Resource Software Solution

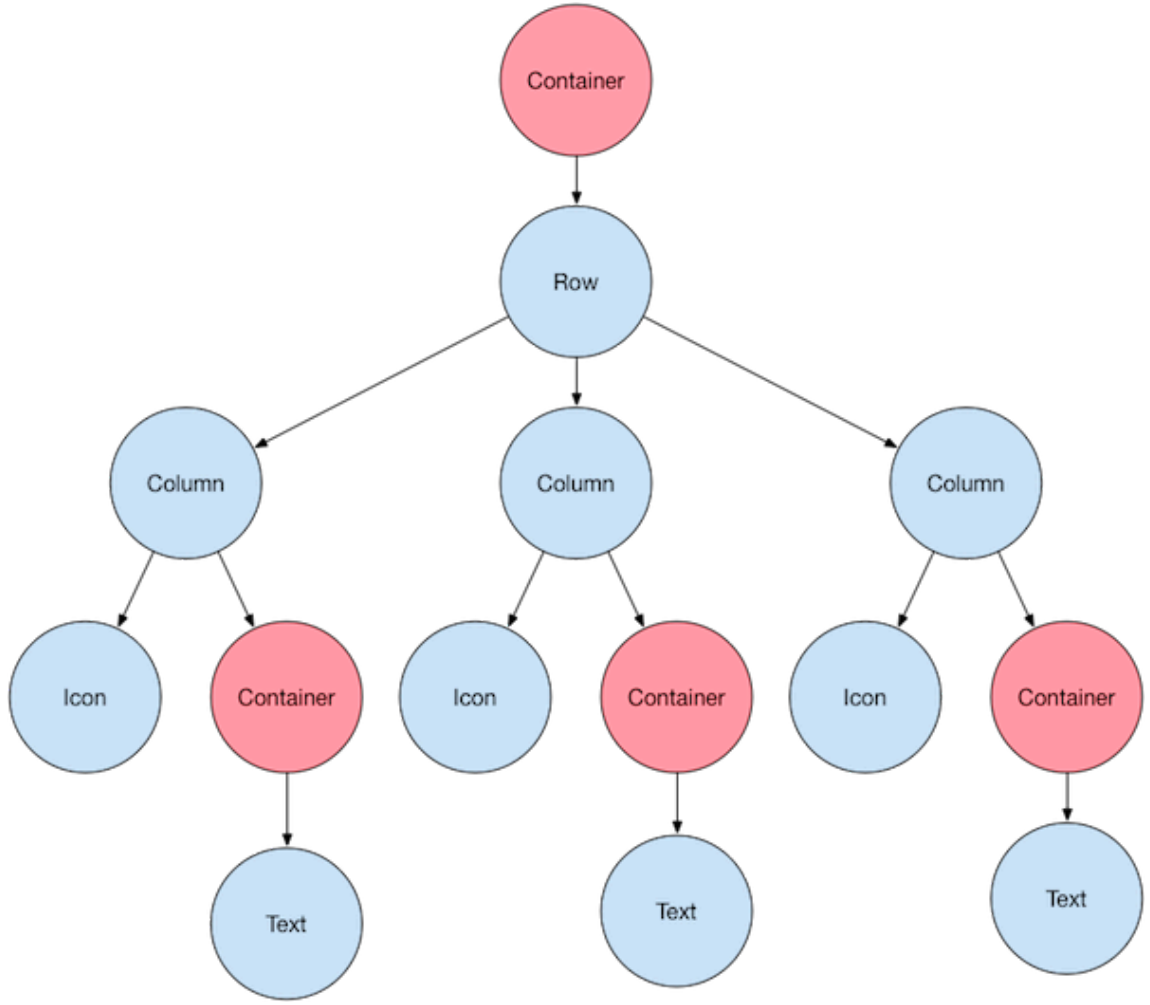
Building Layouts

Layouts in Flutter

- Widgets are classes used to build UIs.
- Widgets are used for both **layout** and **UI elements**
  - The images, icons, and text that you **see** in a Flutter app are all widgets (**UI elements**).
  - But things you **don't see** are also widgets, such as the rows, columns, and grids that arrange, constrain, and align the visible widgets (**layout**).

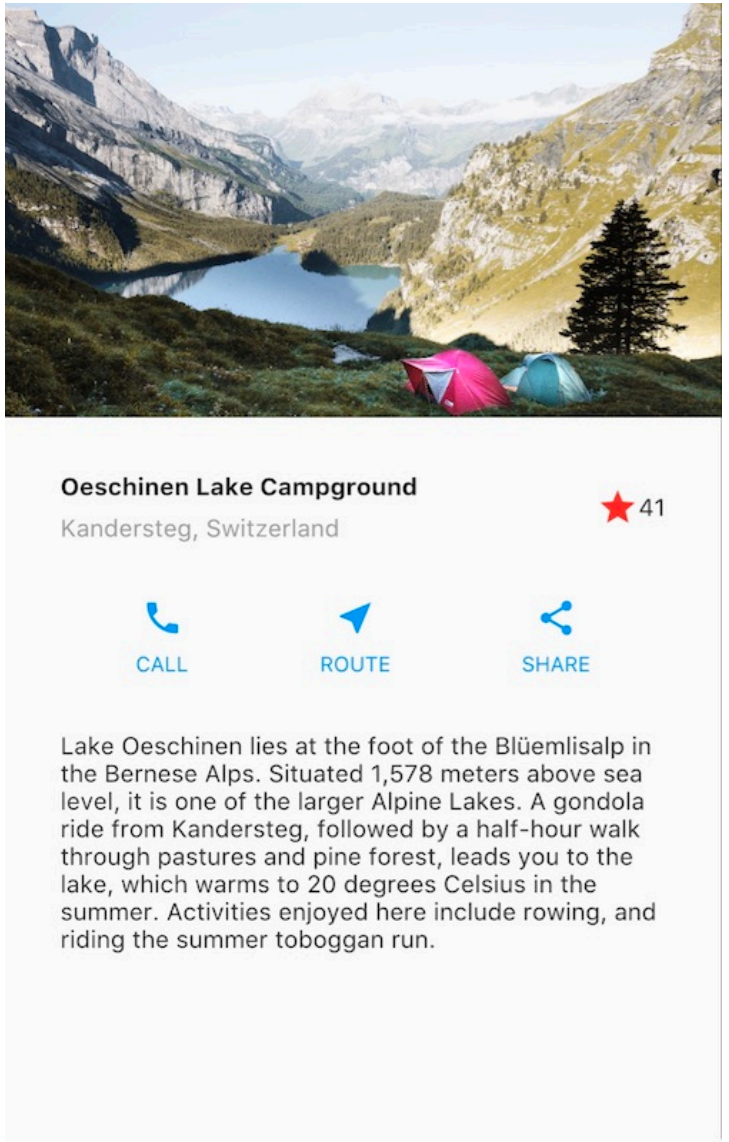
Layouts in Flutter

- For example, the this screenshot below shows 3 icons with a label under each one.



