

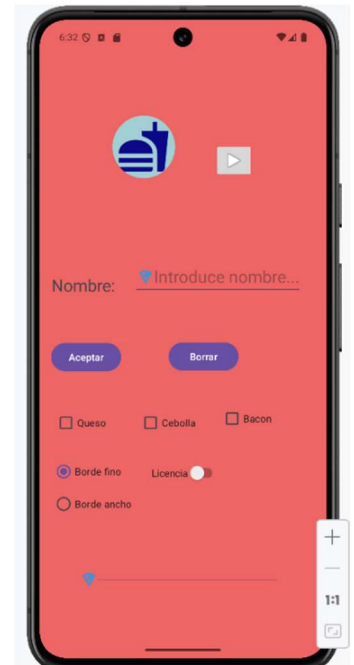
MÁS CONTROLES: IMÁGENES, SWITCH, SEEKBAR

Creamos un proyecto (ya hecho este punto anteriormente) : File→New Project → Empty Views Activity , el cual llamaremos, por ejemplo: **ImagenesYSeekBar**

El objetivo es crear la siguiente pantalla:

1. En el Activity usar binding para acceder a los recursos del layout.
2. Empezaremos por crear un **TextView** en la parte izquierda con texto “Nombre” y un **PlainText** al que le añadáis un hint (“Introduce tu nombre”), alineado a este por la derecha y al contenedor en el resto.

Poner un icono utilizando la propiedad **drawable** del PlainText (elige para que aparezca al comienzo de la misma como ves en la figura con la pizza). Pon un **padding** para que se separe un poco la pizza del borde (como veis las propiedades se pueden poner desde el asistente o bien desde código, ve comprobando que lo que añades se añade en código y prueba a cambiar o añadir una de ellas desde ahí).



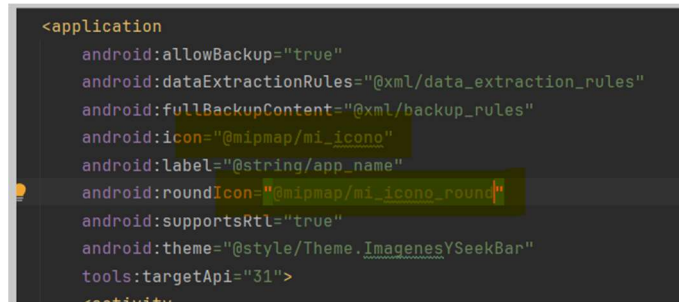
Colocaremos dos botones debajo. Esos botones serán Aceptar y Borrar. Poner un id a todos los controles y la etiqueta, a partir del fichero Strings.

3. Ahora vamos a proceder a añadir un **ImageView** en la parte superior de la ventana. Tenemos dos ubicaciones.
 - a. Mipmap: imágenes multiresolución.
 - b. Drawable: para poner imágenes. Pueden ser jpg, pero no son eficientes. Es mejor vectoriales que se adaptan a la resolución en función de fórmulas matemáticas y se adaptan a la misma. (al contrario que los mapas de bits).

Si pinchas en botón derecho sobre la carpeta res→New Vector Asset (ahí puedes coger un Clip Art existente, o importar un archivo .svg que previamente hayas creado o descargado).



4. Crearemos un nuevo icono para la app (res -> new Image), con un nombre diferente, para así tener que ir al archivo AndroidManifest.xml, y asociarlo.



```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/mi_icono"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/mi_icono_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.ImagenesYSeekBar"
    tools:targetApi="31">
```

5. Añadir un **ImageButton** y ponlo pegado a la imagen que pusiste en la parte superior (elige el play que es representativo en este ejemplo).
6. Investiga cómo puedes hacer para que al pulsar el play, la imagen superior cambie, y si vuelves a pulsar, vuelva a quedarse igual.
7. Probad a añadir un png (menos recomendable que vectorial, pero más que jpg, que tenga un fondo transparente), a la carpeta drawable, y que, por ejemplo, salga al pulsar en el propio ImageView principal.
8. Para alinear los botones es útil usar una guía, y así poner las constraint hacia ella. Utilízala para alinear la imagen principal y la del play.
9. Añadir 3 **Check** y un **RadioButton** con 3 opciones. Al pulsar aceptar, deberán mostrarse las opciones seleccionadas en el Toast
 - a. Para crear un Check, es sencillo, arrastrarlo simplemente.
 - b. Para crear un Radio Button, deben ir dentro de un mismo Radio Group y lo correcto es dejar al menos uno marcado por defecto.
10. Ahora adaptar el ejercicio, para que al pulsar el Aceptar, se genere un objeto de tipo Pedido (guardado en un paquete Modelo), y se muestre el Pedido con el toString.
11. Añadamos un switch, de que hemos leído las condiciones del pedido. Si está marcado mostrará el mensaje, si no, se indicará que hay que leer las condiciones.
12. Añadiremos un **SeekBar**. Aquí es importante saber que se asocian 3 listener, para poder actuar al comienzo, durante o al final del uso del elemento. Algunas propiedades interesantes son:
 - a. Max: valor máximo.
 - b. Progress: valor seleccionado.
 - c. Thumb: Se puede añadir un icono personalizado en vez del puntito que sale por defecto.
 - d. ProgressTint: va marcando de un color la parte seleccionada.

Aquí tenéis un ejemplo de código de la parte del SeekBar:

```
binding.sbSatisfaccion.setOnSeekBarChangeListener(object :
SeekBar.OnSeekBarChangeListener {
    override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser:
Boolean) {
        // Called when the progress level has changed.
        // Use the progress value here (ranges from 0 to max value)
        Log.i("Santa", "Progress: $progress")
    }

    override fun onStartTrackingTouch(seekBar: SeekBar?) {
        // Called when the user starts changing the progress value.
        Log.i("Santa", "Start tracking ${binding.sbSatisfaccion.progress}")
    }

    override fun onStopTrackingTouch(seekBar: SeekBar?) {
        // Called when the user stops changing the progress value.
        Log.i("Santa", "Stop tracking ${binding.sbSatisfaccion.progress}")
    }
})
```

Como recibe como parámetro un object que es una interfaz, esto te obliga a implementar los 3 métodos, aunque alguno lo puedas dejar vacío si no quieres que haga nada. Como veis la potencia de las funciones lambda permite hacer cosas como esta al definir un parámetro.