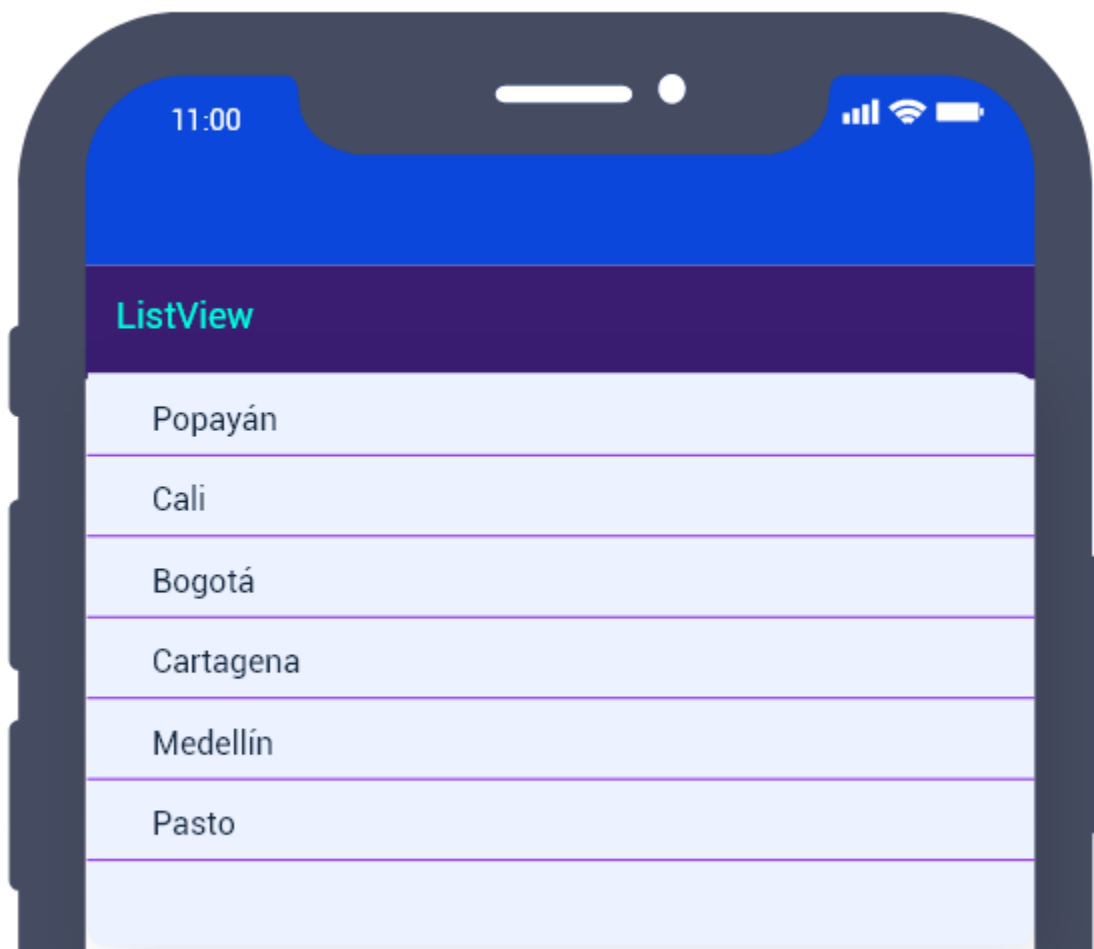
Cuando los elementos que se quieren mostrar en la lista o en el Spinner no son estáticos, se crea el arreglo desde el código Java.

Ejemplo de creación de un ListView:

```
<ListView  
  
    android:layout_gravity="center"  
  
    android:id="@+id/listaCiudades"  
  
    android:layout_width="wrap_content"  
  
    android:layout_height="wrap_content"  
  
    android:entries="@array/ciudades" />
```

El código describe la creación de un ListView, utilizando las mismas propiedades que el Spinner, y la propiedad entries, donde se relaciona el array "ciudades" creado en los recursos.

