SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER APPLICATIONS

23CAT607- CROSS-PLATFORM APP DEVELOPMENT

I YEAR II SEM

UNIT 3 – INTRODUCTION TO LAYOUTS

TOPIC 1 – Type of Layout Widgets

One thing to keep in mind that “Everything in Flutter is Widget”.

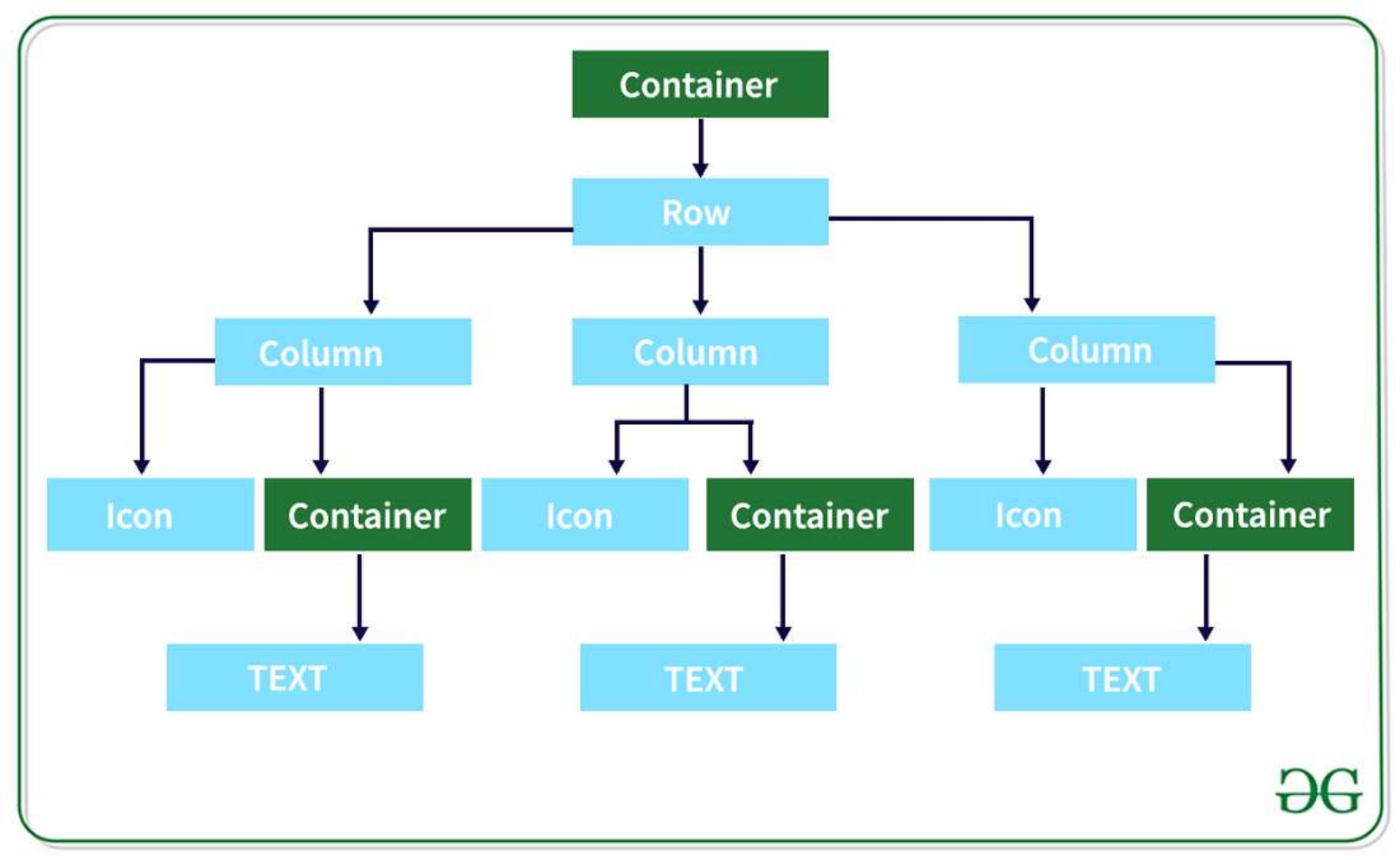
Meaning the core of the layout in any Flutter Application is the widget. Putting it simply, all the images, icons, labels and text, etc are technically widgets of different types and layouts.

The concept let’s take a single example and break down those components for better understanding.

