Creating Scrolling Lists and Gridview

USING THE CARD

- The **Card** widget is part of the Material Design and has minimal rounded corners and shadows. To group and lay out data,
- The Card widget is customizable with properties such as **elevation**, **shape**, **color**, **margin**, and others.
- The **elevation** property is a value of double, and the higher the number, the larger the shadow that is cast.
- To customize the shape and borders of the **Card** widget, you modify the shape property. Some of the **shape** properties are **StadiumBorder**, **UnderlineInputBorder**, **OutlineInputBorder**, and others.

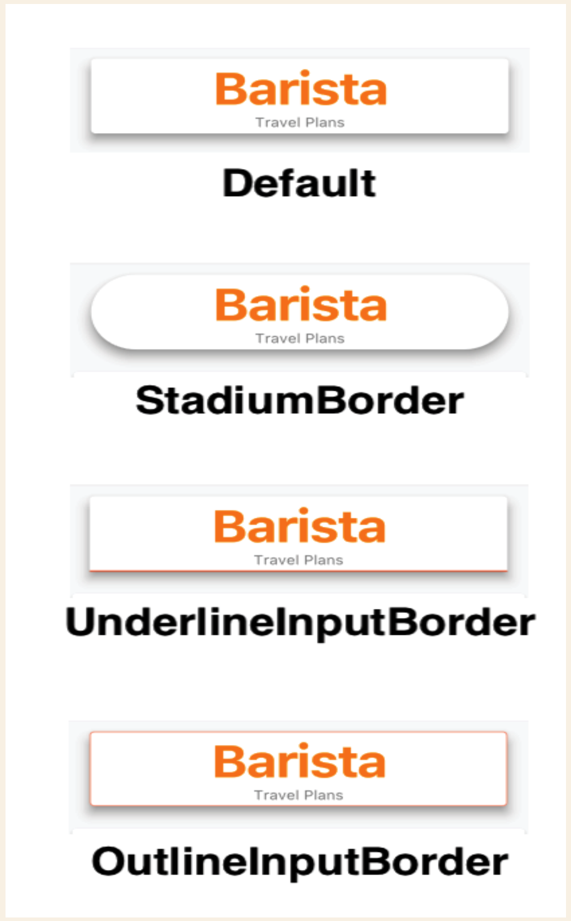
The following are a few ways to customize the Card's shape property

- **// Create a Stadium Border**

  shape: StadiumBorder(),
- **// Create Square Corners Card with a Single Orange Bottom Border**

  shape: UnderlineInputBorder(borderSide: BorderSide(color: Colors.deepOrange)),
- **// Create Rounded Corners Card with Orange Border**

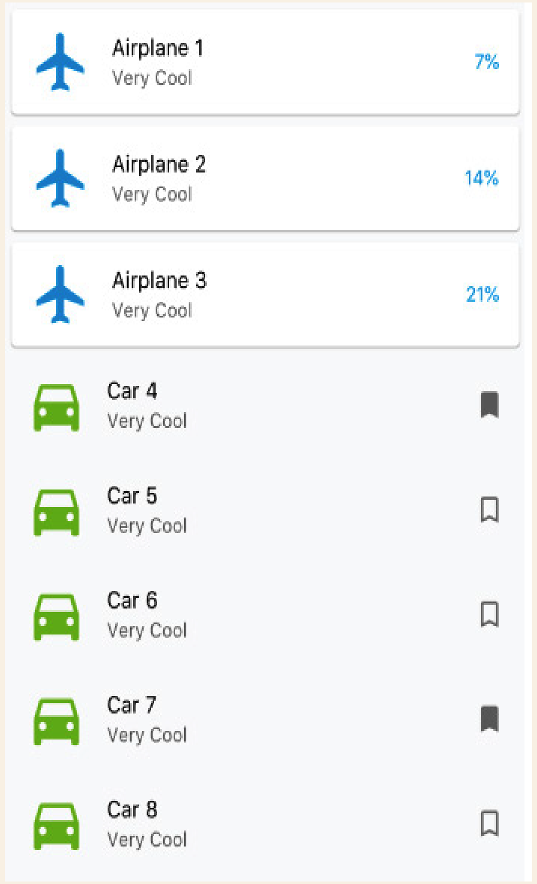
  shape: OutlineInputBorder(borderSide: BorderSide(color: Colors.deepOrange.withOpacity(0.5)),),



