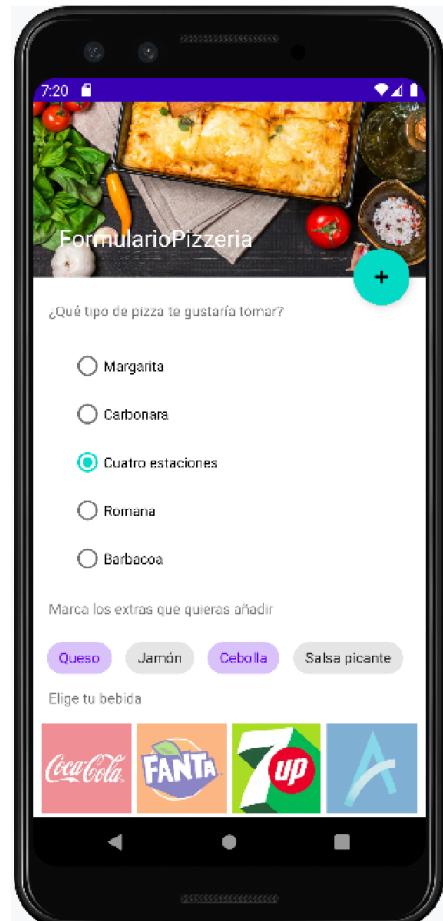


PROYECTO 4

FORMULARIO PIZZERÍA



En este proyecto haremos una app que mostrará un formulario de entrada de datos para encargar una pizza a domicilio.

Durante su desarrollo se tratarán estos conceptos:

- Concepto de Fragment
- El FragmentContainerView
- El diseñador gráfico
- Material design
- Concepto de ConstraintLayout
- Constraints
- Guidelines
- Barrier
- Chip
- Chain
- Flow
- El view binding en un Fragment
- Match constraint
- CheckBox
- ScrollView
- La default app bar
- Material toolbar
- Scrollable toolbar
- Collapsing toolbar
- Floating Action Button
- Snackbar

1 – Concepto de Fragment

Hemos visto que una **Activity** es una pantalla que contiene los elementos de la interfaz de usuario. Sin embargo, actualmente, las interfaces de usuario no se diseñan en las **Activity**, como hemos hecho hasta ahora, sino en los **Fragment**s.

Un **Fragment** es una interfaz de usuario completa (con su layout y su controlador) que puede ser colocada dentro de un **Activity**. Es posible reemplazar un **Fragment** por otro, y también es posible navegar entre los distintos **Fragment**, de forma que el **Fragment** visitado reemplace al antiguo dentro de la **Activity**.

En este proyecto, nuestra **Activity** contendrá un solo **Fragment** con todos los elementos de la interfaz. Aunque podríamos diseñar la interfaz directamente sobre la **Activity**, Google recomienda usar siempre **Fragment**, aunque se trate de una app de una sola pantalla.

- Abre Android Studio y crea un nuevo proyecto usando la plantilla **Empty views activity**.
- Pulsa el botón derecho del ratón sobre el paquete donde está el código fuente de la app y elige **new → Fragment → Fragment (Blank)**
- En la pantalla que aparece rellena estos datos:
 - Nombre del fragment: **FormularioFragment**
 - Archivo de layout del fragment: **fragment_formulario**
 - Lenguaje: **Kotlin**
- Se abrirá una ventana con el código fuente del archivo **FormularioFragment**, pero el código que nos genera Android Studio está obsoleto. Borra todo lo necesario hasta que se quede así:

```
1. class FormularioFragment : Fragment() {  
2.     override fun onCreateView(  
3.         inflater: LayoutInflater,  
4.         container: ViewGroup?,  
5.         savedInstanceState: Bundle?  
6.     ): View? {  
7.         // Inflate the layout for this fragment  
8.         return inflater.inflate(R.layout.fragment_formulario, container, false)  
9.     }  
10. }
```

En los **Fragment** el método **onCreateView** juega un papel similar al **onCreate** de las **Activity**, y es llamado cuando el dispositivo necesita crear la interfaz del **Fragment**

El método **inflate** que aparece, juega un papel similar al **onCreate** de las **Activity**, y su misión es construir los objetos de la interfaz a partir del archivo xml de su diseño.

En el caso de los **Fragment**, **onCreateView** debe devolver el objeto contenedor de la interfaz cargada, que es justo lo que devuelve la llamada a **inflate**.

- Observa que además de **FormularioFragment**, se ha generado un archivo de layout llamado **fragment_formulario.xml**, que podemos abrir con el diseñador.

Al igual que sucede con las **Activity**, los **Fragment** vienen con dos archivos: una vista (layout) definida en **fragment_formulario.xml** donde se diseña la interfaz del **Fragment**, y un archivo de código fuente **FormularioFragment.kt** donde se programarán las acciones del usuario sobre la interfaz (controlador).

2 – El FragmentContainerView

Los **Fragment** no se pueden mostrar directamente dentro de una **Activity**, sino que deben incluirse un componente llamado **FragmentContainerView**.

Un **FragmentContainerView** es un componente que se inserta en una **Activity** y en cuyo interior se visualiza un **Fragment**. El **FragmentContainerView** tiene métodos para poner un **Fragment** dentro de él y cambiarlo por otro.

- Abre el código fuente **activity_main.xml** y añade un elemento **FragmentContainerView** con estas características:
 - ID: **fragment_container_view**
 - Anchura: la de su contenedor
 - Altura: la de su contenedor
 - **android:name** → `dam.moviles.formulariopizzeria.FormularioFragment`

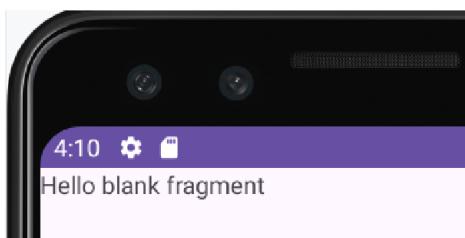
```

1. <androidx.fragment.app.FragmentContainerView
2.   xmlns:android="http://schemas.android.com/apk/res/android"
3.   xmlns:app="http://schemas.android.com/apk/res-auto"
4.   xmlns:tools="http://schemas.android.com/tools"
5.   android:id="@+id/fragment_container_view"
6.   android:layout_width="match_parent"
7.   android:layout_height="match_parent"
8.   android:name="dam.moviles.formulariopizzeria.FormularioFragment"/>

```

El atributo **android:name** contiene el nombre del **Fragment** que se mostrará en el **FragmentContainerView**

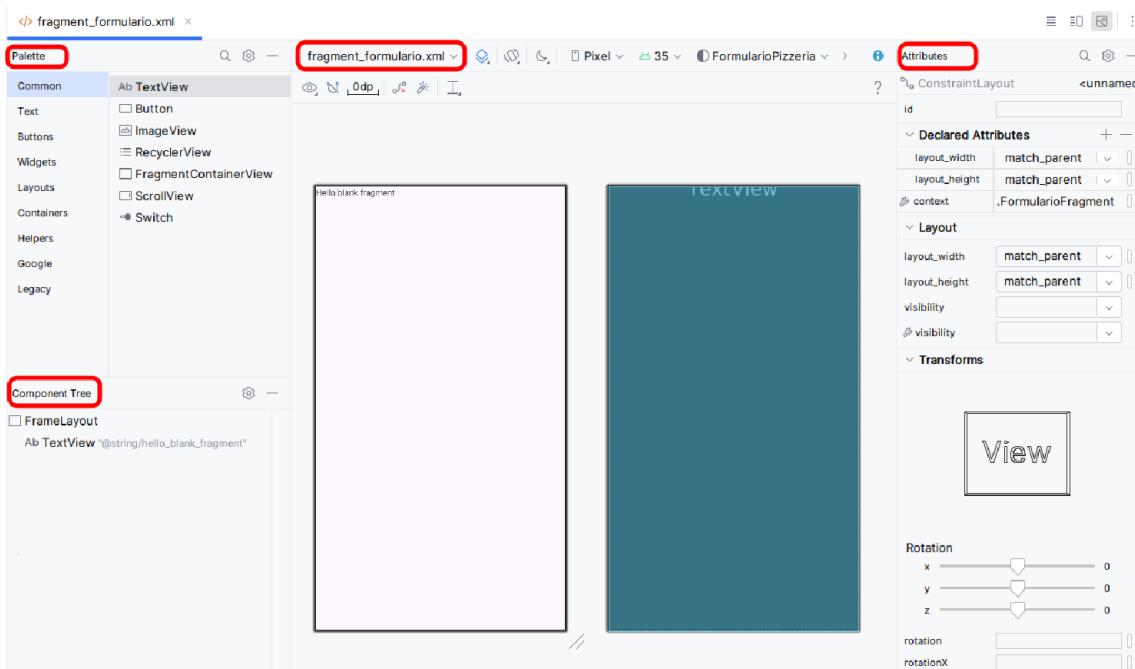
- Ejecuta la app y comprueba que se muestra la vista de **FormularioFragment**.



