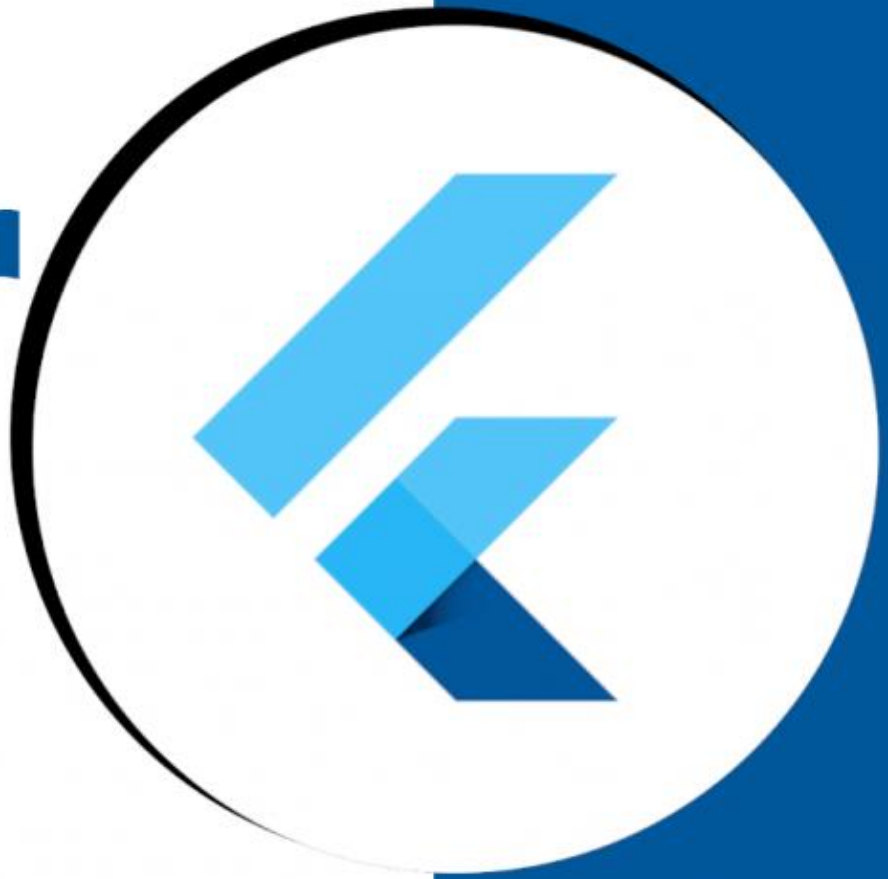
Widgets Flutter

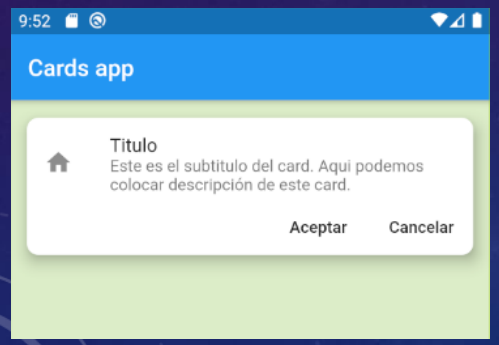


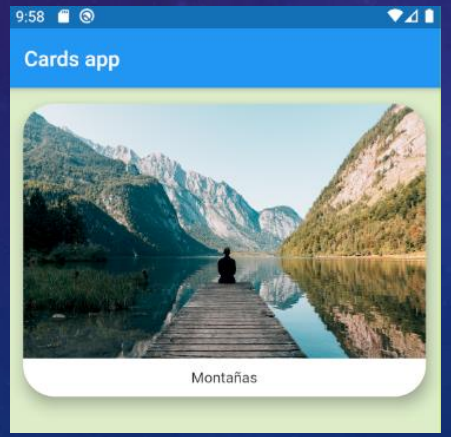
DESARROLLO PARA DISPOSITIVOS INTELIGENTES

GILBERTO RODRÍGUEZ RAMÍREZ

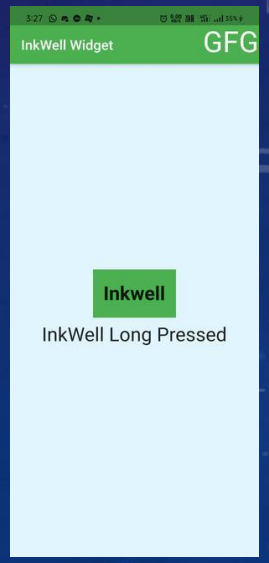
CARD: Es un widget que nos proporciona el aspecto visual de una tarjeta. Crear un card en Flutter se reduce a usar el widget **card( )**, al cual mediante sus propiedades le daremos el aspecto que queramos.

Es importante mencionar que un Card no es más que una especie de contenedor con algunas caracteristicas que lo hacen especial. Al fin al cabo tenemos que diseñar el Card utilizando Column o Row para colocar y ordenar elementos en su interior.





InkWell: **E**s el widget de material en flutter. Responde a la acción táctil realizada por el usuario. Inkwell responderá cuando el usuario haga clic en el botón. Hay tantos gestos como tocar dos veces, presionar prolongadamente, tocar hacia abajo, etc.