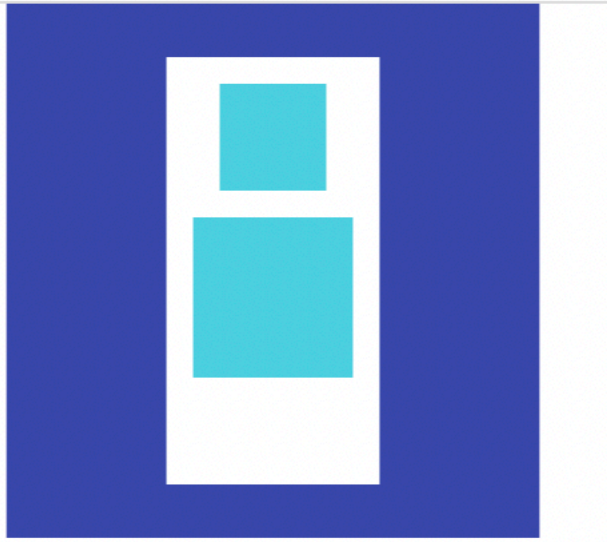
How to build a layout

- Step 1: Select a **layout widget**
- Step 2: Create a visible widget (**UI elements**)
- Step 3: Add the **visible widget** to the **layout widget**
- Step 4: Add the **layout widget** to the **page**



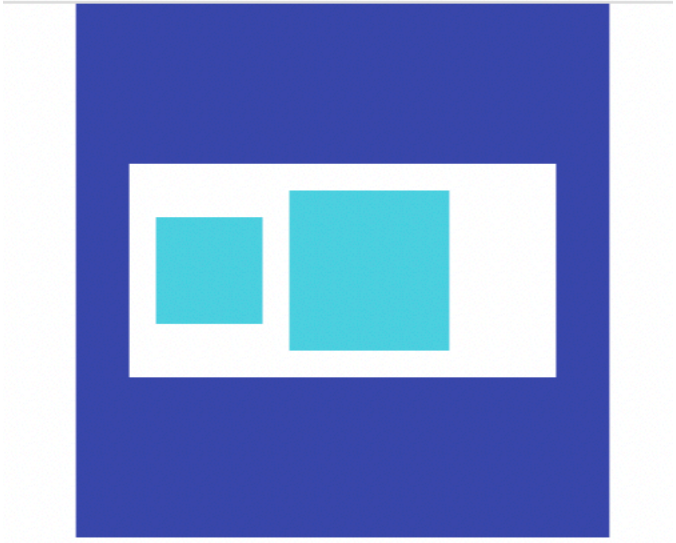
How to build a layout

- **Step 1: Select a layout widget**
  - Choose from a variety of layout widgets based on how you want to align



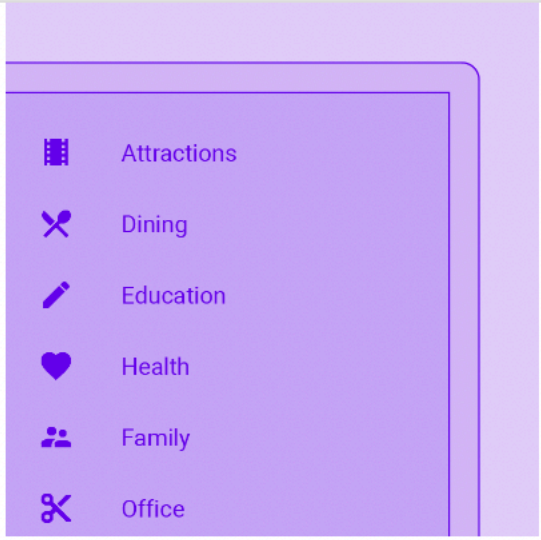
Column

Layout a list of child widgets in the vertical direction.



Row

Layout a list of child widgets in the horizontal direction.



ListView

A scrollable, linear list of widgets. ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction....

How to build a layout

- **Step 2: Create a visible widget**
  - For example, create a **`Text`** widget:
  - For example, create an **`Image`** widget:
  - For example, create an **`Icon`** widget:

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

```
Image.asset(
  'images/lake.jpg',
),
```

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

How to build a layout

- **Step 3: 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`**
  - For example:

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

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

How to build a layout

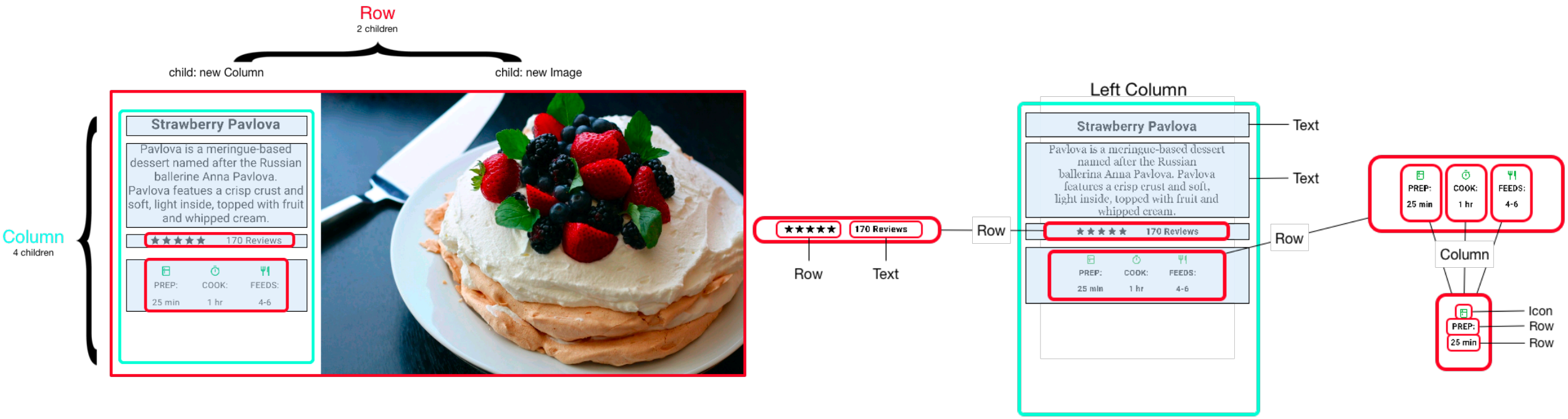
- **Step 4: Add the layout widget to the page**
  - Instantiating and returning a widget in the app's **build()** method displays the widget.