3 – El diseñador gráfico

Android Studio incorpora un diseñador muy potente para diseñar gráficamente las interfaces de usuario.

- Abre el archivo **fragment_formulario.xml** y pulsa el icono  para que se abra el diseñador gráfico.
- Observa que la pantalla se divide en las secciones que se ven en la siguiente imagen:



- **Palette:** Es una zona donde están todos los componentes que podemos arrastrar en la interfaz
- **Component tree:** Es la zona donde se ven todos los elementos que tenemos en la interfaz, y su anidamiento.
- **Diseño (fragment_formulario.xml):** Se muestra la vista previa de nuestra interfaz (pantalla izquierda) y el **blueprint** (pantalla derecha), que es una vista que se centra en la estructura de bloques de los elementos y sus medidas.
- **Attributes:** Cuando seleccionamos un elemento en la vista de diseño, en esta zona aparecerán todos los atributos de ese elemento y podremos cambiarle sus valores.

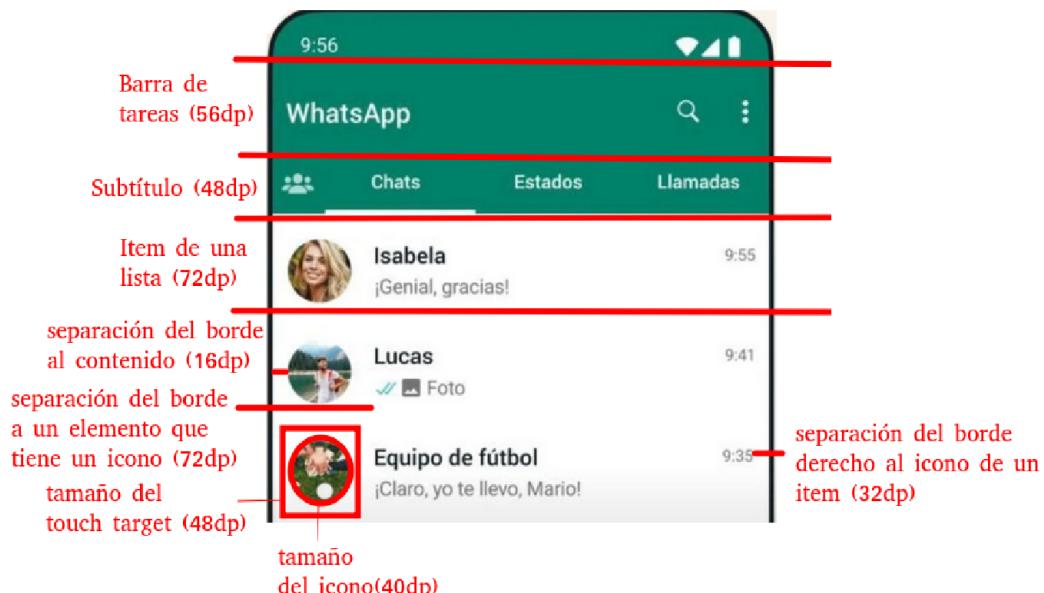
- Abre **fragment_formulario.xml** con el diseñador y selecciona el **TextView** que Android Studio ha colocado por defecto (*puedes hacerlo pinchando sobre él en la zona de diseño, o pinchando sobre él en el component tree*)
- Pulsa la tecla **supr** y el **TextView** deberá borrarse

4 – Material Design

Material Design es una guía de diseño creada por Google para hacer que las apps tengan una apariencia consistente entre ellas y los elementos que intervienen (botones, listas, etc) se comporten de forma intuitiva para el usuario, como si fuesen objetos del mundo real.

Algunas recomendaciones de esta guía a la hora de diseñar interfaces:

- Separaciones:
 - Separación de los bordes izquierdo y derecho de la pantalla a la zona de contenido → **16dp**
 - Separación vertical entre secciones → **8dp**
- Separaciones en una lista de elementos:
 - Separación desde el borde izquierdo de la pantalla al contenido que tiene un ícono/avatar → **72pxn**
 - Separación desde el borde derecho de la pantalla al ícono de un elemento → **32px**
- Tamaños:
 - Barra de tareas → **56dp**
 - Subtítulo → **48dp**
 - Altura de cada ítem de una lista → **72dp**
 - Tamaño de la zona asociada (**touch target**) al ícono de una lista → **48dp**
 - Si el ícono es grande, el tamaño del ícono será → **40dp**
 - Si el ícono es pequeño, el tamaño del ícono será → **24dp**
- Tamaño de los botones:
 - Altura → **36 dp** (siempre que se pueda)
 - Anchura → El del texto, más **16dp** por cada lado



5 – Concepto de ConstraintLayout

Un **ConstraintLayout** es un tipo de contenedor muy eficiente, que además está optimizado para ser diseñado de forma gráfica, con el diseñador de Android Studio.

Cuando un elemento de la interfaz se inserta en el **ConstraintLayout**, indicamos restricciones (constraints) que indican hacia donde queremos que se coloque, de forma que al redimensionar la pantalla, se conserven dichas preferencias.

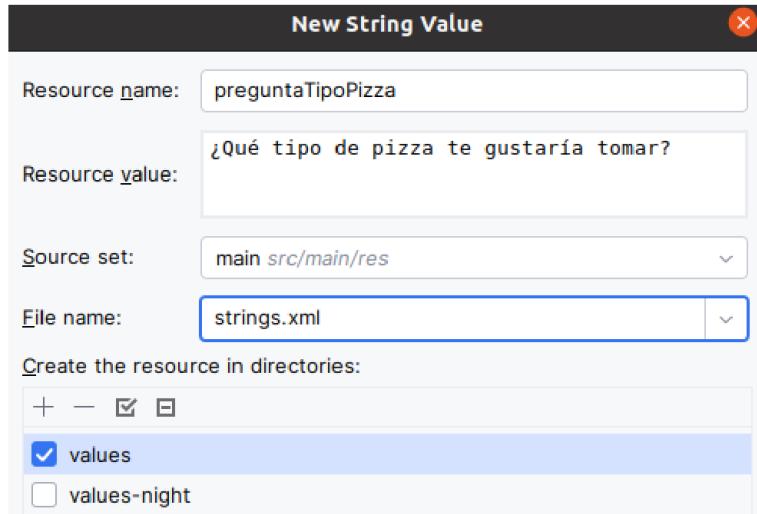
Gracias al **ConstraintLayout** podemos diseñar gráficamente interfaces muy sofisticadas que se comportan igual en todos los dispositivos. Además se cargan más rápido que si se usa anidamiento de layouts, como pasa con los **LinearLayout**

- En la zona **Component Tree** comprueba que el elemento contenedor principal de la interfaz sea un **ConstraintLayout**, y en caso de que no sea así, pulsa el botón derecho sobre el elemento contenedor y después **convert view → Constraint Layout** y pulsa **apply**

6 – Constraints

Cuando añadimos un elemento al **ConstraintLayout**, podemos indicar hacia donde debe pegarse dicho elemento, por cada uno de sus 4 lados, y la cantidad de margen que servirá para separarlo.

- En la zona **palette** busca **Common → TextView**
- Pulsa el ratón sobre **TextView** y arrastra el ratón hacia la vista de diseño, de forma que se coloque un **TextView** en algún lugar de la pantalla (nos da igual el lugar).
- Selecciona el **TextView** en la zona de **diseño** y modifica, en la zona **attributes** el valor de sus atributos de esta forma:
 - Id: **txtPreguntaTipoPizza**
 - Anchura y altura: la de su contenido
- Con el **TextView** seleccionado, vete al atributo **text**, pulsa el pequeño botón que hay a la derecha de la zona para escribir el texto, y aparecerá una ventana donde verás un signo **+**.
- Pulsa el signo **+** y elige **string value**
- Rellena la ventana que aparece con estos datos:
 - Resource name: **preguntaTipoPizza**
 - Resource value: **¿Qué tipo de pizza te gustaría tomar?**

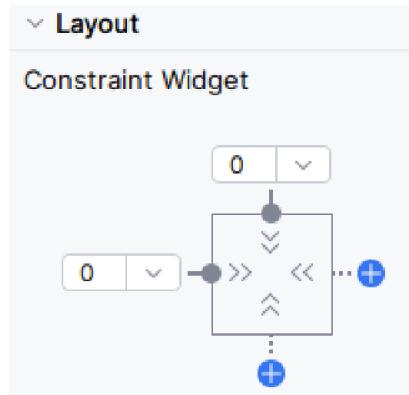


Esta pantalla nos permite añadir directamente un string al archivo **strings.xml** sin necesidad de tocar dicho archivo.