INK: Un práctico widget para dibujar imágenes y otras decoraciones en los widgets de materiales , de modo que las salpicaduras de InkWell e InkResponse se representen sobre ellos.

Este widget dibuja la Decoración dada directamente en el Material , de la misma manera que InkWell e InkResponse dibujan allí. Esto permite que las salpicaduras se dibujen sobre los gráficos opacos.

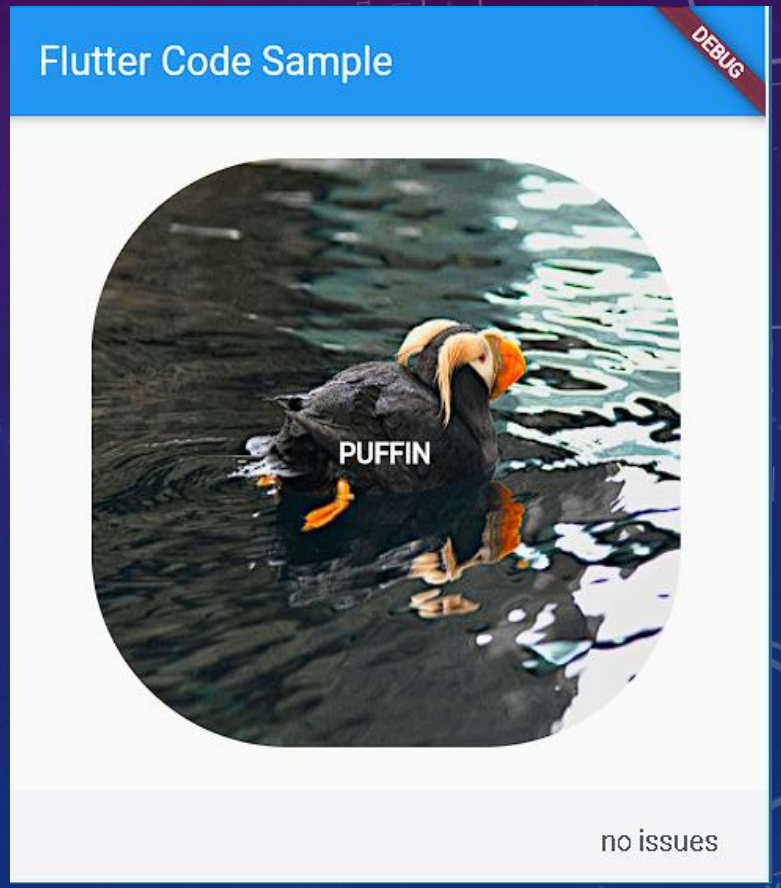
Constructores

- Ink({Key? key, EdgeInsetsGeometry? padding, Color? color, Decoration? decoration, double? width, double? height, Widget? child})

  Pinta una decoración (que puede ser un color simple) en un Material

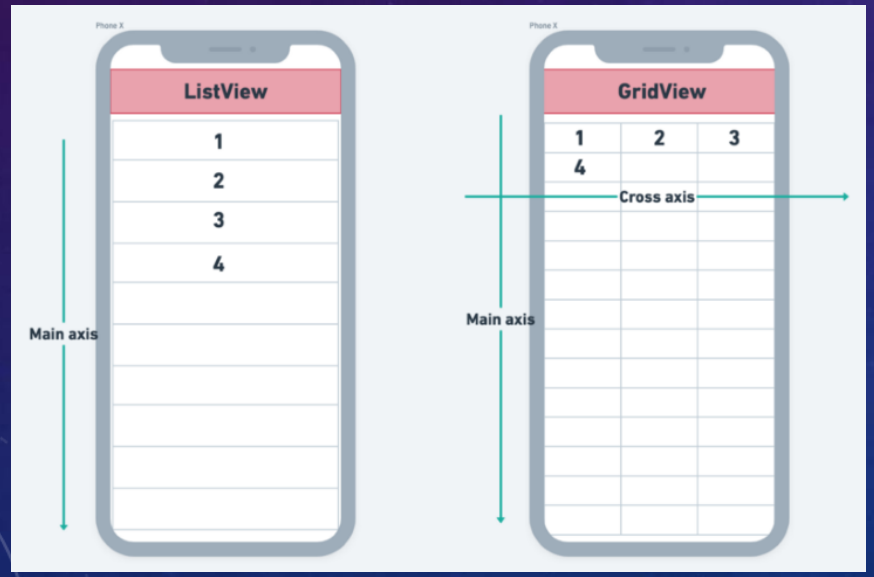
- Ink.image({Key? key, EdgeInsetsGeometry? padding, required ImageProvider<Object> image, ImageErrorListener? onImageError, ColorFilter? colorFilter, BoxFit? fit, AlignmentGeometry alignment = Alignment.center, Rect? centerSlice, ImageRepeat repeat = ImageRepeat.noRepeat, bool matchTextDirection = false, double? width, double? height, Widget? child})

  Crea un widget que muestra una imagen (obtenida de un ImageProvider) en un Material



