

Diseño de User Interface

Projecte Integrat de Software (PIS)

Universitat de Barcelona

victor.campello@ub.edu

c.izquierdo@ub.edu

carlos.martinisla@ub.edu

18-20 de Febrero de 2020

1 Diseño de la aplicación - Interfaz de usuario

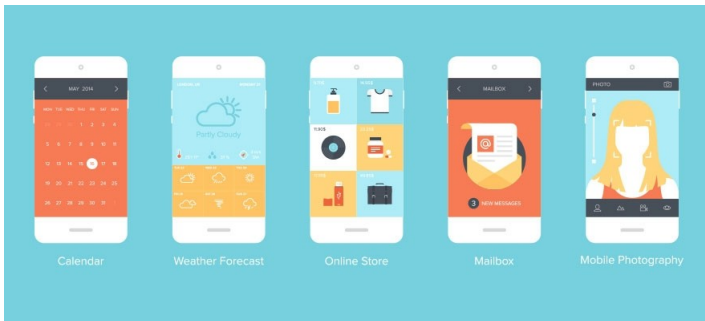
- Diseño, edición y parámetros importantes
- Layouts
- Tipos de Layouts
- Estilos y temas

2 Intents

- Ciclo de transmisión de los Intents
- Cómo crear un Intent

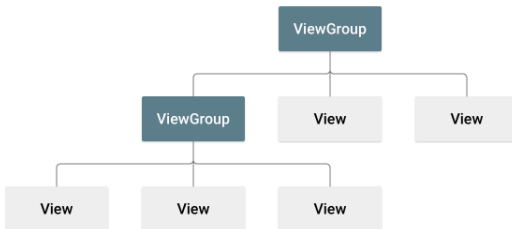
Interfaz

La interfaz de usuario de tu app es todo aquello que el usuario puede ver y con lo que puede interactuar. Android ofrece una variedad de componentes de IU previamente compilados, como objetos de diseño estructurados y controles de IU que te permiten compilar la interfaz gráfica de usuario para tu app.



La estructura de la interfaz usa la siguiente jerarquía de objetos:

- **ViewGroup**: Contenedor invisible que define la estructura donde estarán contenidos otros **ViewGroup** o los **View**. Se denominan "diseños" o "layouts".
- **View**: Principal clase que define todo aquello con lo que el usuario puede interactuar. Se les denomina "widget".



Para editar los layout de tu app, existen dos métodos:

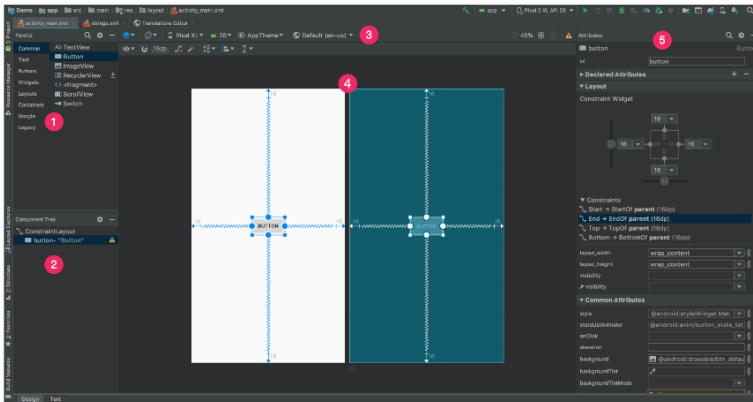
- Con código XML definiendo las diferentes instancias de tu diseño

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

- *Layout editor*, un editor para diseñar de forma interactiva con generación automática del código XML.

Layout Editor

- 1 **Palette:** Es la lista de vistas y grupos de vistas que puedes arrastrar a tu diseño.
- 2 **Component Tree:** Corresponde a la jerarquía de vistas para tu diseño.
- 3 **Toolbar:** Tiene los botones para configurar la apariencia de tu diseño en el editor y cambiar algunos atributos de diseño.
- 4 **Design editor:** Corresponde al diseño en la vista Design o Blueprint, o ambas.
- 5 **Attributes:** Contiene controles para los atributos de las vistas seleccionadas.