- Para pegar el **TextView** a la esquina superior de la pantalla, pulsa el ratón en el círculo que tiene arriba el **TextView** y arrástralos hacia arriba, de forma que se forme una flecha que vaya desde el círculo hasta el borde superior de la pantalla.

Al hacer esto, el **TextView** se mueve hacia arriba, porque hemos añadido una restricción (**constraint**) que hace que el texto debe pegarse hacia el borde superior de la pantalla.

- Para añadir otra restricción, de forma que el texto se pegue a la izquierda de la pantalla, basta con repetir el mismo procedimiento, pero ahora arrastrando el ratón desde el círculo que hay a la izquierda del **TextView** hasta el borde izquierdo de la pantalla.
- El siguiente paso es añadir un poco de margen para que el **TextView** no esté tan pegado a la esquina de la pantalla. Con el **TextView** seleccionado, localiza en la sección **Attributes** una parte llamada **Constraint widget**



En el **constraint widget** podemos ver las **constraints** que hay definidas, y el número que hay en cada una, nos indica el margen que lo separa de aquello hacia lo que debe pegarse (el borde de la pantalla, otro elemento, etc)

- Usa el **constraint widget** para que el **TextView** tenga un margen de 16dp por arriba y por la izquierda. *Si lo haces bien, el TextView se despegará un poco de los bordes de la pantalla.*
- En la sección **Button** arrastra cuatro **RadioButton** (da igual donde se pongan)
- Selecciona de uno en uno cada botón y ponle estos atributos:
 - Id: de arriba hacia abajo, **optPizza1, optPizza2, optPizza3, optPizza4**
 - Ancho y alto: los de su contenido
 - Texto: de arriba hacia abajo, los strings **pizza1, pizza2, pizza3, pizza4**
- Con la tecla **Mayúsculas** pulsada, selecciona de uno en uno los cuatro botones de radio, de forma que veas que todos se quedan marcados
- Selecciona ahora a **optPizza1** y ponle una restricción que lo pegue hacia el borde izquierdo de la pantalla y otra que lo pegue hacia el borde derecho de la pantalla. *Si lo haces bien, verás que ambas condiciones opuestas "se anulan" y el botón se queda en el centro de la pantalla, con unos muelles que representan las dos condiciones*
- Selecciona **optPizza2** y ponle estas dos restricciones:
 - Una que vaya desde el borde izquierdo de **optPizza2** hasta el borde izquierdo de **optPizza1**
 - Una que vaya desde el borde superior de **optPizza2** hasta el borde inferior de **optPizza1**
- Siguiendo el procedimiento del punto anterior, haz que **optPizza3** se pegue por la izquierda y por debajo a **optPizza2**
- Repite lo mismo con **optPizza4** y **optPizza3**. *Si haces bien los últimos apartados, verás que los cuatro botones aparecen uno encima de otro, y alineados por la izquierda con optPizza1*
- Selecciona **optPizza1** y observa que en la parte inferior del **constraint widget** ha aparecido una barra llamada **bias**, con un 50 en su interior
- Desplaza la barra del **bias** para que tenga valor **12**. *Si lo haces bien, verás que los botones se pegan hacia la izquierda, pero dejando un pequeño margen*

Cuando sobre un elemento tenemos dos restricciones opuestas, el **bias** nos permite indicar si dar más peso a una que a la otra.

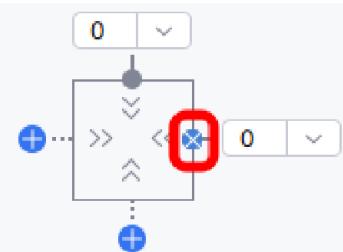
- Por último, selecciona **optPizza1** y añade una restricción que pegue su parte superior con la inferior del **TextView**, dejando un margen de 8dp

6 – Guidelines

Una técnica muy útil para alinear los componentes es usar una **guideline**, que es una línea invisible (que solo se ve en el diseñador) y hacer que los elementos se peguen a ella mediante **constraints**.

- Selecciona **optPizza1** y borra todas las restricciones que le has puesto

Si te cuesta trabajo borrar una restricción porque un elemento se ha pegado al borde y no puedes verla, en el **constraint widget** puedes pasar el ratón por el lado donde está la restricción y verás una X con la que puedes eliminarla



- Pulsa la tecla **Mayúsculas** y pulsa en los botones **optPizza2**, **optPizza3** y **optPizza4** de uno en uno, de forma que todos se queden seleccionados
- Pulsa el botón derecho del ratón sobre la selección y luego **Clear constraints from selection**. *Si lo haces bien, todas las constraints que has puesto en el apartado anterior sobre los botones se borrarán*
- En la zona **palette**, busca un **RadioGroup** y arrástralo hacia la interfaz, y ponle id **grupoPizza**. *Como el RadioGroup está vacío, no verás nada en él, salvo un pequeño punto.*
- En el **component tree** selecciona los botones de radio y arrástralos para meterlos dentro del **RadioGroup**. *Si lo haces bien, verás que el RadioGroup forma un bloque con los cuatro botones en su interior*
- Busca **Helpers** y añade a la interfaz un **Guideline (vertical)**. *Si lo haces bien, verás una línea vertical que cruza la pantalla*
- En el **component tree** busca la **Guideline** creada, seleccionala y en la zona **attributes** ponle el id **guideline1**
- Pulsa varias veces en el circulito que aparece en la parte inferior del **Guideline** hasta que puedas ver las distintas formas para expresar el valor en que se sitúa el **Guideline**.
- Mueve el **Guideline** para que se sitúe a **72px** del borde izquierdo de la pantalla.
- Añade ahora un **Guideline (horizontal)** y arrástralo para que ocupe un **7%** de la pantalla.
- Ponle como id **guideline2**
- Selecciona el **RadioGroup** y ponle estas restricciones:
 - o Su borde superior se pegará al **guideline2**

- o Su borde izquierdo se pegará al **guideline1**
- Selecciona el **TextView**, quítale la restricción que lo pega hacia la izquierda y arrástralolo lejos de esa zona (para poder trabajar mejor)
- Coloca una nueva **Guideline (Vertical)** que esté a **16dp** del borde izquierdo de la pantalla y ponle como id **guideline3**
- Pon una restricción en el **TextView** para que se pegue por la izquierda a **guideline3**

7 – Barrier

Una barrera (**Barrier**) es como un **Guideline**, pero con la particularidad de que no está ocupando una posición fija, sino que puede moverse si los elementos de la interfaz incluidos en ella cambian de tamaño.

La barrera se incluirá incluir elementos que pueden cambiar de tamaño, de forma que cuando la barrera se mueva, lo hagan también los elementos que tengan alguna restricción que les haga pegarse a ella.

- En **Helpers** busca el **Barrier (Horizontal)** y arrastra uno hacia la pantalla.
- En el **component tree** busca la barrera, selecciónala y en la zona de **attributes** ponle el id **barrera1**

Cuando se arrastra la barrera, no tiene ningún elemento y por ese motivo no se muestra nada en la pantalla.

- Para vincular el **RadioGroup** con la barrera, vete al **component tree** y arrastra el **RadioGroup** dentro de **barrera1**. *Si lo haces bien, se verá que la barrera contiene al RadioGroup y en la zona de diseño se verá que aparece una barrera en la parte superior del RadioGroup.*

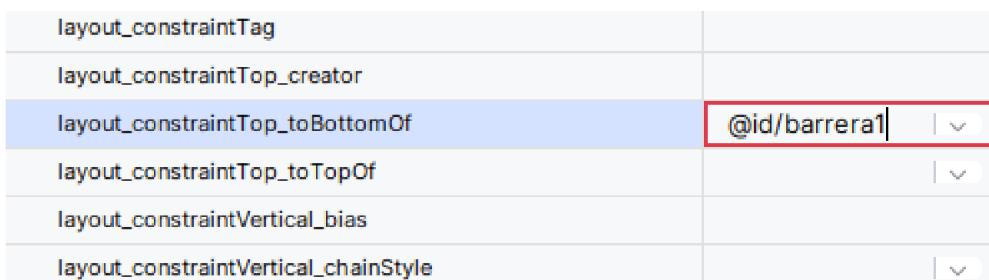


- Selecciona **barrera1** y en la zona de atributos, pon **barrierDirection** a **bottom**.
Si lo haces bien, verás que la barrera se desplaza hacia el fondo del RadioGroup

Poniendo la barrera en la parte inferior del **RadioGroup** hacemos que si el **RadioGroup** crece hacia abajo, la barrera se desplace con él.