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

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

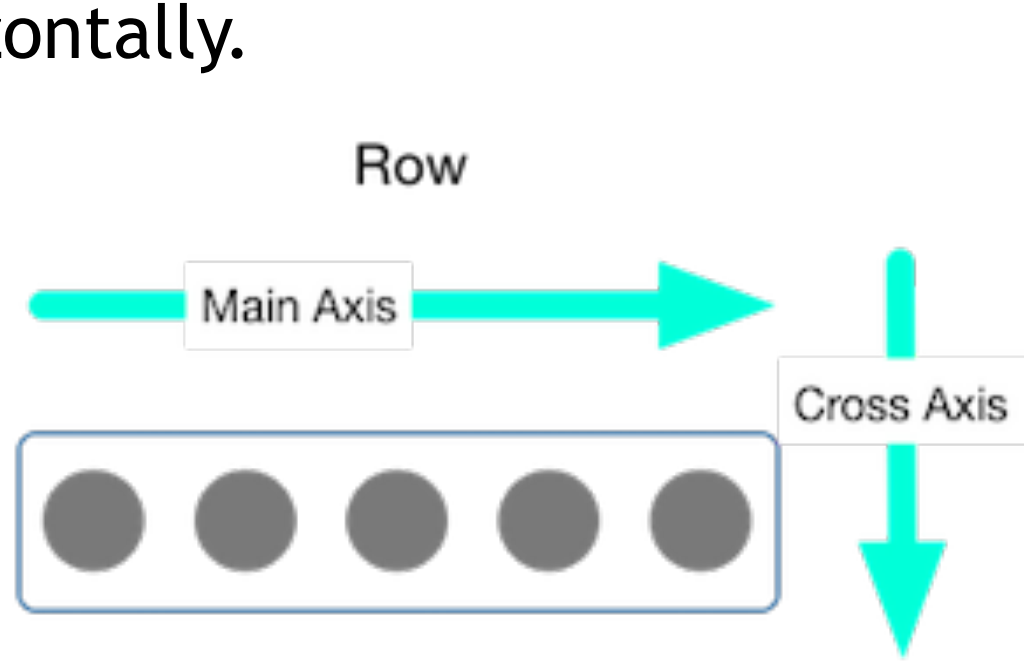
Multiple widgets

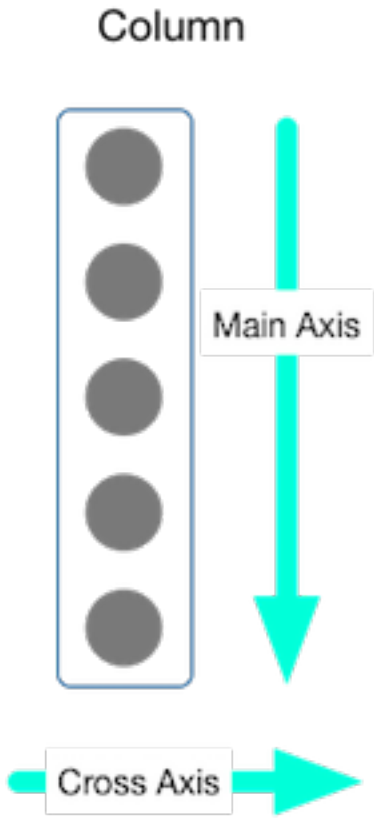
- You can use a **`Row`** widget to arrange widgets **horizontally**, and a **`Column`** widget to arrange widgets **vertically**.



Aligning widgets

- You control how a row or column aligns its children using the **`mainAxisAlignment`** and **`crossAxisAlignment`** properties.
- For a row, the main axis runs horizontally and the cross axis runs vertically. For a column, the main axis runs vertically and the cross axis runs horizontally.

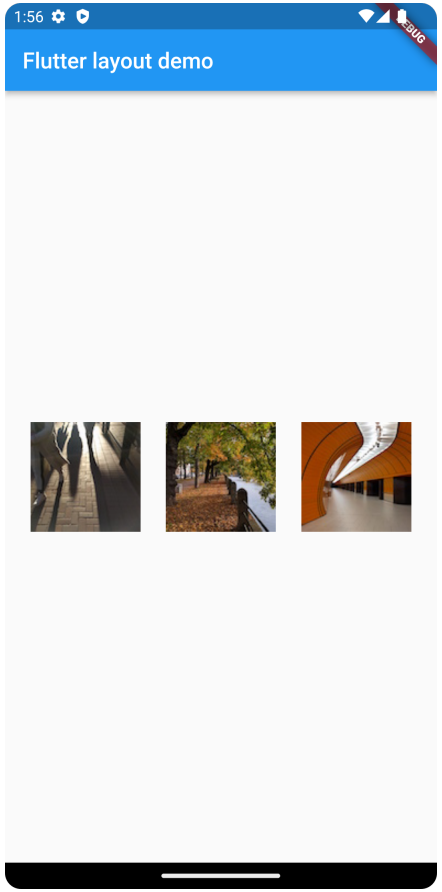




Aligning widgets

- In the following example, each of the 3 images is 100 pixels wide. The render box (in this case, the entire screen) is more than 300 pixels wide, so setting the main axis alignment to **`spaceEvenly`** divides the free horizontal space evenly between, before, and after each image.