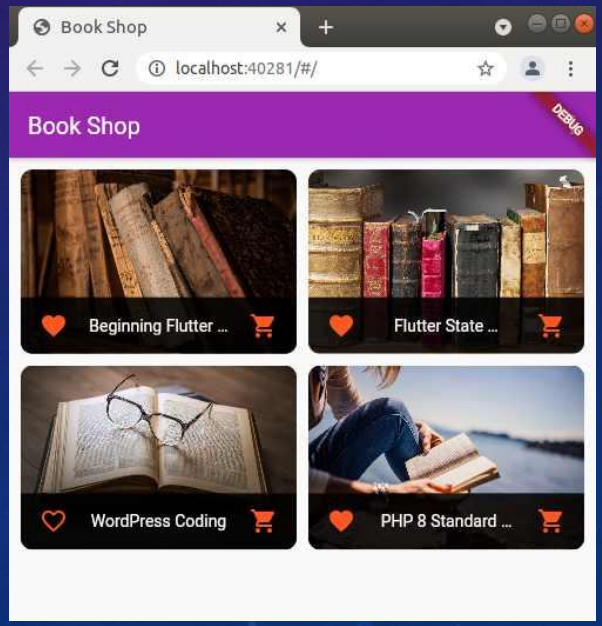
GridView: En Flutter, GridView es un widget que muestra una lista de elementos como una matriz 2D. En términos simples, los elementos se muestran en formato de tabla.

A diferencia de una lista normal, en la que los elementos se representan solo en una dirección, GridView representa los elementos tanto horizontal como verticalmente. La siguiente figura representa cómo GridView es diferente de una lista normal en una aplicación Flutter



GridTile: en Flutter es una subclase de Stateless Widget, que es una cuadrícula de mosaicos desplazable. Y GridTile generalmente usa GridTileBar Widget en el encabezado o pie de página. No ambos, al mismo tiempo.

También GridTile debe tener un hijo. Sin embargo, el widget sin estado GridTileBar que devuelve un widget IconButton.



GridTileBar: Un encabezado utilizado en un GridTile de diseño de materiales .

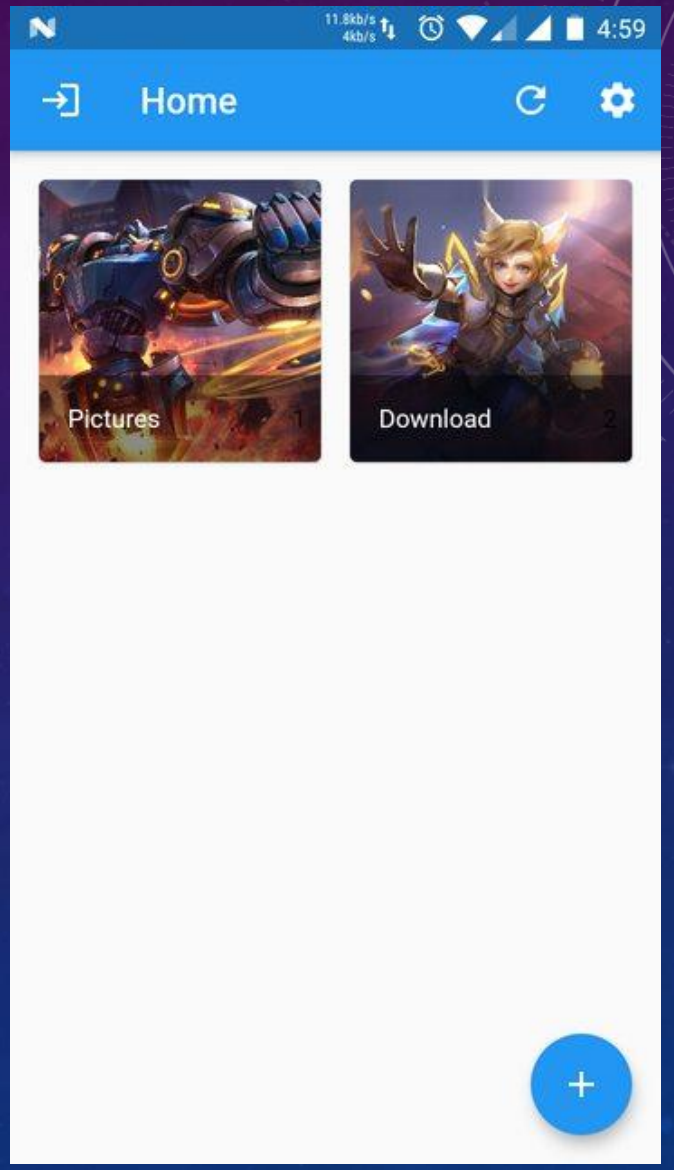
Por lo general, se usa para agregar un encabezado o pie de página de una o dos líneas en un GridTile .

Para un encabezado de una línea, incluya un widget de título . Para agregar una segunda línea, incluya también un widget de subtítulos . Use el encabezado o el final para agregar un ícono.