Parámetros importantes en layouts

En los diferentes layouts definidos, existen parámetros básicos que debéis conocer:

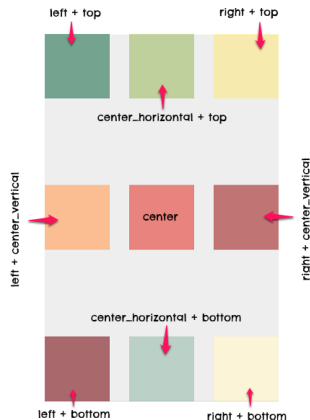
- **android:id="@+id/name"** Identificador único para cada elemento. Si añadimos @ significa parser interno de XML. Si añadimos +. significa que el id no está creado aún, y lo crea automáticamente en el archivo R.strings.
- **layout_width, layout_height** Definen el ancho y la altura de cualquier view. Si usamos "wrap_content", el tamaño se ajusta a lo mínimo para una visualización correcta. Si, en cambio, usamos "match_parent", se ajusta al máximo permitido por el padre ("layout")

Tipos de Layouts - Frame layout

Frame Layout: Está pensado para mostrar solo un elemento. Aún así, se puede elaborar una especie de cuadrícula. En la siguiente diapositiva veremos un ejemplo.

Para el *FrameLayout*, se puede usar el término **gravity** para alinear las *FrameLayout* hijos como se desee.

```
android:layout_gravity="right|bottom"
```



Frame layout ejemplo

```
<FrameLayout 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"
    tools:context=".ActividadPrincipal">

    <Button
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:text="Saltar"
        android:id="@+id/boton_saltar"
        android:layout_gravity="center_horizontal|bottom"/>

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/imagen_background"
        android:layout_gravity="top|center"
        android:src="@drawable/background_frame_layout"
        android:scaleType="centerCrop" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imagen_estadistica"
        android:layout_gravity="center"
        android:src="@drawable/ejemplo_estadistica"
        android:padding="16dp" />

</FrameLayout>
```

Ejercicio Frame layout

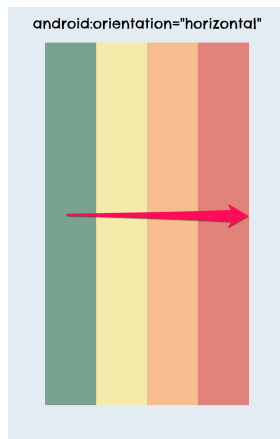
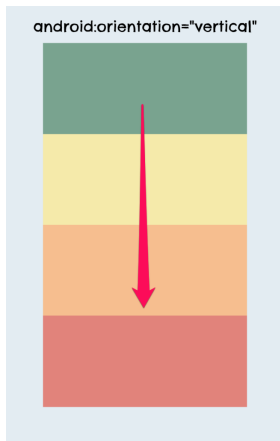
Diseña un layout del estilo frame, con los siguientes elementos:

- 4 TextView indicando Norte, Sur, Este y Oeste.
- Un boton central con el Texto: Brujula.

Trata de cambiar el color del texto de cada TextView.

Tipos de Layout - Linear Layout

Linear layout distribuye todos los contenidos (Views) dentro del ViewGroup a lo largo de la dimensión establecida. La orientación la puedes escoger a través del atributo **android:orientation**.



Linear Layout

También existe la opción **android:layout_weight** que te permite definir la importancia de cada View dentro de la interfície.



El parametro weight se define a través de un número entero **Int**.

```
android:weightSum="6"
```

El número te permite saber si la división de los distintos pesos.

