

Interface Gráfica

Profesor: Ana Isabel Vegas

INDICE

1. INTRODUCCIÓN	3
2.- LAYOUTS	5
FRAME LAYOUT	5
LINEAR LAYOUT	6
TABLE LAYOUT	10
GRID LAYOUT	13
RELATIVE LAYOUT	15
LIST VIEW	18
GRID VIEW.....	19
CARDVIEW.....	22
RECYCLERVIEW	23
3. CONTROLES DE ENTRADA	31
BOTONES	31
CAJAS DE TEXTO.....	33
ETIQUETAS.....	33
IMAGENES.....	34
CHECKBOXES	35
RADIO BUTTONS	36
SPINNERS.....	38
4.- EVENTOS	41
5.- GESTION DE TEXTOS.....	42
EL ARCHIVO STRINGS.XML	42
INTERNACIONALIZACION.....	42
6.- MATERIAL DESIGN.....	46
COLORES	46
ESTILOS	48
TEMAS.....	49
ÍNDICE DE GRÁFICOS	50

1. INTRODUCCIÓN

Para diseñar la interface de usuario de nuestras aplicaciones necesitamos los siguientes componentes:

- Un layout que será donde definimos la estructura de la interface.
- Los objetos view que son los componentes básicos con los que se construye la interfaz gráfica de la aplicación, por ejemplo cajas de texto, botones, ...etc.

Para construir una interface necesitamos de dos archivos:

- Un documento xml que será donde diseñemos la pantalla.
- Una actividad, que será clase Java donde programamos el comportamiento de la interface.

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

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/nombre" />

    <EditText
        android:id="@+id/TxtNombre"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

    <Button
        android:id="@+id/BtnHola"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hola" />
</LinearLayout|
```

Gráfico 1. Diseño de la pantalla

```
package com.example.android;

import android.app.Activity;

public class HolaUsuario extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Localizar los controles
        final EditText txtNombre = (EditText)findViewById(R.id.TxtNombre);
        final Button btnHola = (Button)findViewById(R.id.BtnHola);

        btnHola.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                //Creamos el Intent
                Intent intent = new Intent(HolaUsuario.this, FrmMensaje.class);

                //Creamos la información a pasar entre actividades
                Bundle b = new Bundle();
                b.putString("NOMBRE", txtNombre.getText().toString());

                //Añadimos la información al intent
                intent.putExtras(b);

                //Iniciamos la nueva actividad
                startActivity(intent);
            }
        });
    }
}
```

Gráfico 2. Lógica de negocio de la pantalla

2 . - LAYOUTS

FRAME LAYOUT

Éste es el más simple de todos los layouts de Android.

Un FrameLayout coloca todos sus controles hijos alineados con su esquina superior izquierda, de forma que cada control quedará oculto por el control siguiente (a menos que éste último tenga transparencia). Por ello, suele utilizarse para mostrar un único control en su interior, a modo de contenedor (placeholder) sencillo para un sólo elemento sustituible, por ejemplo una imagen.

Los componentes incluidos en un FrameLayout podrán establecer sus propiedades android:layout_width y android:layout_height, que podrán tomar los valores “fill_parent” (para que el control hijo tome la dimensión de su layout contenedor) o “wrap_content” (para que el control hijo tome la dimensión de su contenido).

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <EditText
        android:id="@+id/TxtNombre"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:inputType="text"
        android:text="Ejemplo FrameLayout" />

</FrameLayout>
```

Gráfico 3. Diseño FrameLayout

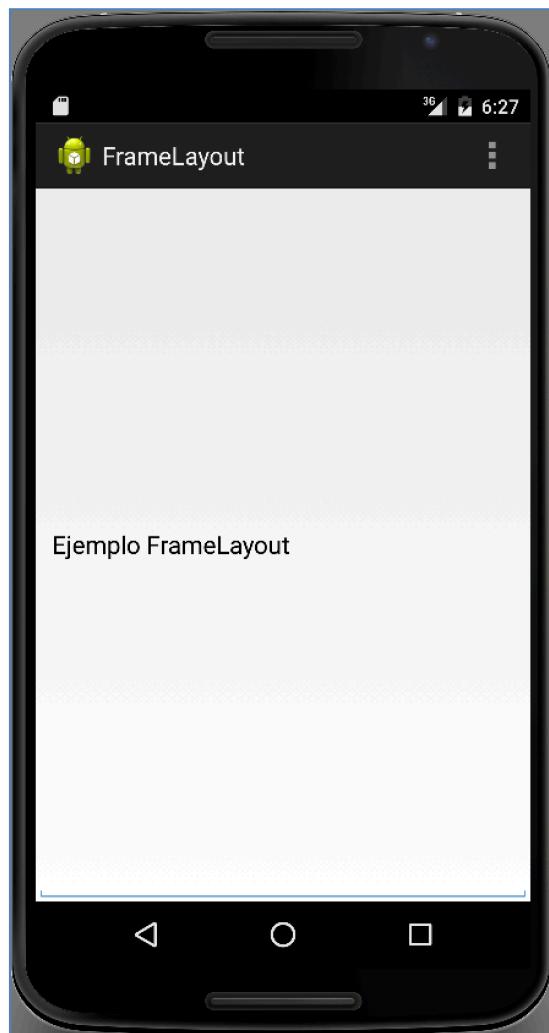


Grafico 4. Vista FrameLayout

LINEAR LAYOUT

Mediante esta plantilla podemos agregar los componentes uno tras otro tanto en orientación horizontal como vertical.

Los elementos contenidos en un LinearLayout pueden establecer sus propiedades android:layout_width y android:layout_height para determinar sus dimensiones dentro del layout.

Pero en el caso de un LinearLayout, tendremos otro parámetro con el que jugar, la propiedad android:layout_weight.

Esta propiedad nos va a permitir dar a los elementos contenidos en el layout unas dimensiones proporcionales entre ellas.

Ejemplo. Si incluimos en un LinearLayout vertical dos cuadros de texto (EditText) y a uno de ellos le establecemos un layout_weight="1" y al otro un layout_weight="2" conseguiremos como efecto que toda la superficie del layout quede ocupada por los dos cuadros de texto y que además el segundo sea el doble (relación entre sus propiedades weight) de alto que el primero.

ATRIBUTOS XML

android:orientation; los valores que podemos escribir son horizontal o vertical; si es horizontal entonces trabajaremos el layout a modo de fila y si es vertical lo trabajaremos a modo de columna.

android:layout_width y android:layout_height; Tenemos tres posibles opciones:

Podemos asignar una dimensión específica, por ejemplo: 150px que se deberá mostrar de esta forma sin importar la resolución de la pantalla que se tenga.

Podemos definir el valor wrap_content, por medio del cual le indicamos a la aplicación que el elemento sólo deberá ocupar lo correspondiente a su tamaño natural, a menos que sea demasiado grande, en este caso Android usará la propiedad word-wrap según sea necesario para que el elemento se ajuste.

Por último, podemos utilizar el valor fill_parent, que indica que el elemento es libre de utilizar todo el espacio disponible del contenedor en el que se encuentra.

Comúnmente, utilizarás las dos últimas opciones ya que son independientes del tamaño de pantalla del dispositivo.

android:layout_weight; Por otro lado si tenemos definido un widget con el valor de "1" en este atributo y "2" para otro widget, esto hará que el segundo widget ocupe el doble del espacio libre con respecto al primero y así sucesivamente.

android:layout_gravity; Por default, los elementos se alinean a partir de la esquina superior izquierda. Por lo tanto, si creamos una fila de widgets de forma horizontal dentro de un contenedor de tipo LinearLayout, todos ellos se empezarán a alinear a partir de este punto y seguirá ese flujo hasta que todos los widgets aparezcan en la interfaz. Utilizamos este atributo para indicarle al elemento y a su contenedor cómo deberá alinearse en la pantalla. Los valores que le podemos asignar a este atributo son: left, center_horizontal , center_vertical y right.

android:padding; Por default, los widgets se apilan de manera seguida una junto a otro, cosa que algunas veces no resulta muy agradable visualmente. Por ello, para aumentar el espacio en blanco entre cada uno de ellos utilizamos este atributo.

La propiedad android:padding nos permite definir un sólo valor de relleno para los cuatro lados del widget. Si queremos que el relleno aplique únicamente a alguno de los lados del elemento o bien, definir valores distintos para cada lado, podemos utilizar las propiedades **android:paddingLeft**, **android:paddingRight**, **android:paddingTop**, y **android:paddingBottom**.

Para definir el valor de este atributo tenemos una serie de unidades de medidas que podemos utilizar:

- **dp** – píxeles independientes de la densidad: es una unidad abstracta que se basa en la densidad de píxeles de la pantalla. Sirve para garantizar que las cosas tengan las mismas medidas físicas en cualquier pantalla. Como regla, hay que tener en cuenta que 160dp equivaldrá a una pulgada (o, lo que es lo mismo, 63dp serán un centímetro) en cualquier pantalla.
- **sp** – píxeles independientes de la escala: es igual que la unidad dp, pero escalada según el tamaño de fuente escogido por el usuario. Es la unidad recomendada para definir el tamaño de los textos que se muestran en pantalla.
- **pt** – puntos: representan 1/72 partes de una pulgada (o, más o menos, 1/28 de un centímetro) en cualquier pantalla.
- **px** – píxeles: no se recomienda utilizar esta unidad, porque las pantallas tienen diferentes densidades y tamaños, por lo que los elementos dimensionados en píxeles se pueden ver de formas muy diferentes.
- **mm** – milímetros: medida directa del elemento en pantalla, en unidades más internacionales.