Constructor

GridTileBar({Key? key, Color? backgroundColor, Widget? leading, Widget? title, Widget? subtitle, Widget? trailing})

Crea una barra de mosaico de cuadrícula



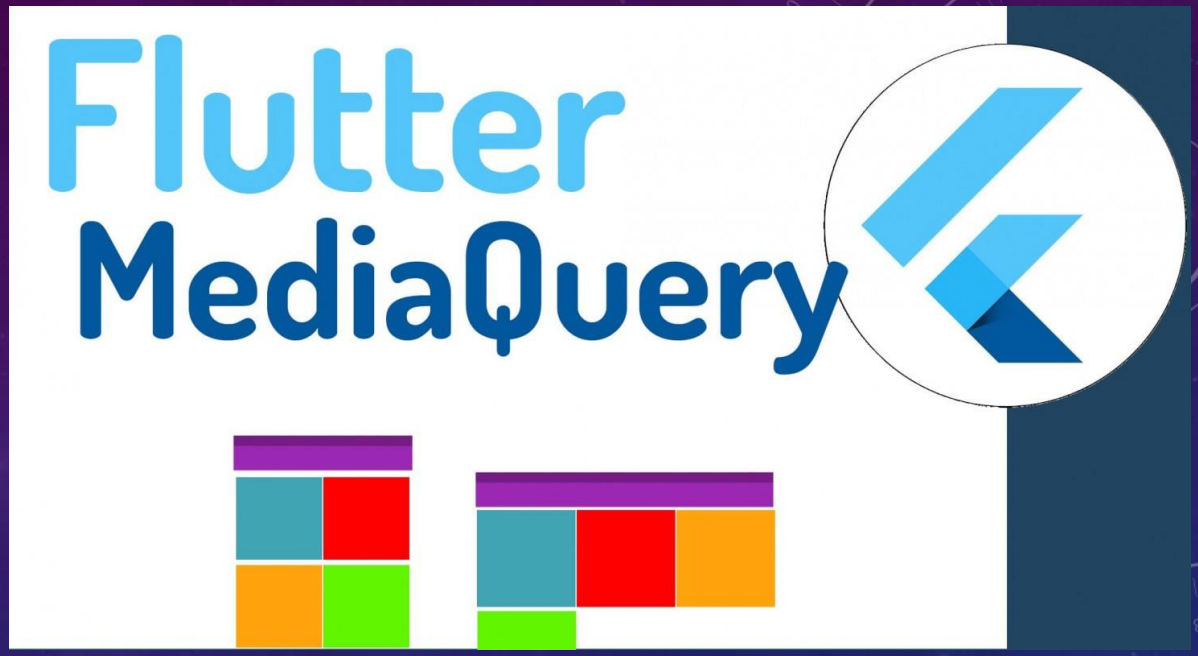
SEGUNDA

PARTE

MediaQuery

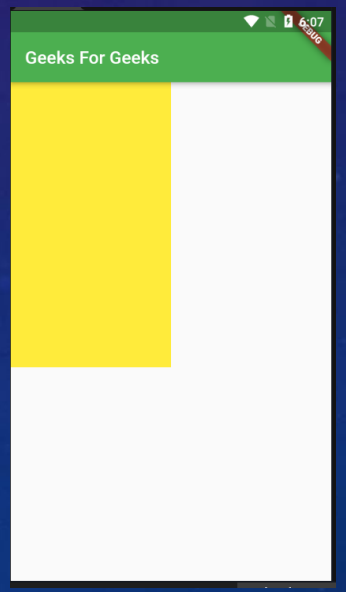
En Flutter tenemos una clase que nos permite dar rápidamente información sobre el equipo que está renderizando nuestra aplicación, datos como la orientación y el tamaño total de pantalla y con esto hacer nuestros cálculos.

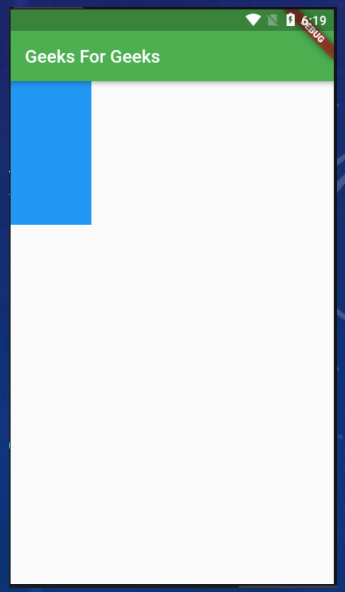
La clase de MediaQuery tiene un comportamiento o idea similar al que tenemos en CSS con cuentan con el mismo nombre de esta tecnología, adaptar la aplicación en base a reglas que definimos sea por la orientación o el tamaño de pantalla:




```
@media (max-width: 600px) {
  .facet_sidebar {
    display: none;
  }
}

@media (min-width: 700px) and (orientation: landscape) { ... }
```