Linear Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="48dp">

    <TextView
        android:id="@+id/texto_conectar"
        android:layout_width="wrap_content"
        android:layout_height="60dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:text="Conectar"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/input_usuario"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:hint="Correo" />

    <EditText
        android:id="@+id/input_contrasena"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:ems="10"
        android:hint="Contraseña"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/boton_iniciar_sesion"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:text="Iniciar Sesión" />

    <TextView
        android:id="@+id/texto_olvidaste_contrasena"
        android:layout_width="wrap_content"
        android:layout_height="60dp"
        android:layout_gravity="center_horizontal"
        android:layout_weight="1"
        android:gravity="center_vertical"
        android:text="¿Olvidaste tu contraseña?"
        android:textAppearance="?android:attr/textAppearanceMedium"
        android:textColor="#0E8AEE" />

</LinearLayout>
```



Ejercicio de Linear Layout

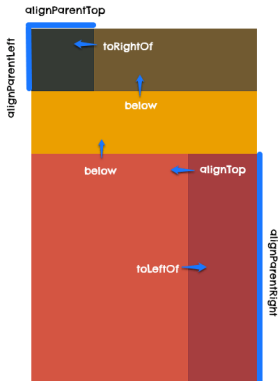
Crea un Linear Layout que contenga:

- 1 Dos botones
- 2 Dos entradas de texto.
- 3 Un TextView en la parte superior de color rojo.
- 4 Un TextView centrado en la parte inferior.
- 5 El tema debe ser oscuro.

Los botones deben tener sus respectivos nombres, etc.

Tipos de layout - Relative layout

Relative layout es el elemento más flexible de todos. Permite una distribución más avanzada y viene determinada en función de como declaramos la posición relativa de los View con respecto a otros. Los parámetros para declarar la posición relativa vienen determinados de la siguiente manera:



- **android:layout_above**: Posiciona el borde inferior del elemento actual con el borde superior del View declarado con la id.
- **android:layout_centerHorizontal**: Usa true para indicar que el view será centrado horizontalmente con respecto al padre.
- **android:layout_alignParentBottom**: Usa true para alinear el borde inferior de este View con el borde inferior del padre.
- **android:layout_alignStart**: Alinea el borde inicial de este elemento con el borde inicial del view referido por id.

Relative layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">

    <EditText
        android:id="@+id/input_nombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:ems="10"
        android:hint="Nombres"
        android:inputType="textPersonName" />

    <EditText
        android:id="@+id/input_apellido"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/input_nombre"
        android:ems="10"
        android:hint="Apellidos"
        android:inputType="textPersonName" />

    <TextView
        android:id="@+id/texto_estado_civil"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/input_apellido"
        android:layout_marginRight="48dp"
        android:paddingBottom="8dp"
        android:paddingTop="16dp"
        android:text="Estado civil"
        android:textAppearance="?android:attr/textAppearanceMedium" />
```

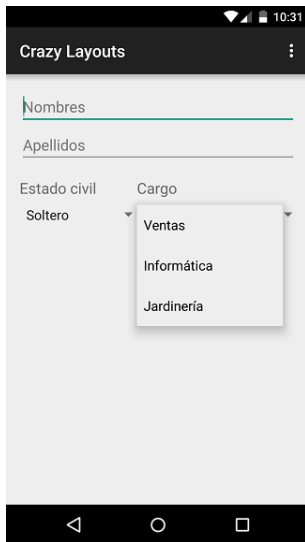
```
<Spinner
    android:id="@+id/spinner_estado_civil"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/texto_estado_civil"
    android:layout_toLeftOf="@+id/spinner_cargo"
    android:entries="@array/lista_estado_civil" />

<TextView
    android:id="@+id/texto_cargo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/input_apellido"
    android:layout_centerHorizontal="true"
    android:layout_toRightOf="@+id/texto_estado_civil"
    android:paddingBottom="8dp"
    android:paddingTop="16dp"
    android:text="Cargo"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<Spinner
    android:id="@+id/spinner_cargo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/texto_cargo"
    android:layout_alignParentRight="true"
    android:layout_alignStart="@+id/texto_cargo"
    android:layout_below="@+id/texto_cargo"
    android:entries="@array/lista_cargo" />

</RelativeLayout>
```

Relative layout



Ejercicio de Relative Layout

Crea un Relative Layout que contenga:

- 1 Dos botones uno al lado del otro.
- 2 Un spinner en la esquina superior derecha.
- 3 Un TextView centrado en la parte inferior.