Figura 27. Lista diseño ListView ciudades



3. Eventos Listeners

En Android, la interacción del usuario con la aplicación se gestiona a través de eventos. Por ejemplo, cuando se toca una vista (como un botón), se llama al método `onTouchEvent()` del objeto, utilizando los objetos de escucha de eventos para detectar la interacción del usuario.

Los Event Listeners (objetos de escucha de eventos) son interfaces de la clase `View` que contienen un método de llamada, el cual será invocado cuando la vista registre una interacción del usuario con el elemento de la interfaz gráfica. Esto significa que los `View`, como `Button`, `ImageView`, `RadioButton` y `Checkbox`, al interactuar con el usuario, el objeto escuchará y llamará al método correspondiente, ejecutando una acción determinada.

La ejecución de los eventos se realiza en el código Java ubicado en la carpeta `/java`.

Para fijar un `Listeners`, se debe implementar y agregar los métodos del evento correspondiente a cada `View` mediante el método `setOn`. Para realizar el ejemplo del evento `Listeners` de un `Button`, se deben seguir algunos pasos:

Crear el `Button` en el `Layout`

Crear el `Button` en el `Layout` con su propiedad `ID`, ya que es la que nos va a permitir referenciarlo desde el código Java. Se muestra la creación de un botón en el `layout` llamado `activity_main.xml`. El código XML tiene un contenedor de tipo `LinearLayout` con su propiedad vertical y el `Button` es creado con sus propiedades básicas. La propiedad que se necesita para referenciar es la del `id`. El nombre del identificador que se colocó para el ejemplo fue `btn clic`, y en la vista de diseño se puede

ver cómo queda el botón. Estas son las diferentes vistas en las que se puede trabajar en el layout:

- **Code:** permite visualizar solo el código XML.
- **Split:** permite visualizar el código y el diseño como se ve en la figura.
- **Design:** permite visualizar solo el editor de diseño.

```
<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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="80dp"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/btn clic"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/click" />
</LinearLayout>
```

Declarar el tipo de view

En la clase Java, declarar el tipo de view y referenciarlo. La clase en Java está compuesta por un método principal que se llama onCreate. Todo el código que esté dentro de este método se ejecuta inmediatamente se corra la aplicación. Cada clase tiene relacionado su layout en el método setContentView, por lo tanto, se debe referenciar los view en cada clase que lo contenga. En el ejemplo, la clase Java MainActivity contiene el layout activity_main que es donde se creó el Button.

```
import android.os.Bundle;

import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    Button btncllic; // Declarar el botón

    @Override

    protected void onCreate(Bundle savedInstanceState) { // Método principal

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        /* Para referenciar el Button que está en el layout

        se hace por medio del Id y con el método findViewById */

        btncllic = findViewById(R.id.btncllic);

        /* En la variable btncllic quedan almacenadas las propiedades del botón
```



```
para poder manejar el evento listener */  
  
}  
  
}
```

Implementar el evento Listeners

Implementar el evento listener: En la figura se ve la implementación del evento `setOnClickListener` para que cuando el usuario dé clic en el botón se ejecute una instrucción. El código debe escribirse en el método `onClick`.

Los View tiene diferentes métodos escuchadores, en la tabla que se encuentra a continuación se pueden observar los más utilizados:

- **ListView**. `SetOnItemClickListener`
- **Spinner**. `SetOnItemSelectedListener`.
- **ImageView**. `setOnClickListener`
- **RadioButton**. `setOnClickListener`

```
public class MainActivity extends AppCompatActivity {  
  
    Button btnclic; // Declarar el botón  
  
    @Override  
  
    protected void onCreate(Bundle savedInstanceState) { // Método principal  
  
        super.onCreate(savedInstanceState);  
  
        setContentView(R.layout.activity_main);  
  
        /* Para referenciar el Button que está en el layout
```

```
se hace por medio del Id y con el método findViewById */  
  
btn clic = findViewById(R.id.btn clic);  
  
/* En la variable btn clic quedan almacenadas las propiedades del botón  
para poder manejar el evento listener */  
  
btn clic.setOnClickListener(new View.OnClickListener() {  
  
    @Override  
  
    public void onClick(View v) {  
  
        // Aquí se colocan las instrucciones que se quieran ejecutar al dar clic en el botón  
  
    }  
  
});  
  
}  
  
}
```