LayoutBuilder

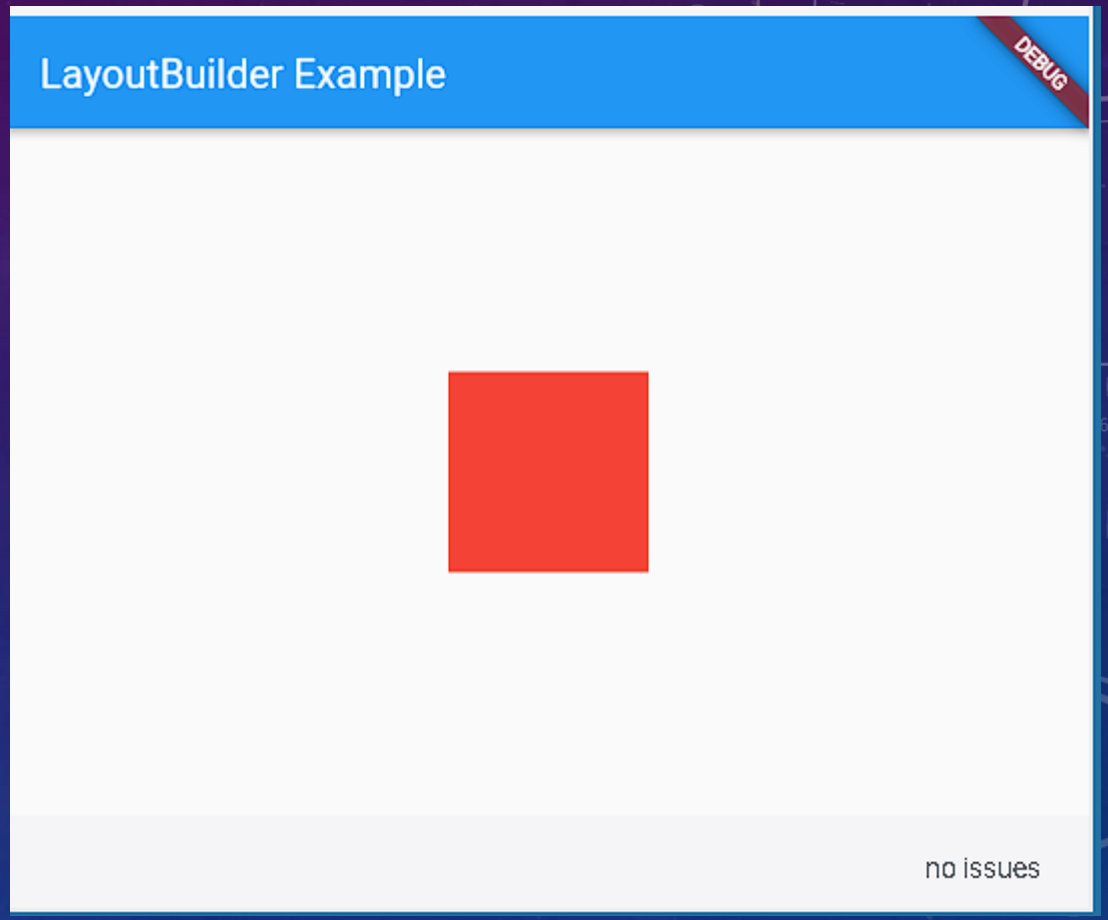
Crea un árbol de widgets que puede depender del tamaño del widget principal.

Similar al widget de Generador , excepto que el marco llama a la función de Generador en el momento del diseño y proporciona las restricciones del widget principal. Esto es útil cuando el padre restringe el tamaño del niño y no depende del tamaño intrínseco del niño. El tamaño final de LayoutBuilder coincidirá con el tamaño de su hijo.

Constructores

LayoutBuilder ({ Clave ? clave , constructor LayoutWidgetBuilder requerido })

Crea un widget que difiere su construcción hasta el diseño.



OrientationBuilder

En algunas situaciones, desea actualizar la visualización de una aplicación cuando el usuario gira la pantalla del modo vertical al modo horizontal. Por ejemplo, la aplicación puede mostrar un elemento tras otro en modo vertical, pero colocar esos mismos elementos uno al lado del otro en modo horizontal.

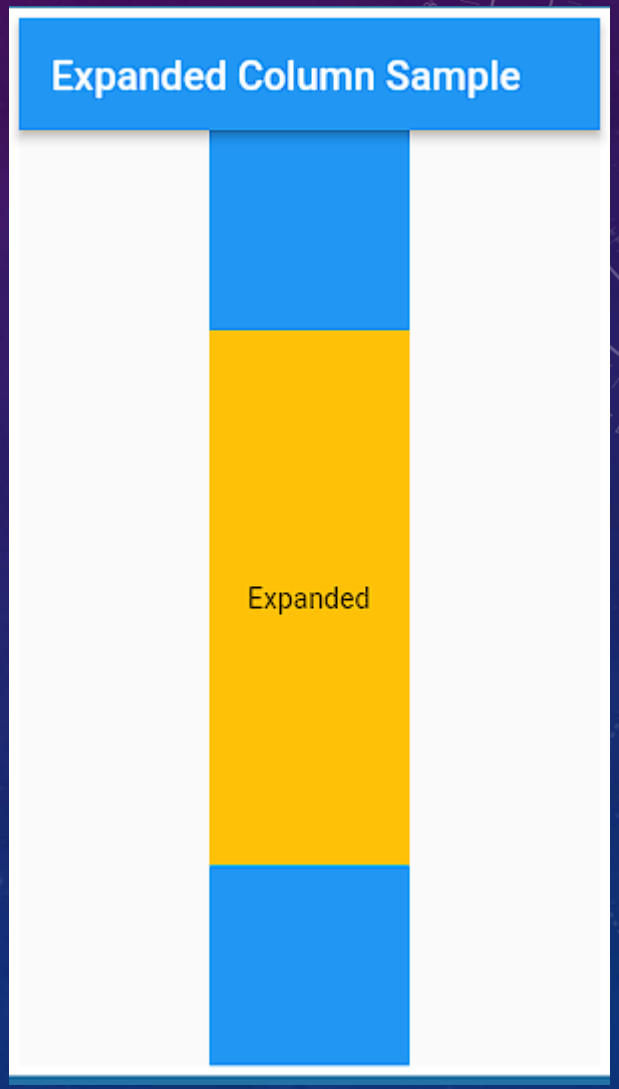

```
cuerpo : OrientationBuilder (
  constructor : ( contexto , orientación ) { return GridView . count ( // Crea
      crossAxisCount : orientación == Orientación . retrato ? 2 : 3 , ); }, ),
```