- **in** – pulgadas: medida directa del elemento en pantalla, en unidades anglosajonas.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Para:" />

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Asunto:" />

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="Mensaje:" />

    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Enviar" />

</LinearLayout>

```

Grafico 5. Diseño LinearLayout

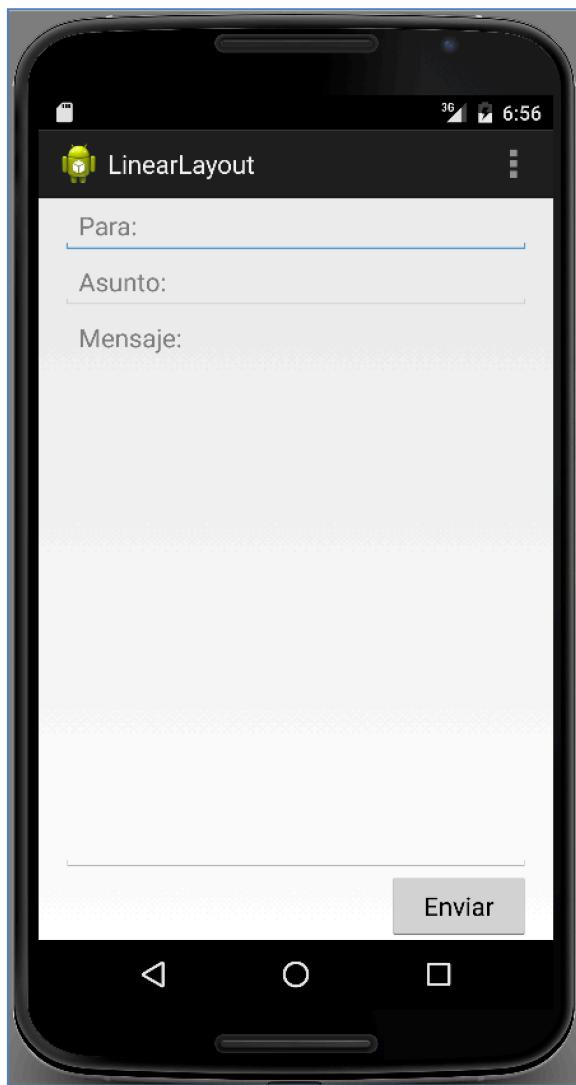


Grafico 6. Vista LinearLayout

TABLE LAYOUT

Un TableLayout permite distribuir sus elementos hijos de forma tabular, definiendo las filas y columnas necesarias, y la posición de cada componente dentro de la tabla.

La estructura de la tabla se define de forma similar a como se hace en HTML, es decir, indicando las filas que compondrán la tabla (objetos TableRow), y dentro de cada fila las columnas necesarias, con la salvedad de que no existe ningún objeto especial para definir una columna (algo así como un

TableColumn) sino que directamente insertaremos los controles necesarios dentro del TableRow y cada componente insertado (que puede ser un control sencillo o incluso otro ViewGroup) corresponderá a una columna de la tabla. De esta forma, el número final de filas de la tabla se corresponderá con el número de elementos TableRow insertados, y el número total de columnas quedará determinado por el número de componentes de la fila que más componentes contenga.

Por norma general, el ancho de cada columna se corresponderá con el ancho del mayor componente de dicha columna, pero existen una serie de propiedades que nos ayudarán a modificar este comportamiento:

- **android:stretchColumns**. Indicará las columnas que pueden expandir para absorver el espacio libre dejado por las demás columnas a la derecha de la pantalla.
- **android:shrinkColumns**. Indicará las columnas que se pueden contraer para dejar espacio al resto de columnas que se puedan salir por la derecha de la pantalla.
- **android:collapseColumns**. Indicará las columnas de la tabla que se quieren ocultar completamente.

Todas estas propiedades del TableLayout pueden recibir una lista de índices de columnas separados por comas (ejemplo: android:stretchColumns="1,2,3") o un asterisco para indicar que debe aplicar a todas las columnas (ejemplo: android:stretchColumns="*").

Otra característica importante es la posibilidad de que una celda determinada pueda ocupar el espacio de varias columnas de la tabla (análogo al atributo colspan de HTML). Esto se indicará mediante la propiedad android:layout_span del componente concreto que deberá tomar dicho espacio.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="5dp"
    android:padding="15dip">

    <TableRow>
        <TextView android:text="Celda 1.1" />
        <TextView android:text="Celda 1.2" />
        <TextView android:text="Celda 1.3" />
    </TableRow>

    <TableRow>
        <TextView android:text="Celda 2.1" />
        <TextView android:text="Celda 2.2" />
        <TextView android:text="Celda 2.3" />
    </TableRow>

    <TableRow>
        <TextView
            android:layout_span="2"
            android:text="Celda 3.1" />
        <TextView android:text="Celda 3.2" />
    </TableRow>

</TableLayout>
```

Gráfico 7. Diseño TableLayout

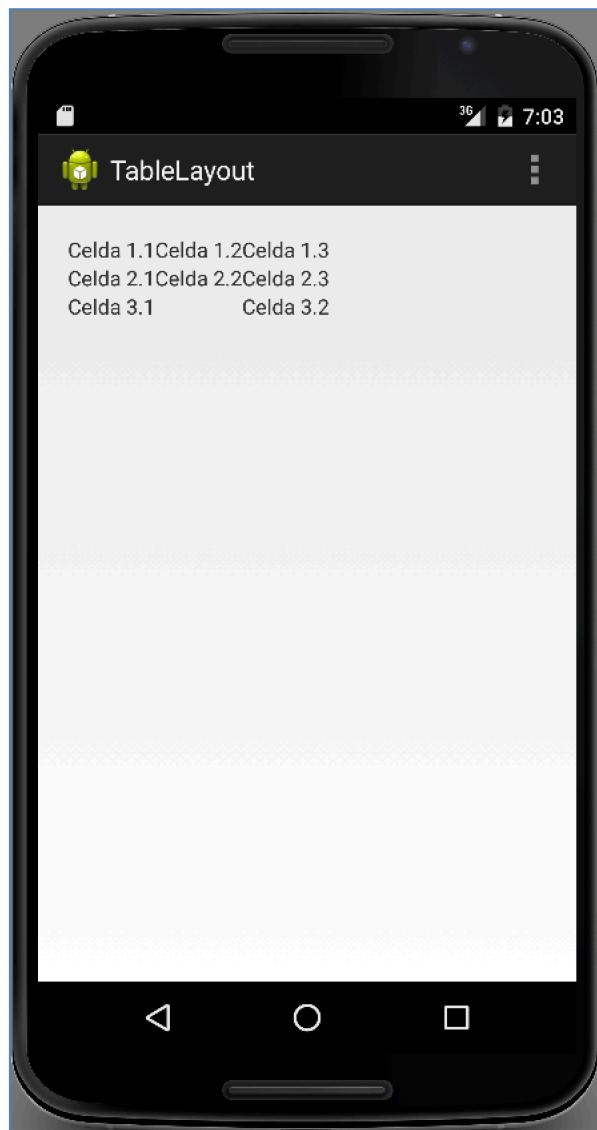


Gráfico 8. Vista TableLayout

GRID LAYOUT

Este tipo de layout fue incluido a partir de la API 14 (Android 4.0) y sus características son similares al TableLayout, ya que se utiliza igualmente para distribuir los diferentes elementos de la interfaz de forma tabular, distribuidos en filas y columnas. La diferencia entre ellos estriba en la forma que tiene el GridLayout de colocar y distribuir sus elementos hijos en el espacio disponible. En este caso, a diferencia del TableLayout indicaremos el número de filas y

columnas como propiedades del layout, mediante **android:rowCount** y **android:columnCount**. Con estos datos ya no es necesario ningún tipo de elemento para indicar las filas, como hacíamos con el elemento TableRow del TableLayout, sino que los diferentes elementos hijos se irán colocando ordenadamente por filas o columnas (dependiendo de la propiedad android:orientation) hasta completar el número de filas o columnas indicadas en los atributos anteriores.

Adicionalmente, igual que en el caso anterior, también tendremos disponibles las propiedades **android:layout_rowSpan** y **android:layout_columnSpan** para conseguir que una celda ocupe el lugar de varias filas o columnas.

Existe también una forma de indicar de forma explícita la fila y columna que debe ocupar un determinado elemento hijo contenido en el GridLayout, y se consigue utilizando los atributos **android:layout_row** y **android:layout_column**. De cualquier forma, salvo para configuraciones complejas del grid no suele ser necesario utilizar estas propiedades.

