Capitulo 2

UD02. INTERFACE DE USUARIO. CICLOS DE VIDA

Resultados de avaliación

RA1. Aplica tecnoloxías de desenvolvemento para dispositivos móbiles, e avalía as súas características e as súas capacidades.

RA2. Desenvolve aplicacións para dispositivos móbiles, para o que analiza e emprega as tecnoloxías e as librarías específicas.

Criterios de avaliación

- CA1.6 Describíronse os perfís que establecen a relación entre o dispositivo e a aplicación.
- CA1.7 Analizouse a estrutura de aplicacións existentes para dispositivos móbiles, e identificáronse as clases utilizadas.
- CA1.8 Realizáronse modificacións sobre aplicacións existentes.
- CA1.9 Utilizáronse emuladores para comprobar o funcionamento das aplicacións.
- CA2.1 Xerouse a estrutura de clases necesaria para a aplicación.
- CA2.2 Identificáronse as características das interfaces de usuario para dispositivos móbiles e técnicas específicas para o seu desenvolvemento e a súa adaptación.
- CA2.3 Analizáronse e utilizáronse as clases que modelan ventás, menús, alertas e controis para o desenvolvemento de aplicacións gráficas sinxelas.
- CA2.4 Utilizáronse as clases necesarias para a conexión e a comunicación con dispositivos sen fíos.
- CA2.5 Utilizáronse as clases necesarias para o intercambio de mensaxes de texto e multimedia.
- CA2.8 Realizáronse probas de interacción entre o usuario e a aplicación para mellorar as aplicacións desenvolvidas a partir de emuladores.
- CA2.9 Empaquetáronse e despregáronse as aplicacións desenvolvidas en dispositivos móbiles reais.
- CA2.10 Documentáronse os procesos necesarios para o desenvolvemento das aplicacións.

BC1. Análise de tecnoloxías para desenvolvemento de aplicacións en dispositivos móbiles.

- Estrutura dunha aplicación para dispositivo móbil.
- Modificación de aplicacións existentes.
- Uso do contorno de execución do administrador de aplicacións.
- Contornos integrados de traballo.
- Módulos para o desenvolvemento de aplicacións móbiles.
- Emuladores.
- Perfís: características, arquitectura e requisitos. Dispositivos soportados.

- Ciclo de vida dunha aplicación: descubrimento, instalación, execución, actualización e borrado.
- Ferramentas e fases de construción.
- Eventos da interface.
- Probas de interacción.
- Empaquetaxe e distribución.
- Documentación do desenvolvemento das aplicacións.
- Estrutura de clases dunha aplicación.
- Interfaces de usuario. Clases asociadas.
- Comunicacións: clases asociadas. Tipos de conexións

Última actualización: 08.02.2024

SUBSECCIONES DE UD02. INTERFACE DE USUARIO. CICLOS DE VIDA

Capitulo 1

LINEAR LAYOUT

LINEAR LAYOUT

`LinearLayout` es un tipo de diseño (layout) en Android que se utiliza para organizar elementos de interfaz de usuario en una vista en **línea recta**, ya sea horizontal o verticalmente. Es parte del sistema de diseño de Android y se utiliza para crear interfaces de usuario **simples** y **lineales**.

Un `LinearLayout` organiza sus elementos secuencialmente **uno después del otro** en el orden en que se agregan al diseño. Puedes controlar si los elementos se colocan horizontalmente o verticalmente utilizando la propiedad `android:orientation` en el archivo de diseño XML. Puedes establecer `android:orientation` en “horizontal” para que los elementos se alineen de izquierda a derecha o en “vertical” para que se alineen de arriba a abajo.

Aquí hay un ejemplo de un `LinearLayout` en XML con orientación vertical:

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

    <!-- Aquí puedes agregar elementos de interfaz de usuario, como botones, etiquetas, etc. -->

</LinearLayout>
```

Dentro de este `LinearLayout`, puedes agregar elementos como botones, etiquetas, campos de texto u otros elementos de interfaz de usuario que se organizarán verticalmente uno debajo

del otro. Puedes personalizar las propiedades de diseño, como márgenes, pesos, alineación, etc., para controlar la apariencia exacta de los elementos en el diseño.

Última actualización: 08.02.2024

SUBSECCIONES DE LINEAR LAYOUT

CREAR PROYECTO

Para ver cómo funcionan los ciclos de vida en Android vamos a generar un proyecto sobre el que veremos los distintos estados de forma práctica. Así, lo primero que haremos será generar nuestro protecyo en Android Studio siguiendo los siguientes pasos:

1. **Abrir Android Studio**: Abre Android Studio en tu ordenador.
2. **Crear un Nuevo Proyecto**:
    - Selecciona “Start a new Android Studio project” en la pantalla de inicio.
    - Elige “Phone and Tablet” como tipo de dispositivo.
    - Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.
3. **Configuración del Proyecto**:
    - En la siguiente pantalla, completa la información básica sobre el proyecto:
        - **Name**: Ingresa “UF1_UD2_1_LinearLayout”.
        - **Package name**: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.
        - **Save location**: Elige la ubicación donde deseas guardar el proyecto en tu sistema.
        - **Language**: Selecciona “Kotlin” como lenguaje de programación.
        - **Minimum API level**: Selecciona “API 24: Android 7.0 (Nougat)” como SDK mínimo.
4. **Finalizar Configuración**:
    - Revisa la configuración y ajusta cualquier otra opción según tus preferencias.
    - Haz clic en “Finish” para crear el proyecto.

Android Studio generará automáticamente la estructura básica del proyecto, incluyendo los archivos necesarios para la actividad principal que has creado. Puedes comenzar a desarrollar tu aplicación agregando código a la actividad `MainActivity.kt` y diseñando la interfaz de usuario en el archivo de diseño correspondiente.

Última actualización: 08.02.2024

DISEÑO

Elimanos todo el contenido del archivo `activity_main.xml`, que define el diseño de nuestra aplicación, y generamos un nuevo `LinearLayout` con las siguientes configuraciones:

- Utilizamos autocompletado para el atributo `android:xmlns`.
- Utilizamos autocompletado para el atributo `xmlns:tools`.

- Establecemos el **ancho** y **alto** para que se ajusten a las dimensiones del contenedor principal.
- Seleccionamos una orientación **vertical**. Es importante tener en cuenta que si tuviéramos una orientación horizontal, los elementos se distribuirían de derecha a izquierda o de izquierda a derecha, dependiendo de la configuración del idioma. Por ejemplo, si el idioma por defecto fuera el árabe, los componentes se distribuirán de izquierda a derecha. Esto se puede especificar en el archivo `AndroidManifest.xml` con la propiedad:

  ```
  android:supportsRtl="true"
  ```

  Esta propiedad indica si admite Right to Left (es decir, una orientación horizontal de derecha a izquierda).

- Añadimos el **contexto** para asociar este layout con la actividad desde la que se va a lanzar. Este paso no es obligatorio, pero puede ser útil para determinadas herramientas. En la actividad, el diseño ya está asociado.
- Añadimos un **margen** de 16dp.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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:orientation="vertical"
    tools:context=".MainActivity"
    android:padding="16dp">
```

Última actualización: 08.02.2024

COMPONENTES

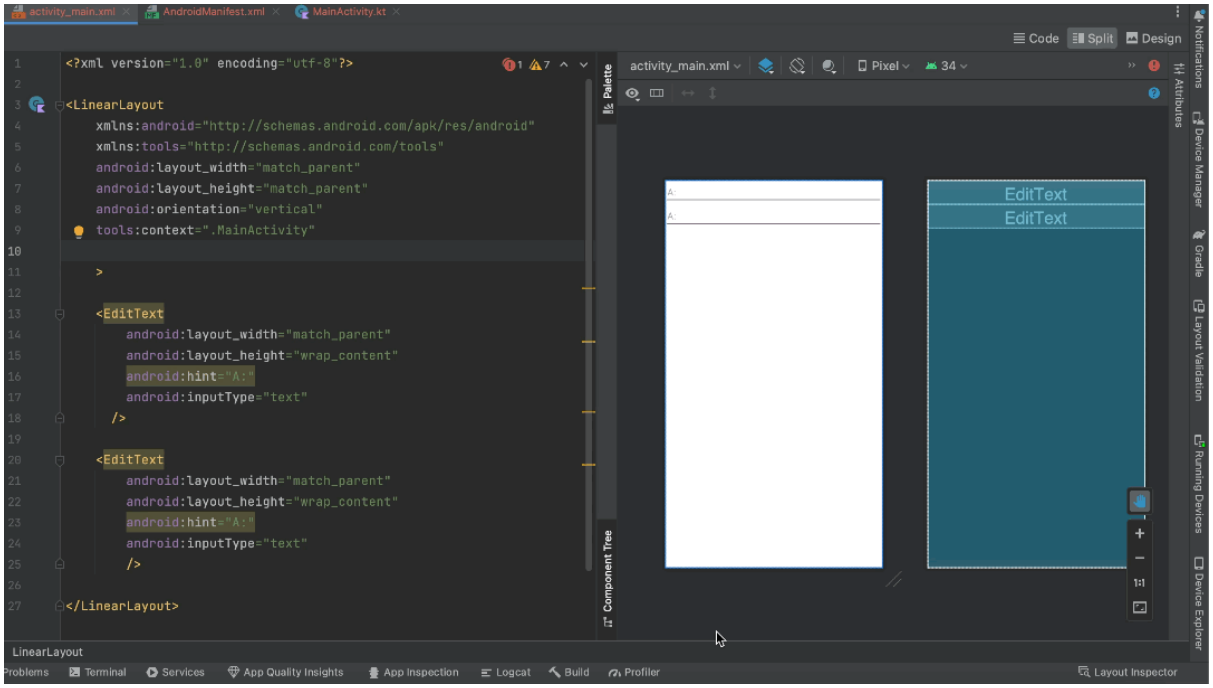
Agregar Campos de Texto

Campo de Texto Destinatario

Para crear un campo de texto destinatario, seguiremos estas configuraciones:

- El **ancho** se ajusta automáticamente al tamaño del elemento padre, que a su vez está configurado para adaptarse al tamaño de la pantalla.
- La **altura** se ajustará dinámicamente según el contenido ingresado.
- Utilice el atributo `android:hint` para proporcionar una pista o mensaje de ejemplo en el campo de texto. Por ejemplo: `android:hint="A:"` (donde “hint” es una pista).
- Puede personalizar el formato de los contenidos según sea necesario, por ejemplo, para contraseñas, identificadores u otros tipos de datos, utilizando la propiedad `inputType`.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="A:"
    android:inputType="text"
    />
```



Caja de texto Asunto

Para crear un campo de texto para el “Asunto”, siga estas configuraciones:

- El ancho se ajusta automáticamente al tamaño del elemento padre, que a su vez está configurado para adaptarse al tamaño de la pantalla.
- La altura se ajustará dinámicamente según el contenido ingresado.
- Utilice el atributo `android:hint` para proporcionar una pista o mensaje de ejemplo en el campo de texto. Por ejemplo: `android:hint="Mensaje"` (donde “hint” es una pista).
- Si desea permitir la entrada de múltiples líneas de texto, configure la propiedad `textMultiLine` para admitir un formato de contenido diferente, como contraseñas, identificadores, etc.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Mensaje"
    android:inputType="textMultiLine"
    />
```

Agregar un Botón

Para añadir un botón a la interfaz, siga estas configuraciones:

- El tamaño y la altura del botón se ajustarán automáticamente al contenido que contiene.
- Utilice el atributo `android:text` para establecer el texto del botón. Por ejemplo: `android:text="Enviar"`.