Los botones deben tener sus respectivos nombres, etc.

Tipos de layouts - Table layout

Tipo de disposición que sirve para crear tablas estilo "excel". Diseño secuencial, hay que agruparlo en ramas de código usando `<Table Row>`.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:stretchColumns="1">

    <TableRow android:background="#128675">

        <TextView
            android:layout_column="0"
            android:padding="3dip"
            android:text="Producto"
            android:textColor="@android:color/white" />

        <TextView
            android:layout_column="2"
            android:padding="3dip"
            android:text="Subtotal"
            android:textColor="@android:color/white" />
    </TableRow>

    <TableRow>

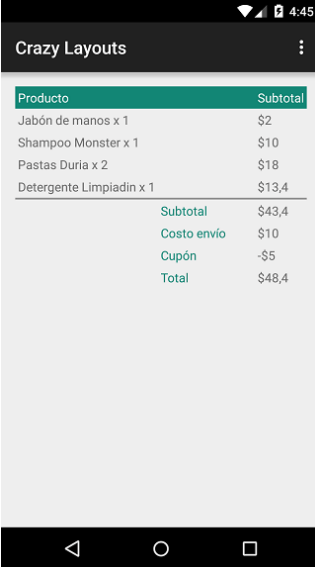
        <TextView
            android:layout_column="0"
            android:padding="3dip"
            android:text="Jabón de manos x 1" />

        <TextView
            android:layout_column="2"
            android:gravity="left"
            android:padding="3dip"
            android:text="$2" />
    </TableRow>

</TableLayout>
```

- El parámetro **android:layout_column** asigna la columna a la que pertenece cada View declarada. También se puede usar el parámetro **weight**.

Table layout



The screenshot shows a mobile application interface with a dark header bar containing the title 'Crazy Layouts' and a menu icon. Below the header is a table with two columns: 'Producto' and 'Subtotal'. The table lists four items: 'Jabón de manos x 1' for \$2, 'Shampoo Monster x 1' for \$10, 'Pastas Duria x 2' for \$18, and 'Detergente Limpiadin x 1' for \$13,4. A horizontal line separates this list from a summary section. The summary section includes 'Subtotal' (\$43,4), 'Costo envío' (\$10), 'Cupón' (-\$5), and 'Total' (\$48,4). The bottom of the screen shows a standard Android navigation bar.

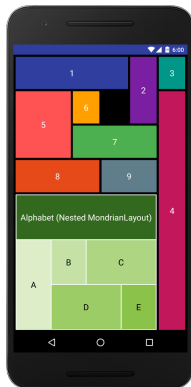
Producto	Subtotal
Jabón de manos x 1	\$2
Shampoo Monster x 1	\$10
Pastas Duria x 2	\$18
Detergente Limpiadin x 1	\$13,4
<hr/>	
Subtotal	\$43,4
Costo envío	\$10
Cupón	-\$5
Total	\$48,4

Ejercicio Table layout

Crea una tabla con distintas columnas y elementos. Estilo inventario.

Tipos de layout - GridLayout

Se distribuyen los elementos de forma tabular, en filas y columnas. A diferencia de Table layout, aquí hay que declarar el número de filas y de columnas como propiedades del layout, mediante **android:rowCount** y **android:columnCount**. De este modo, los elementos se van ordenando uno a uno (No hay que declarar `<Table Row>` como hacíamos en Table Layout).



Grid Layout

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:alignmentMode="alignBounds"
    android:columnCount="4"
    android:rowCount="4">

    <TextView
        android:id="@+id/numero_7"
        style="@style/AppTheme.BotonCalculadora.Azul"
        android:layout_column="0"
        android:layout_row="0"
        android:text="7" />

    <TextView
        android:id="@+id/numero_8"
        style="@style/AppTheme.BotonCalculadora.Azul"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:layout_row="0"
        android:text="8" />

    <TextView
        android:id="@+id/numero_9"
        style="@style/AppTheme.BotonCalculadora.Azul"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_column="2"
        android:layout_row="0"
        android:text="9" />
```

- Cada índice marca una posición en el grid
- Para textos sencillos, no es necesario declarar una string.