Expanded

Un widget que expande un elemento secundario de Row , Column o Flex para que el elemento secundario llene el espacio disponible.

El uso de un widget Expandido hace que un hijo de Fila, Columna o Flex se expanda para llenar el espacio disponible a lo largo del eje principal (por ejemplo, horizontalmente para una Fila o verticalmente para una Columna). Si se expanden varios hijos, el espacio disponible se divide entre ellos según el factor flexible.

Un widget Expandido debe ser un descendiente de Row , Column o Flex , y la ruta desde el widget Expandido hasta la Fila , Columna o Flex que lo contiene debe contener solo StatelessWidget s o StatefulWidget s (no otros tipos de widgets, como RenderObjectWidget s).



Flexible

Un widget que controla cómo se flexiona un elemento secundario de Row , Column o Flex .

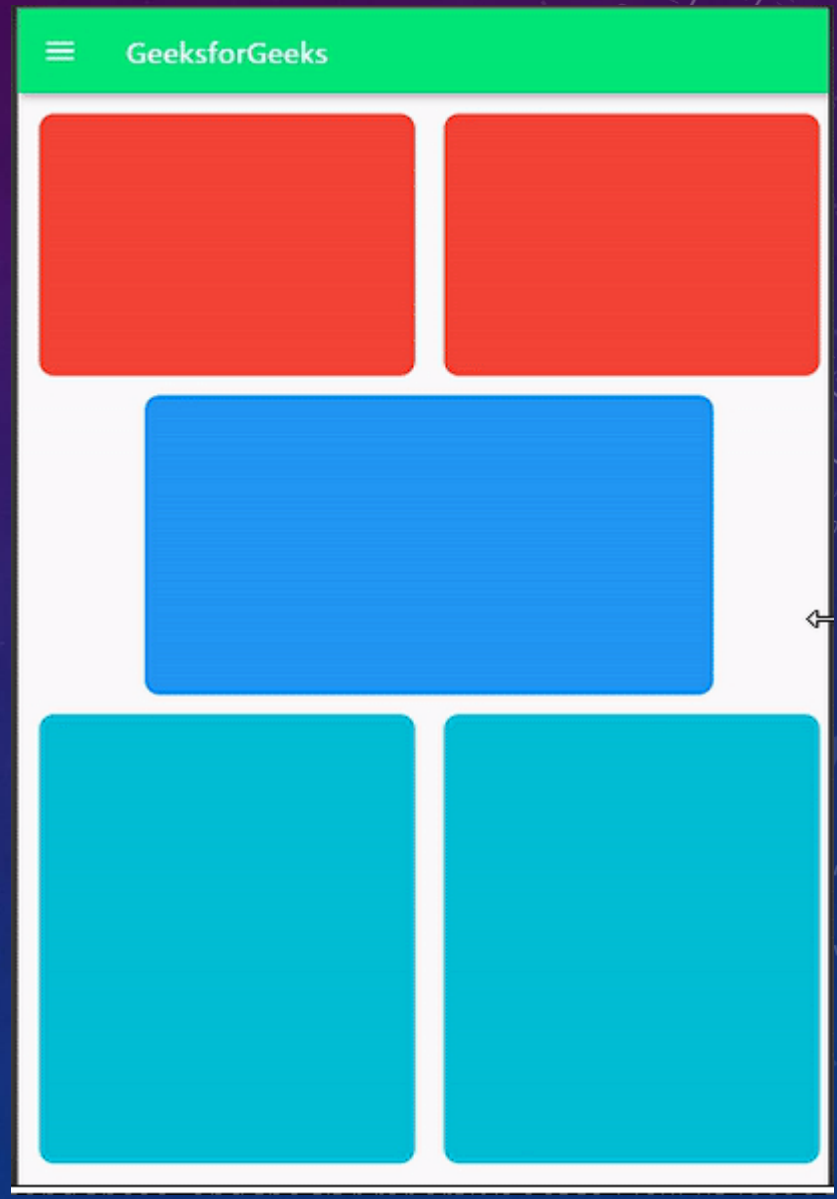
El uso de un widget Flexible le da a un hijo de Fila , Columna o Flex la flexibilidad de expandirse para llenar el espacio disponible en el eje principal (por ejemplo, horizontalmente para una Fila o verticalmente para una Columna), pero, a diferencia de Expandido , Flexible no lo hace. exigir que el niño llene el espacio disponible.