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Enviar"  />
```

Estilo de componentes

Una vez que hemos añadido los elementos básicos, es hora de darles formato:

1. **Agregar Márgenes**:

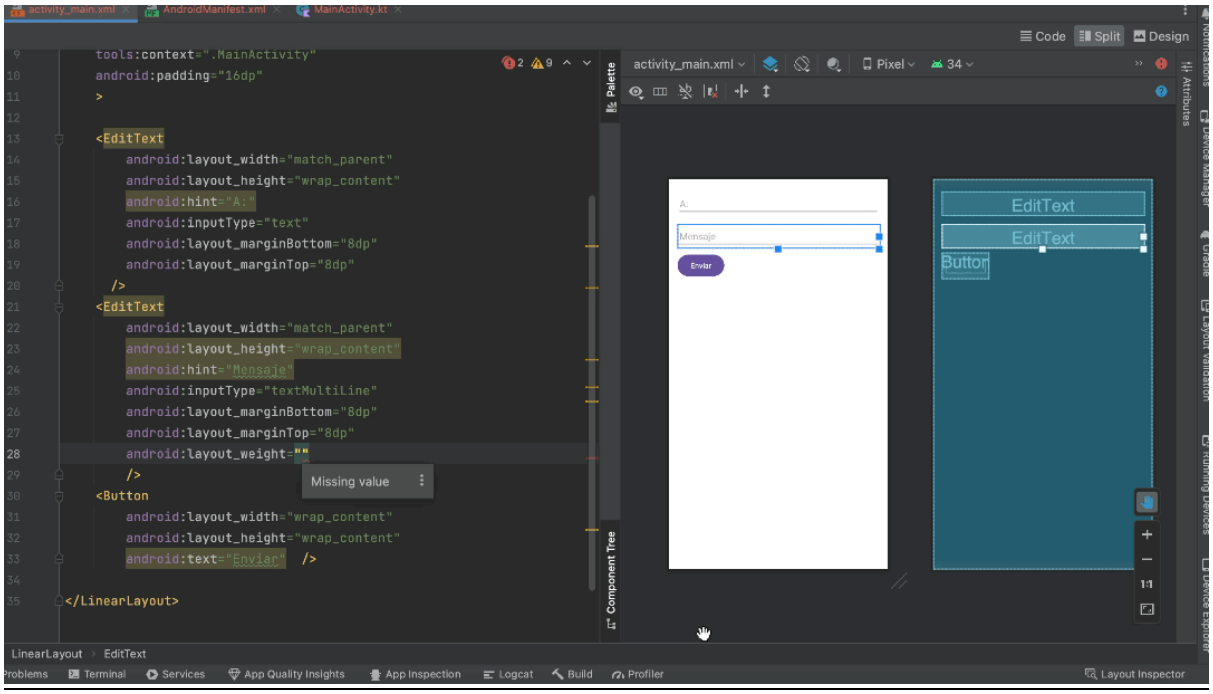
   Para mejorar la apariencia visual, vamos a añadir márgenes tanto arriba como abajo a cada elemento. Esto se puede lograr con las siguientes líneas de código en las etiquetas de los elementos:

```
android:layout_marginBottom="8dp"
android:layout_marginTop="8dp"
```

2. **Aumentar el Tamaño de la Caja de Texto**:

   Para hacer que la caja de texto sea más grande, vamos a utilizar la propiedad `android:layout_weight`. Un valor igual o mayor que 1 distribuirá el espacio disponible entre los elementos. Puedes agregarlo así:

```
android:layout_weight="1"
```



1. **Centrar el Contenido del Placeholder en la Caja de Texto**:

   Podemos centrar el contenido dentro de la caja de texto utilizando la propiedad `android:gravity`. Esto asegurará que el texto dentro de la caja de texto esté alineado verticalmente en el centro. Puedes lograrlo de esta manera:

```
android:gravity="top"
```

   También puedes combinar valores si necesitas centrar horizontalmente y verticalmente:

```
android:gravity="top|center"
```

2. **Posicionar el Botón**:

   Para posicionar el botón a la derecha, puedes utilizar la propiedad `android:layout_gravity`. Esto establecerá la ubicación del elemento dentro de su contenedor. Para colocar el botón a la derecha, puedes agregar lo siguiente:

```
android:layout_gravity="right"
```

Además de todo esto, debemos cambiar los **literales** del texto.

Última actualización: 08.02.2024

PRÁCTICA

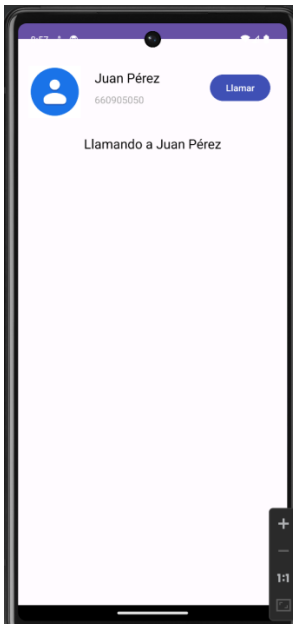
En este ejercicio, crearemos una aplicación que muestra una lista de contactos en un `LinearLayout` vertical y permite a los usuarios interactuar con ellos.

Instrucciones

1. Abre Android Studio y crea un nuevo proyecto de Android llamado "UDO2_5_Practica_LinearLayout".
2. En el archivo `activity_main.xml`, crea una interfaz de usuario que contenga un `LinearLayout` vertical. Dentro de este `LinearLayout`, agregaremos elementos de vista para representar una lista de contactos.
3. Personaliza los elementos de vista con texto, imágenes y atributos de diseño según tus preferencias. Puedes usar elementos como `ImageView` profile, `TextView`, y `Button` para representar un contacto. Para que la app se vea de la siguiente manera:

4. Implementa el código Java en el archivo `MainActivity.java` para que los elementos de vista puedan interactuar de la siguiente manera:

- Cuando se hace clic en el botón dentro del botón de “Llamar”, muestra un `mensaje` específico para ese contacto, como “Llamando a [nombre del contacto]”.



Capitulo 2

FRAME LAYOUT

FRAME LAYOUT

FrameLayout es uno de los tipos de contenedores de diseño (layouts) que se utilizan en el desarrollo de aplicaciones Android. Es un tipo de diseño que organiza los elementos secundarios (views) de forma superpuesta uno encima del otro, de modo que solo se muestra un elemento a la vez. Es decir, los elementos secundarios se apilan en capas y el último agregado se superpone a los anteriores.

Algunas características clave de FrameLayout incluyen:

- **Apilamiento de vistas**: Los elementos secundarios se colocan en la misma posición dentro del FrameLayout. Esto significa que ocupan la misma área y se superponen uno encima del otro. El último elemento agregado se muestra en la parte superior, ocultando los elementos anteriores.
- **Útil para superposición**: FrameLayout es especialmente útil cuando se necesita superponer vistas, como botones, imágenes o fragmentos, en una actividad o fragmento. Puedes usarlo para crear interfaces de usuario con capas.
- **Simplicidad**: Es uno de los layouts más simples y livianos disponibles en Android. No tiene reglas de alineación complejas, como LinearLayout o RelativeLayout.

Características Principales:

- **Diseños Sencillos**: FrameLayout es ideal para diseñar interfaces de usuario simples y eficaces.
- **Apilamiento de Vistas**: Las vistas hijas se apilan unas sobre otras en función del orden en el que se añadieron. Esto permite controlar la superposición de elementos en pantalla.
- **Control de Posicionamiento**: Puedes controlar la posición de cada vista dentro del FrameLayout, lo que facilita la creación de diseños personalizados.

Última actualización: 08.02.2024

SUBSECCIONES DE FRAME LAYOUT

CREAR PROYECTO

Para ver cómo funcionan los ciclos de vida en Android vamos a generar un proyecto sobre el que veremos los distintos estados de forma práctica. Así, lo primero que haremos será generar nuestro protecyo en Android Studio siguiendo los siguientes pasos:

1. **Abrir Android Studio**: Abre Android Studio en tu ordenador.
2. **Crear un Nuevo Proyecto**:
    - Selecciona “Start a new Android Studio project” en la pantalla de inicio.
    - Elige “Phone and Tablet” como tipo de dispositivo.
    - Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.
3. **Configuración del Proyecto**:
    - En la siguiente pantalla, completa la información básica sobre el proyecto:
        - **Name**: Ingresa “UF1_UD2_2_FrameLayout”.
        - **Package name**: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.
        - **Save location**: Elige la ubicación donde deseas guardar el proyecto en tu sistema.
        - **Language**: Selecciona “Kotlin” como lenguaje de programación.
        - **Minimum API level**: Selecciona “API 24: Android 7.0 (Nougat)” como SDK mínimo.
4. **Finalizar Configuración**:
    - Revisa la configuración y ajusta cualquier otra opción según tus preferencias.
    - Haz clic en “Finish” para crear el proyecto.

Android Studio generará automáticamente la estructura básica del proyecto, incluyendo los archivos necesarios para la actividad principal que has creado. Puedes comenzar a desarrollar tu aplicación agregando código a la actividad `MainActivity.kt` y diseñando la interfaz de usuario en el archivo de diseño correspondiente.

Última actualización: 08.02.2024

DISEÑO

Para lograr el resultado deseado, puedes reescribir las instrucciones de la siguiente manera:

Reestructuración del Layout en activity_main.xml

Para configurar el diseño de la aplicación, primero eliminaremos todo el contenido del archivo `activity_main.xml`. Luego, procederemos a crear un `FrameLayout` con las siguientes características:

- Utilizaremos la función de autocompletado para el atributo `xmlns` de Android.
- Asimismo, emplearemos el autocompletado para el atributo `xmlns` de la herramienta "tools".
- Estableceremos el ancho y alto del diseño para que se ajusten a las dimensiones del contenedor principal.
- Elegiremos una orientación vertical. Es importante tener en cuenta que, en caso de una orientación horizontal, la disposición de los elementos variará de derecha a izquierda o de izquierda a derecha, según la configuración del idioma del dispositivo. Para gestionar esto, podemos definir la propiedad `android:supportsRtl="true"` en el archivo `AndroidManifest.xml`. Esta propiedad indica si la aplicación admite la orientación de derecha a izquierda (RTL).
- Agregaremos el contexto para asociar este diseño con la actividad desde la que se lanzará. Aunque este paso no es obligatorio, puede ser útil para ciertas herramientas.
- Por último, aplicaremos un margen de 16dp al diseño.

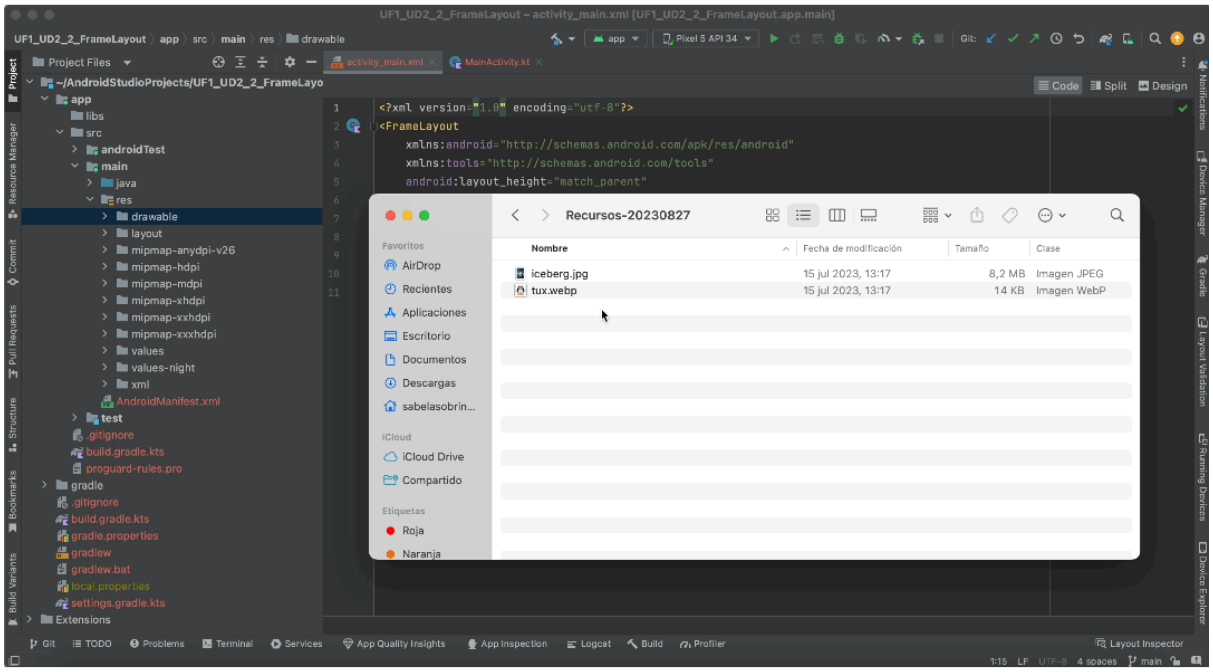
A continuación se muestra el código XML resultante:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android:layout_margin="16dp"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <!-- Aquí puedes agregar tus vistas y elementos de diseño -->
</FrameLayout>
```