- Añade a la interfaz un **TextView** con estas características:
 - Id: **txtPreguntaExtra**
 - Ancho y alto: el de su contenido
 - Texto: El del string **preguntaExtra**
- Selecciona **txtPreguntaExtra** y haz que se pegue por la izquierda a **guideline3** y por arriba a **barrera1**, con un margen de 16dp

Si tienes dificultad para unir la parte superior del **TextView** con **barrera1**, puedes seleccionar el **TextView** y en la zona de atributos, desplegar el menú **All attributes** y buscar la sección **layout_constraints**. Allí hay un montón de restricciones posibles, y una de ellas es **layout_constraintTopToBottomOf**. Selecciona **barrera1** en ella



- Añade un nuevo **RadioButton** al **RadioGroup** y comprueba que al añadirlo, **txtPreguntaExtra** se desplaza hacia abajo.

Si en lugar de una barrera, hubiésemos puesto un **guideline** y hubiéramos pegado el texto a él, al añadir el **RadioButton**, el texto hubiese seguido pegado al **guideline**.

- Pon al nuevo **RadioButton** estas características:
 - Id: **optPizza5**
 - Ancho y alto: El de su contenedor
 - Texto: El string **pizza5**

8 – Chip

Un **Chip** es un botón que se puede pulsar y soltar y su principal característica es que pueden tomar distintas apariencias. Se pueden agrupar en un **ChipGroup**

- En **Buttons** localiza **Chip** y arrastra tres de ellos a la interfaz (da igual donde los coloques)
- Configura cada **Chip** con estas características:
 - Id: **btnChip1, btnChip2, btnChip3, btnChip4**
 - Ancho y alto: El de su contenido
 - Texto: el referenciado en los strings **chip1, chip2, chip3** y **chip4**
 - style: **@style/Widget.MaterialComponents.Chip.Choice**

El atributo **style** permite indicar la apariencia que tendrá el **Chip**. Las posibles opciones son:

- **Action:** Actúa como un botón normal y corriente que se puede pulsar
- **Choice:** Cuando se hace clic en él, permanece pulsado
- **Filter:** Se usa para marcar condiciones
- **Entry:** Se usa para marcar condiciones, y tiene una X que permite cerrarlo

9 – Chain

Una cadena (**Chain**) es una restricción que permite formar una línea uniendo varios elementos de la interfaz y adaptándose al espacio libre de la pantalla.

Los elementos encadenados se pueden repartir equitativamente el espacio libre de la pantalla (**spread**), también se lo pueden repartir pero colocando los elementos de los extremos en los bordes de la pantalla (**spread inside**) o agruparlos todos juntos (**packed**)

- Pulsa la tecla **Mayúsculas** y selecciona de uno en uno los cuatro **Chip**
- Pulsa el botón derecho del ratón sobre los elementos seleccionados y elige **Align → Vertical center**. *Si lo haces bien, verás que se crea una restricción que alinea verticalmente los Chip*

Antes de formar una cadena de elementos, conviene alinearlos, ya que las cadenas funcionan mejor cuando los elementos están alineados.

- Pulsa el botón derecho sobre los elementos seleccionados y elige **Chain → Create horizontal chain**. *Si lo haces bien, verás que se forma una línea con los 4 chips, que el espacio entre ellos es el mismo, y que hay restricciones que unen el primero y el último con los bordes de la pantalla*
- Selecciona **btnChip1** y ponle una restricción que una su borde superior con el borde inferior de **txtPreguntaExtra**, con un margen de **16dp**
- Añade una nueva **guideline (horizontal)** debajo de los **Chip** y ponle id **guideline4**
- Añade un **TextView** con estas características

- Id: **txtPreguntaBebida**
- Alto y ancho: el de su contenido
- Texto: El del string **preguntaBebida**
- Colócale una restricción que por la izquierda lo una a **guideline3** y por arriba a **guideline4**

10 – Flow

Un **Flow** es como una cadena, pero con la particularidad de que cuando el tamaño de la pantalla cambia y se hace más pequeño, los elementos que no caben en la línea, se pasan a la siguiente (esto no pasa en la cadena, donde los elementos que no caben se pierden)

- En **Common** localiza **ImageView** y arrastra 6 a la interfaz, colocándolos donde quieras.
- Pon a los **ImageButton** estas características:
 - Id: **btnBebida1, btnBebida2, btnBebida3**, etc
 - Ancho y alto: **90dp**
 - **src:** las imágenes **bebida1.webp, bebida2.webp, bebida3.webp**, etc
 - **scaleType: fitXY**
- Con la tecla **mayúsculas** pulsada, selecciona de uno en uno todos los **ImageView**
- Pulsa el botón derecho del ratón sobre la selección y elige **Add helpers → Flow**. *Verás que los ImageView se alinean en la parte superior de la pantalla y algunos no caben y se salen por fuera*
- En el **component tree** selecciona **flow** y cambia su atributo **flow_wrapMode** al valor **chain**. *Si lo haces bien, verás que los elementos que no caben se bajan a la siguiente línea y se ven centrados*

El atributo **flow_wrapMode** permite indicar lo que sucede con los elementos que no caben. Sus posibilidades son:

- **chain:** Los elementos que no caben se pasan a la siguiente línea y forman una cadena (**Chain**)
- **aligned:** Los elementos que no caben se disponen en otra línea, colocándose de izquierda a derecha.

- Con el **flow** seleccionado, busca el atributo **flow_horizontalStyle** y selecciona el icono (**packed**).
- Selecciona el **flow** y añádele estas restricciones:
 - su borde superior se pegará a **txtPreguntaBebida** con un margen de **16dp**
 - sus bordes izquierdo y derecho se pegan a los bordes de la pantalla

- Selecciona el **flow** y con el botón derecho pulsa **Add helpers → Barrier (Horizontal)** y se creará una barrera
- En la barrera creada:
 - Pon el id **barrera2**
 - Modifica el atributo **barrierDirection** a **bottom**

Como el flow puede cambiar de tamaño si se añaden nuevas bebidas, es buen lugar para añadir una barrera en su parte inferior

- Pulsa el icono de la esquina inferior derecha del diseñador y haz más grande el tamaño de la pantalla.



11 – El view binding en un Fragment

El **view binding** que hemos visto en las **Activity** se puede poner también en los **Fragment**, de forma que podamos agrupar automáticamente los componentes de la interfaz de usuario en un objeto **MainActivityBinding**

Sin embargo, en el caso de los **Fragment** se suele tomar una precaución adicional, para evitar que cometamos errores cuando la app se hace muy compleja

- Habilita el **view binding** en el proyecto
- En **FormularioFragment** crea una variable de instancia **binding** y haz un método privado para inicializarla, de la misma forma que hemos visto para las **Activity**.

```

1. class FormularioFragment : Fragment() {
2.     private lateinit var binding: FragmentFormularioBinding
3.     override fun onCreateView(
4.         inflater: LayoutInflater,
5.         container: ViewGroup?,
6.         savedInstanceState: Bundle?
7.     ): View? {
8.         inicializarBinding(inflater, container)
9.     }
10.    private fun inicializarBinding(inflater: LayoutInflater, container: ViewGroup?){
11.        binding=FragmentFormularioBinding.inflate(inflater, container, false)
12.    }
13. }
```

El método **FragmentFormularioBinding.inflate** necesita tres parámetros, y dos de ellos solo los tenemos dentro de **onCreateView**. Por ese motivo, el método **inicializarBinding** necesita que se le pasen esos dos parámetros

- Haz que el método `onCreateView` devuelva el objeto `root` del `binding` (*esto es porque `onCreateView` debe devolver el objeto contenedor que hay en el xml, y eso es justo lo que guarda `binding.root`*)

```

1. override fun onCreateView(
2.     inflater: LayoutInflater,
3.     container: ViewGroup?,
4.     savedInstanceState: Bundle?
5. ): View? {
6.     inicializarBinding(inflater,container)
7.     return binding.root
8. }

```

De esta forma, conseguimos inicializar el `binding` de la misma manera que en las **Activity** y ya lo podríamos utilizar. Sin embargo, en los **Fragment** esta forma de inicializar el `binding` puede hacer que cometamos errores en apps más complejas.