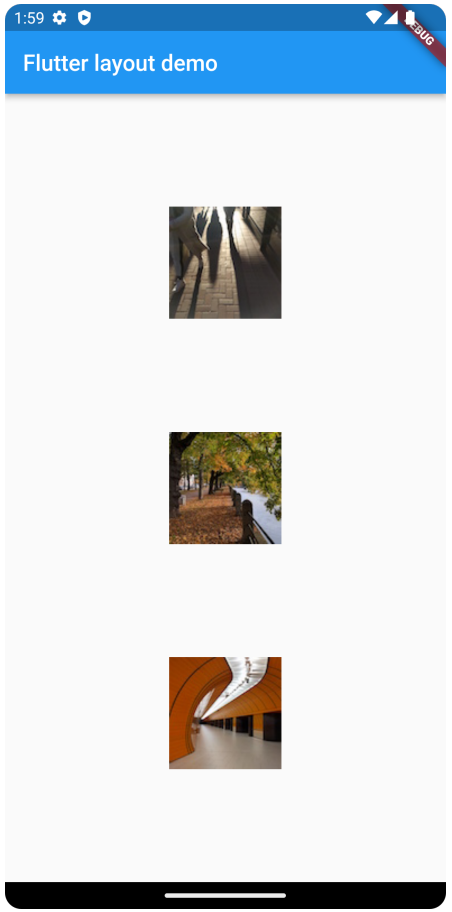
```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Image.asset('images/pic1.jpg'),
    Image.asset('images/pic2.jpg'),
    Image.asset('images/pic3.jpg'),
  ],
);
```



Aligning widgets

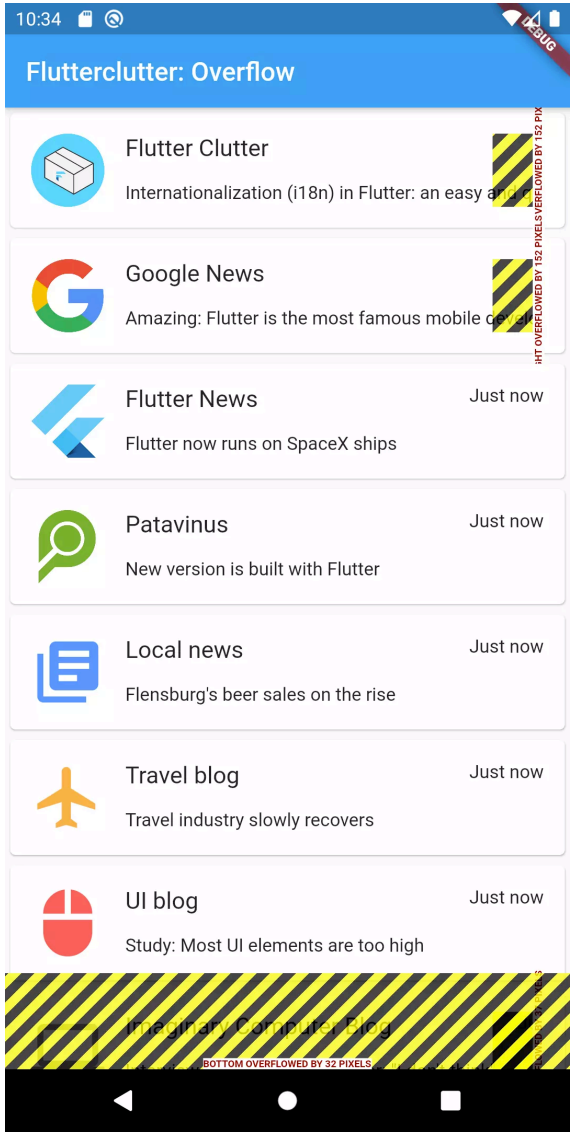
- The following example shows a column of 3 images, each is 100 pixels high. The height of the render box (in this case, the entire screen) is more than 300 pixels, so setting the main axis alignment to **`spaceEvenly`** divides the free vertical space evenly between, above, and below each image.

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



Sizing widgets

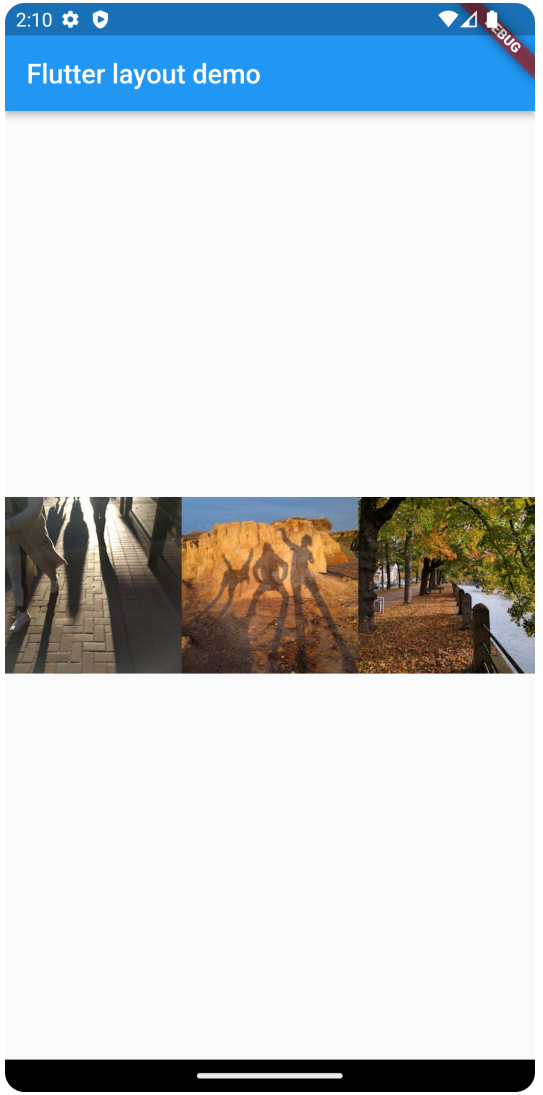
- When a layout is **too large** to fit a device, a yellow and black striped pattern appears along the affected edge.



Sizing widgets

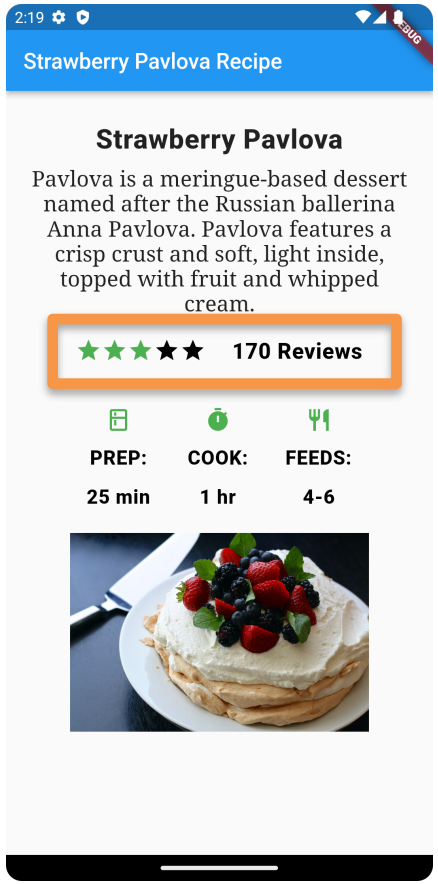
- Widgets can be sized to fit within a row or column by using the `Expanded` widget.