Añadir imágenes

Las tenemos que añadir en al carpeta de recursos. SEleccionamos la vista de ficheros del proyecto (veremos más recursos).

Arrastramos a la carpeta drawable. Si pinchamos con el botón derecho dentro de la imagen veremos que ya nos da la opción de convertirla a webp:



Para añadirla a nuestra app tenemos que crear un nuevo elemento, tenemos que establecer el ancho y el alto, lo dejamos al tamaño del recurso.

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

Ahora tenemos que añadir el recurso, es decir, la imagen indicando la carpeta y el nombre de la imagen:

```
    android:src="@drawable/iceberg"
```

Pero vamos a escalar la imagen:

- centreCrop: va a ocupar todo el contenido del layout. El escalado se hace uniforme.
- centerInside: lo mismo que en el anterior pero cuando uno de los coja los bordes pasa.
- center: La imagen es mucho más grande que el contenedor, al pone en el centro pero no la escala por lo que no se ve bien.

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/iceberg"
    android:scaleType="centerCrop"/>
```

Multiresoluciones

Podemos tener diferentes resoluciones de nuestra imagen en función del dispositivo en el que se vaya a utilizar.

Cómo brindar compatibilidad con diferentes densidades de píxeles

Hay que crear varias carpetas en la carpeta drawable para las dierentes imágenes con las diferentes resoluciones. En caso de que no tuviéramos

Añadir la orta imagen (Ejercicio)

Aparece encima de la otra imagen -> se apilan las distintas vistas en el orden en el que generamos código. Como tiene el fondo transparente podemos ver la imagen que hay abajo. la centramos y la bajamos:

```
    android:layout_gravity="center_horizontal|bottom" />
```

Texto

color: #RGB Se puede poner en hexadecimal Tamaño: android:textSize=“40sp” Estilo: android:textStyle=“bold”

Última actualización: 08.02.2024

Capitulo 2

FRAME LAYOUT

Un `ScrollView` en Android es un contenedor que hereda de `FrameLayout` y se utiliza para permitir el desplazamiento de su contenido cuando no hay suficiente espacio para mostrarlo en la pantalla. Este contenedor solo admite un único elemento hijo y ofrece desplazamiento vertical. En lugar de agregar múltiples contenedores, podemos utilizar un solo `ScrollView` para lograr esta funcionalidad. Si necesitamos desplazamiento horizontal, podemos emplear un `HorizontalScrollView`.

La principal característica de un ScrollView es que permite a los usuarios desplazarse verticalmente (y en algunos casos horizontalmente) para acceder a todo el contenido que está fuera de la vista actual. Por ejemplo, si tienes un formulario largo o una lista de elementos que no caben completamente en la pantalla, puedes colocarlos dentro de un ScrollView para que los usuarios puedan desplazarse hacia arriba y abajo para interactuar con todo el contenido.

Última actualización: 08.02.2024

SUBSECCIONES DE FRAME LAYOUT

CREAR PROYECTO

Para ver cómo funcionan los ciclos de vida en Android vamos a generar un proyecto sobre el que veremos los distintos estados de forma práctica. Así, lo primero que haremos será generar nuestro protecyo en Android Studio siguiendo los siguientes pasos:

1. **Abrir Android Studio**: Abre Android Studio en tu ordenador.
2. **Crear un Nuevo Proyecto**:
    - Selecciona “Start a new Android Studio project” en la pantalla de inicio.
    - Elige “Phone and Tablet” como tipo de dispositivo.
    - Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.
3. **Configuración del Proyecto**:
    - En la siguiente pantalla, completa la información básica sobre el proyecto:
        - **Name**: Ingresa “UF1_UD2_3_ScrollView”.
        - **Package name**: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.
        - **Save location**: Elige la ubicación donde deseas guardar el proyecto en tu sistema.
        - **Language**: Selecciona “Kotlin” como lenguaje de programación.
        - **Minimum API level**: Selecciona “API 24: Android 7.0 (Nougat)” como SDK mínimo.
4. **Finalizar Configuración**:
    - Revisa la configuración y ajusta cualquier otra opción según tus preferencias.
    - Haz clic en “Finish” para crear el proyecto.

Android Studio generará automáticamente la estructura básica del proyecto, incluyendo los archivos necesarios para la actividad principal que has creado. Puedes comenzar a desarrollar tu aplicación agregando código a la actividad `MainActivity.kt` y diseñando la interfaz de usuario en el archivo de diseño correspondiente.

Última actualización: 08.02.2024

DISEÑO

Oara comenzar con nuestro proyecto, eliminamos todo el contenido del fichero activity_main.xml que establece nuestro diseño de la aplicación y generaremos un ScrollView:

- Utilizaremos autocompletado para el atributo android **xmlns**.
- Utilizaremos autocompletado para el atributo **xmlns** de tools.
- Estableceremos el ancho y alto para que se ajusten a las **dimensiones** del contenedor principal.
- Añadimos el contexto para asociar este layout con la actividad desde la que se va a lanzar.

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

</ScrollView>
```

Como sabemos, este elemento sólo podemos tener un único componente, pero nosotros queremos tener dos: un **elemento de texto** y un **botón**. Por lo que, añadimos un `LinearLayout` con las siguientes características:

- Estableceremos el ancho y alto para que se ajusten a las **dimensiones** del contenedor principal.
- Añadir un margen de 16dp.
- Seleccionaremos una orientación **vertical**.

Dentro de este Layout crearemos los elementos que necesitamos:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    >
    <EditText
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:hint="Mensaje"
       android:inputType="textMultiLine" />
    <Button
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Enviar" />
  </LinearLayout>
```

Vídeo

Si eliminamos el ScrollView y dejamos solo el LinearLayout como elemento raíz, notaríamos que no podemos realizar un desplazamiento vertical. Aunque podríamos desplazarnos dentro de la caja de texto, no tendríamos la capacidad de visualizar el botón que está fuera de la vista.

Última actualización: 08.02.2024

IMÁGENES

Para agregar imágenes a tu aplicación, es necesario seguir algunos pasos importantes. A continuación, se describe el proceso detalladamente:

1. Organización en la Carpeta de Recursos:

Primero, debes asegurarte de que tus imágenes estén ubicadas en la carpeta de recursos de tu proyecto. Para ello, realiza los siguientes pasos:

- Abre la vista de **ficheros** de tu proyecto (donde puedes ver la estructura de archivos).
- Navega hasta la carpeta **“drawable”** dentro de la carpeta de recursos. Si no existe, créala.

2. Conversión a Formato WebP (Opcional):

Si deseas optimizar tus imágenes para mejorar la eficiencia, puedes convertirlas al formato WebP. Esto se puede hacer fácilmente:

- En la carpeta “drawable”, haz clic derecho en la imagen que deseas convertir.
- Verás una opción para convertirla a formato WebP.

3. Agregar una ImageView:

Ahora, debes incorporar la imagen en tu aplicación utilizando una `ImageView`. Define la `ImageView` en el archivo XML de diseño de tu actividad de la siguiente manera:

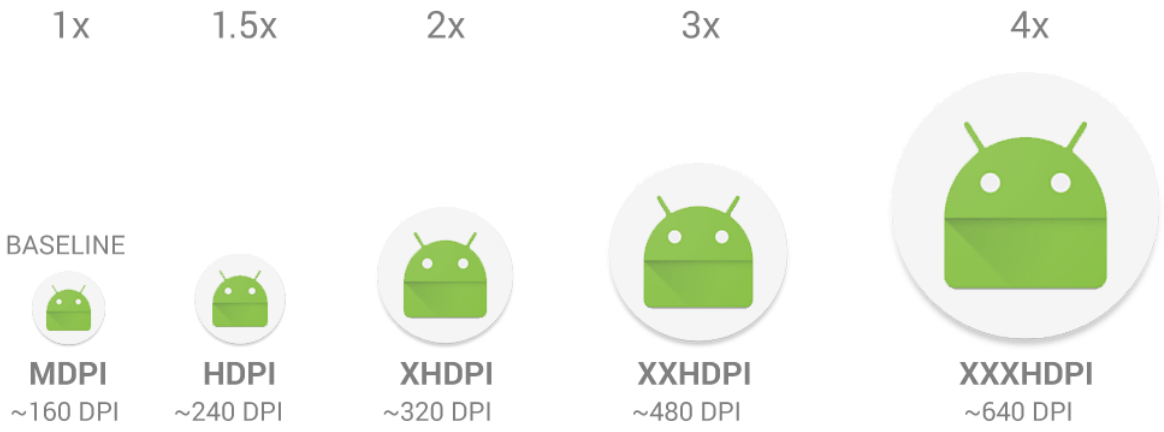
```
<ImageView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:src="@drawable/iceberg"
   android:scaleType="centerCrop"/>
```

El código anterior especifica las siguientes características:

- `android:layout_width` y `android:layout_height` establecen el ancho y alto de la `ImageView` según las dimensiones de tu recurso de imagen.
- `android:src` especifica la imagen que deseas mostrar, haciendo referencia al nombre de la imagen en la carpeta “drawable”.
- `android:scaleType` define cómo se escalará la imagen dentro de la `ImageView`. En este caso, se utiliza “centerCrop” para llenar todo el contenido del diseño, manteniendo la relación de aspecto y recortando los bordes si es necesario.

Densidades de pantalla

Podemos tener diferentes resoluciones de nuestra imagen en función del dispositivo en el que se vaya a utilizar.



Android proporciona carpetas de recursos específicas para diferentes densidades de pantalla, como drawable-mdpi, drawable-hdpi, drawable-xhdpi, drawable-xxhdpi, y más. Puedes colocar imágenes y recursos gráficos optimizados para cada densidad en estas carpetas.

Android seleccionará automáticamente los recursos adecuados según la densidad de pantalla del dispositivo.

Añadir la orta imagen (Ejercicio)

Añade la otra imagen a continuación de la anterior.

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/tux"
    android:layout_gravity="center_horizontal|bottom" />
```

Las distintas vistas se **superponen**, apareciendo una encima de la otra en el orden en que se generó el código. Dado que estas vistas tienen un fondo **transparente**, permiten que la imagen subyacente sea visible. Para centrar la vista y moverla hacia abajo, utilizamos la propiedad `android:layout_gravity` de la siguiente manera:

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

Esta configuración posiciona la vista en el centro horizontal de su contenedor y la alinea en la parte inferior del mismo.