El motivo está en que es muy frecuente que el usuario navegue de un **Fragment** a otro, siempre dentro de **MainActivity**. Cada vez que el usuario visita un **Fragment** se llama a su método `onCreateView`, y cada vez que se cambia a otro, se llama a `onDestroyView`. O sea, los componentes de la interfaz solo existen entre las llamadas a `onCreateView` y `onDestroyView`.

Sin embargo, los **Fragment** tienen otros métodos, como por ejemplo `onCreate`, que es llamado cuando se crea el **Fragment** y eso ocurre antes que `onCreateView`. Como los componentes debe crearlos `onCreateView`, `onCreate` no puede usarlos porque eso puede hacer que reviente la app.

Para evitar que podamos usar la variable `binding` en métodos (como `onCreate`) en los que los componentes no están creados, se suele inicializar `binding` en `onCreateView` y en `onDestroyView` hacer que `binding` guarde `null`. Como veremos, Kotlin nos permite mostrar un mensaje específico si usamos `binding` fuera del ámbito permitido.

- Cambia el nombre a la variable de instancia `binding` y ponle `_binding`
- Convierte en `nullable` la variable de instancia `binding` e inicialízala con el valor `null` (*al hacer esto ya no es necesario usar `lateinit` porque estamos dando un valor a `binding` en el constructor*)

```

1. class FormularioFragment : Fragment() {
2.     private var _binding: FragmentFormularioBinding? = null
3.     override fun onCreateView(
4.         inflater: LayoutInflater,
5.         container: ViewGroup?,
6.         savedInstanceState: Bundle?
7.     ): View? {
8.         inicializarBinding(inflater,container)
9.         return _binding.root
10.    }
11.    private fun inicializarBinding(inflater:LayoutInflater, container: ViewGroup?){
12.        _binding=FragmentFormularioBinding.inflate(inflater,container,false)
13.    }
14. }

```

En Kotlin un tipo de dato **nullable** es aquel que admite el valor **null** y su nombre termina en **?**. Por ejemplo, el tipo **Int** no puede guardar **null**, pero el tipo **Int?** si

- Programa el método **onDestroyView** y haz que **_binding** guarde **null**

```
1. override fun onDestroy() {  
2.     super.onDestroy()  
3.     _binding=null  
4. }
```

- Observa que aparece un error en la línea **_binding.root**

```
1. override fun onCreateView(  
2.     inflater: LayoutInflater,  
3.     container: ViewGroup?,  
4.     savedInstanceState: Bundle?  
5. ): View? {  
6.     inicializarBinding(inflater,container)  
7.     return _binding.root  
8. }
```

Kotlin no permite llamar a métodos de un objeto que guarda **null**¹. Por ese motivo, no es posible llamar a métodos de un objeto nullable si previamente no se ha comprobado que el objeto no es nulo (por ejemplo, haciendo un **if**)

En nuestro caso, vamos a añadir una variable de instancia auxiliar que permita acceder a **_binding** solo cuando esta no sea **null**

- Añade una variable de instancia llamada **binding** cuyo tipo sea **ResultadoFragmentBinding**, y que cuando sea consultada, nos devuelva **_binding**, o se lance un error en caso de que **_binding** sea **null**

```
1. class FormularioFragment : Fragment() {  
2.     private var _binding: FragmentFormularioBinding?=null  
3.     private val binding:FragmentFormularioBinding  
4.         get() = checkNotNull(_binding) {"Debes usar binding entre onCreateView y onDestroyView"}  
5.         // resto omitido  
6. }
```

La función **checkNotNull** comprueba si **_binding** es **null**. En caso afirmativo, lanza un error con el mensaje “Debes usar binding...”. Si **_binding** no es **null**, entonces **binding** devuelve **_binding**.

Por tanto, a partir de ahora, podremos usar **binding** y olvidarnos de **_binding**

- Haz que **onCreateView** devuelva **binding.root**

```
1. override fun onCreateView(  
2.     inflater: LayoutInflater,  
3.     container: ViewGroup?,  
4.     savedInstanceState: Bundle?  
5. ): View? {
```

¹ Se evita así el temido error **NullPointerException**, presente en la mayoría de los lenguajes de programación

```
6.     inicializarBinding(inflater, container)
7.     return binding.root
8. }
```

Podemos escribir **binding.root** sin error, porque **binding** es **FormularioFragment**, que nunca puede guardar **null**

- Añade a **FormularioFragment** un método auxiliar **getListaBotones** que nos devuelva un **List<ImageView>** con todas las imágenes de las bebidas

```
1. private fun getListaBotones():List<ImageView> =listOf(
2.     binding.btnBebida1,
3.     binding.btnBebida2,
4.     binding.btnBebida3,
5.     binding.btnBebida4,
6.     binding.btnBebida5,
7.     binding.btnBebida6,
8. )
```

Vamos a hacer que las imágenes se comporten como botones jugando con su canal alfa (o grado de transparencia). Las imágenes con un canal alfa 0.5 se consideran no pulsadas, mientras que las que tienen canal alfa 1 se consideran pulsadas.

- Añade a **FormularioFragment** un método auxiliar **desactivarBotones**, que ponga a todos los **ImageView** de las bebidas canal alfa 0.5F (*la F indica que es un Float, que es el tipo que admite el canal alfa*)

```
1. private fun desactivarBotones(){
2.     getListaBotones()
3.         .forEach { boton -> boton.alpha = 0.5F }
4. }
```

- Añade a **FormularioFragment** un método auxiliar **pulsarBoton**, que reciba un **ImageView** y le ponga canal alfa a 1, desactivando previamente todos los botones

```
1. private fun pulsarBoton(boton: ImageView){
2.     desactivarBotones()
3.     boton.alpha=1F
4. }
```

- Añade a **FormularioFragment** un método auxiliar **inicializarBotones** que primero desactive todos los botones de las bebidas y luego, haga que cuando se pulse cada uno, se active

```
1. private fun inicializarBotones(){
2.     desactivarBotones()
3.     getListaBotones().forEach { boton ->
4.         boton.setOnClickListener{
5.             pulsarBoton(boton)
6.         }
7.     }
8. }
```

- Por último, llama a **inicializarBotones** en **onCreateView**

```

1. override fun onCreateView(
2.     inflater: LayoutInflater,
3.     container: ViewGroup?,
4.     savedInstanceState: Bundle?
5. ): View? {
6.     inicializarBinding(inflater, container)
7.     inicializarBotones()
8.     return binding.root
9. }
```

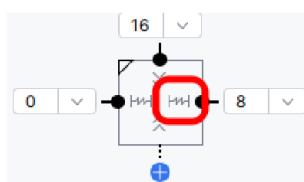
12 – Match constraint

A menudo hay ocasiones en las que queremos que un elemento se extienda todo lo posible, hasta donde le permitan las restricciones (**constraints**) que tiene configuradas.

Para hacer eso, hay que poner en **layout_width** o **layout_height** el valor **0dp**, que significa **match constraints**, o sea, que el elemento se extiende lo máximo posible que le permitan sus restricciones.

- Añade un **TextView** y configúralo de esta forma:
 - Id: **txtPreguntaDireccion**
 - Texto: **Escribe tus datos de envío**
 - Alto y ancho: El de su contenido
- Haz que la parte superior de **txtPreguntaDireccion** se pegue a **barrera2**
- Añade tres **MultilineText** y configúralos de esta forma:
 - Id: **txtNombre**, **txtDireccion**, **txtTeléfono**
 - Alto y ancho: El de su contenido
 - Hint: **Nombre completo, Dirección, Teléfono**
 - inputType: **text, text, phone**
 - Tamaño del texto: **12sp**
- Añade restricciones a **txtNombre** para que se pegue por la izquierda a **guideline3** y por la derecha al borde derecho de la pantalla, con un margen de **8dp** en dicho caso. *Si lo haces bien verás que **txtNombre** se centrará en la pantalla.*
- Cambia el **layout_width** de **txtNombre** y ponle **0dp**. *Si lo haces bien, verás que **txtNombre** se extiende en anchura todo lo que le permiten sus restricciones.*

Cuando se pone **0dp** a un elemento, su altura/anchura se extiende lo máximo que le permiten sus restricciones. Esto se puede hacer gráficamente en el **constraint widget** pinchando dentro del cuadro hasta que se vea el icono .



- Repite lo mismo con **txtDireccion** y **txtTelefono**, de forma que se extiendan en anchura por la pantalla.
- Alinea los tres cuadros de texto para que estén uno por debajo de otro
- Ejecuta la app en un móvil de tamaño mediano y verás que el formulario no cabe en la pantalla.