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout 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:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:rowCount="2"
    android:columnCount="3"
    tools:context="com.grupoatrium.gridlayout.MainActivity">

    <TextView android:text="Celda 1.1" />
    <TextView android:text="Celda 1.2" />
    <TextView android:text="Celda 1.3" />

    <TextView android:text="Celda 2.1" />
    <TextView android:text="Celda 2.2" />
    <TextView android:text="Celda 2.3" />

    <TextView android:text="Celda 3.1" android:layout_columnSpan="2" />
    <TextView android:text="Celda 3.2" />

</GridLayout>
```

Gráfico 9. Diseño GridLayout

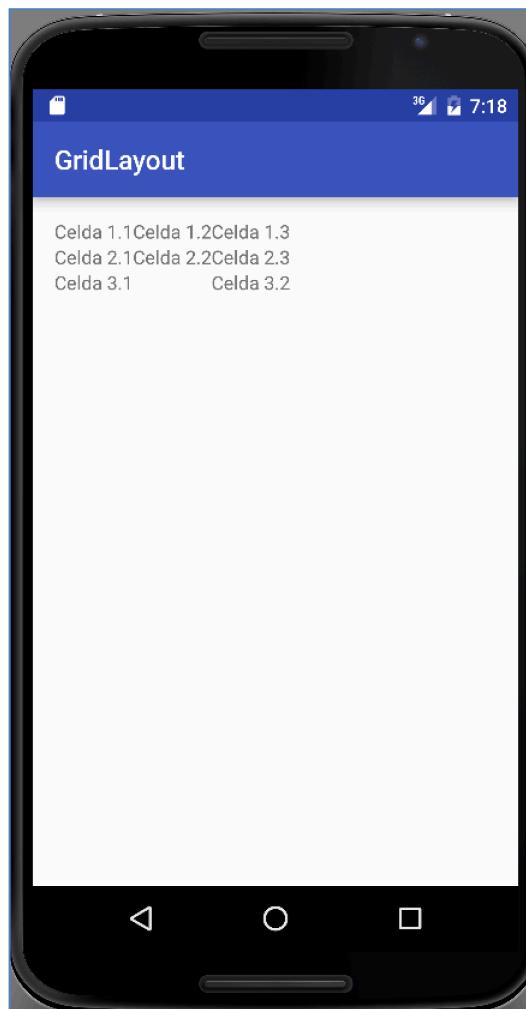


Gráfico 10. Vista GridLayout

RELATIVE LAYOUT

Permite mostrar los view en una posición relativa a otros elementos de la interface.

ATRIBUTOS XML

android:layout_alignParentTop; Si le establecemos el valor a true alinea el componente arriba del layout padre.

android:layout_centerVertical; Si le establecemos el valor a true centra el componente verticalmente respecto a su componente padre.

android:layout_below; Coloca el componente debajo del elemento con el ID especificado.

android:layout_toRightOf; Coloca el componente a la derecha del elemento con el ID especificado.

Además de estos atributos tendremos un sinfín de propiedades para colocar cada control justo donde queramos.

- Posición relativa a otro control:
 - android:layout_above.
 - android:layout_below.
 - android:layout_toLeftOf.
 - android:layout_toRightOf.
 - android:layout_alignLeft.
 - android:layout_alignRight.
 - android:layout_alignTop.
 - android:layout_alignBottom.
 - android:layout_alignBaseline.
- Posición relativa al layout padre:
 - android:layout_alignParentLeft.
 - android:layout_alignParentRight.
 - android:layout_alignParentTop.
 - android:layout_alignParentBottom.
 - android:layout_centerHorizontal.
 - android:layout_centerVertical.
 - android:layout_centerInParent.
- Opciones de margen (también disponibles para el resto de layouts):
 - android:layout_margin.
 - android:layout_marginBottom.
 - android:layout_marginTop.
 - android:layout_marginLeft.
 - android:layout_marginRight.

- Opciones de espaciado o padding (también disponibles para el resto de layouts):
 - android:padding.
 - android:paddingBottom.
 - android:paddingTop.
 - android:paddingLeft.
 - android:paddingRight.

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >

    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="Recuerda:" />

    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/name"
        android:layout_toLeftOf="@+id/times" />

    <Spinner
        android:id="@+id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/name" />

    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/times"
        android:text="Aceptar" />

</RelativeLayout>

```

Gráfico 11. Diseño RelativeLayout

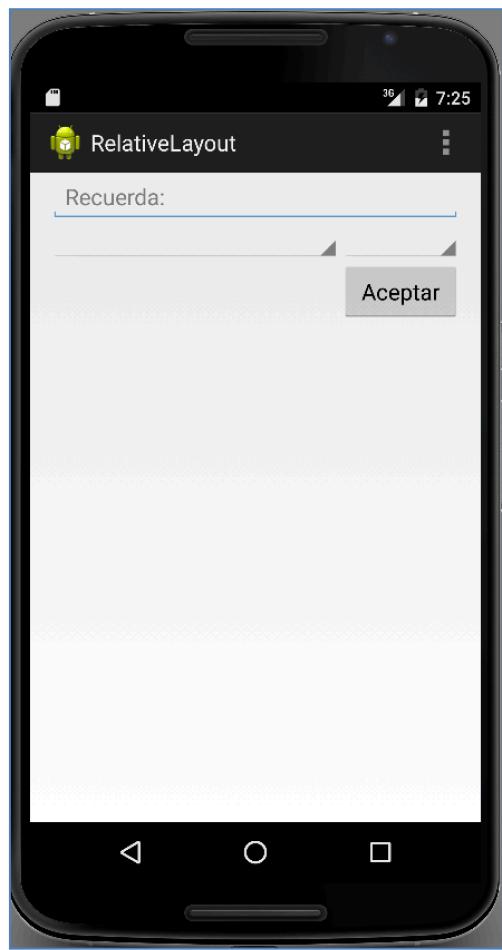


Gráfico 12. Vista RelativeLayout

LIST VIEW

Una vista ListView visualiza una lista deslizable verticalmente de varios elementos, donde cada elemento puede definirse como un Layout .Su utilización es algo compleja, pero muy potente.

Para utilizar un ListView dentro de un Layout has de usar la siguiente estructura:

```

<FrameLayout>
    <ListView
        android:id="@+id/list"
        ... />
    <TextView
        android:id="@+id/empty"
        ... />
</FrameLayout>

```

Gráfico 13. Estructura ListLayout

Donde tenemos un FrameLayout que me permite visualizar dos posibles elementos, uno u otro, pero no los dos simultáneamente. El primero es el ListView que se visualizará cuando haya algún elemento en la lista. El segundo puede ser cualquier tipo de vista y se visualizará cuando no existan elementos en la lista. El sistema controla la visibilidad de forma automática, solo has de tener cuidado de identificar cada uno de los elementos con el valor exacto que se muestra.

GRID VIEW

Nos permite crear una cuadrícula bidimensional de elementos entre los que podemos elegir. Cuando trabajamos con GridView podemos definir el número y el tamaño de las columnas; el número de filas por el contrario se define dinámicamente en función del número de ítems que el adaptador que estemos utilizando indique como disponibles para la vista a desplegar en la pantalla.

Existen una serie de propiedades que cuando se combinan, nos permiten determinar el número de columnas y sus respectivos tamaños:

- **android:numColumns** indica el número de columnas que tendrá la cuadrícula. O en el caso de que indiquemos el valor `auto_fit` en este atributo, Android calculará el número de columnas basado en el espacio disponible y en las propiedades que veremos a continuación.
- **android:verticalSpacing** y **android:horizontalSpacing** indican cuántos pixeles de espacio en blanco deben existir entre cada ítem de la cuadrícula.
- **android:columnWidth** indica cuántos pixeles de ancho deberá tener cada columna.

- **android:stretchMode** define, para las cuadrículas cuyo valor para el atributo android:numColumns sea `auto_fit`, cómo debe aprovecharse cada espacio disponible que no esté siendo ocupado por el ancho de las columnas o por los espacios en blanco que se definan. De esta forma, utilizaremos el valor `android:stretchMode=columnWidth` para indicar que las columnas deberán expandirse y aprovechar el espacio en blanco disponible y el valor `android:stretchMode=spacingWidth` para hacer que el espacio en blanco entre las columnas se expanda.

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridView1"
    android:numColumns="auto_fit"
    android:gravity="center"
    android:columnWidth="50dp"
    android:stretchMode="columnWidth"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

</GridView>
```

Gráfico 14. Diseño GridView

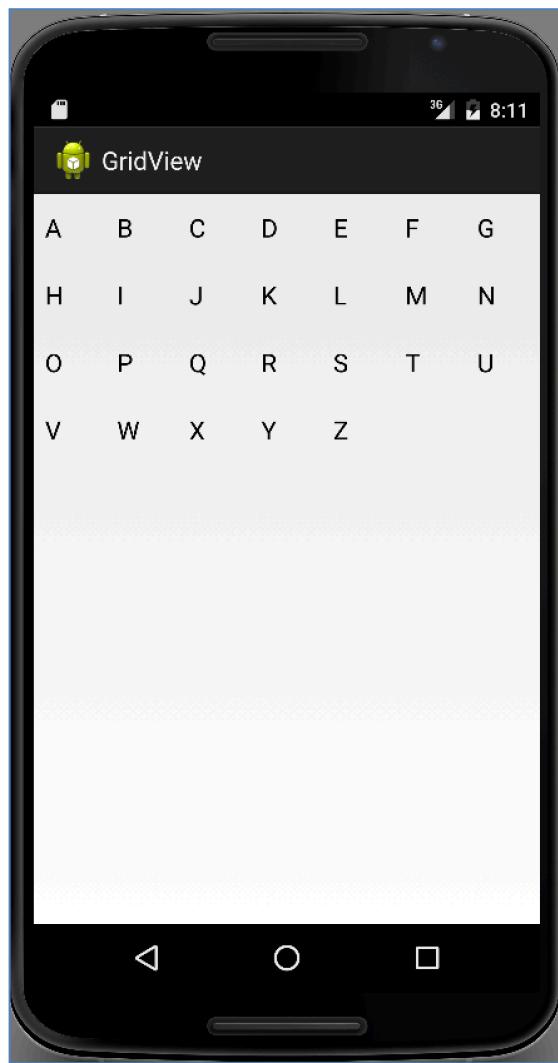


Gráfico 15. Vista GridView

Android 5 añade dos nuevos layouts CardView y ReciclerView. Para usar los dos nuevos widgets necesitamos compilar las dependencias correspondientes, estas se encuentran dentro de el conjunto de librerías de soporte y pueden ser incluidas de la siguiente manera, en el archivo build.gradle de la raíz del directorio app:

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:cardview-v7:+'  
    compile 'com.android.support:recyclerview-v7:+'  
}
```