Un widget flexible debe ser un descendiente de Row , Column o Flex , y la ruta desde el widget Flexible hasta su Row , Column o Flex adjunto debe contener solo StatelessWidget s o StatefulWidget s (no otros tipos de widgets, como RenderObjectWidgets).

Constructor

Flexible({Key? key, int flex = 1, FlexFit fit = FlexFit.loose, required Widget child})

Crea un widget que controla cómo se flexiona un elemento secundario de Row , Column o Flex



FractionallySizedBox

FractionallySizedBox Widget es un widget que dimensiona a su hijo a una fracción del espacio total disponible. Para que sea fácil de entender cómo funciona FractionallySizedBox , por ejemplo, hay un contenedor con un tamaño de 200 x 100. Dentro, hay un botón en relieve y vamos a establecer el tamaño del botón en relación con el tamaño del contenedor.

```
Contenedor (
    altura: 24,0,
    color: Colors.black,
    hijo: Columna (
        hijos: [
            Flexible (
                hijo: FractionallySizedBox (
                    factor de altura: 1, factor de ancho: 0,25,
                    hijo: Contenedor (color: Colors.orange),
                ),
            ),
            Fexible (
                child: FractionallySizedBox(
                    heightFactor: 1, widthFactor: 0.15,
                    child: Container(color: Colors.green)),
            ),
            Fexible(
                child: FractionallySizedBox(
                    heightFactor: 1, widthFactor: 0.05,
                    niño: Contenedor (color: Colors.blue)
                ),
            ),
        ]
    )
)
```

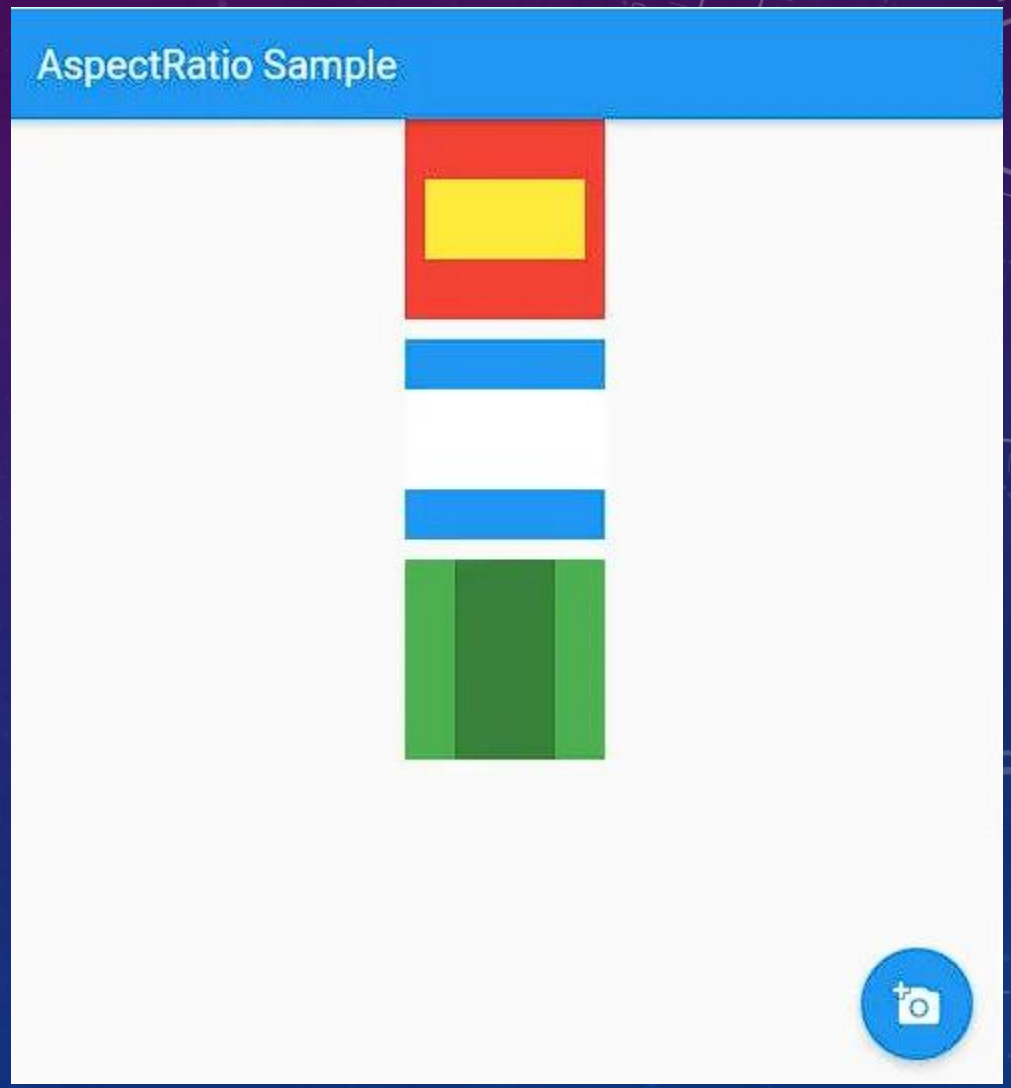
AspectRatio

El widget AspectRatio intenta ajustar el tamaño del niño a una relación de aspecto específica.