13 – CheckBox

El **CheckBox** es una casilla de verificación que un usuario puede marcar. Las casillas de verificación no son excluyentes

- En **buttons** localiza **Checkbox** y arrastra uno debajo del último cuadro de texto, poniéndole estas características:
 - Id: **chkAceptar**
 - Texto: **He leído y acepto los términos de uso de la app**
 - Restricciones: su borde superior estará pegado al borde inferior del último cuadro de texto, con una separación de **16dp**
 - Margen hacia el fondo de la pantalla: **64dp**

14 – ScrollView

El **ScrollView** es un tipo de **FrameLayout** que incluye barras de desplazamiento para que el usuario pueda acceder al contenido que no cabe en la pantalla.

- Abre **fragment_formulario.xml** con la vista de código fuente
- Encierra todo el contenido dentro de una etiqueta **ScrollView**, de manera que la estructura del diseño sea este:

```

1. <ScrollView
2.     xmlns:android="http://schemas.android.com/apk/res/android"
3.     xmlns:app="http://schemas.android.com/apk/res-auto"
4.     xmlns:tools="http://schemas.android.com/tools"
5.     android:layout_width="match_parent"
6.     android:layout_height="match_parent">
7.     <androidx.constraintlayout.widget.ConstraintLayout
8.         android:layout_width="match_parent"
9.         android:layout_height="match_parent">
10.         <!-- Aquí viene el resto de código de la interfaz -->
11.         <androidx.constraintlayout.widget.ConstraintLayout/>
12.     <ScrollView/>
```

- Ejecuta la app y comprueba que ahora es posible desplazar el formulario hacia arriba y abajo

15 – La Default App Bar

Las primeras apps de Android incorporaban una barra de tareas por defecto (**default app bar**) que ofrecía una serie de posibilidades fijas, como poder regresar a la pantalla anterior o mostrar un menú de opciones. Hoy día, esta barra se considera obsoleta y lo habitual es reemplazarla por otros tipos de barras con más capacidades.

- Abre el archivo `res → values → themes → themes.xml` y verás el tema usado por la app

Cada tema utiliza un tema “padre” del que hereda los estilos, y que el tema puede sobreescribir.

- Cambia el valor del atributo `parent` para que sea `Theme.MaterialComponents.DayNight.DarkActionBar`
- Ejecuta la app y comprueba que aparece una barra de tareas, que es la **default app bar**.
- Vuelve a cambiar el valor del atributo `parent` para ahora sea `Theme.MaterialComponents.DayNight.NoActionBar`

Cuando se pone un estilo que incluya `NoActionBar` significa que queremos un estilo que no utilice la barra de tareas que llevan por defecto las apps.

- Pon la app a funcionar y comprueba que, además de no mostrarse la barra de tareas, ha cambiado el color de los botones de radio y los chips al ser pulsados.

16 – MaterialToolbar

Una **MaterialToolbar** es una barra de tareas en la que habitualmente se pone el nombre de la app e incluye menús que permiten acceder a las diferentes pantallas (**Fragment**) de la app y su vuelta hacia atrás.

Cuando usamos la **MaterialToolbar**, la estructura del xml cambia para que tenga esta nueva situación:

```

1. <androidx.coordinatorlayout.widget.CoordinatorLayout
2.   xmlns:android="http://schemas.android.com/apk/res/android"
3.   xmlns:app="http://schemas.android.com/apk/res-auto"
4.   xmlns:tools="http://schemas.android.com/tools"
5.   android:layout_width="match_parent"
6.   android:layout_height="match_parent">
7.   <com.google.android.material.appbar.AppBarLayout
8.     android:layout_width="match_parent"
9.     android:layout_height="wrap_content">
10.    <!-- AQUÍ VA LA TOOLBAR -->
11.    </com.google.android.material.appbar.AppBarLayout>
12.    <androidx.core.widget.NestedScrollView
13.      android:layout_width="match_parent"
14.      android:layout_height="match_parent"
15.      app:layout_behavior="@string/appbar_scrolling_view_behavior">
16.        <!-- AQUÍ VA EL FORMULARIO -->
17.        </androidx.core.widget.NestedScrollView>
18.    </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Las etiquetas que aparecen en esta estructura son:

- **CoordinatorLayout:** Es el elemento principal de una pantalla que contiene vistas en las que se producen animaciones (como por ejemplo, desplazar un scroll) y las animaciones de una vista afectan a las otras vistas.
- **AppBarLayout:** Es un contenedor que lleva dentro una barra de tareas, y le proporciona la capacidad de realizar una animación.
- **NestedScrollView:** Es una versión mejorada del **ScrollView** que puede ser colocada en un **CoordinatorLayout** y así, coordinar el desplazamiento del formulario con el de la barra de tareas.

- Abre **fragment_formulario.xml** con la vista de código fuente
- Borra las etiquetas relacionadas con el **ScrollView**, de forma que se quede el formulario plano, sin posibilidad de moverse.

Hacemos esto porque al reestructurar el archivo **fragment_formulario.xml** para incluir la **MaterialToolbar**, estas volverán a aparecer.

- Dentro de la etiqueta **AppBarLayout**, coloca la etiqueta **MaterialToolbar**, de esta forma:

```

1. <com.google.android.material.appbar.MaterialToolbar
2.     android:id="@+id/toolbar"
3.     style="@style/Widget.MaterialComponents.Toolbar.Primary"
4.     android:layout_width="match_parent"
5.     android:layout_height="?attr/actionBarSize"/>

```

El atributo **style** sirve para asignar un estilo a la barra de progreso. En este proyecto estamos usando un estilo predefinido que está en la librería **Material Design**.

Cuando se pone **?attr actionBarSize** significa que la altura la coge de una variable (atributo) definida en el tema

- A continuación, abrimos el archivo **FormularioFragment** y añadimos un método **inicializarToolbar** en el que tenemos que llamar a **setSupportActionBar** para indicar que nuestra **MaterialToolbar** sustituye a la antigua barra de progreso (**default app bar**) de las apps de Android

```

1. class FormularioFragment : Fragment() {
2.     // resto omitido
3.     override fun onCreateView(
4.         inflater: LayoutInflater, container: ViewGroup?,
5.         savedInstanceState: Bundle?
6.     ): View? {
7.         inicializarBinding(inflater, container)
8.         inicializarMaterialToolbar()
9.         return binding.root
10.    }
11.    private fun inicializarMaterialToolbar(){
12.        val mainActivity:MainActivity = activity as MainActivity
13.        mainActivity.setSupportActionBar(binding.toolbar)
14.    }
15. }

```

En las primeras versiones de Android, las apps tenían siempre una barra de tareas (**default app bar**), que no se podía cambiar. Actualmente, podemos añadir otras barras de tareas que aportan más funciones (como la **MaterialToolbar**), y cuando la ponemos, debemos llamar al método **setSupportActionBar** para que haga que esa barra sustituya a la barra de tareas antigua.

El método **setSupportActionBar** se encuentra en **MainActivity**, pero estamos programando dentro de **FormularioFragment** y no tenemos acceso a dicho método.

Para conseguir tener acceso, obtenemos el **activity** en el que está **FormularioFragment** y le hacemos un casting² al tipo **MainActivity**.

- Ejecuta la app y comprueba que aparece una barra de tareas fija, pero el formulario vuelve a poder desplazarse

El formulario ahora se puede mover porque se encuentra en un **NestedScrollView**, que es una versión del **ScrollView** que es compatible con la **MaterialToolbar**

17 – Scrollable toolbar

Ahora mismo la barra de tareas está fija y cuando nos desplazamos por el formulario siempre está visible.

Una técnica que se usa actualmente para ahorrar espacio consiste en que al bajar por el formulario, la barra se desplace y desaparezca, volviendo a aparecer cuando se sube al principio del formulario. Esto se llama una **scrollable toolbar**

- Coloca el atributo **layout_scrollFlags** con valores **scroll|enterAlways** en la etiqueta **MaterialToolbar**

```
1. <com.google.android.material.appbar.MaterialToolbar
2.     android:id="@+id/toolbar"
3.     style="@style/Widget.MaterialComponents.Toolbar.Primary"
4.     app:layout_scrollFlags="scroll|enterAlways"
5.     android:layout_width="match_parent"
6.     android:layout_height="?attr/actionBarSize"
7.     app:layout_scrollFlags="scroll|enterAlways"/>
```

El atributo **app:layout_scrollFlags** permite indicar el comportamiento de la barra de tareas en relación al desplazamiento de la interfaz.

- **scroll** → indica que la barra de tareas desaparece y aparece con el formulario
- **enterAlways** → la barra recupera su posición de forma más rápida cuando el usuario desplaza hacia arriba el formulario

² En Kotlin el casting se hace con la palabra **as**

18 – Collapsing toolbar

Una **collapsing toolbar** es una barra de tareas que tiene una altura grande y que se colapsa en la barra de tareas normal cuando el usuario desplaza hacia arriba el formulario.

- Copia el archivo adjunto al proyecto **pizzeria.webp** en la carpeta **res/drawable**
- Borra el atributo **style** de la etiqueta **MaterialToolbar**
- Coloca en la etiqueta **AppBarLayout**, el atributo **android:theme**, indicando el estilo que va a tener la etiqueta **AppBarLayout** y todas las etiquetas que contiene:

```
1. <com.google.android.material.appbar.AppBarLayout  
2.     android:layout_width="match_parent"  
3.     android:layout_height="wrap_content"  
4.     android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar">
```

Al usar **android:theme** el estilo que se pone es aplicado a todos los elementos contenidos en la etiqueta **AppBarLayout**.

Es necesario hacer este paso para que la **collapsing toolbar** se vea correctamente (si no se pone, se ve una pequeña línea dividiendo la barra de tareas con la parte extendida)

- Dentro de **AppBarLayout**, incluye una **CollapsingToolbarLayout** que contenga a la **MaterialToolbar** (*observa que la línea 10, donde definimos **app:layout_scrollFlags**, se borra de la **MaterialToolbar** porque se sube al **CollapsingToolbarLayout***)

```
1. <com.google.android.material.appbar.CollapsingToolbarLayout  
2.     android:id="@+id/collapsing_toolbar_layout"  
3.     android:layout_width="match_parent"  
4.     android:layout_height="300dp"  
5.     app:layout_scrollFlags="scroll|exitUntilCollapsed">  
6.     <com.google.android.material.appbar.MaterialToolbar  
7.         android:id="@+id/materialToolbar"  
8.         android:layout_width="match_parent"  
9.         android:layout_height="?attr/actionBarSize"  
10.        app:layout_scrollFlags="scroll|enterAlways"/>  
11.    </com.google.android.material.appbar.CollapsingToolbarLayout>
```

La etiqueta **CollapsingToolbarLayout** convierte una **MaterialToolbar** en una **collapsing toolbar**.

El atributo **app:layout_scrollFlags** indica como es el desplazamiento de la barra, y poniendo combinación **scroll|exitUnitCollapsed** hacemos que se colapse la barra

- Ejecuta la app y comprueba que la barra de tareas tiene una altura muy grande (300dp) y que al deslizar el formulario, se colapsa.

Ahora mismo la barra se ve transparente porque tenemos que ponerle un fondo, que puede ser un color sólido o una imagen.

- Para poner un color sólido a la barra, añadimos el atributo **android:background** a la etiqueta **CollapsingToolBarLayout**, indicando el color que queremos usar.

```
1. <com.google.android.material.appbar.CollapsingToolbarLayout  
2.     android:id="@+id/collapsing_toolbar_layout"  
3.     android:layout_width="match_parent"  
4.     android:layout_height="300dp"  
5.     app:contentScrim="?attr/colorPrimary"  
6.     android:background="?attr/colorPrimary"  
7.     app:layout_scrollFlags="scroll|exitUntilCollapsed">
```

- **android:background** → Permite indicar un color sólido de fondo
- **app:contentScrim** → Es el color que queremos poner a la barra cuando esta se encuentre colapsada.

- Por último, para poner una imagen de fondo a la barra de tareas, inserta un **ImageView** justo después de la etiqueta **CollapsingToolbarLayout**, así: (*no olvides borrar el atributo android:background*)

```
1. <com.google.android.material.appbar.AppBarLayout  
2.     android:layout_width="match_parent"  
3.     android:layout_height="wrap_content"  
4.     android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar">  
5.     <com.google.android.material.appbar.CollapsingToolbarLayout  
6.         android:id="@+id/collapsing_toolbar_layout"  
7.         android:layout_width="match_parent"  
8.         android:layout_height="300dp"  
9.         app:contentScrim="?attr/colorPrimary"  
10.        app:layout_scrollFlags="scroll|exitUntilCollapsed">  
11.        <ImageView  
12.            android:layout_width="match_parent"  
13.            android:layout_height="match_parent"  
14.            android:scaleType="centerCrop"  
15.            android:src="@drawable/pizzeria"  
16.            app:layout_collapseMode="parallax"/>  
17.        <com.google.android.material.appbar.MaterialToolbar  
18.            android:id="@+id/toolbar"  
19.            android:layout_width="match_parent"  
20.            android:layout_height="?attr/actionBarSize" />  
21.    </com.google.android.material.appbar.CollapsingToolbarLayout>  
22. </com.google.android.material.appbar.AppBarLayout>
```

El atributo **app:layout_collapseMode** con valor **parallax** sirve para indicar que la imagen se irá colapsando a una velocidad diferente al desplazamiento, produciendo así un efecto de profundidad.

19 – Floating action button

Un **Floating Action Button (FAB)** es un botón que está flotando en la pantalla y que por su apariencia, destaca sobre todos los demás

- Abre el archivo **fragment_formulario.xml** con la vista de código fuente

- Justo al final del archivo, en el momento en que se cierra la etiqueta **NestedScrollView**, añade la etiqueta **FloatingActionButton**:

- Id: **btnEnviar**
- Icono: **@android:drawable/ic_input_add**
- **layout_gravity: bottom|end**
- **layout_margin:16dp**

```

1. <com.google.android.material.floatingactionbutton.FloatingActionButton
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:id="@+id	btnEnviar"
5.     android:layout_gravity="bottom|end"
6.     android:layout_margin="16dp"
7.     android:src="@android:drawable/ic_input_add"/>

```

El atributo **layout_gravity** permite indicar hacia donde queremos que el botón se ubique en la pantalla

- Ejecuta la app y verás que el botón aparece pegado a la esquina inferior derecha de la pantalla y permanece flotante, de manera que podemos subir y bajar por el formulario sin que el botón cambie de posición.

Como puedes ver, en esa ubicación el botón tapa elementos interactuables de la interfaz, con lo que molesta. Vamos a hacer que se pegue a la **collapsing toolbar**

- Abre el archivo **fragment_formulario.xml** con la vista de código fuente
- Borra el atributo **layout_gravity** del **FloatingActionButton**
- Añade al **FloatingActionButton** los atributos **app:layout_anchor** y **app:layout_anchorGravity** de esta forma:

```

1. <com.google.android.material.floatingactionbutton.FloatingActionButton
2.     android:layout_width="wrap_content"
3.     android:layout_height="wrap_content"
4.     android:id="@+id	btnEnviar"
5.     app:layout_anchor="@+id/collapsing_toolbar_layout"
6.     app:layout_anchorGravity="bottom|end"
7.     android:src="@android:drawable/ic_input_add"/>

```

- **app:layout_anchor**: Indica donde queremos colocar el botón flotante
- **app:layout_anchorGravity**: Indica donde deberá posicionarse el botón flotante, en relación al lugar donde lo hayamos colocado con **app:layout_anchor**

20 – Snackbar

Un **Snackbar** es un botón alargado que aparece en la pantalla por un cierto tiempo, como los **Toast**, pero el usuario puede interactuar con ellos.

Los **Snackbar** permiten ser deslizados, incluir botones que lanzan acciones, etc.

- Haz que al pulsar el botón **btnEnviar** se compruebe si la casilla **chkAceptar** está habilitada. En caso de que no lo esté, se mostrará un **Snackbar** informando al usuario que debe aceptar los términos de uso

```

1. private fun inicializarBotones(){
2.     // resto omitido
3.     binding.btnEnviar.setOnClickListener{
4.         if(!binding.chkAceptar.isChecked){
5.             Snackbar.make(
6.                 binding.btnEnviar, // elemento que origina el snackbar
7.                 "Debe aceptar los términos para continuar", // texto del snackbar
8.                 Snackbar.LENGTH_SHORT
9.             ).show()
10.        }
11.    }
12. }
```

El método **Snackbar.make** nos permite crear un **Snackbar** pasando los siguientes parámetros:

- El objeto que da lugar al **Snackbar**
- El texto del **Snackbar**
- La duración en la pantalla

Una vez creado el **Snackbar**, su método **show** lo muestra en pantalla

- Ejecuta la app y comprueba que al pulsar **btnEnviar** sin haber aceptado los términos, se muestra el **Snackbar** y que puedes deslizarlo para que desaparezca
- Añade al **Snackbar** una opción para que el usuario pueda aceptar los términos dentro del **Snackbar**, haciendo que el formulario baje hasta el fondo.