Última actualización: 08.02.2024

TEXTO

Por último nos quedaría añadir el texto de nuestra aplicación, vamos a definir un TextView con el texto “Tux Loves Linux!” que se muestra en negritas, con un tamaño de texto grande. El TextView se coloca en la parte superior y se centra horizontalmente dentro de su contenedor, y el color del texto se define como blanco (asumiendo que haya un recurso de color “white” definido en el proyecto).

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tux Loves Linux!"
    android:layout_gravity="center_horizontal|top"
    android:textColor="@color/white"
    android:textSize="40sp"
    android:textStyle="bold
```

Última actualización: 08.02.2024

PRÁCTICA

En este diseño, crearemos un diseño de pila de tarjetas en el FrameLayout. Cada tarjeta contendrá una imagen y un título, y las tarjetas se apilarán una encima de la otra.

Instrucciones

1. Abre Android Studio y crea un nuevo proyecto de Android llamado “UDO2_6_Practica_FrameLayout”.
2. En el archivo `activity_main.xml`, crea una interfaz de usuario que contenga un `FrameLayout`.
3. Personaliza los elementos de vista con texto, imágenes y atributos de diseño según tus preferencias. Puedes usar elementos como `ImageView` y `TextView` pra que la app se vea de la siguiente manera:

- Ourense
- Santiago

4. Implementa el código Java en el archivo `MainActivity.java` para que los elementos de vista puedan interactuar de la siguiente manera:

- Cuando se hace clic en la imagen, muestra un `mensaje` específico para esa ciudad.
- Cuando se hace clic en la imagen, se escucha un sonido.

Última actualización: 20.01.2025

CONSTRAIT LAYOUT

Los **Constraint Layouts** son una característica importante en el desarrollo de aplicaciones Android. Estos diseños son el enfoque **predeterminado** en Android Studio y se han convertido en un componente esencial para crear interfaces de usuario flexibles y eficientes.

Está disponible como una librería de soporte que puede utilizarse desde el nivel de API 9, lo que significa que son compatibles con una amplia variedad de dispositivos Android. Los Constraint Layouts son especialmente valiosos para **evitar jerarquías** complejas de vistas, lo que a su vez mejora el rendimiento de la aplicación.

Una de las características distintivas de Constraint Layout es su capacidad para definir **restricciones** que controlan la **posición** y el **tamaño** de los componentes dentro de él. Estas restricciones pueden abordar márgenes, posicionamiento relativo y dimensionamiento, lo que proporciona un control preciso sobre la disposición de los elementos de la interfaz de usuario.

Puedes encontrar una guía detallada sobre cómo utilizar Constraint Layout en la documentación oficial de Android, que te ayudará a dominar esta potente herramienta de diseño.

Constraint Layout es solo una parte del conjunto de herramientas conocido como Android Jetpack. El objetivo de Jetpack es acelerar el desarrollo de aplicaciones al tiempo que se siguen las mejores prácticas. Proporciona bibliotecas para la arquitectura, manejo de datos, navegación y diseño de interfaces de usuario, entre otras cosas.

En resumen, Constraint Layouts son esenciales para el desarrollo de interfaces de usuario **flexibles** en Android, y forman parte de las herramientas poderosas que Android Jetpack ofrece a los desarrolladores para crear aplicaciones eficientes y de alta calidad.

Todas las actividades que podemos hacer con esas librerías se pueden organizar en cuatro grandes grupos:

- Arquitectura.
- Datos.
- Navegación entre las pantallas.
- Interfaz de usuario; layouts, Emojis, etc.

Otro objetivo es que el código funcione con independencia del código que se está ejecutando. Entre las muchas tareas que nos proporciona vamos a empezar a ver el ConstraitLayout.

Última actualización: 08.02.2024

SUBSECCIONES DE CONSTRAIT LAYOUT

CREAR PROYECTO

Para ver cómo funcionan los ciclos de vida en Android vamos a generar un proyecto sobre el que veremos los distintos estados de forma práctica. Así, lo primero que haremos será generar nuestro protecyo en Android Studio siguiendo los siguientes pasos:

1. **Abrir Android Studio**: Abre Android Studio en tu ordenador.
2. **Crear un Nuevo Proyecto**:
    - Selecciona “Start a new Android Studio project” en la pantalla de inicio.
    - Elige “Phone and Tablet” como tipo de dispositivo.
    - Selecciona “Empty Activity” como plantilla para comenzar con una actividad vacía.
3. **Configuración del Proyecto**:
    - En la siguiente pantalla, completa la información básica sobre el proyecto:
        - **Name**: Ingresa “UF1_UD2_4_ConstraitLayout”.
        - **Package name**: Deja el nombre de paquete predeterminado o personalízalo según tus necesidades.
        - **Save location**: Elige la ubicación donde deseas guardar el proyecto en tu sistema.
        - **Language**: Selecciona “Kotlin” como lenguaje de programación.
        - **Minimum API level**: Selecciona “API 24: Android 7.0 (Nougat)” como SDK mínimo.
4. **Finalizar Configuración**:
    - Revisa la configuración y ajusta cualquier otra opción según tus preferencias.
    - Haz clic en “Finish” para crear el proyecto.

Android Studio generará automáticamente la estructura básica del proyecto, incluyendo los archivos necesarios para la actividad principal que has creado. Puedes comenzar a desarrollar tu aplicación agregando código a la actividad `MainActivity.kt` y diseñando la interfaz de usuario en el archivo de diseño correspondiente.

Última actualización: 08.02.2024

DEPENDENCIAS

Dentro de la carpeta “app”, encontramos un archivo llamado `build.gradle`, que es un script de configuración fundamental para nuestro proyecto Android. En este archivo, definimos las bibliotecas y dependencias que nuestro proyecto utilizará. Entre estas dependencias, podemos observar la presencia de la biblioteca `ConstraintLayout`. Si esta biblioteca no está presente en nuestro proyecto en el momento de la construcción, se descargará automáticamente.

El archivo `build.gradle` se ve así:

```
dependencies {

    implementation("androidx.core:core-ktx:1.9.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.9.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")
}
```

Las dependencias, como `ConstraintLayout`, se agregan mediante la función `implementation`. Esto significa que estamos incluyendo esas bibliotecas en nuestro proyecto.

Para asegurarnos de que las bibliotecas se descarguen correctamente, especificamos los repositorios desde los cuales se obtendrán estas dependencias en el archivo `settings.gradle`:

```
dependencyResolutionManagement {
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories {
        google()
        mavenCentral()
    }
}
```

En este caso, estamos utilizando dos repositorios principales: “google” y “mavenCentral”. Gracias a esta configuración, el sistema descargará las bibliotecas necesarias desde estos repositorios automáticamente cuando construyamos el proyecto.

En resumen, el archivo `build.gradle` es esencial para gestionar las dependencias de nuestro proyecto Android, y la biblioteca `ConstraintLayout` se encuentra entre las dependencias que hemos configurado. La configuración de repositorios asegura que estas bibliotecas se descarguen correctamente sin necesidad de una intervención adicional.

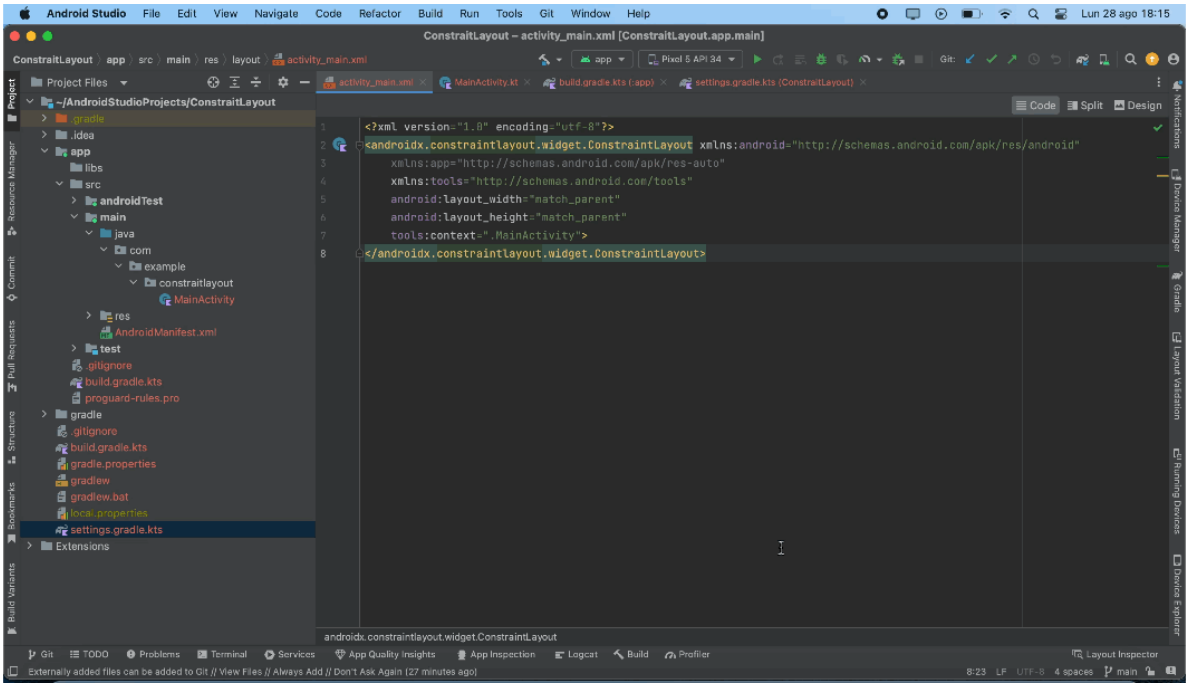
Última actualización: 08.02.2024

DISEÑO

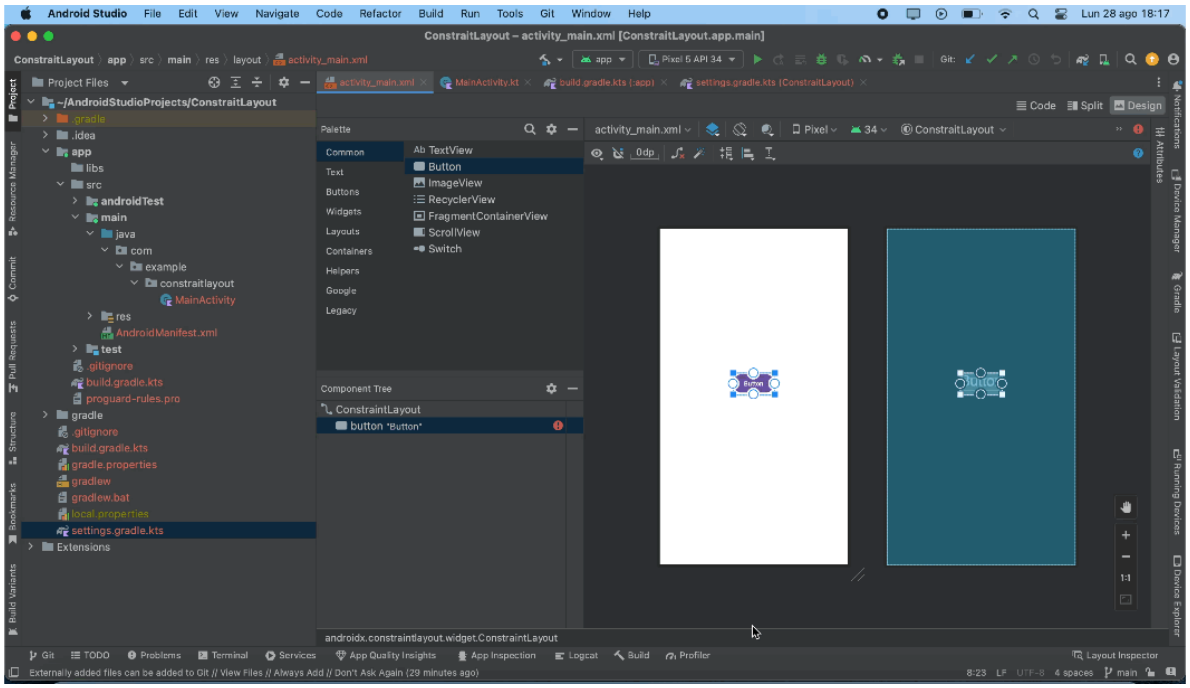
En el archivo `activity_main.xml`, procedemos a eliminar el elemento `TextView`, dejando el archivo de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
</androidx.constraintlayout.widget.ConstraintLayout>
```