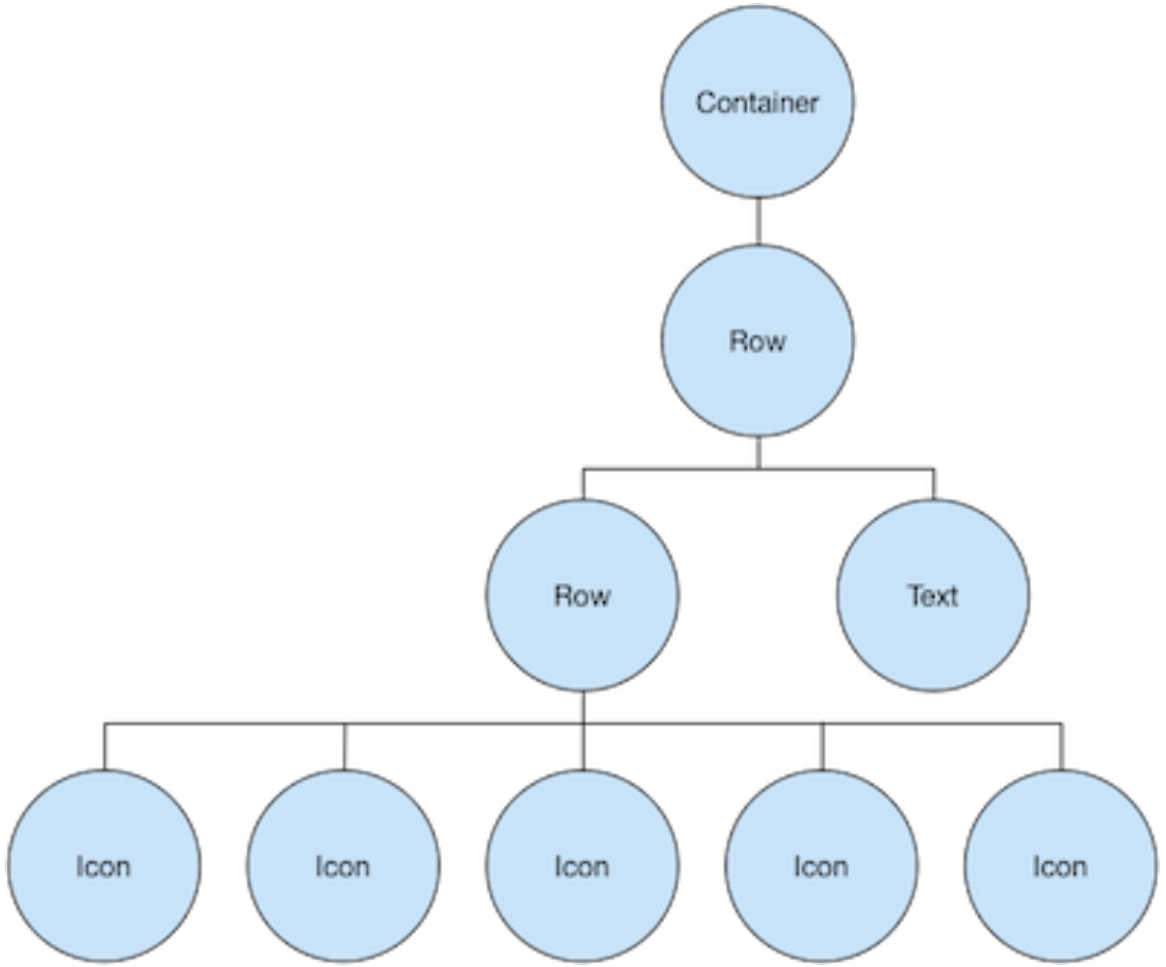
```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic2.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic3.jpg'),
    ),
  ],
);
```



Nesting rows and columns

- The outlined section is implemented as two rows. The ratings row contains five stars and the number of reviews. The icons row contains three columns of icons and text.
- The widget tree for the ratings row:

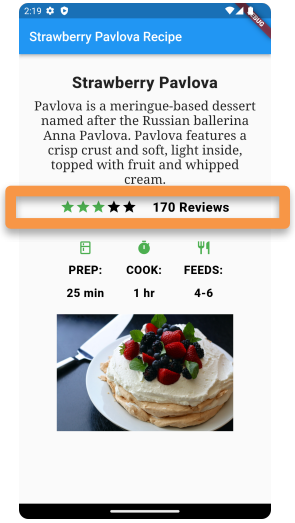


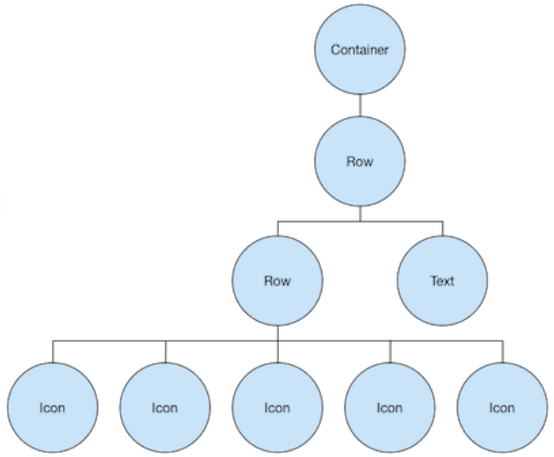


Nesting rows and columns

- Creates a row containing a smaller row of 5 star icons, and text:

```
var stars = 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]),
    Icon(Icons.star, color: Colors.black),
    Icon(Icons.star, color: Colors.black),
  ],
);
```