4. Navegación (Intents)

La navegación entre pantallas en Android se realiza a través de los Intents. El Intent es básicamente una intención de realizar una acción; es la forma de comunicación en Android que permite la interacción entre componentes de la misma aplicación o entre diferentes aplicaciones (Activities, Services, Broadcasts, etc.).

Existen dos tipos de Intents:

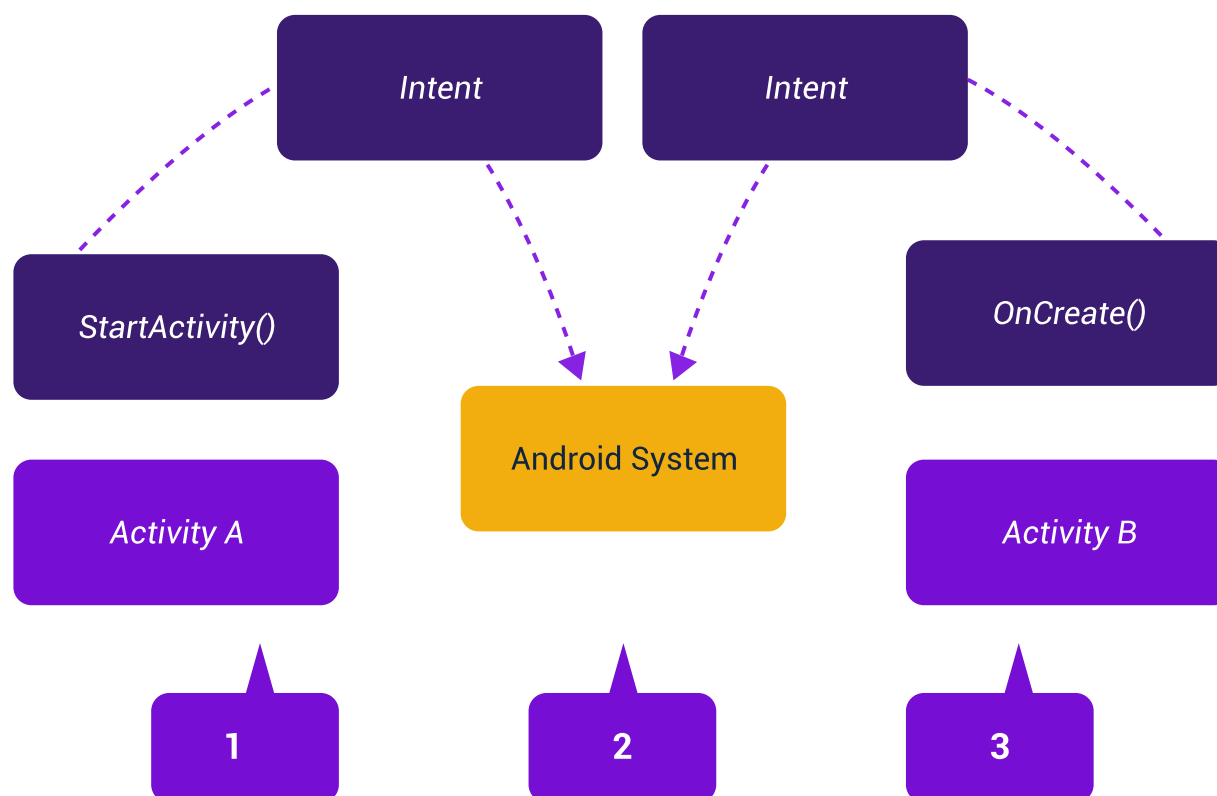
Intent explícito

Se utiliza para llamar a un componente específico. Por ejemplo, si se tiene una aplicación que contiene dos actividades, Actividad A y Actividad B, y se requiere navegar desde la Actividad A hasta la Actividad B, se define una intención explícita.