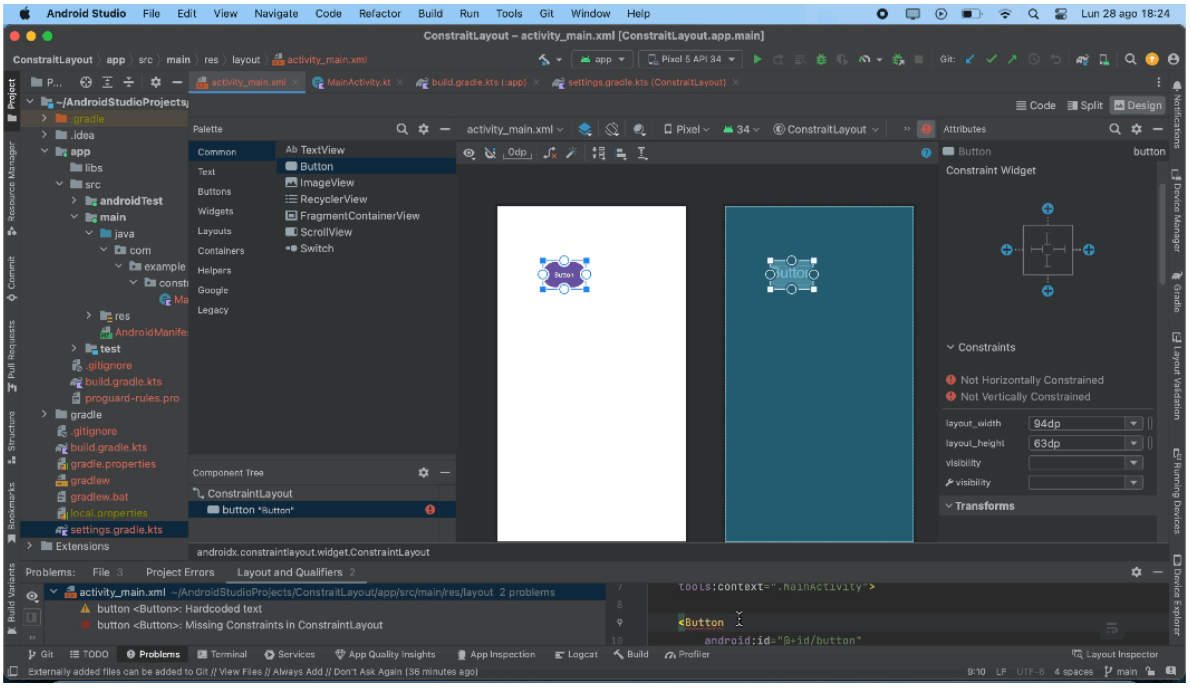
A continuación, en la vista de diseño, agregamos un botón. Vemos que lo podemos arrastrar y mover a cualquier lado de la pantalla, pero es importante destacar que arrastrarlo y posicionarlo libremente en la pantalla no es una práctica adecuada, ya que la posición del botón puede variar según la orientación del dispositivo.



En lugar de eso, vamos a utilizar **restricciones** (*constraints*) para posicionar el botón de manera precisa. Si observamos el botón, notamos que tiene unos accionadores relacionados con el tamaño del botón, y cualquier cambio que realicemos aquí se reflejará en el código.



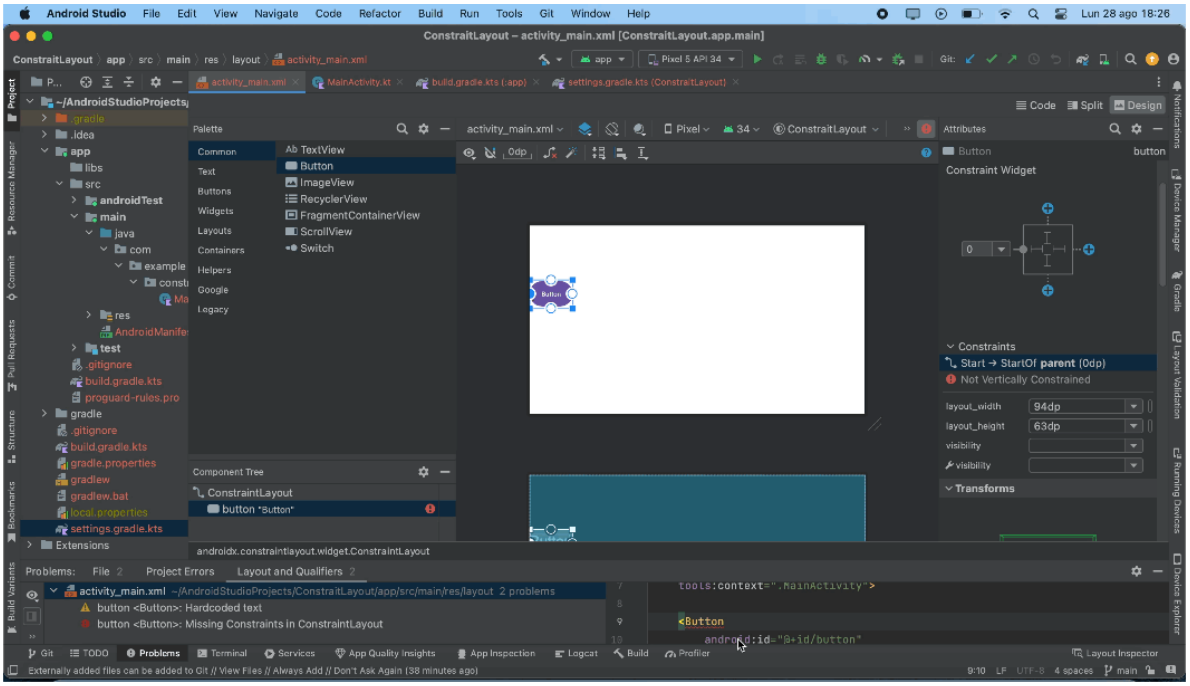
Además, tenemos accionadores **circulares** que nos permiten ajustar el comportamiento de la vista. Por ejemplo, si arrastramos el botón hacia un lado de la pantalla, aparecerá una nueva **restricción** que indica que el “inicio” del botón está conectado al “inicio” del padre. Si cambiamos la orientación del dispositivo, veremos que el botón se mantiene ajustado al lado restringido.



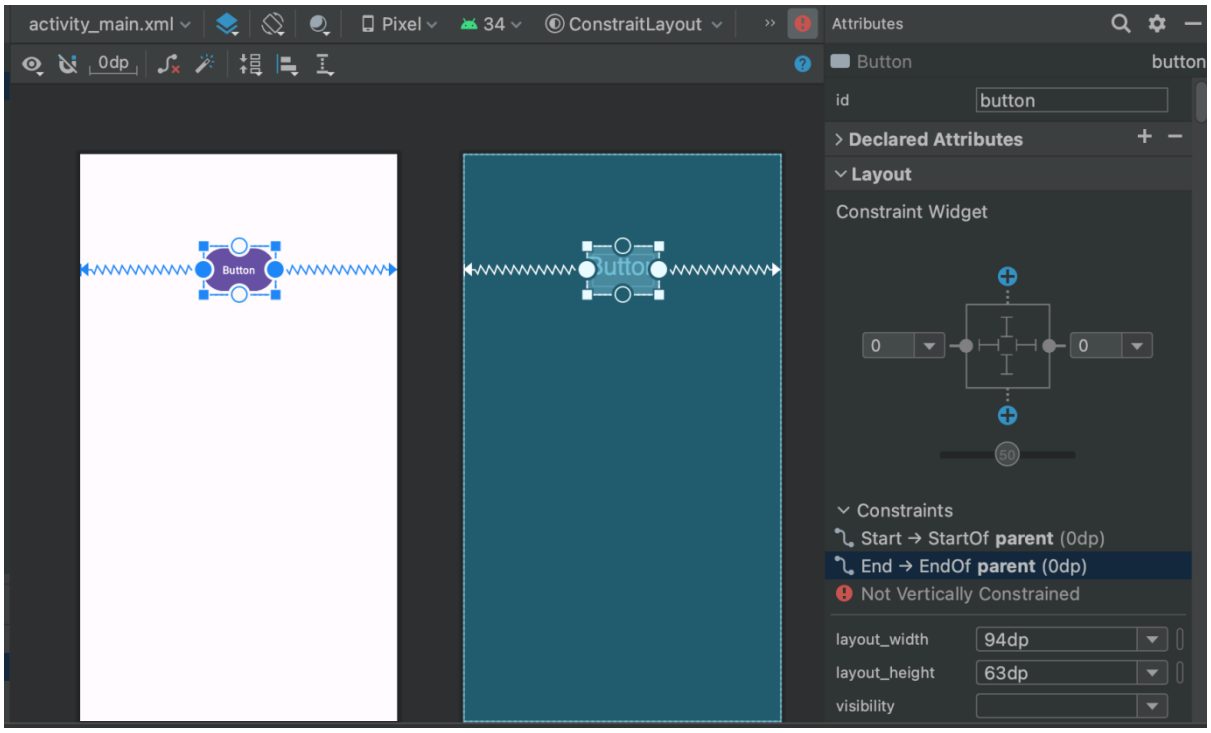
Si ahora ajustamos el lado opuesto a la otra parte de la pantalla, veremos cómo el elemento se posiciona en el **centro**. Podemos pensar en estas restricciones como “muelles” que están tirando del componente en ambas direcciones.