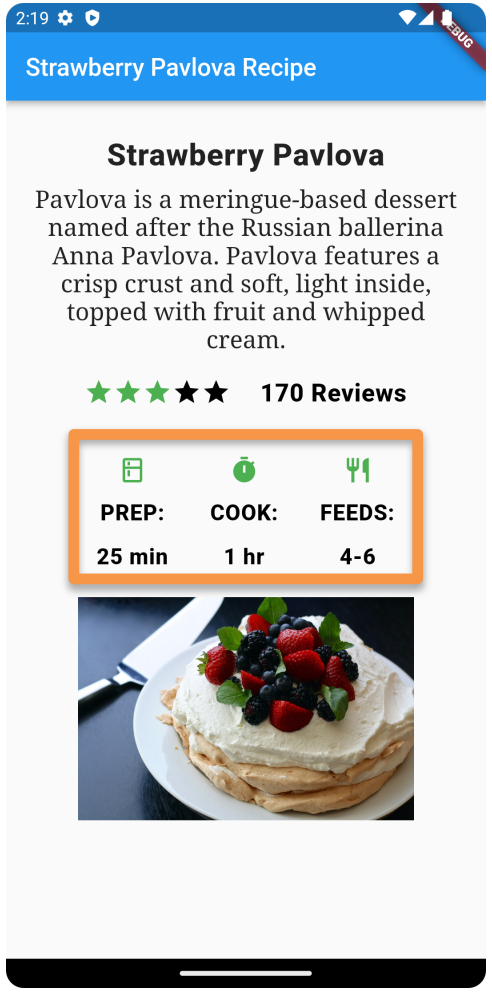


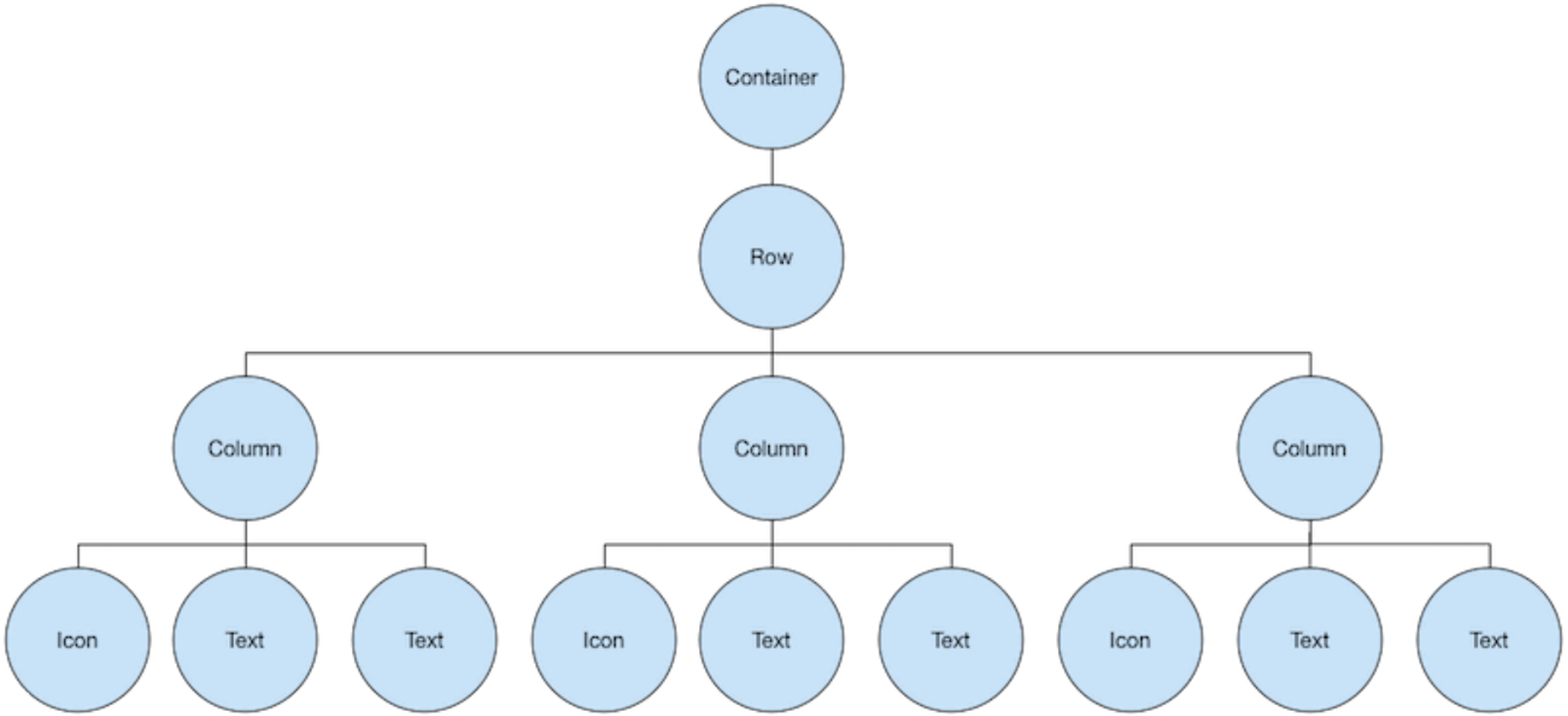



```
final ratings = Container(
  padding: const EdgeInsets.all(20),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      stars,
      const Text(
        '170 Reviews',
        style: TextStyle(
          color: Colors.black,
          fontWeight: FontWeight.w800,
          fontFamily: 'Roboto',
          letterSpacing: 0.5,
          fontSize: 20,
        ),
      ),
    ],
  ),
);
```

Nesting rows and columns

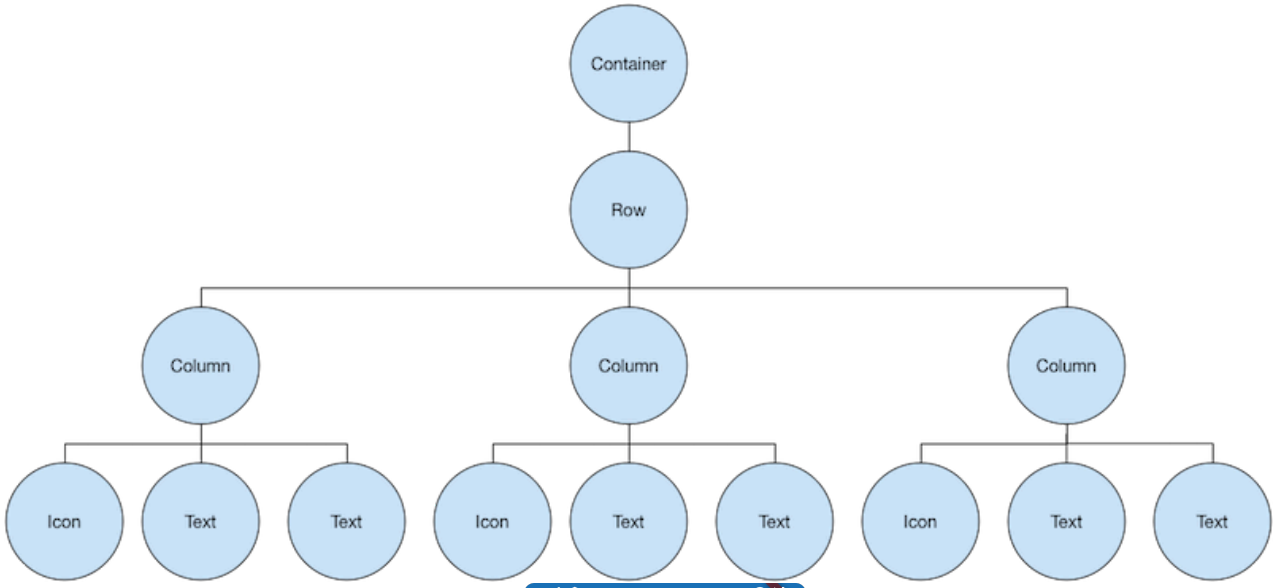
- The icons row, below the ratings row, contains 3 columns; each column contains an icon and two lines of text, as you can see in its widget tree:

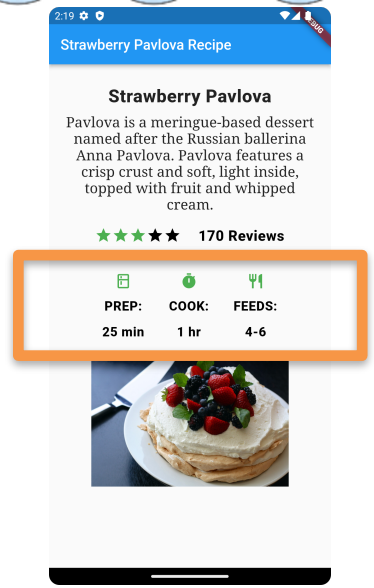




Nesting rows and columns

- Defines the icons row:





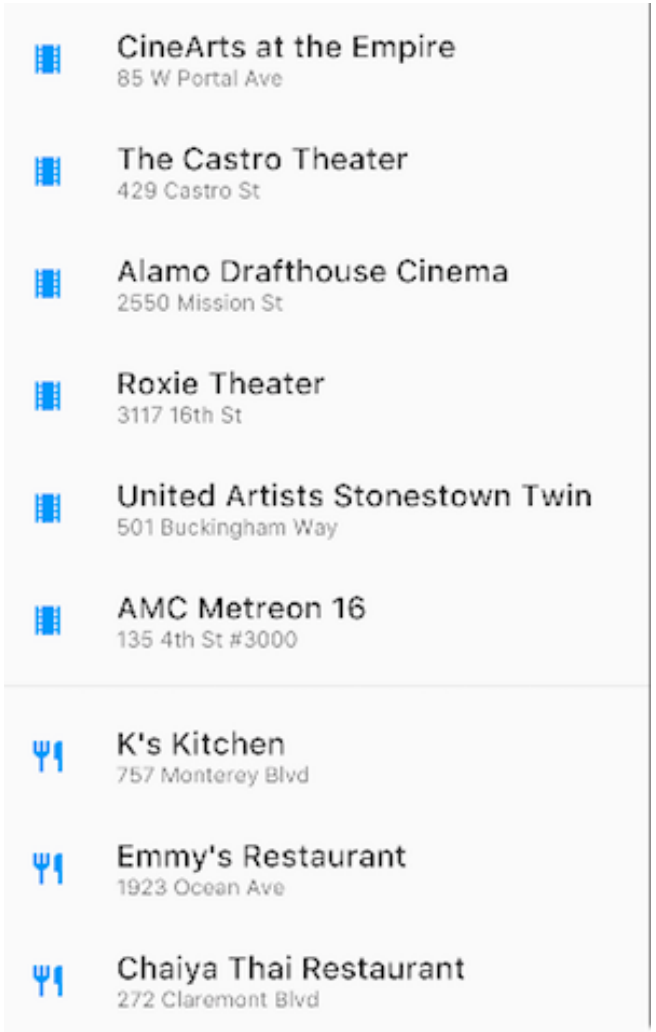

```dart
child: Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Column(
      children: [
        Icon(Icons.kitchen, color: Colors.green[500]),
        const Text('PREP:'),
        const Text('25 min'),
      ],
    ),  // Column
    Column(
      children: [
        Icon(Icons.timer, color: Colors.green[500]),
        const Text('COOK:'),
        const Text('1 hr'),
      ],
    ),  // Column
    Column(
      children: [
        Icon(Icons.restaurant, color: Colors.green[500]),
        const Text('FEEDS:'),
        const Text('4-6'),
      ],
    ),  // Column
  ],
),  // Row
```

Note: Nesting rows and columns

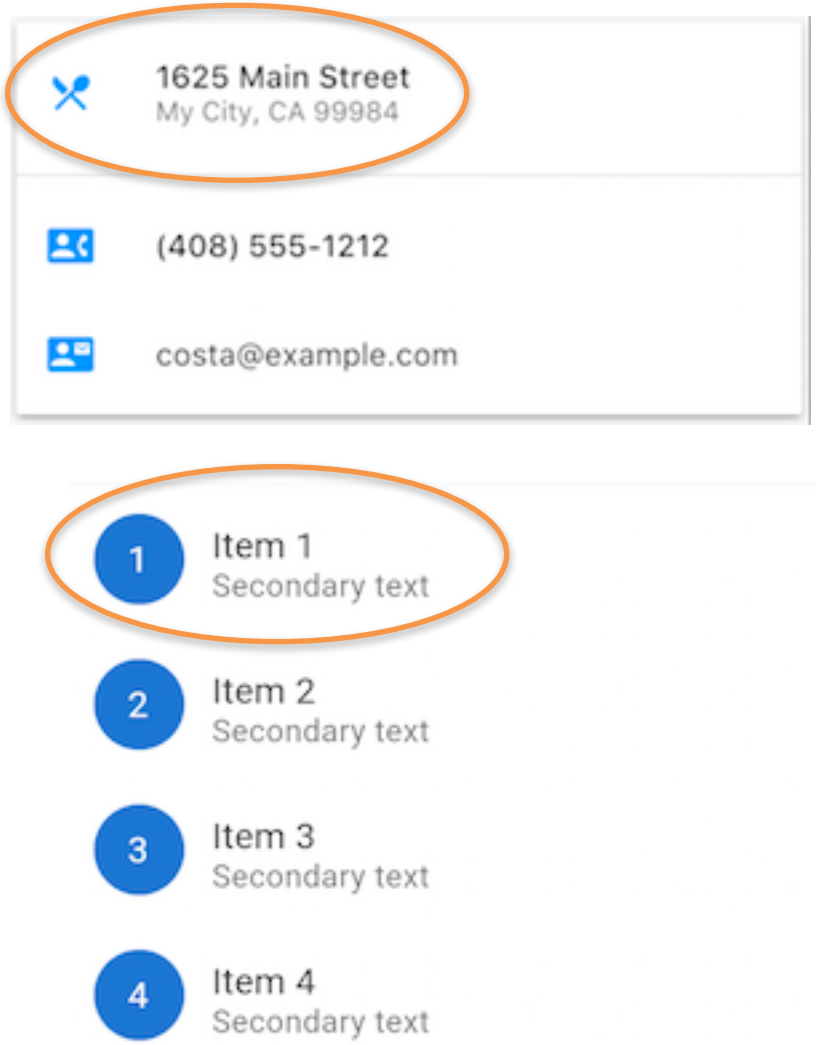
- **Row** and **Column** are basic primitive widgets for horizontal and vertical layouts—these low-level widgets allow for **maximum customization**.
- Flutter also offers specialized, higher level widgets that might be sufficient for your needs.
  - Instead of **Row** you might prefer **ListTile**, an easy-to-use widget with properties for leading and trailing icons, and up to 3 lines of text.
  - Instead of **Column**, you might prefer **ListView**, a column-like layout that **automatically scrolls** if its content is too long to fit the available space.