Para ello, se utiliza la clase Intent donde se especifica dónde está y a dónde se quiere ir:

```
Intent intent = new Intent(ActividadA.this, ActividadB.class);  
  
startActivity(intent);
```

Figura 28. Uso intent explícito



La figura presenta cómo se utiliza un intent para iniciar una actividad. Cuando el objeto Intent nombra un componente de actividad específico de forma explícita, el sistema inicia ese componente al instante.

Intent implícito

A diferencia de los explícitos, los intent implícitos no nombran el componente específico. Se usan para iniciar un componente específico de la aplicación. Por ejemplo, si se desea abrir la cámara, una ubicación en un mapa o una página web, se utilizan los intent implícitos para solicitar que se activen estos servicios.

Ejemplo:

```
Intent intent = new Intent(Intent.ACTION_VIEW,  
Uri.parse("http://www.example.com"));  
  
startActivity(intent);
```

5. Tareas asincrónicas

Todos los componentes de una aplicación Android, como las actividades, los servicios o los broadcast receivers, se ejecutan en el mismo hilo de ejecución, llamado hilo principal, main thread o GUI thread. Como sugiere este último nombre, también es el hilo donde se realizan todas las operaciones que gestionan la interfaz de usuario de la aplicación. Por ello, cualquier operación larga o costosa que se realice en este hilo bloqueará la ejecución del resto de componentes de la aplicación y, por supuesto, también la interfaz, produciendo un efecto evidente de lentitud, bloqueo o mal funcionamiento en general para el usuario.

Android utiliza la clase AsyncTask para ejecutar tareas en segundo plano.

La forma básica de utilizar la clase AsyncTask consiste en crear una nueva clase que extienda de ella y sobrescribir varios de sus métodos, distribuyendo la funcionalidad de la tarea entre ellos. Estos métodos son los siguientes:

- **onPreExecute()**

Se ejecutará antes del código principal de nuestra tarea. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.

- **doInBackground()**

Contendrá el código principal de nuestra tarea.

- **onProgressUpdate()**

Se ejecutará cada vez que llamemos al método publishProgress() desde el método doInBackground().

- **onPostExecute()**

Se ejecutará cuando finalice nuestra tarea, o, dicho de otra forma, tras la finalización del método `doInBackground()`.

- **onCancelled()**

Se ejecutará cuando se cancele la ejecución de la tarea antes de su finalización normal.

Estos métodos tienen una particularidad esencial para nuestros intereses. El método `doInBackground()` se ejecuta en un hilo secundario, por lo tanto, no se puede interactuar con la interfaz en su interior. Sin embargo, todos los demás métodos se ejecutan en el hilo principal, lo que significa que dentro de ellos se puede hacer referencia directa a los controles de usuario para actualizar la interfaz.

6. Persistencia de datos

En el desarrollo móvil, es indispensable que las aplicaciones puedan guardar datos, ya sea para no perder información en una nueva ejecución o para manejar información en bases de datos. La persistencia de datos es una habilidad esencial para que un desarrollador cree aplicaciones de alta calidad. Uno de los métodos más comunes de almacenamiento en Android es el de las Bases de Datos en SQLite.

6.1. Bases de datos local SQLite

SQLite es un motor de bases de datos relacional de código abierto y actualmente muy popular, ya que ofrece particularidades muy útiles como su pequeño tamaño, ligereza, no necesitar servidor, requerir poca configuración y guardar toda la base de datos en un solo fichero. Su uso es muy habitual en aplicaciones de tamaño pequeño.