Gráfico 16. Build.gradle

CARDVIEW

Nos permite agrupar casi cualquier tipo y número de elementos en una vista que, al final, resulta un tipo “Tarjeta” como lo visto en Google Now. Al venir de la librería de soporte, este nuevo widget funciona igual en versiones anteriores a L.

El uso de CardView es muy sencillo, al final se trata de un widget que otorga dicho UI a lo que se encuentre entre el. Uno de sus atributos es el poder asignar bordes redondeados a la carta: card_view:cardCornerRadius="2dp".

Ademas existe un atributo

android:foreground="?android:attr/selectableItemBackground" para que la tarjeta obtenga la animación característica de Material Design al hacer clic sobre ella.

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

<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:foreground="?android:attr/selectableItemBackground"
    android:clickable="true"
    android:layout_margin="10dp"
    android:id="@+id/card_view"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    card_view:cardCornerRadius="2dp">

    <RelativeLayout

        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <ImageView
            android:src="@drawable/backend"
            android:layout_width="150dp"

```

Gráfico 17. Layout CardView

RECYCLERVIEW

Una actualización de ListView. Esta vez es mas fácil de usar, optimiza recursos, y ofrece animaciones predeterminadas. Útil para su uso en colecciones dinámicas (ListView y GridView).

Haciendo que todo fluya como la seda, sin embargo, las aplicaciones ahora requieren más espacio de almacenamiento en tu dispositivo, pero velocidad y optimización de batería lo valen.

RecyclerView es una mejora de lo antes visto en ListView o GridView. Es un widget que despliega vistas que comparten la misma estructura de vista de los datos, ósea , son vistas que se “repiten” en forma, pero no en información. RecyclerView nos proporciona un LayoutManager que es el encargado de asignar el acomodo de los elementos (Grid o Linear), aunque por el momento solo esta disponible Linear (como lista). Para que esto funcione necesitamos construir nuestro Adapter y a su vez, el Adapter necesita de un Dataset, ósea un conjunto (arreglo) de objetos.

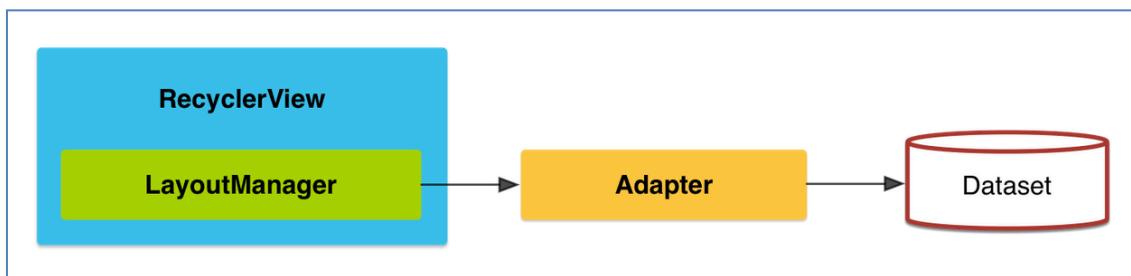


Gráfico 18. Recursos para crear RecyclerView

El Dataset requiere de un arreglo de objetos a mostrar. Regularmente un objeto se crea mediante algo llamado POJO (Plain Old Java Object) y dicho POJO contiene todas las características del objeto.

```
public class Course {

    private Integer id;
    private String name;
    private String description;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
}
```

Gráfico 19. POJO

Crear el Adapter no es tan complicado, lo primero que debemos hacer es crear una clase que extienda de RecyclerView.Adapter<CoursesAdapter.ViewHolder> e implementar los métodos requeridos. El constructor recibe dos parámetros, uno es el Dataset (Arreglo de objetos del tipo Curso) y el otro es el ItemLayout (es el layout que creamos con el CardView y describe cada curso)

```
private ArrayList<Course> courses;

private int itemLayout;

public CoursesAdapter(ArrayList<Course> data, int itemLayout){

    courses = data;

    this.itemLayout = itemLayout;

}
```

Gráfico 20. Adapter

La clase interna ViewHolder que extiende de RecyclerView.ViewHolder contiene todos los llamados a los widgets contenidos en el ItemLayout (compuesto de un ImageView y 2 TextView), por lo que se garantiza que el llamado es solo una vez, haciendo que consuma menos recursos a nivel de batería y procesamiento. Anteriormente un ViewHolder era una buena práctica, ahora es algo obligatorio. Ademas se implementa un AdapterView.OnClickListener para que cada elemento en el listado tenga un evento OnClick.

```

public static class ViewHolder extends RecyclerView.ViewHolder implements
AdapterView.OnClickListener {

    public ImageView image;
    public TextView name;
    public TextView description;

    public ViewHolder(View itemView) {
        super(itemView);

        itemView.setOnClickListener(this);

        image = (ImageView) itemView.findViewById(R.id.image);
        name = (TextView) itemView.findViewById(R.id.name);
        description = (TextView) itemView.findViewById(R.id.description);
    }

    @Override
    public void onClick(View view) {
        Toast.makeText(view.getContext(), "OnItemClick :D", Toast.LENGTH_SHORT).show();
    }
}

```

Gráfico 21. ViewHolder

El método onCreateViewHolder se encarga de inflar la vista del ItemLayout y el método onBindViewHolder es el encargado de pasar la información del Dataset a sus respectivos lugares en el ItemLayout. Ademas de recordarle al viewHolder el estado actual de los elementos del ItemLayout. Al final el método getItemCount nos permite conocer el tamaño o cantidad de elementos que contiene nuestro RecyclerView que es el mismo que el tamaño del dataset. Y listo, todo lo anterior en conjunto crea nuestroAdapter para poder usar nuestro RecyclerView.

```
@Override

public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
    View v = LayoutInflater.from(viewGroup.getContext()).inflate(itemLayout, viewGroup, false);
    return new ViewHolder(v);
}
```

Gráfico 22. Método onCreateViewHolder

```
@Override

public void onBindViewHolder(ViewHolder viewHolder, int position) {

    Course course = courses.get(position);
    viewHolder.name.setText(course.getName());
    viewHolder.description.setText(course.getDescription());

    switch (course.getId()){

        case 1:

            viewHolder.image.setImageResource(R.drawable.disenio);
            break;

        case 2:

            viewHolder.image.setImageResource(R.drawable.android);
            break;
    }
}
```

Gráfico 23. Método onBindViewHolder

```
@Override  
  
    public int getItemCount() {  
  
        return courses.size();  
    }
```

Gráfico 24. Método getItemCount

```
<RelativeLayout  
  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    xmlns:android="http://schemas.android.com/apk/res/android">  
  
    <android.support.v7.widget.RecyclerView  
        android:scrollbars="vertical"  
        android:id="@+id/my_recycler_view"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"/>
```

Gráfico 25. Layout RecyclerView

```
ArrayList<Course> courses;

ReadLocalJSON readLocalJSON = new ReadLocalJSON();

courses = readLocalJSON.getCourses(getActivity());

Recycler View recycler View = (Recycler View) getActivity().findViewById(R.id.my_recycler_view);

recyclerView.setHasFixedSize(true);

recyclerView.setAdapter(new CoursesAdapter(courses, R.layout.row));

recyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

recyclerView.setItemAnimator(new DefaultItemAnimator());
```

Gráfico 26. Activity RecyclerView



RECUERDA QUE...

- Los layouts nos permiten establecer la disposición de los componentes en la pantalla.

3. CONTROLES DE ENTRADA

BOTONES

Android nos proporciona tres tipos de botones:

- el clásico (Button)
- el de tipo on/off (ToggleButton)
- el que puede contener una imagen (ImageButton).

Button

Un control de tipo Button es el botón más básico que podemos utilizar.

En el ejemplo siguiente definimos un botón con las siguientes propiedades:

- android:text; Texto del botón.
- android:background; Color de fondo del botón.
- android:typeface; Estilo de fuente.
- android:textcolor; Color de fuente.
- android:textSize; Tamaño de fuente.