USING THE LISTVIEW AND LISTTILE

- The constructor **ListView.builder** is used to create an on-demand **linear scrollable list** of widgets .
- Within the builder, you use the **itemBuilder** callback to **create the list of children** widgets.
- The **scrollDirection** argument defaults to **Axis.vertical** but can be **changed** to **Axis.horizontal**.

  ....
- The **ListTile** widget is commonly used with the **ListView** widget to easily format and organize icons, **titles**, and **descriptions** in a **linear** layout.
- You can also use the **onTap** and **onLongPress** **callbacks** to execute an action when the user taps the **ListTile**.


Airplane 1
Very Cool
7%
Airplane 2
Very Cool
14%
Airplane 3
Very Cool
21%
Car 4
Very Cool
Car 5
Very Cool
Car 6
Very Cool
Car 7
Very Cool
Car 8
Very Cool

There are different types of ListViews :

ListView
ListView.builder
ListView.separated
ListView.custom

ListView()

This is the default constructor of the ListView class. A ListView simply takes a list of widgets and makes it scrollable. Usually, this is used with a few children as the List will also construct invisible elements in the list, so numerous widgets may render this inefficiently.

```
ListView(
                    padding: EdgeInsets.all(20),
                    children: <Widget>[
                              CircleAvatar(
                              maxRadius: 50,
                              backgroundColor: Colors.black,
                              child: Icon(Icons.person, color: Colors.white, size: 50),
                              ),
                              Center(
                              child: Text(
                                        List View',
                                        style: TextStyle(
                                        fontSize: 50,
                                        ),),),
                              Text(
                              "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem
Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a
gallery of type and scrambled it to make a type
                              style: TextStyle(
                                        fontSize: 20, ),),
                    ],
          ),
```

ListView.builder()

The builder() constructor constructs a repeating list of widgets. The constructor takes two main parameters:

An itemCount for the number of repetitions for the widget to be constructed (not compulsory).
An itemBuilder for constructing the widget which will be generated ‘itemCount‘ times (compulsory).
If the itemCount is not specified, infinite widgets will be constructed by default.

```
ListView.builder(
            itemCount: 20,
            itemBuilder: (context, position) {
                  return Card(
                  child: Padding(
                        padding: const EdgeInsets.all(20.0),
                        child: Text(
                        position.toString(),
                        style: TextStyle(fontSize: 22.0),
                        ),
                  ),
                  );
            },),
```

ListView.separated ()

The ListView.separated() constructor is used to generate a list of widgets, but in addition, a separator widget can also be generated to separate the widgets. In short, these are two intertwined list of widgets: the main list and the separator list. Unlike the builder() constructor, the itemCount parameter is compulsory here.

```
ListView.separated(
                    itemBuilder: (context, position) {
                              return Card(
                              child: Padding(
                                        padding: const EdgeInsets.all(15.0),
                                        child: Text(
                                        'List Item $position’, ), ), );   },
                    separatorBuilder: (context, position) {
                              return Card(
                              color: Colors.grey,
                              child: Padding(
                                        padding: const EdgeInsets.all(5.0),
                                        child: Text(
                                        'Separator $position',
                                        style: TextStyle(color: Colors.white), ), ),);},
                    itemCount: 20,
                    ),
```

ListView.custom()