- Android incluye soporte para SQLite y utiliza el paquete `android.database.sqlite` para trabajar con bases de datos.
- Permite ejecutar la aplicación en tiempo real en el dispositivo móvil.
- Para crear un esquema de base de datos o actualizar el mismo, se debe crear una clase que herede de `SQLiteOpenHelper`, donde se implementa el constructor y dos métodos abstractos (obligatorios de implementar): `onCreate()` y `onUpgrade()`.
- A través de los métodos `getReadableDatabase()` y `getWritableDatabase()`, se abre y se obtiene una instancia de la base de datos de solo lectura o de lectura y escritura, respectivamente.
- Para la creación de las tablas, se ejecutan las sentencias SQL dentro del método `onCreate()` utilizando el método `db.execSQL()`.

Para la administración de una base de datos SQLite se utiliza la clase SQLiteDatabase:

SQLiteDatabase: tiene métodos para crear, eliminar, ejecutar comandos SQL y realizar otras tareas comunes de administración de bases de datos.

Los métodos más utilizados son:

Tabla 7. Métodos de administración de bases de datos de la clase SQLiteDatabase

Acción	Definición
Insertar datos	db.insert
Consultar datos	db.rawQuery() y db.query()
Eliminar datos	db.delete
Actualizar datos	db.update
Ejecutar sentencia SQL	db.execute

Para reflejar la utilidad de las bases de datos SQLite, se realizará un ejemplo de una base de datos para registro de usuarios y de entidades. Se tendrá una actividad de solicitud de datos para la creación, actualización y borrado en la base de datos, permitiendo gestionar la información de forma persistente.

6.2. Bases de datos en tiempo real

Una base de datos en tiempo real se refiere a un sistema de base de datos que procesa datos en tiempo real, tomando los datos entrantes a medida que fluyen del servidor. La base de datos en tiempo real de Firebase (Firebase Realtime Database) es sin duda uno de los servicios más populares de la plataforma.

Firebase es una plataforma para el desarrollo de aplicaciones web y móviles, creada por James Tamplin y Andrew Lee en 2011, y adquirida por Google en 2014. Provee servicios muy útiles que se pueden incorporar en aplicaciones Android (además de otros sistemas operativos), tales como:

- **Analytics**

Firebase Analytics proporciona una visión profunda sobre el uso de la aplicación por parte de los usuarios, con datos que sirven para analizar el comportamiento de los usuarios con la aplicación.

- **Auth**

Servicio que puede autenticar a los usuarios utilizando únicamente código del lado del cliente. Incluye la autenticación mediante Facebook, GitHub, Twitter y Google. Además, incluye un sistema de administración del usuario por el cual los desarrolladores pueden habilitar la autenticación de usuarios con email y contraseña.

- **Database**

Proporciona una base de datos en tiempo real y back-end. El servicio proporciona a los desarrolladores de aplicaciones una API que permite que la información de las aplicaciones sea sincronizada y almacenada en la nube de Firebase.

- **Storage**

Proporciona cargas y descargas seguras de archivos para aplicaciones Firebase, sin importar la calidad de la red. El desarrollador lo puede utilizar para almacenar imágenes, audio, vídeo, u cualquier otro contenido generado por el usuario.

- **Notifications**

Con Firebase se puede usar la API FirebaseMessaging que permite el envío de mensajes desde Firebase hacia un dispositivo concreto o múltiples dispositivos, pudiendo gestionar las notificaciones desde la consola de Firebase.

Principales características de Firebase Realtime Database:

- No requiere la instalación de un servidor ni de software adicional, ya que la base de datos está alojada en la nube.
- El formato de almacenamiento de los datos es JSON, utilizando una base de datos no relacional.
- La actualización de los datos se realiza en milisegundos debido a la sincronización en tiempo real.
- Permite el desarrollo de aplicaciones multiplataforma para iOS, Android y Web (JavaScript).
- Incluso cuando el dispositivo se encuentra sin conexión a Internet, la aplicación sigue funcionando gracias a la caché.