```
<Button  
    android:id="@+id/Boton1"  
    android:text="Pulsame"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@id/Label"  
    android:onClick="pulsar"/>
```

Gráfico 27. Diseño Button

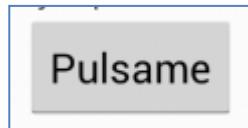


Gráfico 28. Vista Button

ToggleButton

Un control de tipo ToggleButton es un tipo de botón que puede permanecer en dos estados, On/Off.

Las siguientes propiedades se utilizan para mostrar diferentes textos:

- android:textOn; Texto para cuando el botón está en estado On.
- android:textOff; Texto para cuando el botón está en estado Off.

```
<ToggleButton  
    android:id="@+id/Boton2"  
    android:textOn="ON"  
    android:textOff="OFF"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/Boton1"  
    android:onClick="pulsar" />
```

Gráfico 29. Diseño ToggleButton



Gráfico 30. Vista ToggleButton

ImageButton

Este tipo de botones permite establecer una imagen en vez de mostrar un texto como etiqueta.

La propiedad android:src permite establecer la imagen a mostrar.

```
<ImageButton  
    android:id="@+id/Boton3"  
    android:src="@drawable/correcto"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/Boton2"  
    android:onClick="pulsar"/>
```

Gráfico 31. Diseño ImageButton



Gráfico 32. Vista ImageButton

CAJAS DE TEXTO

El control EditText es el componente de edición de texto que proporciona la plataforma Android. Permite la introducción y edición de texto por parte del usuario, por lo que en tiempo de diseño la propiedad más interesante a establecer, además de su posición/tamaño y formato, es el texto a mostrar, atributo android:text.

```
<EditText  
    android:id="@+id/name"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:hint="Recuerda:" />
```

Gráfico 33. Componente EditText

ETIQUETAS

Mediante una etiqueta podemos presentar texto en la pantalla. Sus propiedades principales son:

- android:text; texto a mostrar.
- android:background; color de fondo
- android:textColor; color del texto
- android:textSize; tamaño de la fuente
- android:typeface; estilo del texto: negrita, cursiva, ...

```
<TextView
    android:id="@+id/nombre_profe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="Large Text"
    android:layout_alignParentTop="true"
    android:layout_toEndOf="@+id/avatar_profe" />

<TextView
    android:id="@+id/perfil_profe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:layout_below="@+id/nombre_profe"
    android:layout_toEndOf="@+id/avatar_profe" />

<TextView
    android:id="@+id/experiencia_profe"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Medium Text"
    android:layout_below="@+id/perfil_profe"
    android:layout_toEndOf="@+id/avatar_profe" />
```

Gráfico 34. Componente TextView

IMAGENES

El control ImageView permite mostrar imágenes en la aplicación.

Las propiedades más interesantes son:

- android:src, que permite indicar la imagen a mostrar que debería estar en la carpeta /res/drawable.
- android:maxWidth; anchura máxima de la imagen.
- android:maxHeight; altura máxima de la imagen.

```
<ImageView  
    android:id="@+id/avatar_profe"  
    android:src="@mipmap/anonimo"  
    android:scaleType="centerCrop"  
    android:layout_marginRight="10dp"  
    android:layout_width="150dp"  
    android:layout_height="150dp" />
```

Gráfico 35. Componente ImageView

CHECKBOXES

Los componentes de tipo Check Box permiten una selección múltiple.

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:orientation="vertical" >  
  
<CheckBox  
    android:id="@+id/checkbox_cine"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="onCheckboxClicked"  
    android:text="Me gusta el cine"/>  
  
<CheckBox  
    android:id="@+id/checkbox_teatro"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="onCheckboxClicked"  
    android:text="Me gusta el teatro" />  
  
<CheckBox  
    android:id="@+id/checkbox_circo"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="onCheckboxClicked"  
    android:text="Me gusta el circo" />  
  
<CheckBox  
    android:id="@+id/checkbox_museos"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:onClick="onCheckboxClicked"  
    android:text="Me gustan los museos" />  
  
</LinearLayout>
```

Gráfico 36. Diseño de los CheckBox

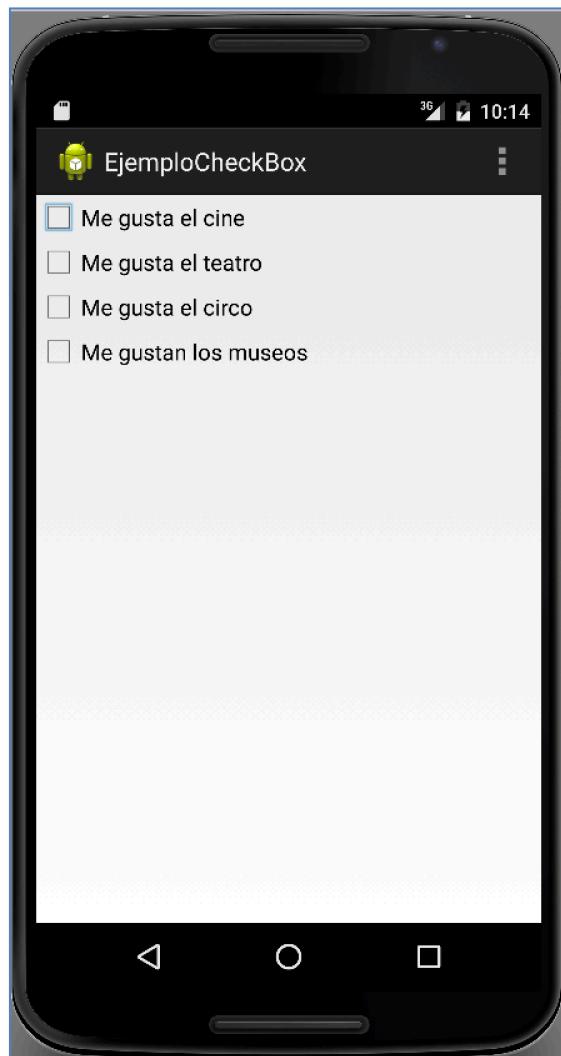


Gráfico 37. Vista de los CheckBox

RADIO BUTTONS

A diferencia de los componentes anteriores los Radio Buttons son excluyentes entre si mientras que pertenezcan al mismo RadioGroup.

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

    <RadioButton
        android:id="@+id/radio_varon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Soy un chico" />

    <RadioButton
        android:id="@+id/radio_mujer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onRadioButtonClicked"
        android:text="Soy una chica" />