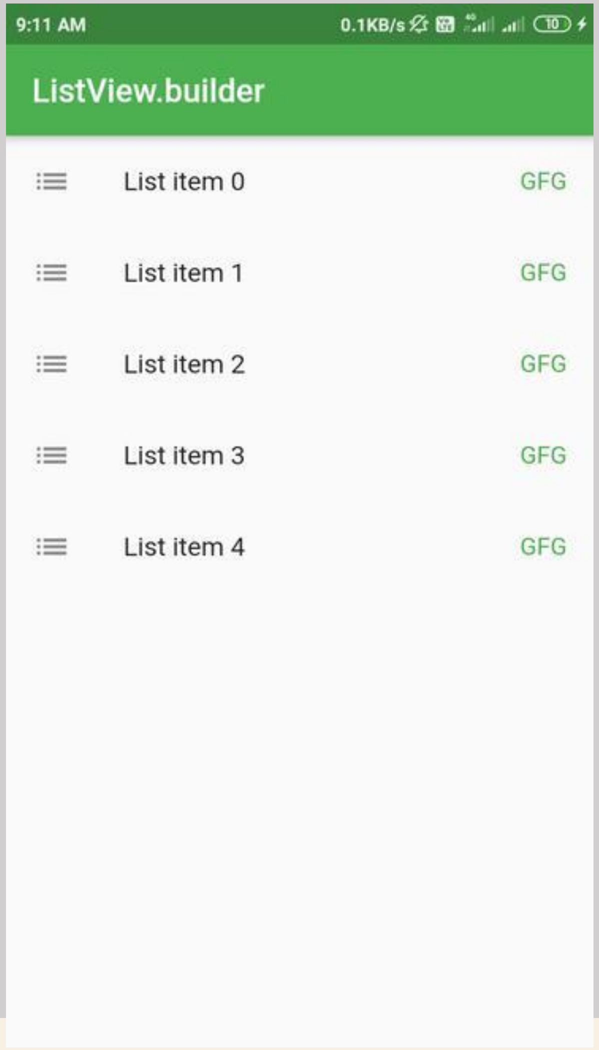
The ListView.custom() constructor build ListViews with custom functionality for how the children of the list are built. The main parameter of this constructor is a SliverChildDelegate which builds the items.

The types of SliverChildDelegates are :

- SliverChildListDelegate
- SliverChildBuilderDelegate

The SliverChildListDelegate accepts a list of children widgets. whereas the SliverChildBuilderDelegate accepts an IndexedWidgetBuilder, simply a builder() function. Digging deeper, we can infer that ListView.builder was created using a ListView.custom with a SliverChildBuilderDelegate. Also, the default ListView() constructor is a ListView.custom with a SliverChildListDelegate.

```
return MaterialApp(
       title: "ListView.builder",
       theme: ThemeData(primarySwatch: Colors.green),
       debugShowCheckedModeBanner: false,
       // home : new ListViewBuilder(),  NO Need To Use Unnecessary New Keyword
       home: const ListViewBuilder());   } }
class ListViewBuilder extends StatelessWidget {
  const ListViewBuilder({Key? key}) : super(key: key);
   @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("ListView.builder")),
      body: ListView.builder(
          itemCount: 5,
          itemBuilder: (BuildContext context, int index) {
            return ListTile(
                leading: const Icon(Icons.list),
                trailing: const Text(
                  "GFG",
                  style: TextStyle(color: Colors.green, fontSize: 15),
                ),
                title: Text("List item $index"));
          }),     );   } }
```


```
9:11 AM                         0.1KB/s
ListView.builder
List item 0    GFG
List item 1    GFG
List item 2    GFG
List item 3    GFG
List item 4    GFG
```

Flutter GridView

GridView is a widget in Flutter that arranges the list of its children in a two-dimensional grid pattern. It allows us to store and display items in a matrix form. This widget is highly useful when you want to display a set of widgets in a grid format. These widgets can be images, cards, icons, or any others. Moreover, it automatically handles scrolling when there are more items that can practically fit on the device's screen. This saves the developer from the hassle of taking care of the scrolling methods.

Types of GridView

There are two main types of GridView that are used often.