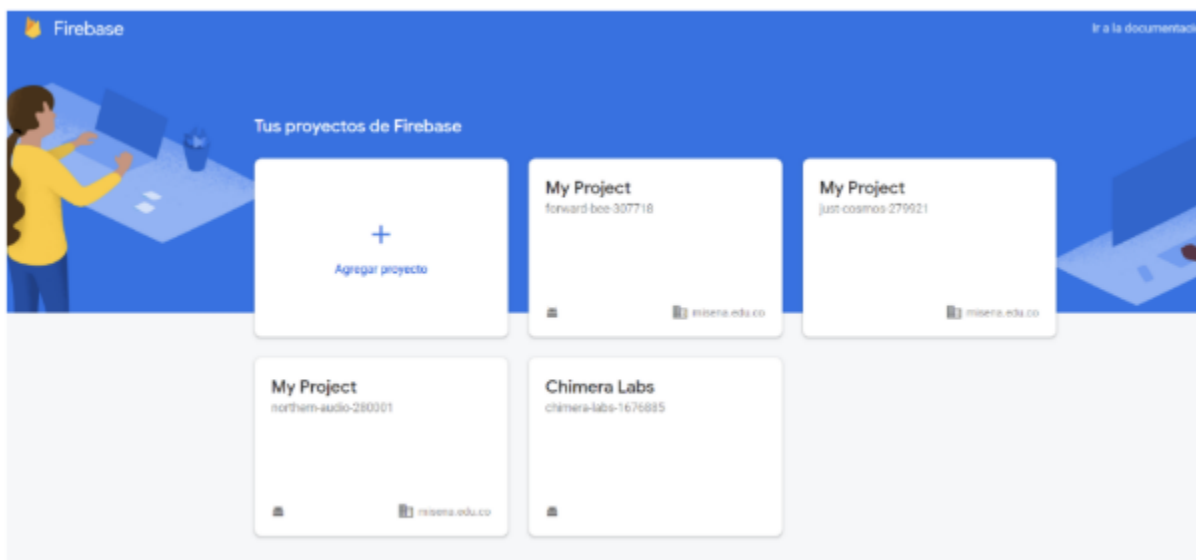
Para crear un proyecto con Firebase se deberán tener lo siguiente:

- Una cuenta de Gmail para acceder a la consola de Firebase
<https://console.firebase.google.com>
- Un proyecto de Android Studio para poder hacer la conexión con la base de datos de Firebase.

Para la configuración de Firebase en el proyecto de Android Studio se enumera los pasos siguientes:

a) Crear un Proyecto en la consola de Firebase.

Figura 29. Pantalla principal de Firebase



b) Agregar el proyecto.

Figura 30. Proceso de creación de un proyecto en Firebase



c) Seleccionar la plataforma para agregar a la APP.

Figura 31. Proyecto recién creado en Firebase



d) Ingresar el nombre del paquete de la APP en el campo Nombre del paquete.

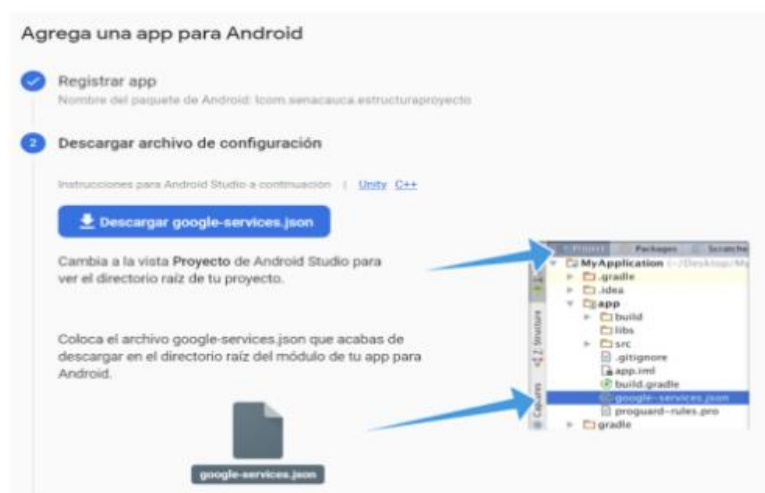
Lo más importante es copiar el paquete de su proyecto de Android Studio que se encuentra en el manifest `com.dominio.proyecto`.