Supongamos que tenemos un widget de contenedor con un ancho de 100 y un alto de 100. En ese caso, la relación de aspecto sería 100/100; es decir, 1.0.

Ahora, cada Widget tiene sus propias restricciones . Como resultado, AspectRatio Widget intenta encontrar el mejor tamaño para mantener la relación de aspecto. Sin embargo, al hacerlo, respeta sus restricciones de diseño.

```
AspectRatio(
    aspectRatio: 3/2
    Child: myWidget(),
)
```

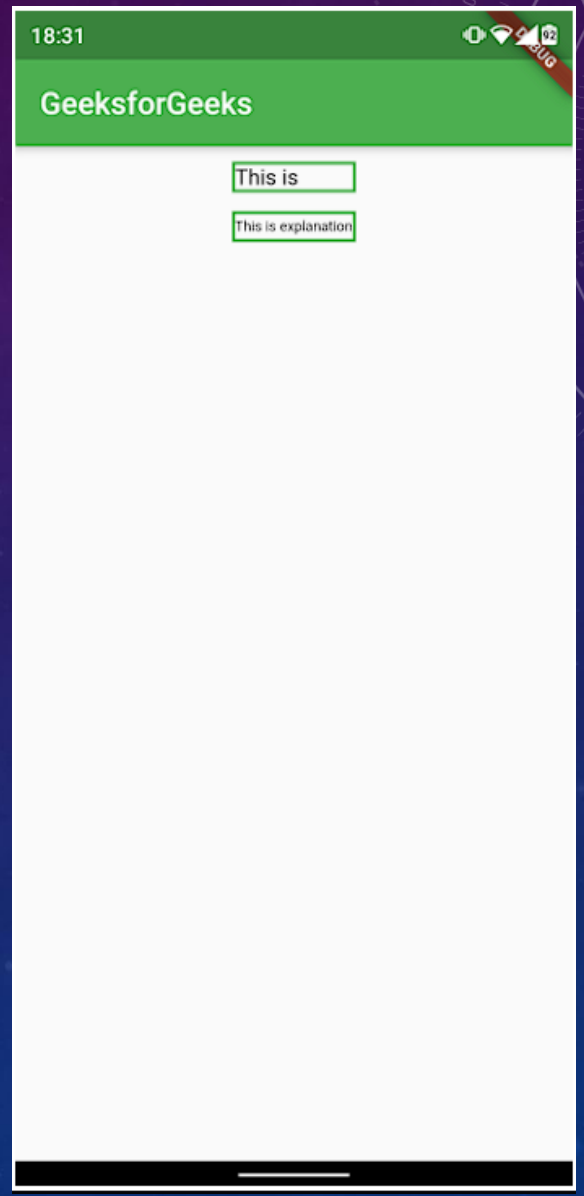


FittedBox

FittedBox restringe el crecimiento de sus widgets secundarios más allá de cierto límite. Los vuelve a escalar según el tamaño disponible. Por ejemplo, si el texto se muestra dentro de un contenedor y el usuario debe ingresar el texto. Si el usuario ingresa una gran cadena de texto, el contenedor crecerá más allá de su tamaño asignado. Pero, si lo envolvemos con FittedBox, se ajustaría al texto según el tamaño disponible. Para una cuerda grande, reduciría su tamaño y, por lo tanto, cabría en el contenedor.

Sintaxis:

```
CajaAjustada({
    Llave llave,
    Ajuste BoxFit: BoxFit.contain,
    AlignmentGeometry alineación: Alignment.center,
    niño widget
     }
)
```



Wrap

Un widget que muestra sus elementos secundarios en varias ejecuciones horizontales o verticales.

Un Wrap coloca a cada niño e intenta colocar al niño junto al niño anterior en el eje principal, dado por la dirección , dejando espacio en el medio. Si no hay suficiente espacio para que quepa el elemento secundario, Wrap crea un nuevo tramo adyacente a los elementos secundarios existentes en el eje transversal.

Una vez que se han asignado todos los elementos secundarios a las ejecuciones, los elementos secundarios dentro de las ejecuciones se colocan de acuerdo con la alineación en el eje principal y de acuerdo con crossAxisAlignment en el eje transversal.

Las carreras en sí se colocan en el eje transversal de acuerdo con runSpacing y runAlignment .

```
Wrap({
  Key key,
  this.direction = Axis.horizontal,
  this.alignment = WrapAlignment.start,
  this.spacing = 0.0,
  this.runAlignment = WrapAlignment.start,
  this.runSpacing = 0.0,
  this.crossAxisAlignment = WrapCrossAlignment.start,
  this.textDirection,
  this.verticalDirection = VerticalDirection.down,
  this.clipBehavior = Clip.hardEdge,
  List children = const [],
})
```