1. GridView.count

2. GridView.builder

GridView.count()

It is the most frequently used grid layout in Flutter because here, we already know the grid's size. It allows developers to **specify the fixed number of rows and columns**. The GriedView.count() contains the following properties:

crossAxisCount: It is used to specify the number of columns in a grid view.

crossAxisSpacing: It is used to specify the number of pixels between each child widget listed in the cross axis.

mainAxisSpacing: It is used to specify the number of pixels between each child widget listed in the main axis.

padding(EdgeInsetsGeometry): It is used to specify the space around the whole list of widgets.

scrollDirection: It is used to specify the direction in which the items on GridView scrolls. By default, it scrolls in a vertical direction.

reverse: If it is true, it will reverse the list in the opposite direction along the main axis.

physics: It is used to determine how the list behaves when the user reaches the end or the start of the widget while scrolling.

```
GridView.count( crossAxisCount: 3, // Adjust the number of columns here
children: _generateGridItems(), // A function that returns a list of widgets ),
```

This constructor is useful when you have a small number of items that are not dynamic and are not planned to change.

Flutter – Dismissible Widget

The **Dismissible** widget in Flutter is used to create items that can be dismissed by swiping them off the screen. It's commonly used in lists or grids where you want to provide a way for users to remove items with a swipe gesture.

Basic Syntax of Dismissible Widget

```
Dismissible(
key: UniqueKey(), // or any unique key for tracking items
child: YourContentWidget(),
background: YourBackgroundWidget(),
secondaryBackground: YourSecondaryBackgroundWidget(),
confirmDismiss: (DismissDirection direction) async {
// Your confirmation logic goes here
// Return true to allow dismissal, false to prevent it
return true;
},
onDismissed: (DismissDirection direction) {
// Your action when item is dismissed goes here
},
onResize: () {
// Your resize animation logic goes here (optional)
},
direction: DismissDirection.endToStart, // or other DismissDirection values
dragStartBehavior: DragStartBehavior.start, // or DragStartBehavior.down
)
```

1: Create MyApp Class

In this class we are going to implement the MaterialApp , here we are also set the Theme of our App.

```
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
        return MaterialApp(
        debugShowCheckedModeBanner: false,
        theme: ThemeData(
                primarySwatch: Colors.green, // Set the app's primary theme color
        ),
        title: 'Dismissible Example',
        home: DismissibleExample(),
        );
}
}
```