Figura 32. Formulario para registrar una aplicación Android en Firebase



- e) Descargar archivo Json google-services.json y copiarlo en la raíz del proyecto de Android Studio.

Figura 33. Proceso de agregar una app para Android en Firebase



- f) Agregar SDK de Firebase.

Copiar las librerías al gradle del proyecto en Android Studio y ya está lista la configuración de Android Studio con Firebase.

Figura 34. Instrucciones para agregar el SDK de Firebase al proyecto



7. Multimedia

En Android Studio, la integración de contenido multimedia en las aplicaciones es muy sencilla gracias a las opciones que presenta la API. Las principales clases que facilitan el acceso a los servicios multimedia son MediaPlayer y VideoView, siendo estas las más utilizadas.

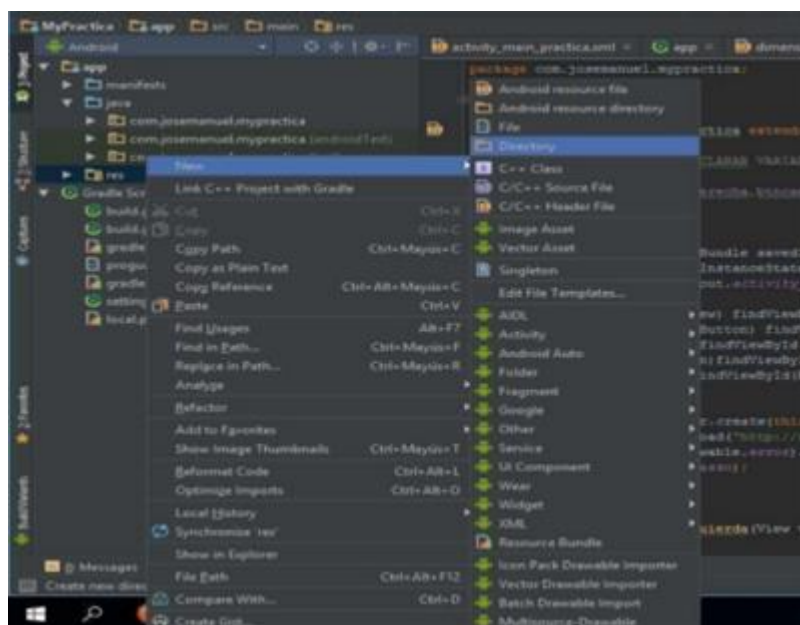
7.1. MediaPlayer

MediaPlayer permite reproducir sonido y video a través de su API. Es uno de los componentes principales del marco de trabajo de medios. Un objeto de esta clase puede recuperar, decodificar y reproducir audio y video con una configuración mínima. Es compatible con varias fuentes de medios diferentes, entre las que se incluyen las siguientes: URI, recursos locales y URL externas.

A continuación, se presenta un ejemplo de cómo reproducir un archivo de audio que está disponible como recurso local (ubicado en el directorio res/raw/ de la app):

- Crear un directorio dentro de la carpeta res llamada raw.

Figura 35. Pantalla de Android Studio que presenta el menú contextual



- Copiar un sonido en la carpeta raw de extensión MP3.

Button Click Sound

How to play audio on button click

Botón 1

Botón 3

Botón 3

- En la clase Java escribe el siguiente código.

```
public class MainActivityPractica extends AppCompatActivity {
    MediaPlayer cancion;
}
```

- Se crea la instancia del sonido.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```



```
cancion = MediaPlayer.create(this, R.raw.bruja);  
}
```

- Ejecución del sonido.

```
public void iniciarSonido(View view) {  
    cancion.start();  
}
```

7.2. VideoView

La clase VideoView permite presentar un archivo de video. Puede cargar videos de varias fuentes (como recursos o proveedores de contenido). La propiedad principal de VideoView es setVideoURI, donde se establece la ruta del video.