</RadioGroup>
```

Gráfico 38. Diseño de los RadioButtons

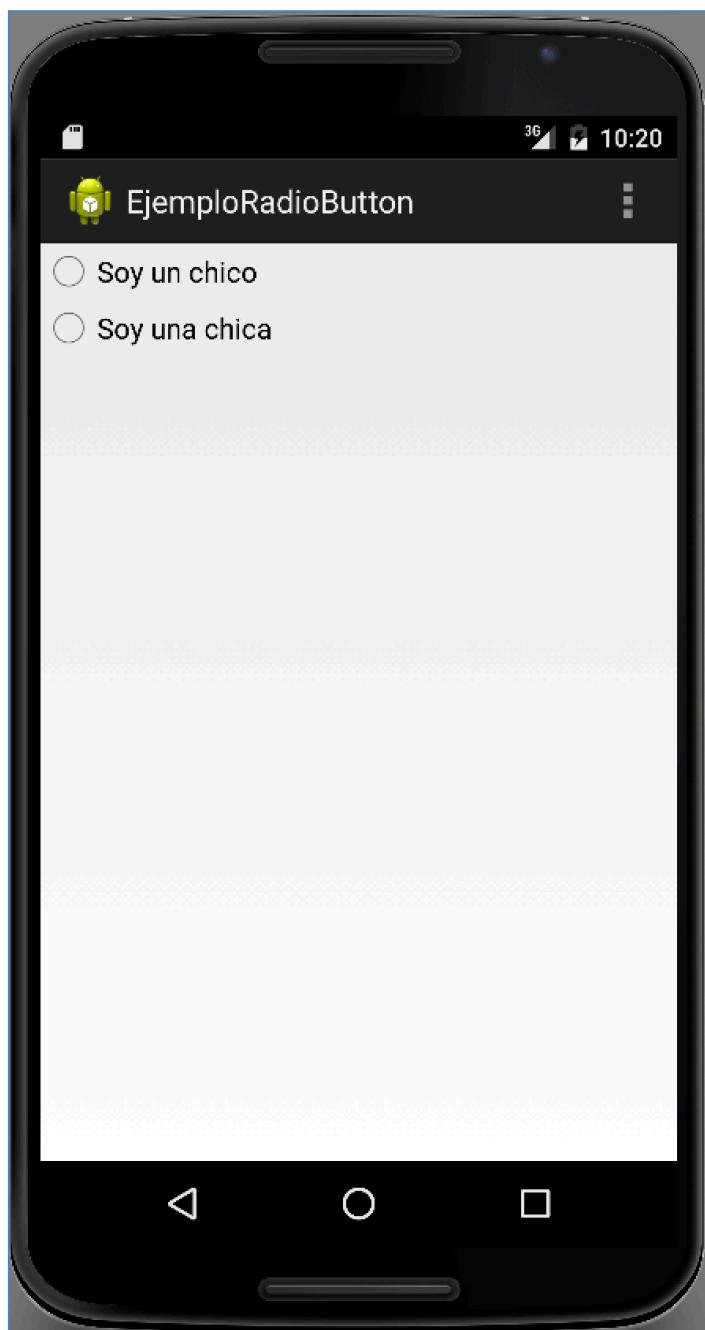


Gráfico 39. Vista de los RadioButtons

SPINNERS

Estos componentes son lo que se conocen como listas desplegables. El usuario pulsa sobre la lista y esta muestra todos sus valores permitiendo la selección de uno de sus elementos.

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

    <TextView
        android:id="@+id/LblMensaje"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Selecciona una opción:" />

    <Spinner
        android:id="@+id/CmbOpciones"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Gráfico 40. Diseño Spinner

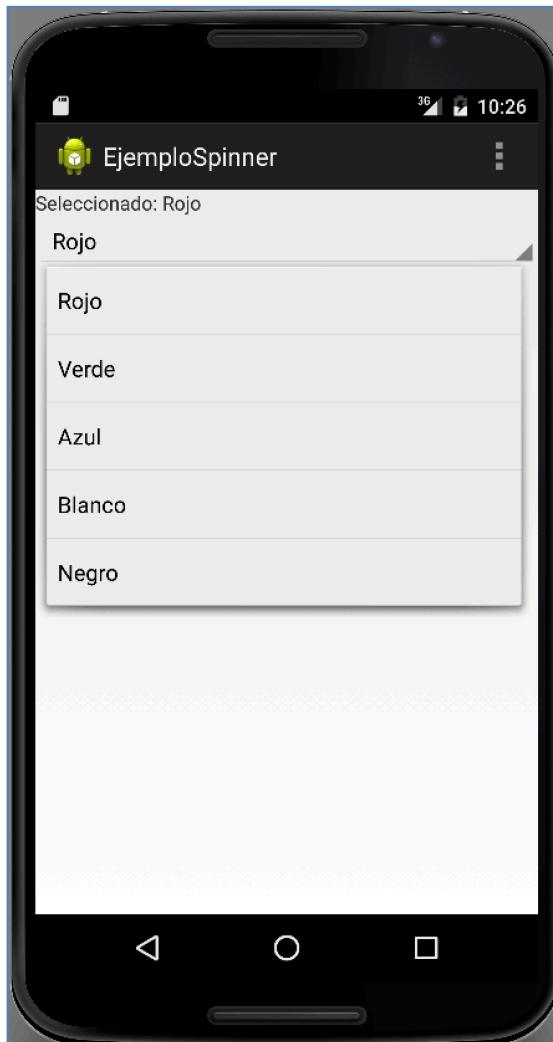


Gráfico 41. Vista de un Spinner



RECUERDA QUE . . .

- Los componentes que hemos visto son los siguientes:
 - **Button:** Botón básico
 - **ToggleButton:** Botón On/Off
 - **ImageButton:** Botón con imagen
 - **EditText:** Cajas de texto
 - **TextView:** Etiquetas
 - **ImageView:** Imágenes
 - **CheckBox:** Casillas de verificación
 - **RadioGroup:** Para agrupar varios Radio Button
 - **RadioButton:** Botones excluyentes
 - **Spinner:** Listas desplegables

4 . - EVENTOS

En Android, hay más de una manera de interceptar los eventos de la interacción del usuario con la aplicación. Al considerar los eventos dentro la interfaz de usuario, el objetivo es capturar los eventos del componente específico con el que el usuario interactúa. La clase View proporciona los medios para hacerlo.

Dentro de las distintas clases de vista que se utilizan para componer su layout, podemos observar varios métodos callback para los eventos de interfaz de usuario. Estos métodos son llamados por el framework Android cuando la acción respectiva ocurre en ese objeto.

A continuación vemos algunos métodos para procesar eventos:

- **onClick();** De la interface **View.OnClickListener**. Este método es invocado cuando el usuario pulsa un componente de la interface.
- **onLongClick();** De la interface **View.OnLongClickListener**. Este método es invocado cuando el usuario pulsa un componente de la interface durante más de un segundo.
- **onFocusChange();** De la interface **View.OnFocusChangeListener**. Este método es invocado cuando se cambia el foco de un componente a otro.
- **onKey();** De la interface **View.OnKeyListener**. Este método es invocado cuando se pulsa una tecla del dispositivo.
- **onTouch();** De la interface **View.OnTouchListener**. Este método es invocado al tocar (para dispositivos táctiles) un componente.
- **onCreateContextMenu();** De la interface **View.OnCreateContextMenuListener**. Este método se invoca al generar un menú contextual.

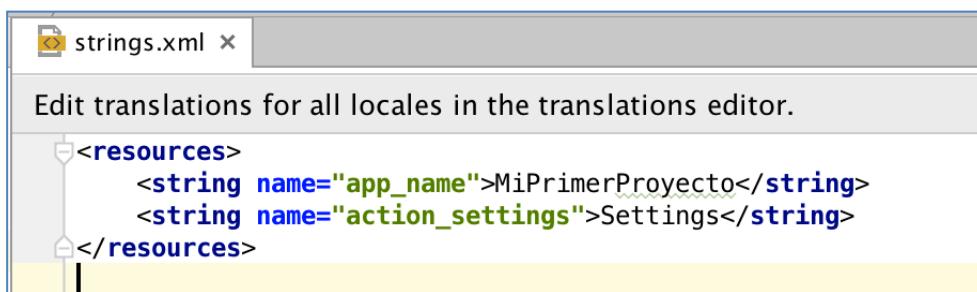
5.- GESTION DE TEXTOS

EL ARCHIVO STRINGS.XML

Todos los textos utilizados en la aplicación se definen en el archivo strings.xml.

De esta forma potenciamos su reutilización y también nos permite aplicar internacionalización.

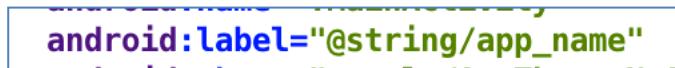
En la siguiente imagen vemos el contenido del archivo:



```
<resources>
    <string name="app_name">MiPrimerProyecto</string>
    <string name="action_settings">Settings</string>
</resources>
```

Gráfico 42. Archivo strings.xml

La forma de mostrar un texto es haciendo referencia al archivo strings.xml y a la clave del texto.