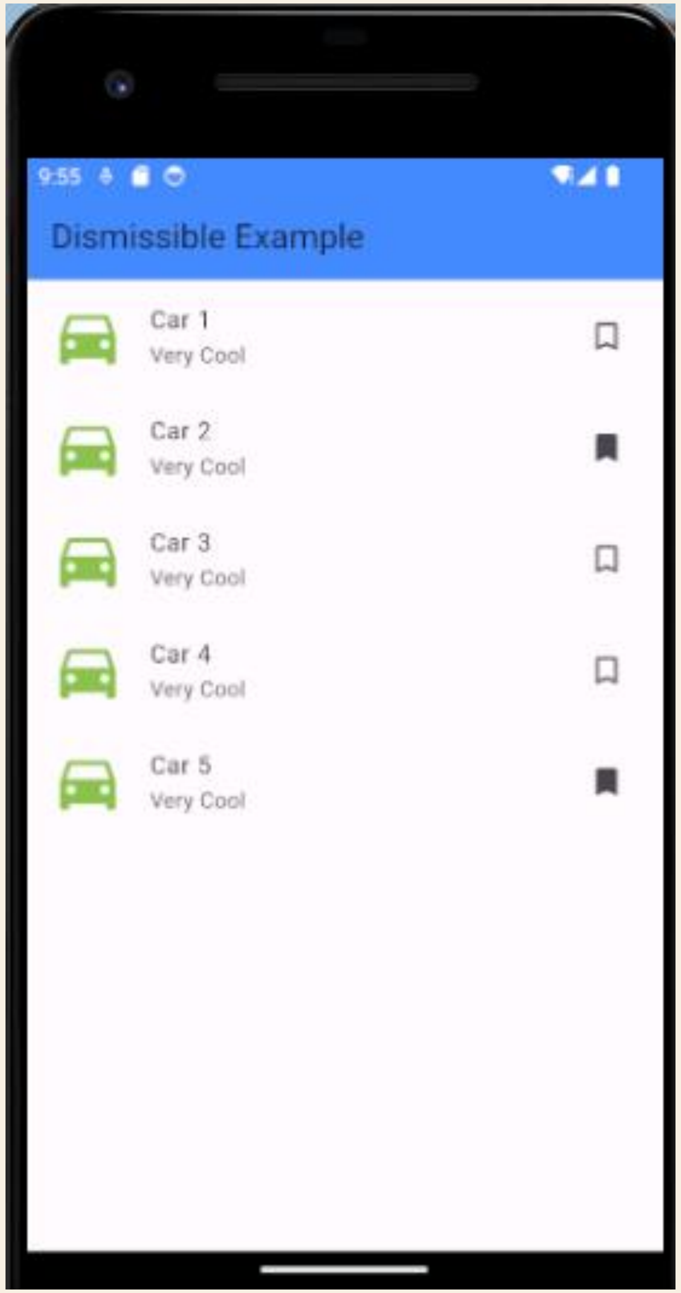
2: Create DismissibleExample Class

In this class we are going to Implement the Dismissible widget whenever the user Swipe the List items then the swiped items are deleted. Comments are added for better understanding.

```
class DismissibleExample extends StatefulWidget {
  @override
  _DismissibleExampleState createState() => _DismissibleExampleState();
}

class _DismissibleExampleState extends State<DismissibleExample> {
// Sample list of items
  List<String> items = List.generate(5, (index) => 'Car ${index + 1}');

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blueAccent,
        title: Text('Dismissible Example'),
      ),
body: ListView.builder(
        itemCount: items.length,
        itemBuilder: (context, index) {
          final item = items[index];
          return Dismissible(
            key: Key(item), // Unique key for each item
            onDismissed: (direction) {
              // Remove the item from the list when dismissed
              setState(() {
                items.removeAt(index);      });
              // Show a snackbar to indicate item removal
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(
                  content: Text('$item dismissed'),   ),  );  },
            background: Container(
              color: Colors.red, // Background color when swiping
              child: Icon( Icons.delete, color: Colors.white, size: 36, ),
              alignment: Alignment.centerRight,
              padding: EdgeInsets.only(right: 20),),
            child: ListTile(
              leading: Icon( Icons.directions_car, size: 48.0, color: Colors.lightGreen,  ),
              title: Text(item),
              subtitle: Text('Very Cool'),
  trailing: (index % 3).isEven ? Icon(Icons.bookmark_border) : Icon(Icons.bookmark), ), );
        },
      ),
    );}}
```