The first screenshot below shows 3 icons with a label under each one

The second screenshot displays the visual layout, showing a row of 3 columns where each column contains an icon and a label.

Widget Tree Diagram.



Parent widget is a row widget, inside that 3rd column widget and in each column, there is an icon and inside the container, there is a text widget.

LAY OUT A WIDGET

Select a layout widget

Choose from a variety of layout widgets based on how you want to align or constrain the visible widget, as these characteristics are typically passed on to the contained widget.

This example uses Center which centers its content horizontally and vertically.

Create a visible widget

For example, create a Text widget:

```
Text('Hello World'),
```

Create an Image widget:

```
return Image.asset(
  image,
  fit: BoxFit.cover,
);
```

Create an Icon widget:

```
Icon(
  Icons.star,
  color: Colors.red[500],
),
```

Add the visible widget to the layout widget

All layout widgets have either of the following:

- A child property if they take a single child—for example, Center or Container
- A children property if they take a list of widgets—for example, Row, Column, ListView, or Stack.

Add the Text widget to the Center widget:

```
const Center(
  child: Text('Hello World'),
),
```

Add the layout widget to the page

For a Material app, you can use a Scaffold widget; it provides a default banner, background color, and has API for adding drawers, snack bars, and bottom sheets. Then you can add the Center widget directly to the body property for the home page.

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    const String appTitle = 'Flutter layout demo';
    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Non-Material apps

For a non-Material app, you can add the Center widget to the app's build() method:

```dart
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: const BoxDecoration(color: Colors.white),
      child: const Center(
        child: Text(
          'Hello World',
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 32,
            color: Colors.black87,
          ),
        ),
      ),
    );
  }
}
```