Ejercicio de Grid Layout

Crea una interficie que represente una calculadora. Los botones deben tener un color naranja predeterminado. Los simbolos operacionales de azul. Asigna a cada boton su respectivo valor. Asume un TextView que permita visualizar las entradas de números.

Action Bar

Para agregar una barra de actividad, hay que comprobar primero si existe la librería de compatibilidad

```
class MyActivity : AppCompatActivity() {  
    // ...  
}
```

Hay que evitar que por defecto, el layout use la barra de acción por defecto. En el manifest hay que modificar esto y añadir la Toolbar en el diseño de la actividad.

```
<application  
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"  
/>
```

```
<android.support.v7.widget.Toolbar  
    android:id="@+id/my_toolbar"  
    android:layout_width="match_parent"  
    android:layout_height="?attr/actionBarSize"  
    android:background="?attr/colorPrimary"  
    android:elevation="4dp"  
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"  
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light" />
```

Ejercicio Layouts

A tu Frame layout creado anteriormente, añade una Action Bar cómo se muestra. Trata de modificarle el color, los tamaños y añadir un spinner desplegable.

- **Estilo** se define como una colección de atributos de apariencia en una sola View (una actividad concreta)
- **Tema** es un estilo aplicado a toda la aplicación y a todas sus vistas.

A continuación se muestra como crear y aplicar un estilo.

Crear un estilo

Para crear un nuevo estilo o tema, abre el archivo `res/values/styles.xml` de tu proyecto. Para cada estilo que desees crear, sigue estos pasos:

1. Agrega un elemento `<style>` con un nombre que identifique el estilo de forma exclusiva.
2. Agrega un elemento `<item>` para cada atributo de estilo que quieras definir.

El `name` en cada elemento especifica un atributo que de otro modo usarías como un atributo XML en tu diseño. El valor del elemento `<item>` es el valor de ese atributo.

Por ejemplo, si defines el siguiente estilo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="GreenText" parent="TextAppearance.AppCompat">
    <item name="android:textColor">#00FF00</item>
  </style>
</resources>
```

Puedes aplicar el estilo a una vista de la siguiente manera:

```
<TextView
  style="@style/GreenText"
  ... />
```

Aplicar un tema

Puedes crear un tema de la misma manera en que creas estilos. La diferencia es cómo lo aplicas: en vez de aplicar un estilo con el atributo `style` en una vista, aplica un tema con el atributo `android:theme` en la etiqueta `<application>` o en una etiqueta `<activity>` en el archivo `AndroidManifest.xml`.

Por ejemplo, aquí se muestra cómo aplicar el tema "oscuro" de material design de la biblioteca de compatibilidad de Android a toda la app:

```
<manifest ... >
  <application android:theme="@style/Theme.AppCompat" ... >
  </application>
</manifest>
```

Aquí se muestra cómo aplicar el tema "claro" a una sola actividad:

```
<manifest ... >
  <application ... >
    <activity android:theme="@style/Theme.AppCompat.Light" ... >
    </activity>
  </application>
</manifest>
```

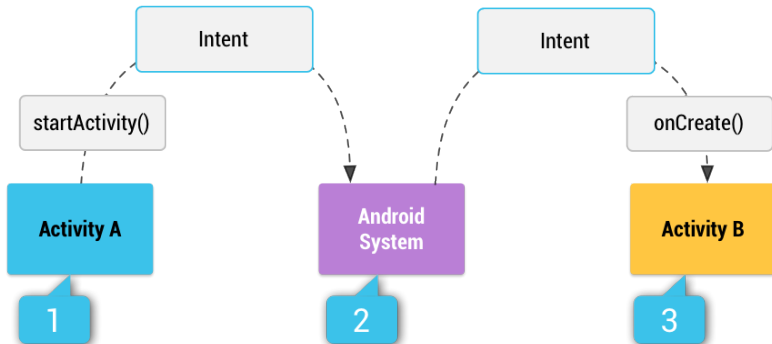
Una **Intent** es un objeto de mensajería que puedes usar para solicitar una acción de otro componente de una app.

Dos tipos de *Intents*:

- **Intents explícitas:** Especifican que aplicación administra (recibe) el intent. Se usan normalmente para iniciar actividades dentro de una misma app, por el simple hecho que en el propio intent se puede determinar con el nombre(conocido) de la clase o actividad que llama.
- **Intents implícitas:** No nombran ningún componente o clase en específico, así puede usarse (interpretarse) por otras apps del sistema. Ejemplo: cuando de una app necesitas abrir el mapa, el intent se traslada a otra aplicación que permita abrir mapas. En este caso, al ser implícita, Google Maps puede leer la información y ejecutar la acción.

Intent específico vs Intent genérico

Intent - Ciclo de transmisión



Cómo crear un Intent

Un objeto **Intent** tiene información que el sistema Android usa para determinar qué componente debe iniciar (como el nombre exacto del componente o la categoría que debe recibir la intent), además de información que el componente receptor usa para realizar la acción correctamente (por ejemplo, la acción que debe efectuar y los datos en los que debe actuar).

Los campos a crear para definir una intent són los siguientes:

- ➊ **Nombre del componente.** Componente que se debe iniciar.
Opcional. Lo que define una intent como explícita.
- ➋ **Acción** String que define la acción genérica a realizar. (View, Send, etc)
- ➌ **Datos** Tipo de datos a manejar por la aplicación que genera y/o recibe
- ➍ **Categoría** String con información sobre los datos a manejar.

Existe información extra que puede ser asociada a un Intent. Para más información, podéis consultar este enlace: <https://developer.android.com/guide/components/intents-filters>

Ejemplos de Intents

1 Intent explícita

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
val downloadIntent = Intent(this, DownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(downloadIntent)
```

2 Intent implícita

```
// Create the text message with a string
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

Selector de apps

Cuando creas un intent implícito, necesariamente pueden existir varias aplicaciones que puedan ejecutar el Intent. Para ello, a veces requerimos de diseñar un selector que nos permita escoger la aplicación con la que queremos interactuar.

```
val sendIntent = Intent(Intent.ACTION_SEND)
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
val title: String = resources.getString(R.string.chooser_title)
// Create intent to show the chooser dialog
val chooser: Intent = Intent.createChooser(sendIntent, title)

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(chooser)
}
```

Para mostrar el diálogo de selección, crea una Intent usando `createChooser()` y transfírela a `startActivity()`, tal como se muestra en el siguiente ejemplo. Aquí se muestra un diálogo con una lista de apps que responden a la intent transferida al método `createChooser()`, con el texto proporcionado como título del diálogo.

En el proceso de recibir una Intent, debes definir que parámetros ha de tener para poder recibir en una actividad determinada. Para esto se diseñan filtros, con un elemento `<intent-filter>` declarados en el `<manifest.xml>`. Un componente de aplicación debe declarar filtros independientes para cada tarea única que puede hacer. Se deben especificar estos elementos:

- `<action>`
- `<data>`
- `<category>`

Ejemplo de filtros

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name="ShareActivity">
    <!-- This activity handles "SEND" actions with text data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
    <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <action android:name="android.intent.action.SEND_MULTIPLE" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="application/vnd.google.panorama360+jpg" />
        <data android:mimeType="image/*" />
        <data android:mimeType="video/*" />
    </intent-filter>
</activity>
```

Diseñar en Android Studio las diferentes interfícies que propusisteis en vuestro individual mock-up.

End