9:55

Dismissible Example

Car 1
Very Cool

Car 2
Very Cool

Car 3
Very Cool

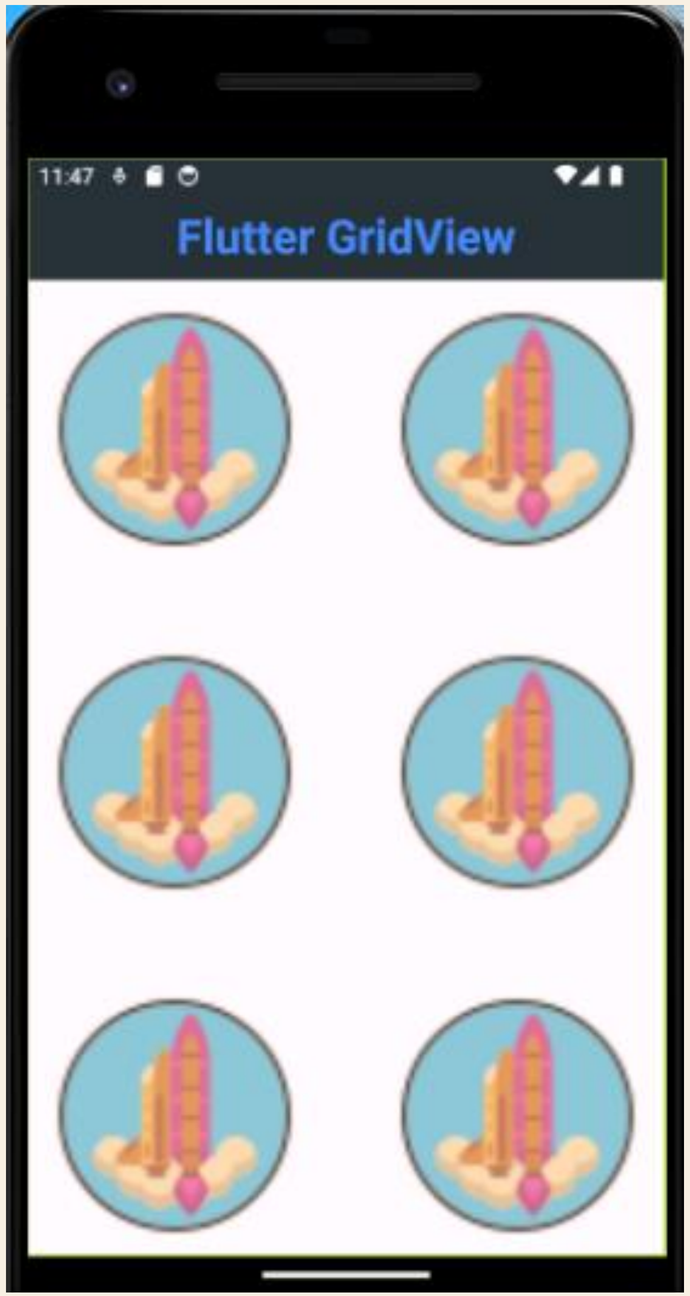
Car 4
Very Cool

Car 5
Very Cool

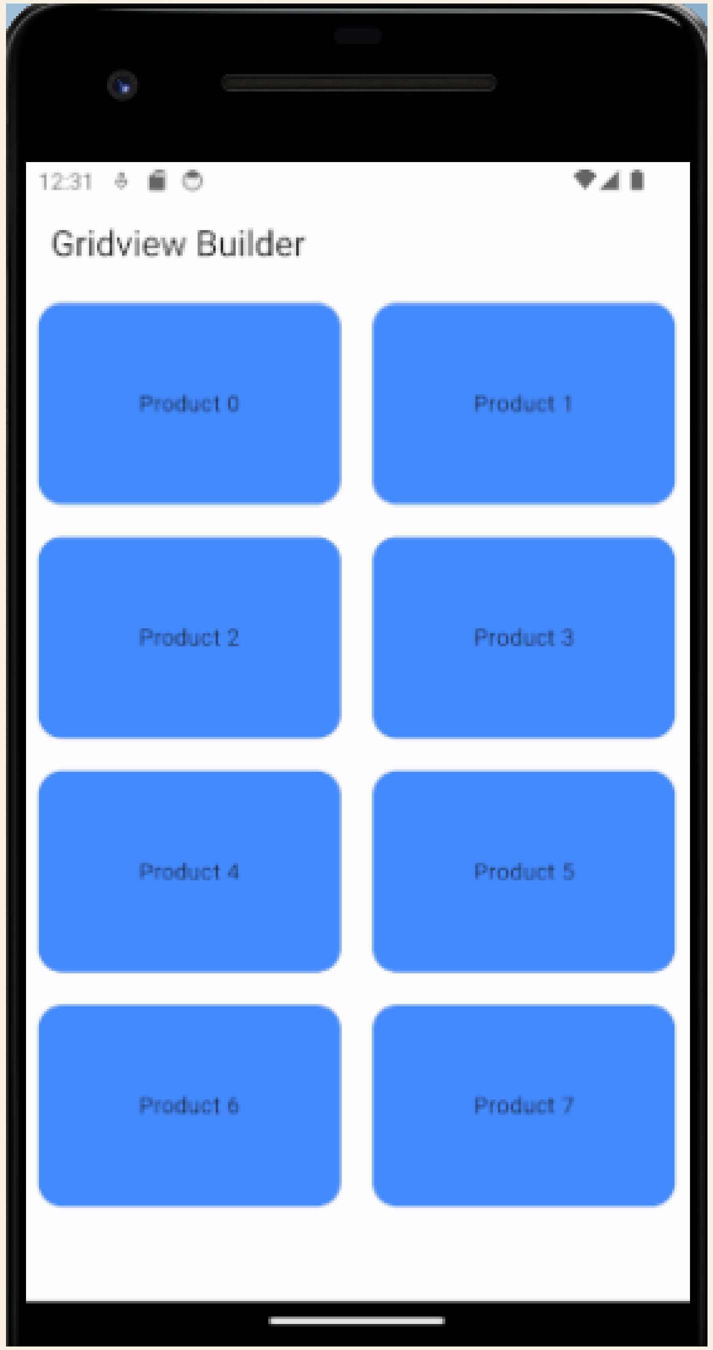
GridView.builder()