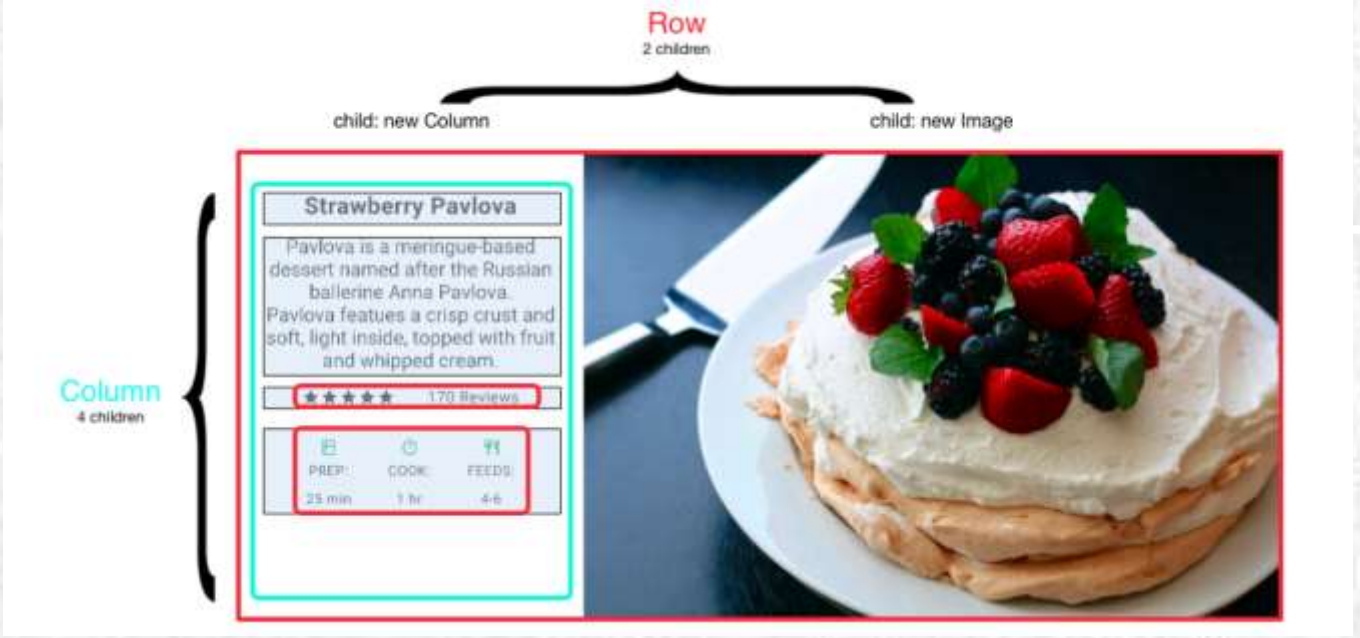
Layout | Flutter

LAY OUT MULTIPLE WIDGETS VERTICALLY AND HORIZONTALLY

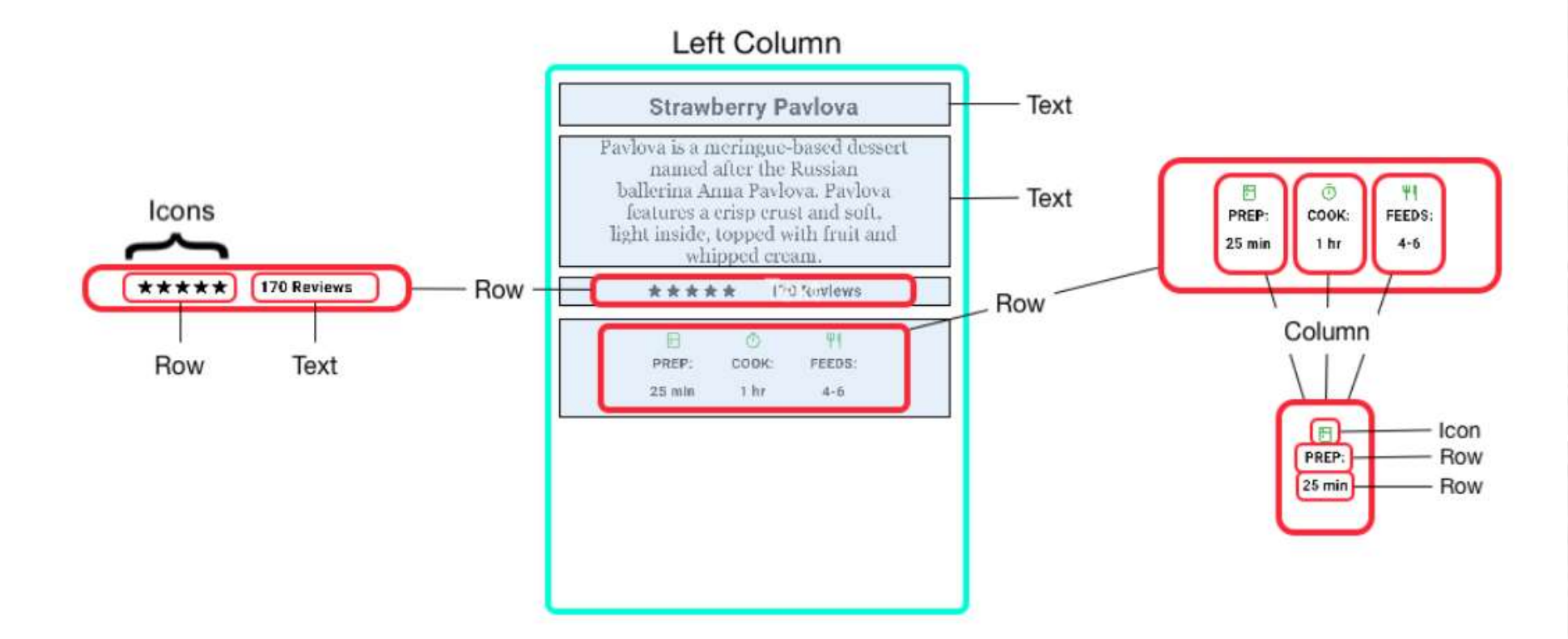
What's the point?

- `Row` and `Column` are two of the most commonly used layout patterns.
- `Row` and `Column` each take a list of child widgets.
- A child widget can itself be a `Row`, `Column`, or other complex widget.
- You can specify how a `Row` or `Column` aligns its children, both vertically and horizontally.
- You can stretch or constrain specific child widgets.
- You can specify how child widgets use the `Row`'s or `Column`'s available space.