```
        android:label="@string/app_name"
```

Gráfico 43. Incluir un texto

INTERNACIONALIZACION

Si queremos que nuestra aplicación sirva los mensajes en diferentes idiomas, en función de la configuración de idioma del dispositivo, debemos hacer lo siguiente:

Pulsamos encima de la carpeta res del proyecto, con el botón derecho, y new resource directory.

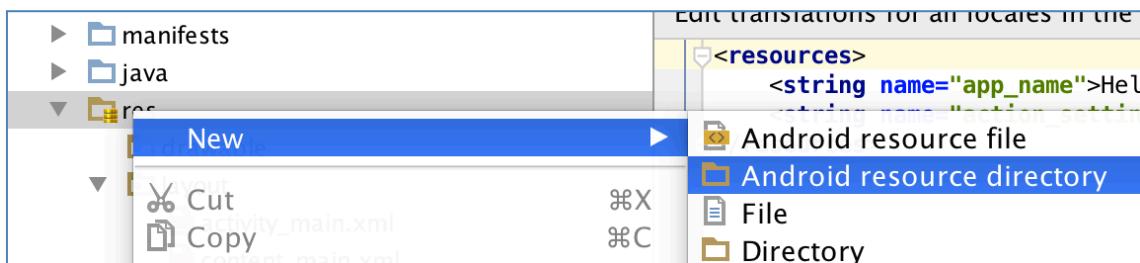


Gráfico 44. Crear un nuevo directorio

Escribimos por ejemplo values-de como nombre de carpeta y elegimos Locale.
Pulsamos el botón >>

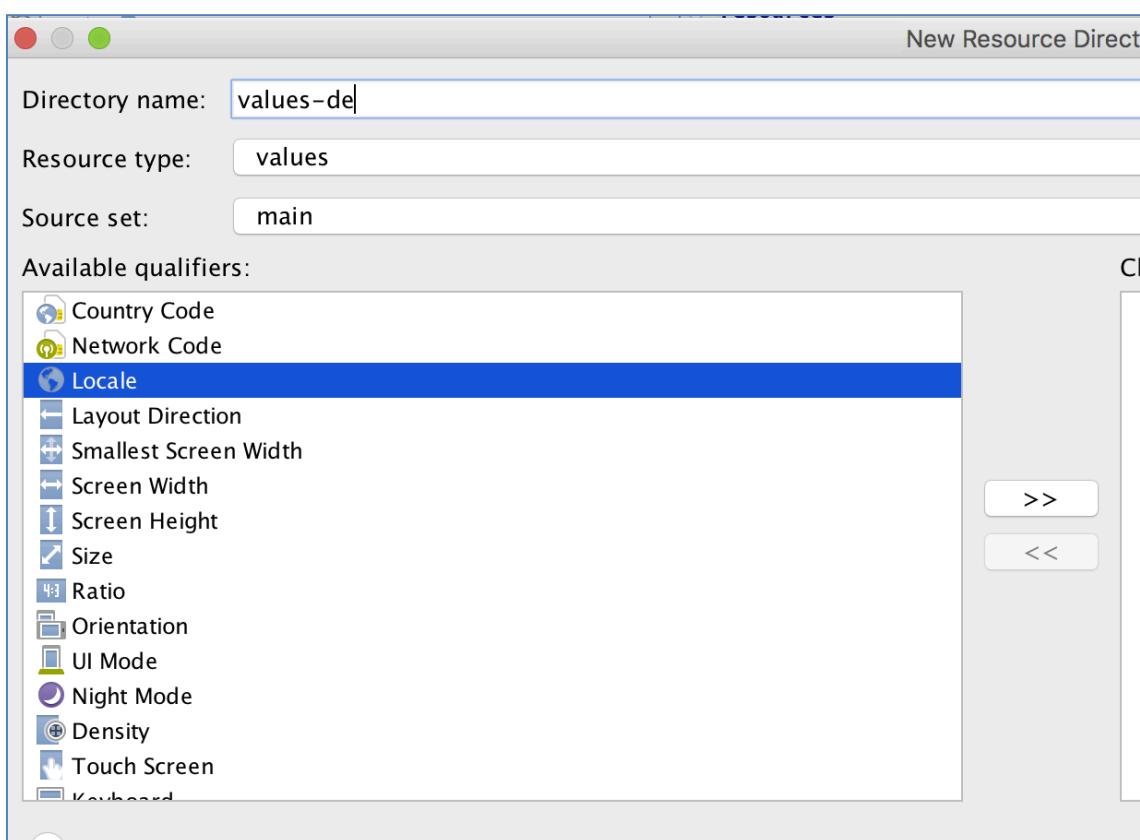


Gráfico 45. Seleccionar el Locale

Ahí elegimos el idioma alemán y la región si es necesario.

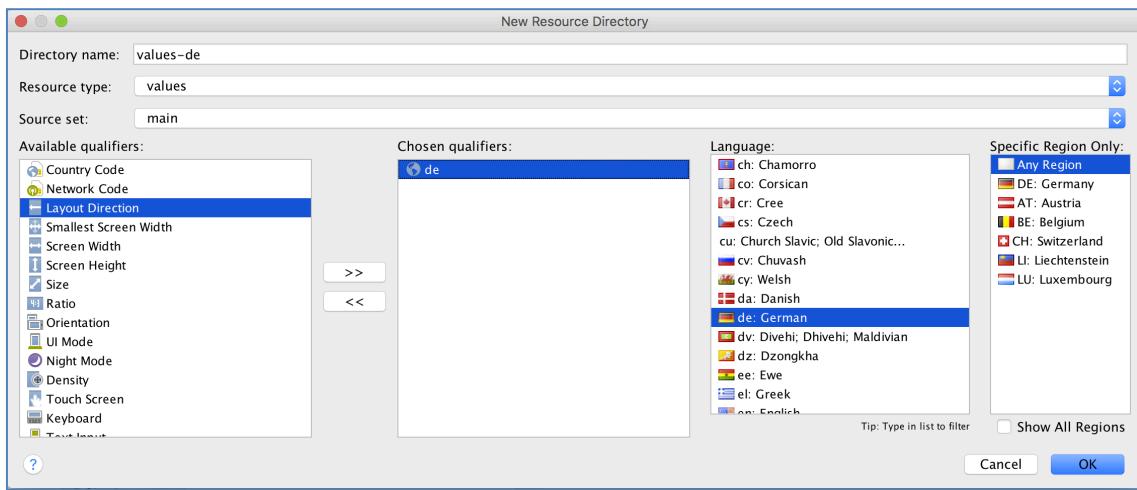


Gráfico 46. Seleccionamos el idioma aleman

Ahora nos vamos a la vista proyecto y vemos que se a creado la carpeta values-de donde copiamos el archivo strings.xml.

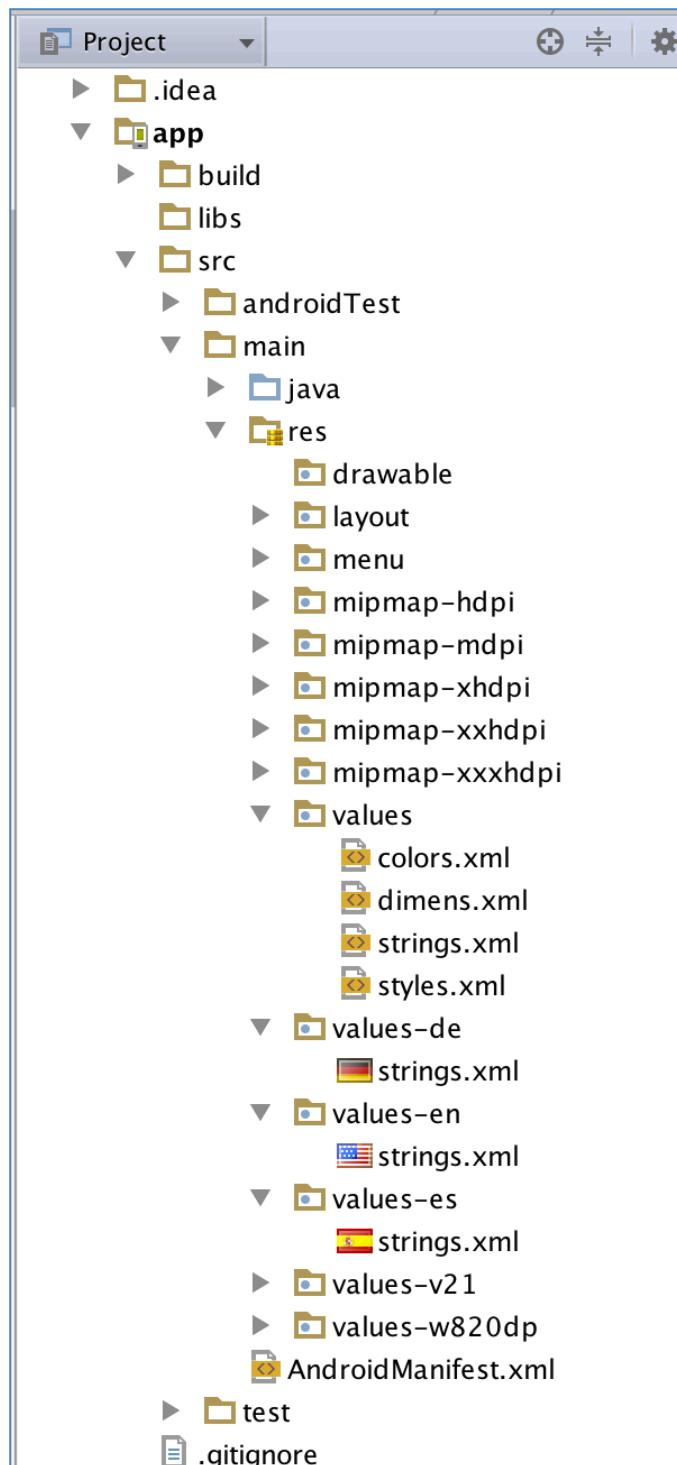


Gráfico 47. Vista del proyecto con internacionalización

Ahora solo queda editar el archivo strings.xml y traducirlo al idioma correspondiente.

6. - MATERIAL DESIGN

Material Design es un concepto, una filosofía, unas pautas enfocadas al diseño utilizado en Android. Es una guía de diseño.

Hasta la versión 4.4 (KitKat) se utilizaba Holo pero la tendencia ahora es Material Design.

Veamos como implementarlo.

COLORES

La sugerencia siempre son 3 colores: Primary, Primary Dark y Accent.

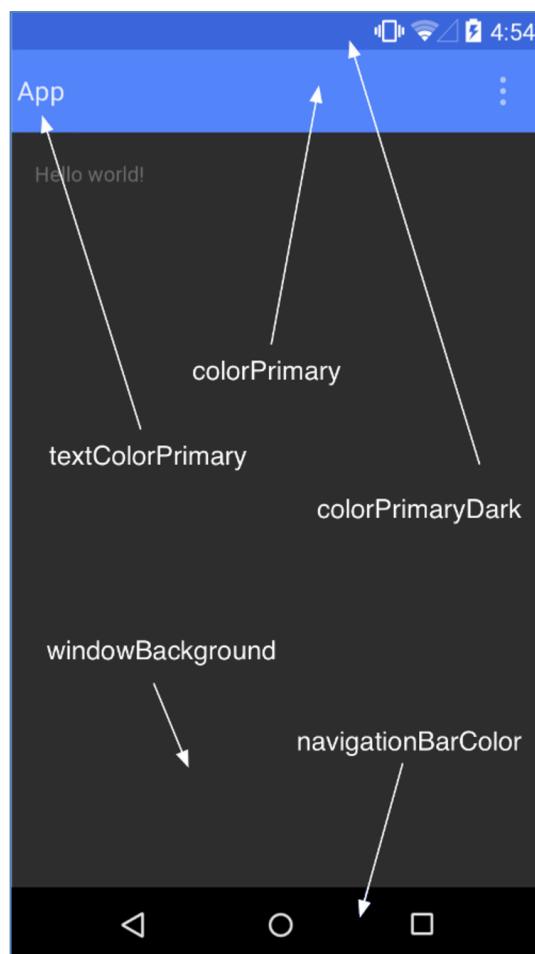


Gráfico 48. Colores con Material Design

Esos 3 colores se pueden elegir de acuerdo a las guías de diseño de material design.

Primary

Cuando se utiliza un color primario en su gama de colores, este color debe ser el más ampliamente utilizado en todas las pantallas y componentes.

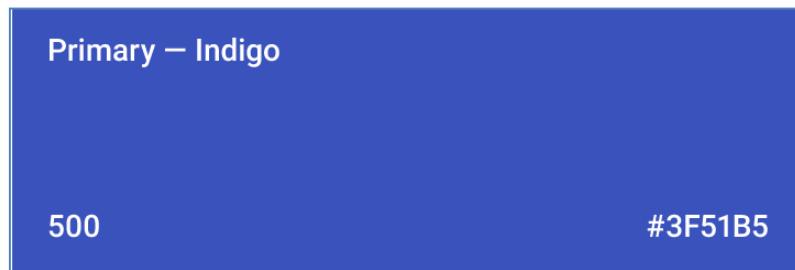


Gráfico 49. Primary Color

Primary Dark

Paletas con un color secundario pueden utilizar este color para indicar una acción o información relacionada.

El color secundario puede ser una variante más oscuro o más claro del color primario. Normalmente suele ser mas oscuro.

Como vemos en la siguiente imagen, si hemos establecido el color primario a 500, el secundario puede ser el 100 o 700 por ejemplo.

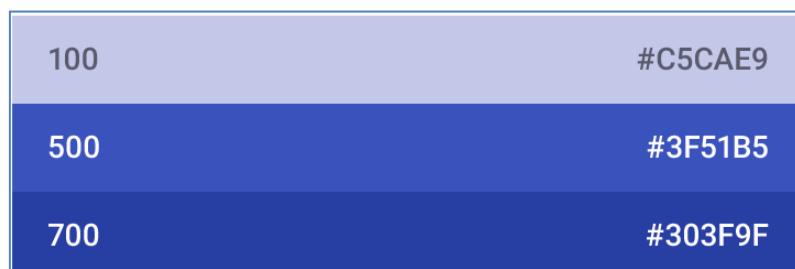


Gráfico 50. Primary Dark Color

Accent

El color accent se debe utilizar para el botón de acción flotante y elementos interactivos, tales como:

- Los campos de texto y cursores
- La selección de texto
- Las barras de progreso
- Los controles de selección, botones y controles deslizantes

Debe ser un color resultante y diferente de los otros como vemos en la imagen.



Gráfico 51. Accent Color

Una vez seleccionados, debemos agregarlos en un archivo **colors.xml** en la carpeta **values**, tal como se muestra en la siguiente imagen:

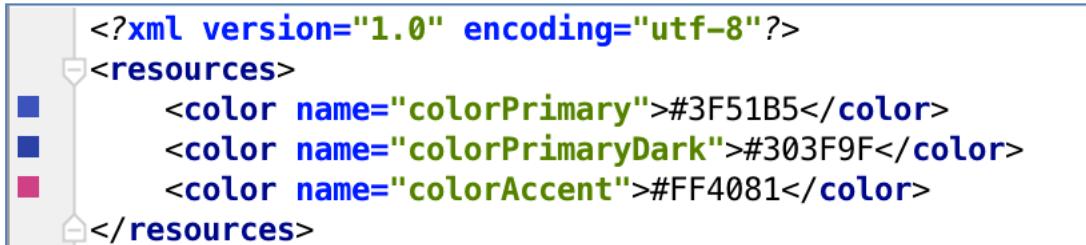


Gráfico 52. colors.xml

ESTILOS

Ahora modifiquemos styles.xml de la carpeta values, el cual tendrá como tema base AppCompat. AppCompat nos permite dentro de sus muchas propiedades:

1. Esconder el ActionBar, ya que este será sustituido por un nuevo Widget llamado Toolbar, que es el encargado de dar ese aspecto Material.

2. Agregar los colores previamente elegidos en colors.xml a toda la aplicación.