This constructor is more dynamic and provides more suitability when you have a large number of items in the grid or when the children widgets are built dynamically during runtime.

```
GridView.builder(
  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 3, // Number of columns in the grid
    crossAxisSpacing: 8.0, // Spacing between columns
    mainAxisSpacing: 8.0, // Spacing between rows
    ),
  itemCount: itemCount,
  itemBuilder: (BuildContext context, int index) {
  return Container(
      color: Colors.white,
      child: Center(
           child: Text(
              'Item $index',
               style: TextStyle(color: Colors.white),
              ),
          ),
       );
    },
 ),
home: Scaffold(
          appBar: AppBar(
          backgroundColor: Colors.blueGrey[900],
          title: Center(
            child: Text(
              'Flutter GridView',
              style: TextStyle(
                color: Colors.blueAccent,
                fontWeight: FontWeight.bold,
                fontSize: 30.0,   ), ), ), ),
        body: GridView.count(
          crossAxisCount: 2,
          crossAxisSpacing: 30.0,
          mainAxisSpacing: 30.0,
          children: List.generate(6,(index) {
              return Padding(
                padding: const EdgeInsets.all(10.0),
                child: Container(
                  decoration: BoxDecoration(
                    image: DecorationImage(
                      image: AssetImage('images/rockets.png'),
                      fit: BoxFit.cover,   ),
                    borderRadius: BorderRadius.all(
                      Radius.circular(20.0), ), ), ),);}, ), ), ), );} }
```


11:47
Flutter GridView

```
return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Grid build',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
        useMaterial3: true,  ),
      home: HomeScreen(), );}}
class HomeScreen extends StatelessWidget {
  HomeScreen({Key? key}) : super(key: key);
  final List<Map> myProducts =
      List.generate(8, (index) => {"id": index, "name": "Product $index"}).toList();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Gridview Builder'), ),
      body: Padding(padding: const EdgeInsets.all(8.0),
          child: GridView.builder(
            gridDelegate: const SliverGridDelegateWithMaxCrossAxisExtent(
                maxCrossAxisExtent: 200,
                childAspectRatio: 3 / 2,
                crossAxisSpacing: 20,
                mainAxisSpacing: 20),
            itemCount: myProducts.length,
            itemBuilder: (BuildContext ctx, index) {
              return Container( alignment: Alignment.center,
                decoration: BoxDecoration(color: Colors.blueAccent,
                    borderRadius: BorderRadius.circular(15)),
                child: Text(myProducts[index]["name"]),); }),), );}}
```


12:31
Gridview Builder
Product 0
Product 1
Product 2
Product 3
Product 4
Product 5
Product 6
Product 7