En el panel de la derecha, al seleccionar un elemento, encontramos un **widget** que nos permite **revisar** las restricciones aplicadas. Además, podemos ajustar el peso de estas restricciones, tanto horizontal como verticalmente, para controlar la disposición precisa del componente en la vista.

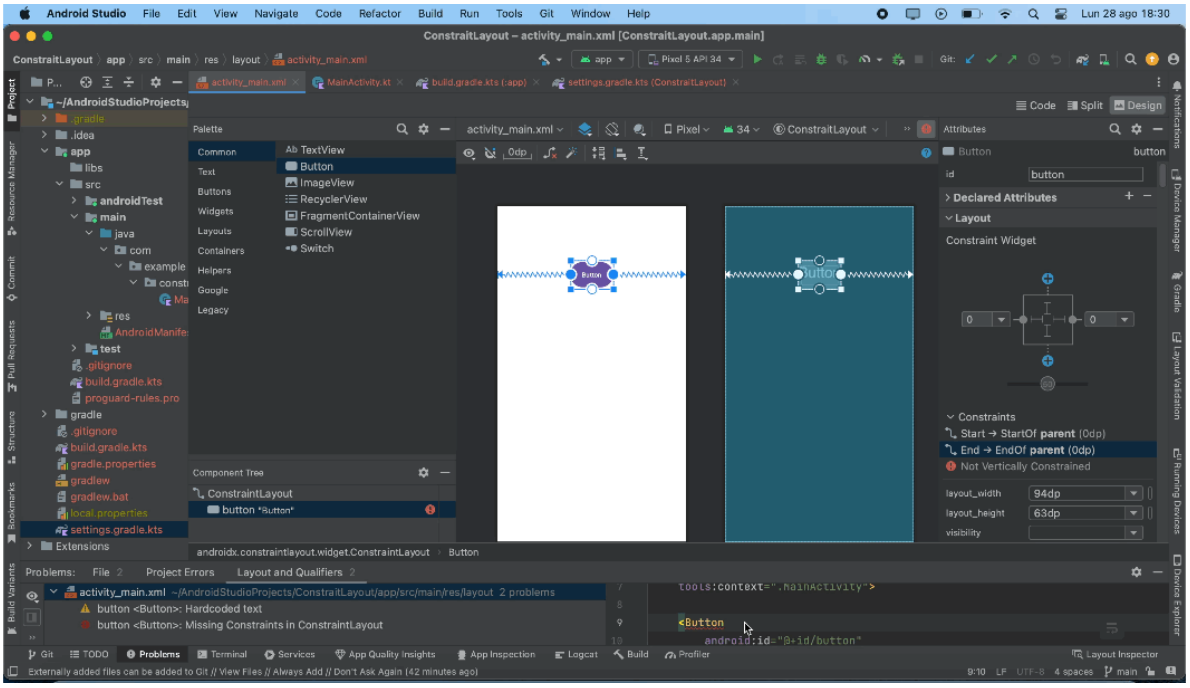
Si ahora ajustamos el lado contrario al otro lado de la pantalla vemos como el elemento se posiciona en medio, podemos comprobar como estas restricciones funcionan como “**muelles**” por decirlo de alguna manera que están tirando del componente a ambos lados.



En el panel de la derecha, al seleccionar un elemento, tenemos un widget donde podemos comprobar las restricciones que tenemos:



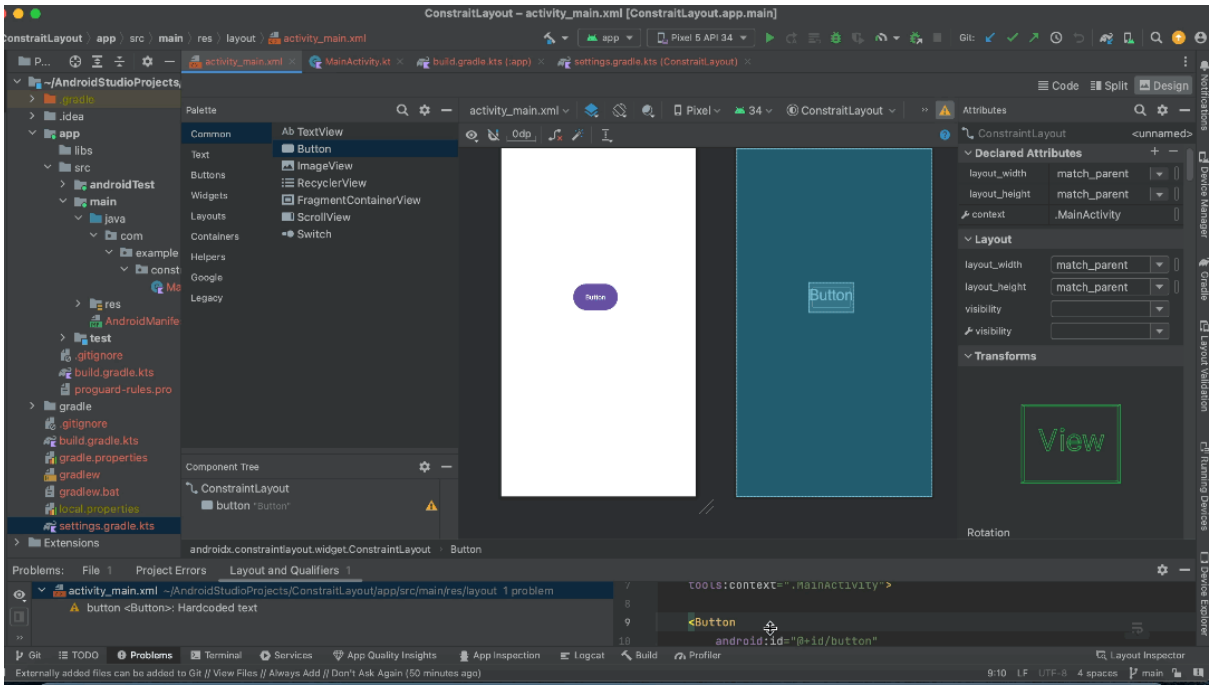
En este componente tenemos un elemento que nos permite controlar el peso que tiene esas restricciones, tanto horizontalmente como verticalmente.



VISTAS

Dimensiones y Márgenes de las Vistas

Además de aplicar restricciones de posición, también podemos aplicar restricciones al tamaño de un elemento en nuestra interfaz. En las propiedades, tenemos la capacidad de definir diferentes restricciones. Por ejemplo, si establecemos el ancho de un botón en 0dp, veremos cómo automáticamente se expande para ocupar todo el espacio definido por esa restricción, que equivale al ancho de la pantalla.



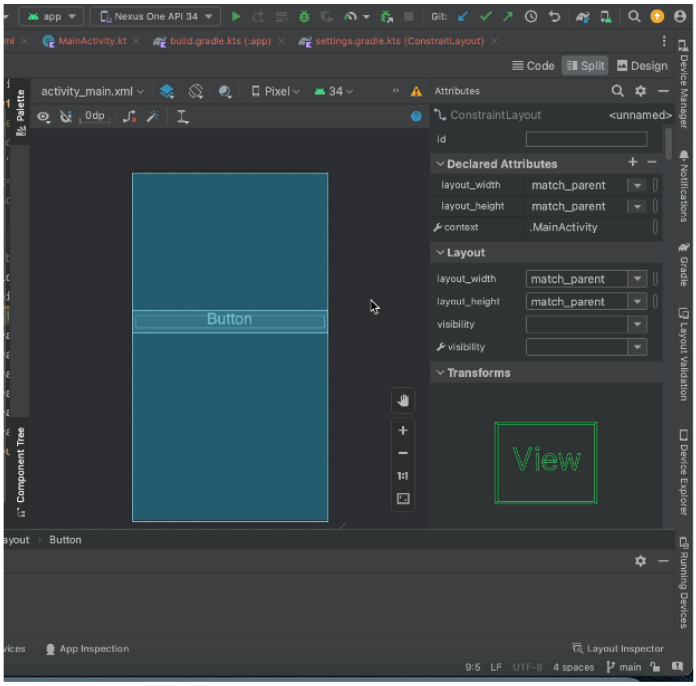
Esta acción es similar a establecer un valor de “match_parent” en el atributo de ancho del botón en el archivo XML:

```
<Button
  android:id="@+id/button"
  android:layout_width="match_parent"
  ...
```

Aunque el resultado visual es el mismo en ambos casos, existe una pequeña **diferencia** conceptual, **“match_parent”** se refiere al ancho del contenedor padre, mientras que cuando ajustamos el tamaño en el ancho mediante una **restricción**, en realidad estamos configurando el tamaño en relación con esa restricción específica.

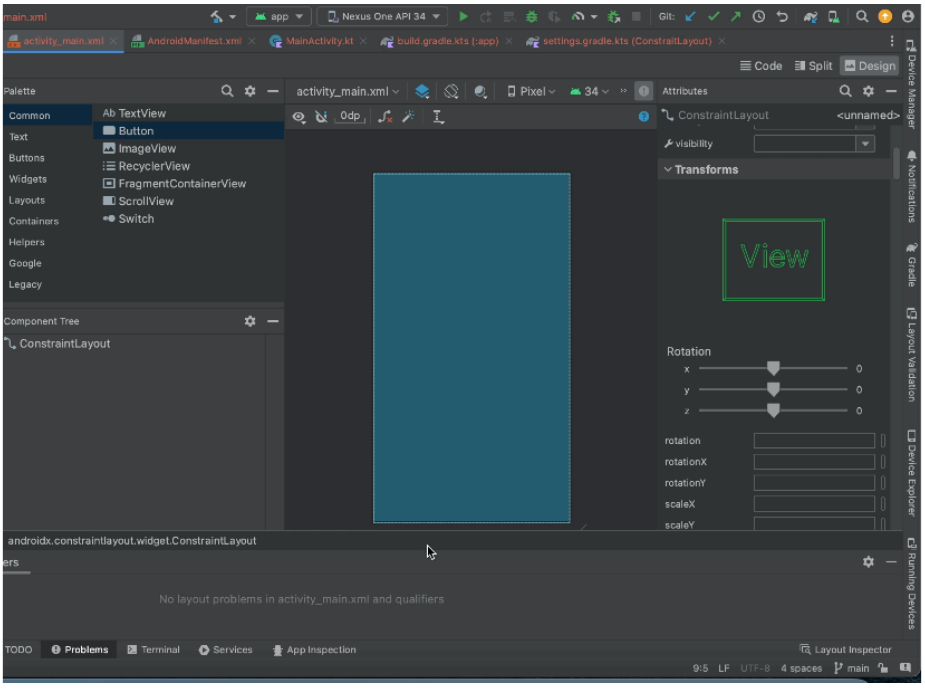
En las cajas de restricciones, se establecen números que son **potencias de dos**. Estos números permiten definir márgenes en relación con la restricción en sí, lo que proporciona un mayor control sobre la disposición de los elementos en la interfaz.