```

1. binding.btnEnviar.setOnClickListener{
2.     if(!binding.chkAceptar.isChecked){
3.         Snackbar.make(
4.             binding.btnEnviar, // elemento que origina el snackbar
5.             "Debe aceptar los términos para continuar", // texto del snackbar
6.             Snackbar.LENGTH_SHORT
7.         ).setAction("Aceptar términos"){
8.             binding.nestedScrollView.fullScroll(View.FOCUS_DOWN)
9.         }.show()
10.    }
11. }
```

El método **setAction** del **Snackbar** sirve para añadir una opción al **Snackbar**. Recibe como parámetros el texto del mensaje y un bloque con el código que se va a ejecutar cuando se pulsa la opción.

- Añade a **FormularioFragment** un método auxiliar **getPizzaSeleccionada** que nos devuelva el nombre de la pizza seleccionada en los botones de radio

```

1. private fun getPizzaSeleccionada():String {
2.     var resultado="No hay ningún tipo de pizza seleccionado"
3.     val idSeleccionado = binding.grupoPizza.checkedRadioButtonId
4.     if(idSeleccionado!= -1) { // si es -1 es porque no hay ningún botón seleccionado
5.         val botonSeleccionado: RadioButton = binding.root.findViewById(idSeleccionado)
6.         resultado=botonSeleccionado.text.toString()
7.     }
8.     return resultado
9. }
```

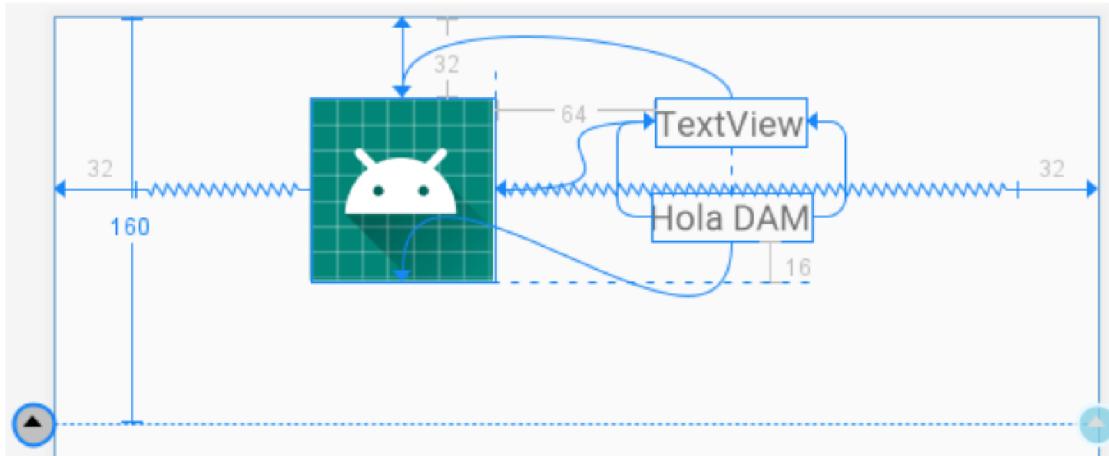
El **RadioGroup** nos permite acceder al id (tal y como sale en el xml) del botón de radio que está seleccionado. Eso nos permite recuperar dicho botón con el método **findViewById** del objeto **binding.root**, que es el contenedor padre de la interfaz

- Haz que si **chkAceptar** está marcado, al pulsar **btnEnviar** se muestre un **Snackbar** con el tipo de pizza elegida.

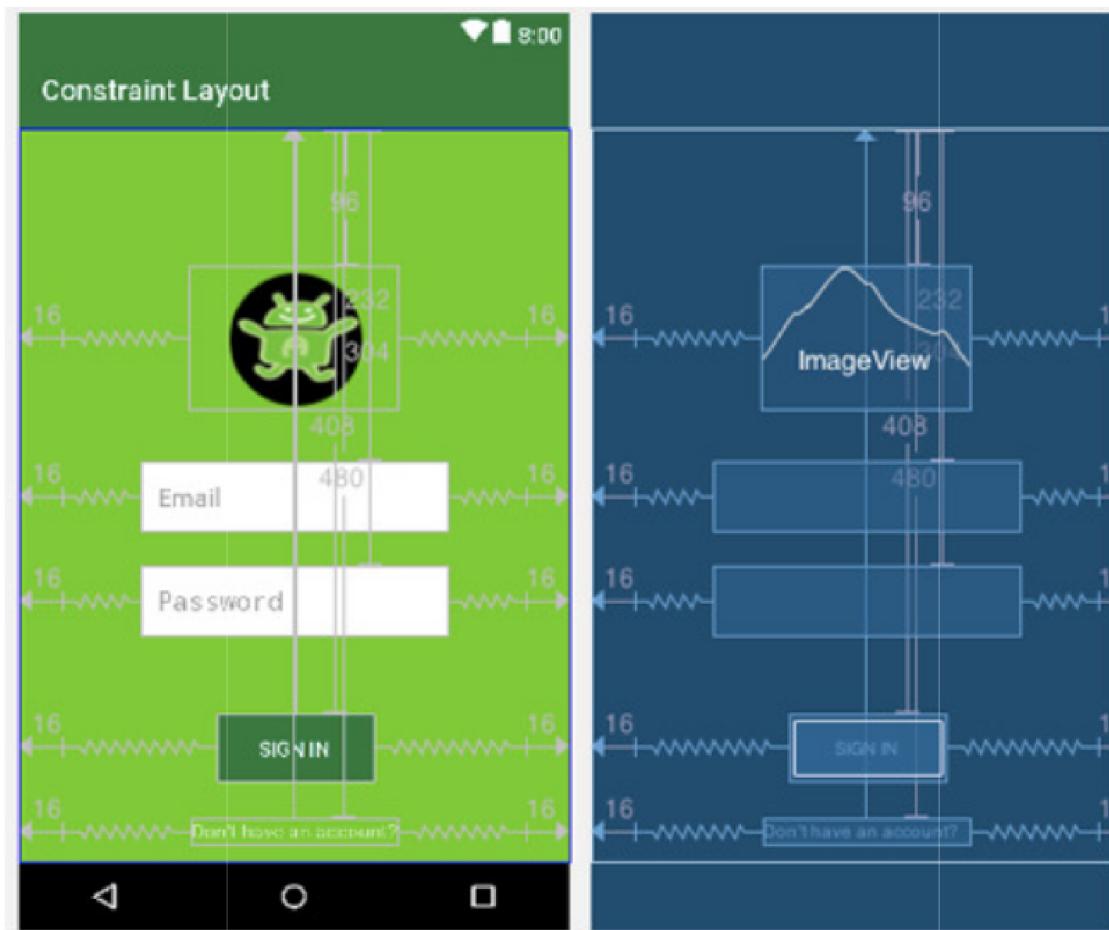
```
1. private fun inicializarBotones(){
2.     // resto omitido
3.     binding.btnEnviar.setOnClickListener{
4.         if(!binding.chkAceptar.isChecked){
5.             // if omitido
6.         }else{
7.             Snackbar.make(
8.                 binding.btnEnviar,
9.                 "Has elegido pizza ${getPizzaSeleccionada()}",
10.                Snackbar.LENGTH_SHORT
11.            ).show()
12.        }
13.    }
14. }
```

Ejercicios

Ejercicio 1: Diseña la siguiente interfaz usando un **ConstraintLayout** y las medidas que se indican:



Ejercicio 2: Diseña la siguiente interfaz, usando las separaciones proporcionadas:



Ejercicio 3: Haz un **Fragment** que tenga la siguiente interfaz, y haz que se muestre en **MainActivity**

Name

Middle Name

Last Name

Phone: 299358 Type: Business

Email

Address: Home

Birthday: 1/1/0001

Group

Save To

Sim

Añade un **FloatingActionButton** pegado a la zona inferior derecha de la pantalla, y que al pulsarlo muestre un **Toast** indicando si falta algún campo por llenar.

Ejercicio 4: Crea una app que calcule el cambio de monedas, siguiendo estas instrucciones:

- Habrá un **EditText** para escribir una cantidad de dinero
- Habrá dos **Spinner**, uno para elegir la moneda origen del cambio y otra para elegir la moneda destino. Deberá poderse cambiar de dinero entre euros, dólares, libras y yens
- Habrá un **Button** que al pulsarlo, permitirá mostrar en un **Snackbar** el resultado de la conversión
- Pon a cada elemento de los **Spinner** un ícono con la bandera de su país correspondiente. *Pista: Usa el atributo **drawable** de los ítem del **Spinner** para establecer los iconos.*