```

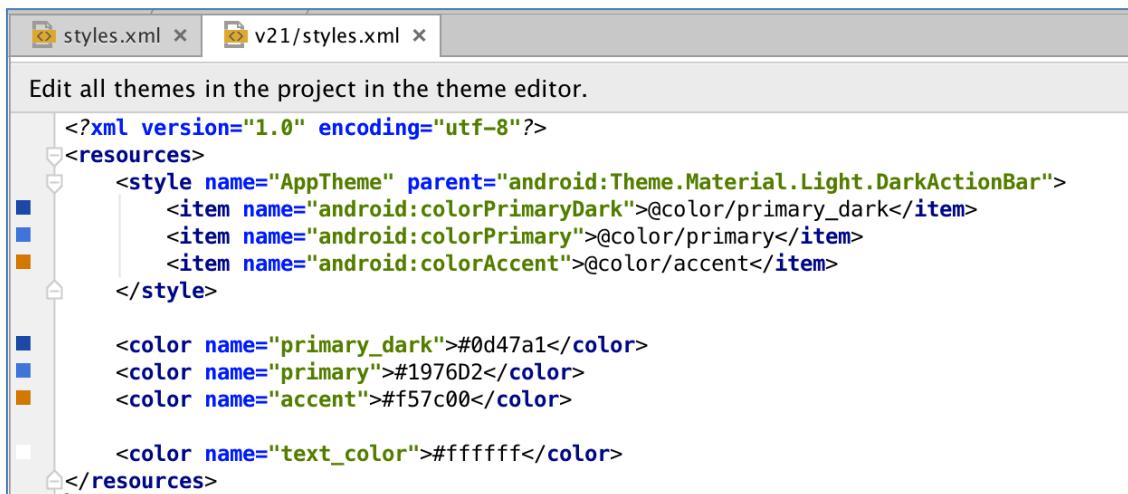
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
    <style name="AppTheme.NoActionBar">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
    </style>
    <style name="AppTheme.AppBarOverlay" parent="ThemeOverlay.AppCompat.Dark.ActionBar" />
    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />
</resources>

```

Gráfico 53. styles.xml

TEMAS

Otra opción es definir el tema directamente en el archivo styles.xml (V21) que será el utilizado a partir de la versión 21 de Android.



```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="AppTheme" parent="android:Theme.Material.Light.DarkActionBar">
        <item name="android:colorPrimaryDark">@color/primary_dark</item>
        <item name="android:colorPrimary">@color/primary</item>
        <item name="android:colorAccent">@color/accent</item>
    </style>

    <color name="primary_dark">#0d47a1</color>
    <color name="primary">#1976D2</color>
    <color name="accent">#f57c00</color>

    <color name="text_color">#ffffff</color>
</resources>

```

Gráfico 54. styles.xml (V21)

ÍNDICE DE GRÁFICOS

Gráfico 1. Diseño de la pantalla.....	3
Gráfico 2. Lógica de negocio de la pantalla	4
Gráfico 3. Diseño FrameLayout.....	5
Grafico 4. Vista FrameLayout.....	6
Grafico 5. Diseño LinearLayout.....	9
Grafico 6. Vista LinearLayout.....	10
Gráfico 7. Diseño TableLayout.....	12
Gráfico 8. Vista TableLayout	13
Gráfico 9. Diseño GridLayout	14
Gráfico 10. Vista GridLayout	15
Gráfico 11. Diseño RelativeLayout.....	17
Gráfico 12. Vista RelativeLayout.....	18
Gráfico 13. Estructura ListLayout.....	19
Gráfico 14. Diseño GridView	20
Gráfico 15. Vista GridView	21
Gráfico 16. Build.gradle	22
Gráfico 17. Layout CardView.....	23
Gráfico 18. Recursos para crear RecyclerView	24
Gráfico 19. POJO	25
Gráfico 20. Adapter	26
Gráfico 21. ViewHolder	27
Gráfico 22. Método onCreateViewHolder	28
Gráfico 23. Método onBindViewHolder.....	28
Gráfico 24. Método getItemCount.....	29
Gráfico 25. Layout RecyclerView	29
Gráfico 26. Activity RecyclerView	30
Gráfico 27. Diseño Button	31
Gráfico 28. Vista Button	31
Gráfico 29. Diseño ToggleButton.....	32
Gráfico 30. Vista ToggleButton.....	32

Gráfico 31. Diseño ImageButton	32
Gráfico 32. Vista ImageButton	33
Gráfico 33. Componente EditText.....	33
Gráfico 34. Componente TextView.....	34
Gráfico 35. Componente ImageView	35
Gráfico 36. Diseño de los CheckBox.....	35
Gráfico 37. Vista de los CheckBox.....	36
Gráfico 38. Diseño de los RadioButtons	37
Gráfico 39. Vista de los RadioButtons	38
Gráfico 40. Diseño Spinner	39
Gráfico 41. Vista de un Spinner	39
Gráfico 42. Archivo strings.xml.....	42
Gráfico 43. Incluir un texto	42
Gráfico 44. Crear un nuevo directorio	43
Gráfico 45. Seleccionar el Locale	43
Gráfico 46. Seleccionamos el idioma aleman	44
Gráfico 47. Vista del proyecto con internacionalización.....	45
Gráfico 48. Colores con Material Design	46
Gráfico 49. Primary Color	47
Gráfico 50. Primary Dark Color.....	47
Gráfico 51. Accent Color.....	48
Gráfico 52. colors.xml.....	48
Gráfico 53. styles.xml.....	49
Gráfico 54. styles.xml (V21)	49