En resumen, además de posicionar elementos en la interfaz, ***ConstraintLayout*** nos permite controlar con **precisión** el **tamaño** y los **márgenes** de las vistas mediante la configuración de restricciones específicas. Esto ofrece una mayor flexibilidad en la creación de interfaces de usuario personalizadas y responsivas.

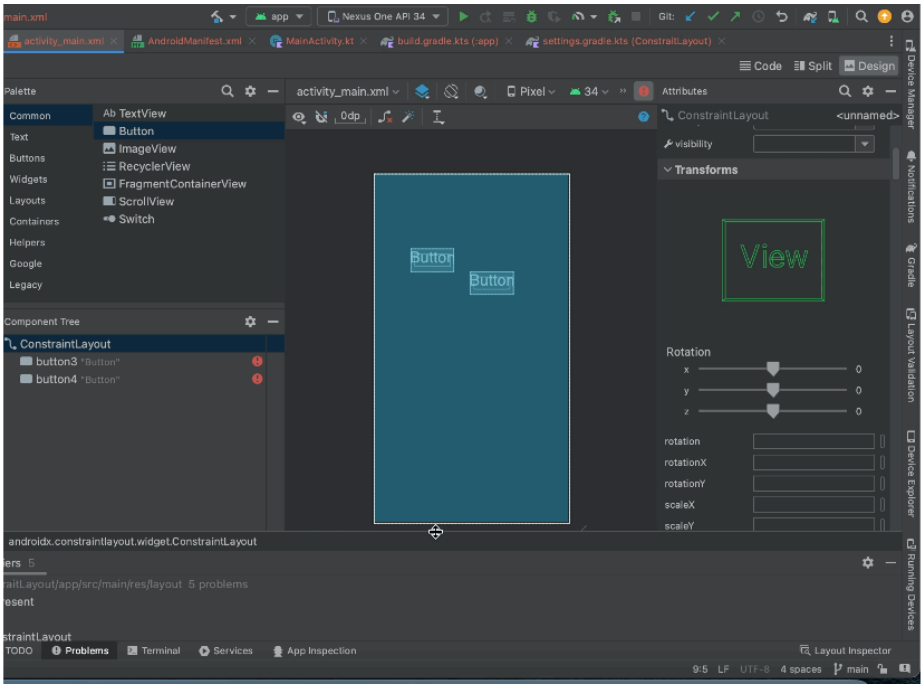


Alineación entre vistas

Vamos a incorporar otro botón y establecer una restricción que conecte la parte inferior del primer botón con la parte superior del segundo botón:



En este escenario, estamos definiendo una **restricción** en la que la **posición** de un botón depende de la posición del otro. Cuando movemos el primer botón, notamos que el segundo botón también se **desplaza**, y esto se debe a la restricción que acabamos de crear:

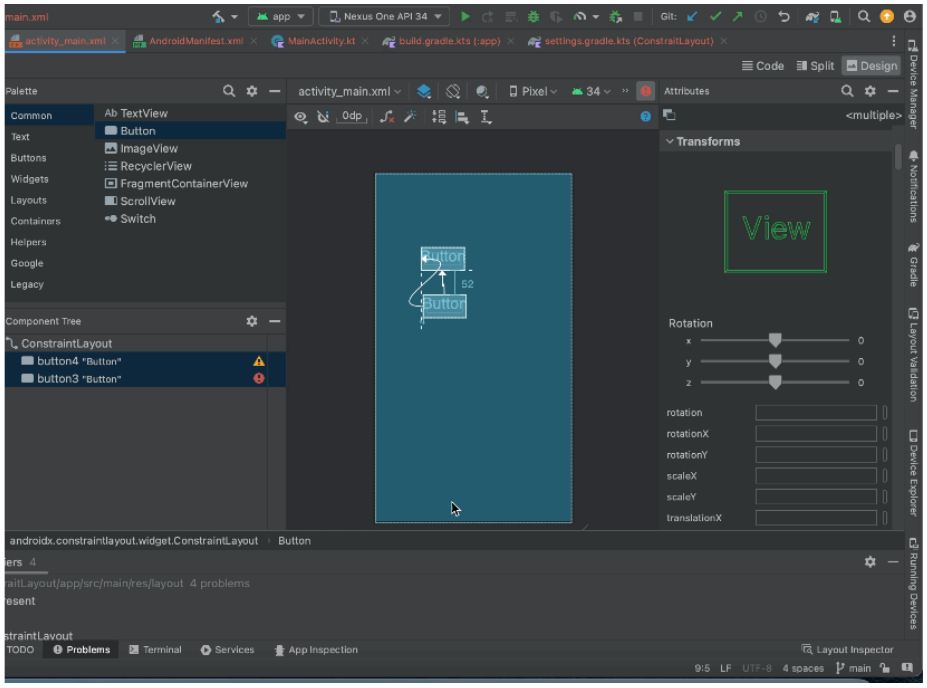


Por lo general, deseamos que los elementos de nuestra aplicación estén alineados de alguna manera para mantener una **apariencia uniforme** en nuestro diseño. Una forma sencilla de lograrlo es seleccionar ambos botones, lo que activará una opción en el menú que nos permitirá explorar diversas alternativas de **alineación**. Al seleccionar una de estas opciones, se generará automáticamente una restricción:

De esta manera, el primer botón tendrá dos restricciones asociadas:

1. La distancia entre ambos botones.
2. La alineación de los bordes izquierdos de ambos botones.

Con estas restricciones en su lugar, si movemos el primer botón, observamos cómo el segundo botón se ajusta automáticamente para mantener ambas restricciones:



Todas estas restricciones también pueden configurarse de manera **manual** seleccionando el borde izquierdo de un botón y conectándolo con el borde izquierdo del otro botón según sea necesario.

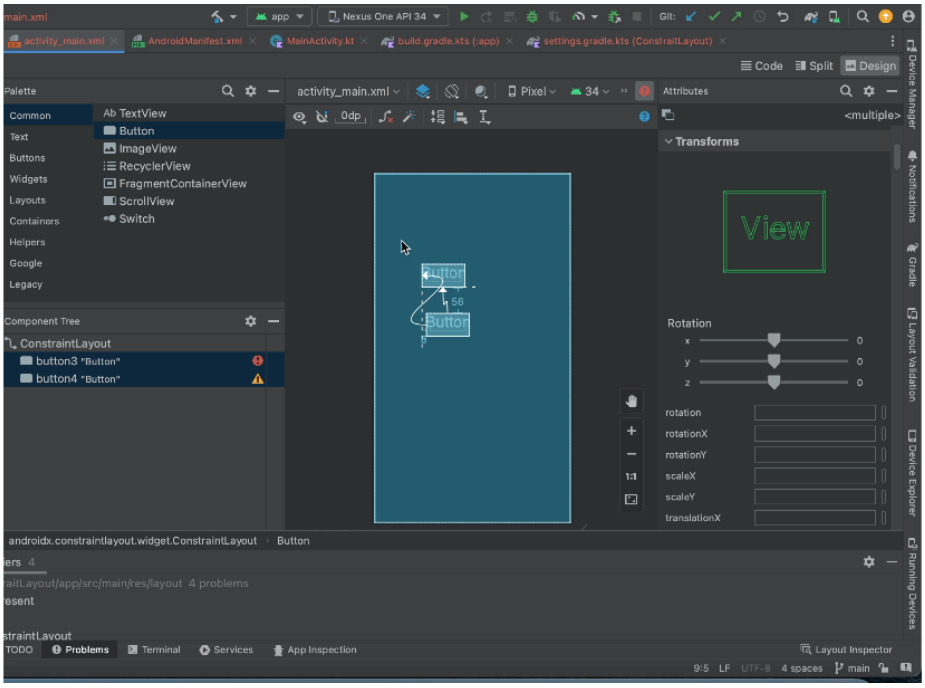
Última actualización: 08.02.2024

GUÍAS

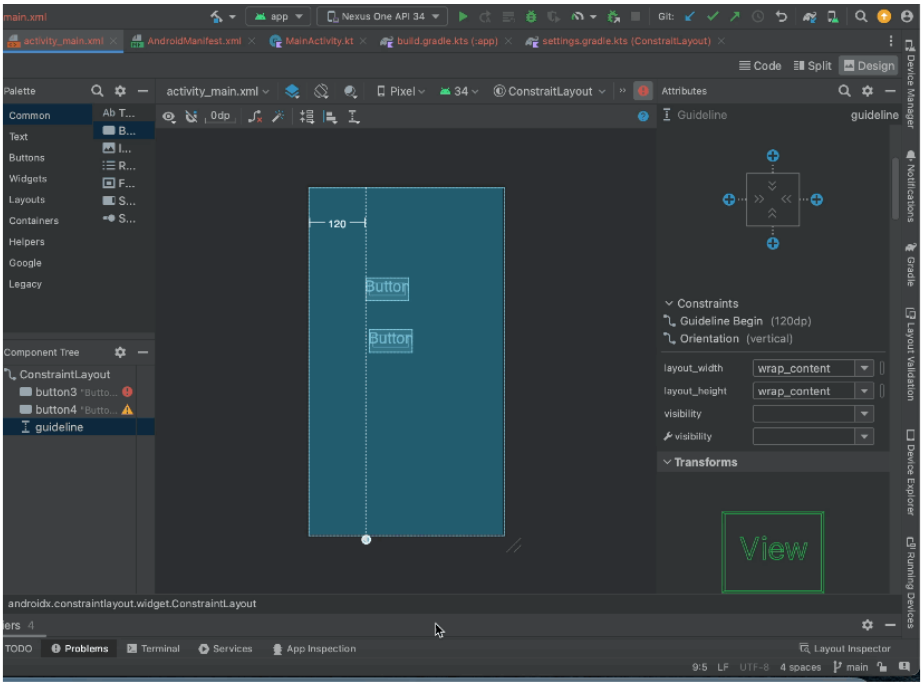
Uso de Guías

Otra técnica útil para alinear componentes es el uso de **guías**, una práctica común en programas de edición de imágenes como Photoshop o Gimp. En *ConstraintLayout*, puedes utilizar guías para lograr alineaciones precisas de la siguiente manera:

1. Dirígete al menú desplegable “Guidelines” y selecciona la opción para agregar una guía vertical.
2. Una vez que hayas añadido la guía, puedes ajustar los elementos de la interfaz para que se alineen con ella. Cuando mueves la guía, notarás que los elementos también se desplazan verticalmente junto con ella:



3. Al seleccionar la guía, aparecerá un número en la pantalla. Si haces clic en la flecha en la parte inferior de la guía, podrás alternar entre un valor en porcentaje y un valor en píxeles (dp):



Esto significa que puedes ajustar la distancia de la guía hacia uno de sus extremos (izquierda o derecha) o en un porcentaje de la pantalla. Es importante tener en cuenta que el uso de porcentajes puede ser más adecuado si deseas que tu diseño sea adaptable a diferentes tamaños de pantalla, mientras que los valores fijos en píxeles pueden generar diferencias en la disposición dependiendo del dispositivo.