This layout is organized as a `Row`. The row contains two children: a column on the left, and an image on the right:

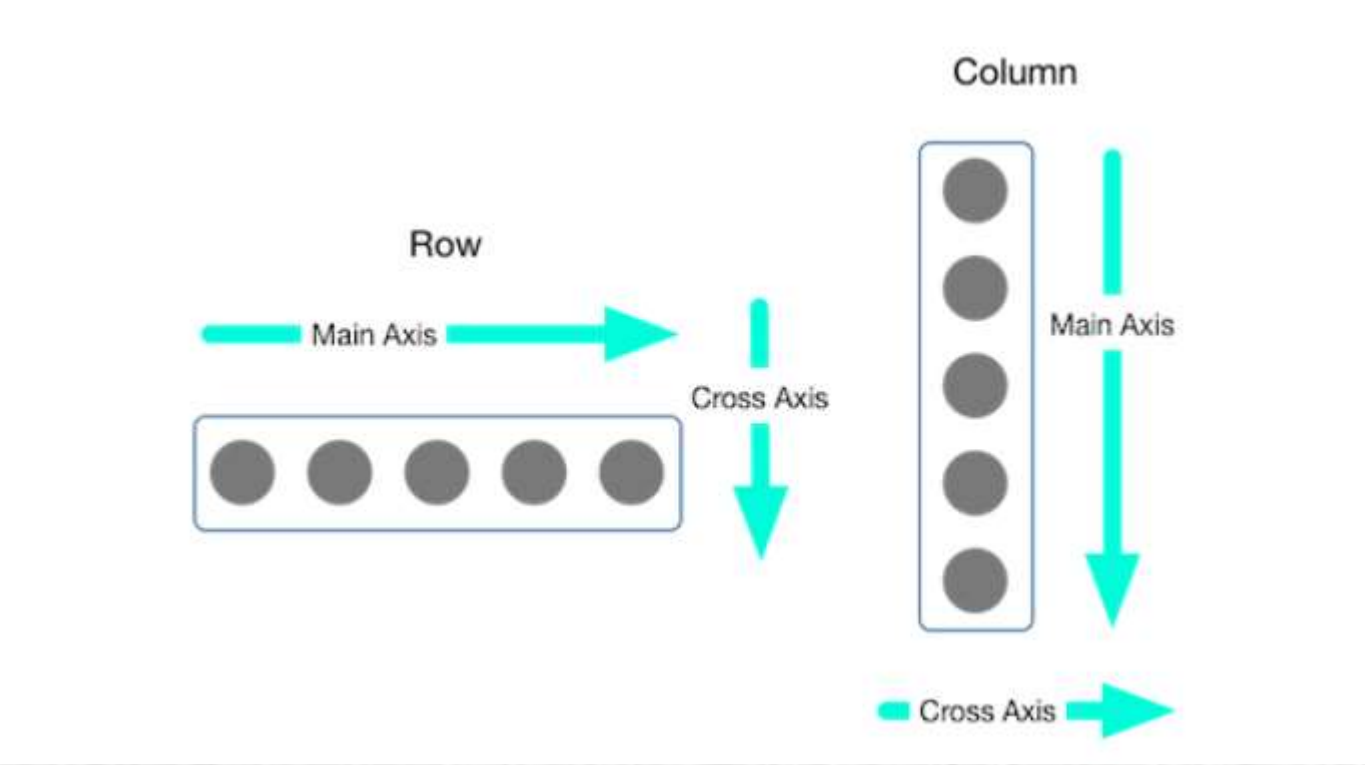


The left column's widget tree nests rows and columns.



Aligning widgets

You control how a row or column aligns its children using the mainAxisAlignment and crossAxisAlignment properties




```
Row( mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
children: [
Image.asset('images/pic1.jpg'),
Image.asset('images/pic2.jpg'),
Image.asset('images/pic3.jpg'),
 ],
 );
```

Sizing widgets

Widgets can be sized to fit within a row or column by using the Expanded widget.

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic2.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic3.jpg'),
    ),
  ],
);
```

App source: sizing

Packing widgets

```
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    const Icon(Icons.star, color: Colors.black),
    const Icon(Icons.star, color: Colors.black),
  ],
)
```

App source: pavlova

Common layout widgets

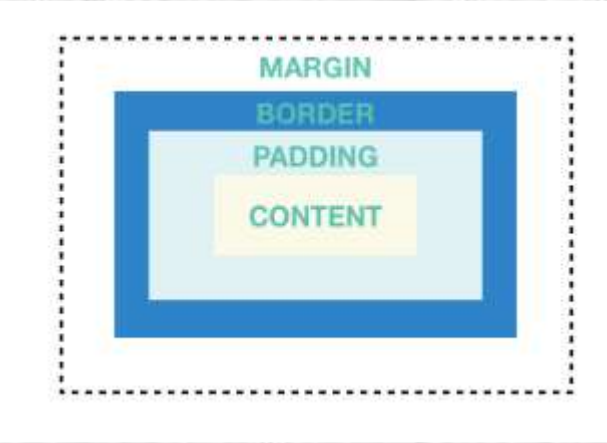
Standard widgets#

- Container: Adds padding, margins, borders, background color, or other decorations to a widget.
- GridView: Lays widgets out as a scrollable grid.
- ListView: Lays widgets out as a scrollable list.
- Stack: Overlaps a widget on top of another.

Material widgets#

- Card: Organizes related info into a box with rounded corners and a drop shadow.
- ListTile: Organizes up to 3 lines of text, and optional leading and trailing icons, into a row.

Container#