A continuación, se describe un ejemplo para utilizar VideoView:

- a) Al igual que con el MediaPlayer, se debe crear la carpeta res/raw y guardar en esta carpeta el video con extensión .mp4.
- b) En el layout, se crea el componente VideoView.

```
<VideoView  
    android:id="@+id/videoview"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content" />
```

- c) En la clase Java se realiza todo el código:

```
public class MainActivity extends AppCompatActivity {  
  
    // Paso 1. Se declara el VideoView  
  
    VideoView videoView;
```

```
@Override

protected void onCreate(Bundle savedInstanceState) { // Método principal

    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_main);

    // Paso 2. Referenciar el VideoView

    videoView = findViewById(R.id.videoview);

    // Paso 3. Se asigna la ruta al VideoView

    videoView.setVideoPath("android.resource://" + getPackageName() + "/" +
R.raw.video);

    // 4. La propiedad SetMediaController permite visualizar los controles de
    pausar, adelantar, iniciar

    videoView.setMediaController(new MediaController(this));

    // 5. Se inicia la ejecución de video

    videoView.start();

}

}
```

Al ejecutar se presentará de la siguiente manera:

Figura 36. Vista Ejecución VideoView



Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
Estructura de un proyecto en Android Studio	Android Developers. (2021, May 25). Aspectos fundamentales de la aplicación.	Página	https://developer.android.com/training/basics/activity-lifecycle
Estructura de un proyecto en Android Studio	Developers. (2021, June 14). Notas de la versión del complemento de Android para Gradle.	Página	https://developer.android.com/studio/releases/gradle-plugin
Interfaz de usuario en Android	Developers. (2021) Cómo crear una IU responsiva con ConstraintLayout.	Página	https://developer.android.com/training/constraint-layout
Interfaz de usuario en Android	Castillo, J. D. L. (2015, May 21). Capítulo 14 - Interacción con Botones, TextView, ImageView. YouTube.	Video	https://www.youtube.com/watch?v=z8zTGTo7vA&ab_channel=Jos%C3%A9DimasLuj%C3%A1nCastillo
Interfaz de usuario en Android	Castillo, J. D. L. (2015, May 21). Capítulo 15 - Android Studio Intent Abrir una Activity. YouTube.	Video	https://www.youtube.com/watch?v=LEVXzsXhO0A&ab_channel=Jos%C3%A9DimasLuj%C3%A1nCastillo

Glosario

Diseño: un diseño define la estructura visual de una interfaz de usuario.

Scrolling: acción de desplazarse internamente.

SDK: kit de desarrollo de software.

String: recurso XML que ofrece una sola string.

View: es la clase base de los widgets.

Wearable: dispositivo conectado que se puede llevar puesto y que se conecta al teléfono móvil.

Referencias bibliográficas

Android Developers. (2021). Documentation | Desarrolladores de Android.

<https://developer.android.com/docs>

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Castellanos	Responsable del Ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de Línea de Producción	Centro de Servicios de Salud - Regional Antioquia
Zulema Yidney León Escobar	Experta Temática	Centro de Teleinformática y Producción Industrial - Regional Cauca
Paola Alexandra Moya Peralta	Evaluadora Instruccional	Centro de Servicios de Salud - Regional Antioquia
Andrés Felipe Herrera Roldán	Diseñador de Contenidos Digitales	Centro de Servicios de Salud - Regional Antioquia
Edward Leonardo Pico Cabra	Desarrollador Fullstack	Centro de Servicios de Salud - Regional Antioquia
Edgar Mauricio Cortés García	Actividad Didáctica	Centro de Servicios de Salud - Regional Antioquia
Daniela Muñoz Bedoya	Animador y Productor Multimedia	Centro de Servicios de Salud - Regional Antioquia
Jaime Hernán Tejada Llano	Validador de Recursos Educativos Digitales	Centro de Servicios de Salud - Regional Antioquia
Margarita Marcela Medrano Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia
Daniel Ricardo Mutis Gómez	Evaluador para Contenidos Inclusivos y Accesibles	Centro de Servicios de Salud - Regional Antioquia