Crea una caja de texto y un botón debajo de ella en ConstraintLayout. Observa cómo el botón se desplaza automáticamente a medida que escribes en la caja de texto. Este comportamiento se logra automáticamente gracias a las restricciones y la alineación de elementos en la interfaz mediante ConstraintLayout.

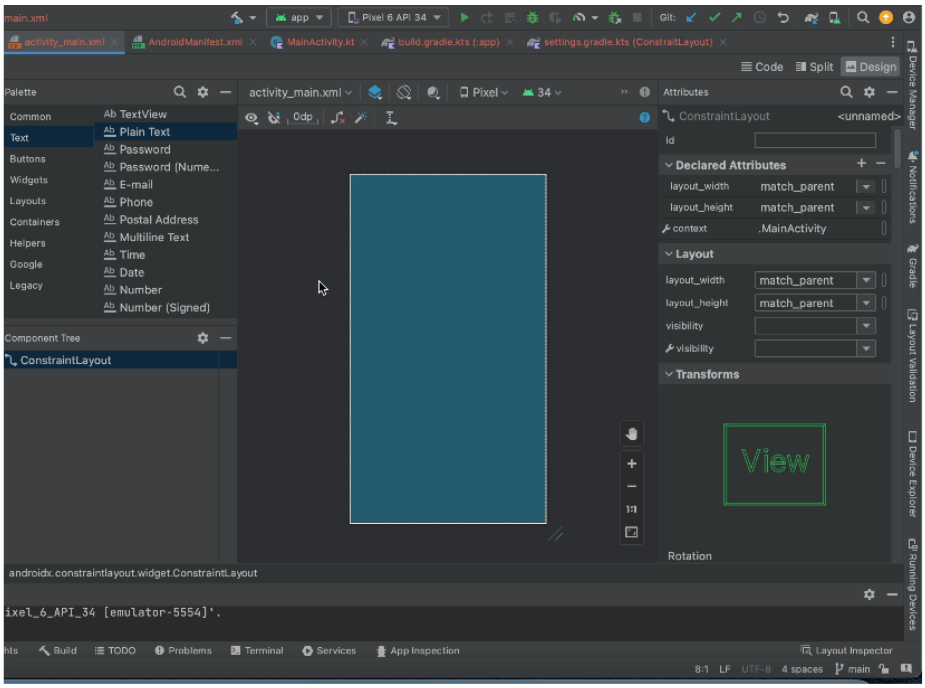
Guías y Barreras

En el caso de que necesitemos algo más flexible o una guía que se modifiquen durante la ejecución de la aplicación y que vaya marcando el posicionamiento de los elementos. Hablamos de las guías móviles. Para poder ver este término, elminamos los elementos que tenemos en nuestra aplicación y vamos a añadir dos cajas de texto multilínea de forma que vaya creciendo a medida que se vaya escribiendo. Debajo de las cajas habrá un botón que se debe ir desplazando a medida que las cajas vayan creciendo.

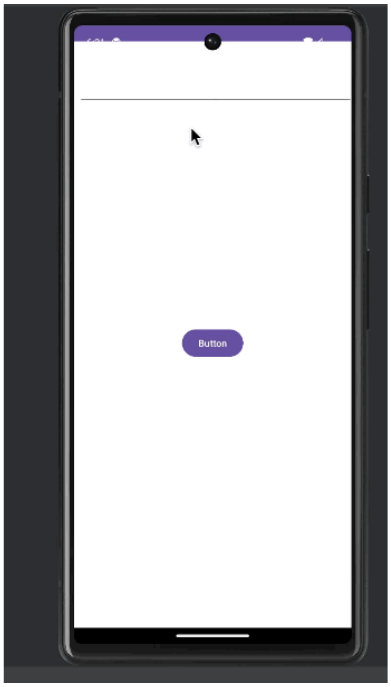
Vamos a empezar creando una guía vertical y la situamos en el medio de la pantalla utilizando un porcentaje del 50%. Lo siguiente será añadir esas cajas de texto editables (Text Multiline), ajustamos su posición añadiendo una restricción con la parte superior, izquierda y derecha de la pantalla añadiendo un margen de 24dp, ajustamos la altura del botón a su contenido:

```
<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
```

Ahora añadimos un botón, lo ajustamos tanto vertical como horizontalmente y lo colocamos en algún punto de la pantalla:



Si ahora ejecutamos nuestra aplicación vemos que si las cajas de texto editables crecen mucho nuestro botón se queda en la misma posición:



Lo que necesitamos es que ese botón se desplazara, ajustara su posición, a medida que el resto de elementos cambien de posición.

Barreras móviles

La diferencia de las barreras y las guías es que las guías son fijas y las barreras pueden desplazarse a medida que haya algún cambio en la aplicación.

- **Añadir una barrera móvil**

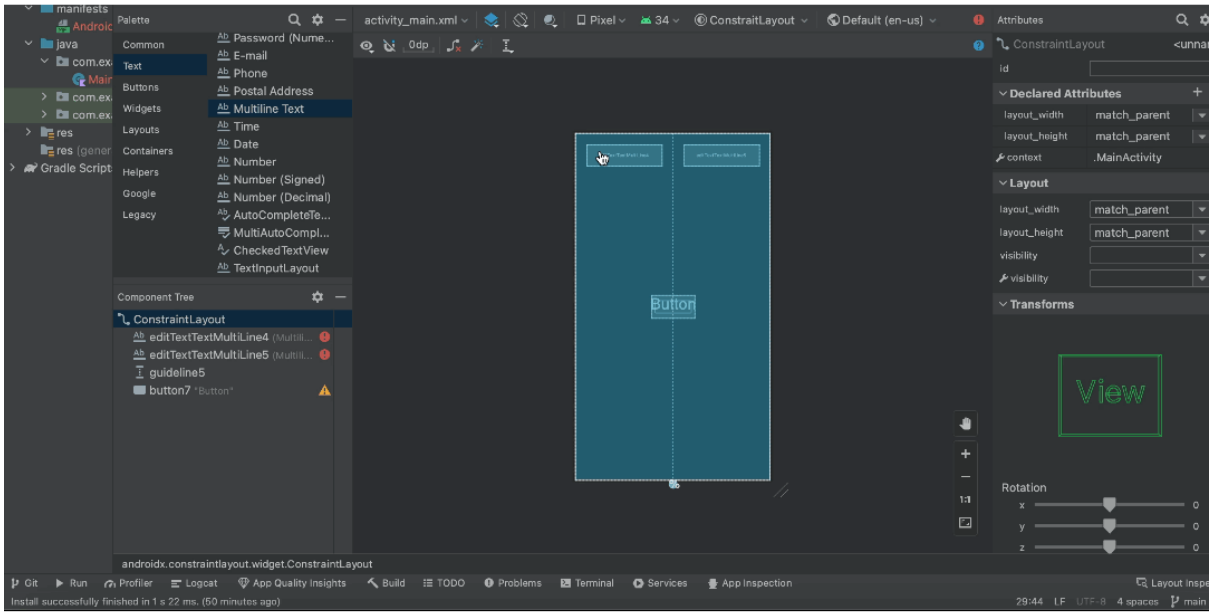
Desde el desplegable de guías (barra Design Tools > Guidelines), pulsar sobre Horizontal Barrier

- **Añadir a la barrera una restricción que referencie las cajas de texto**

El paso anterior nos habrá añadido una nueva barrera horizontal (podemos comprobarlo en el Component Tree donde debería aparecer algo como barrierN). Ahora, debemos añadirle una restricción para que su posición se ajuste en función de la posición de las cajas de texto editables. Para ello, desde el panel Component Tree, seleccionamos las dos cajas de texto y las arrastramos para colocarlas como hijos del componente barrera.

- **Cambiar la restricción a la parte inferior de las cajas de texto**

Por defecto, la nueva restricción se conecta con la parte superior de las cajas de texto. Para modificarlo, de forma que la restricción de la barrera esté conectada con la parte inferior de las cajas de texto (de forma que ésta se desplace al variar el tamaño de las mismas), tenemos que modificar su propiedad barrierDirection estableciendo su valor a bottom.



Ahora ya tenemos una barrera móvil que se irá desplazando a medida que los elementos superiores crezcan. Lo siguiente que tendremos que hacer será añadir una restricción para que lo haga también el botón. Para ello, seleccionamos el botón y buscamos la restricción `layout_constraintTop_toTopOf` al que le asignamos el id de la barrera: `@id/barrier8`:

Not found

Si ahora ejecutamos nuestra aplicación, vemos que el botón se va desplazando a medida que aumentamos el campo de texto.

Cadenas

Son un grupos de vistas que están conectadas entre ellas utilizando las restricciones, normalmente se utilizan para mantener el mismo espaciado entre ellas con independencia de la horientación, para que se distribuya uniformemente el espacio sobrante entre los distintos elementos. Para ver las cadenas vamos a partir de un espacio en blanco, por lo que borramos todo lo que tenemos hasta ahora y añadimos tres botones y añadimos las siguientes restricciones:

- Al primer botón le añadimos una restricción con la parte superior de la pantalla dando un margen de 64.
- Añadimos una restricción a los otros dos botones para que se alineen en la parte superior con el botón actual.

Ahora los tenemos alineados horizontalmente, lo que queremos añadir es un espaciado común entre los tres botones , que no cambie aunque se cambie la posición del dispositivo. Para ello tenemos que hacer clic en el botón derecho en el primer botón y en el menú de “chain” crear una cadena horizontal.

Última actualización: 08.02.2024

PRÁCTICA

En este ejercicio, crearemos una pantalla de inicio de sesión simple con dos campos de entrada (nombre de usuario y contraseña) y un botón de inicio de sesión. Usaremos ConstraintLayout para crear una interfaz de usuario flexible y responsiva.

Instrucciones

1. Abre Android Studio y crea un nuevo proyecto de Android llamado “UDO2_7_Practica_ConstraintLayout”.
2. En el archivo `activity_main.xml`, crea una interfaz de usuario que contenga un `ConstraintLayout` .
3. Personaliza los elementos de vista con texto, imágenes y atributos de diseño según tus preferencias (ambos campos de texto son multilínea) Para que la app se vea de la siguiente manera:

4. Implementa el código Java en el archivo `MainActivity.java` para que los elementos de vista puedan interactuar de la siguiente manera:

- Al hacer clic en el botón “Iniciar Sesión”, se mostrará un mensaje de prueba con los valores ingresados en los campos de entrada.
- Ponle un scroll para que no se pierda el botón.


2:23
UD02_ConstraintLayout_Practica
sabela
abc123.
INICIAR SESIÓN
Usuario: sabela Contraseña: abc123.