Note: Nesting rows and columns

- ListView



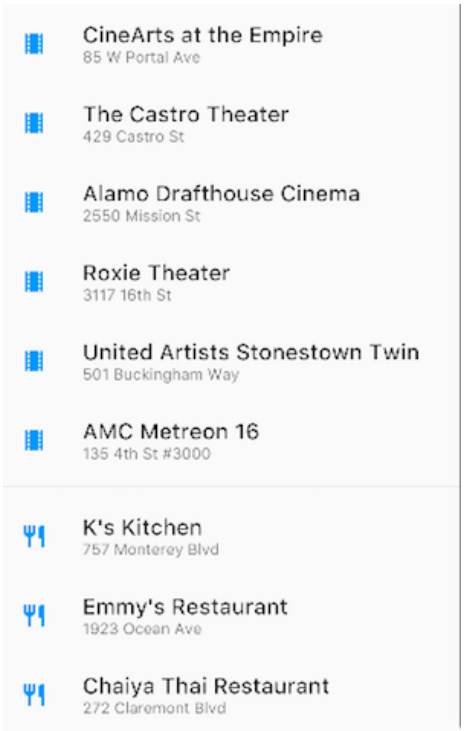
- ListTile



ListView

```
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),
      _tile('Alamo Drafthouse Cinema', '2550 Mission St', Icons.theaters),
      _tile('Roxie Theater', '3117 16th St', Icons.theaters),
      _tile('United Artists Stonestown Twin', '501 Buckingham Way',
          Icons.theaters),
      _tile('AMC Metreon 16', '135 4th St #3000', Icons.theaters),
      const Divider(),
      _tile('K\'s Kitchen', '757 Monterey Blvd', Icons.restaurant),
      _tile('Emmy\'s Restaurant', '1923 Ocean Ave', Icons.restaurant),
      _tile(
          'Chaiya Thai Restaurant', '272 Claremont Blvd', Icons.restaurant),
      _tile('La Ciccia', '291 30th St', Icons.restaurant),
    ],
  );  // ListView
}
```

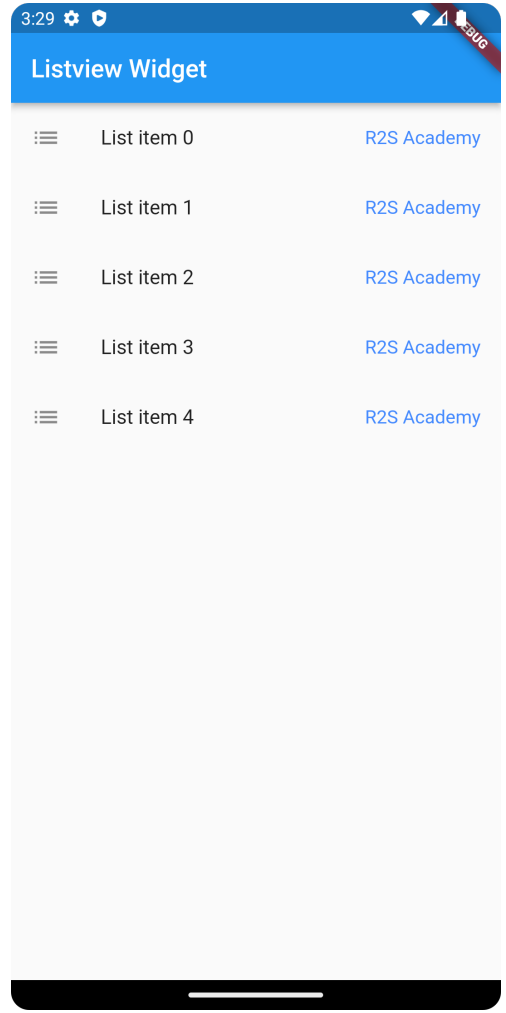
```
ListTile _tile(String title, String subtitle, IconData icon) {
  return ListTile(
    title: Text(title,
        style: const TextStyle(
          fontWeight: FontWeight.w500,
          fontSize: 20,
        )),  // TextStyle, Text
    subtitle: Text(subtitle),
    leading: Icon(
      icon,
   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



ListView

- It is used to create the list of children But when we want to create a list **recursively without writing code again and again** then **ListView.builder** is used.

```
Widget _buildListViewBuilder() {
  return ListView.builder(
      itemCount: 5,
      itemBuilder: (BuildContext context, int index) {
        return ListTile(
            leading: const Icon(Icons.list),
            trailing: const Text(
              "R2S Academy",
              style: TextStyle(color: Colors.blueAccent, fontSize: 15),
            ),  // Text
            title: Text("List item $index"));  // ListTile
      }
      );  // ListView.builder
}
```



ListView

- We have *ListView.builder* with itemCount and *itemBuilder* which will create a new widget **again and again up to 5 times** because we have *itemCount = 5*
- The *itemBuilder* returns ***ListTile*** which has **leading**, **title** and **trailing**. This *ListTile* will build again and again up to 5 times.

```
Widget _buildListViewBuilder() {
  return ListView.builder(
      itemCount: 5,
      itemBuilder: (BuildContext context, int index) {
        return ListTile(
            leading: const Icon(Icons.list),
            trailing: const Text(
              "R2S Academy",
              style: TextStyle(color: Colors.blueAccent, fontSize: 15),
            ),  // Text
            title: Text("List item $index"));  // ListTile
      }
      );  // ListView.builder
}
```