```
Widget _buildImageColumn()
{
  return Container(
    decoration: const BoxDecoration(
      color: Colors.black26,
    ),
    child: Column(
      children: [
        _buildImageRow(1),
        _buildImageRow(3),
      ],
    ),
  );
}
```

GridView

- Build your own custom grid, or use one of the provided grids:
  - GridView.count allows you to specify the number of columns
  - GridView.extent allows you to specify the maximum pixel width of a tile

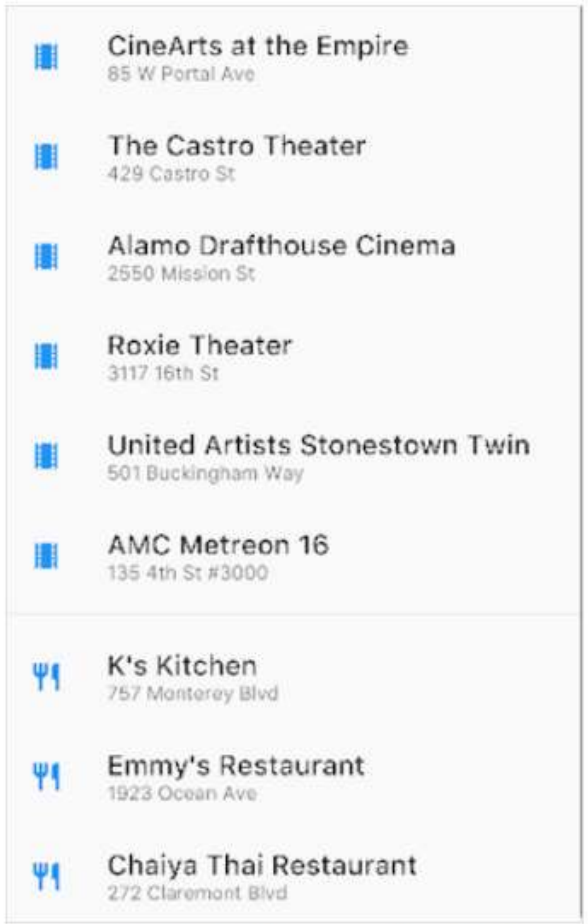
```
Widget _buildGrid() => GridView.extent(
    maxCrossAxisExtent: 150,
    padding: const EdgeInsets.all(4),
    mainAxisSpacing: 4,
    crossAxisSpacing: 4,
    children: _buildGridTileList(30));

// The images are saved with names pic0.jpg, pic1.jpg...pic29.jpg.
// The List.generate() constructor allows an easy way to create
// a list when objects have a predictable naming pattern.
List<Container> _buildGridTileList(int count) => List.generate(
    count, (i) => Container(child: Image.asset('images/pic$i.jpg')));
```

Uses `GridView.extent` to create a grid with tiles a maximum 150 pixels wide.

App source: grid_and_list

ListView

ListView, a column-like widget, automatically provides scrolling when its content is too long for its render box.



Uses `ListView` to display a list of businesses using `ListTile`s. A `Divider` separates the theaters from the restaurants.

App source: grid_and_list

```
Widget _buildList() {
 return ListView(
  children: [
   _tile('CineArts at the Empire', '85 W Portal Ave', Icons.theaters),
   _tile('The Castro Theater', '429 Castro St', Icons.theaters),
],
 );
}

ListTile _tile(String title, String subtitle, IconData icon) {
 return ListTile(
  title: Text(title,
    style: const TextStyle(
     fontWeight: FontWeight.w500,
     fontSize: 20,
    )),
  subtitle: Text(subtitle),
  leading: Icon(
   icon,
   color: Colors.blue[500],
  ), );}
```


thank
you