ListView

- ListView is a scrollable list of widgets.
- It is used to group several items in an array and display them in a scrollable list.
- We can make the ListView scrollable in two directions: horizontal, vertical, using scrollDirection property.
- We can also make it never scroll using physics property set to NeverScrollableScrollPhysics() object.

Types of ListViews:

1. ListView
2. ListView.builder
3. ListView.separated
4. ListView.custom

ListView

- It is the default constructor of the ListView class.
- A ListView simply takes a list of widgets and makes it scrollable.
- By Default in a ListView direction set as a Vertical.

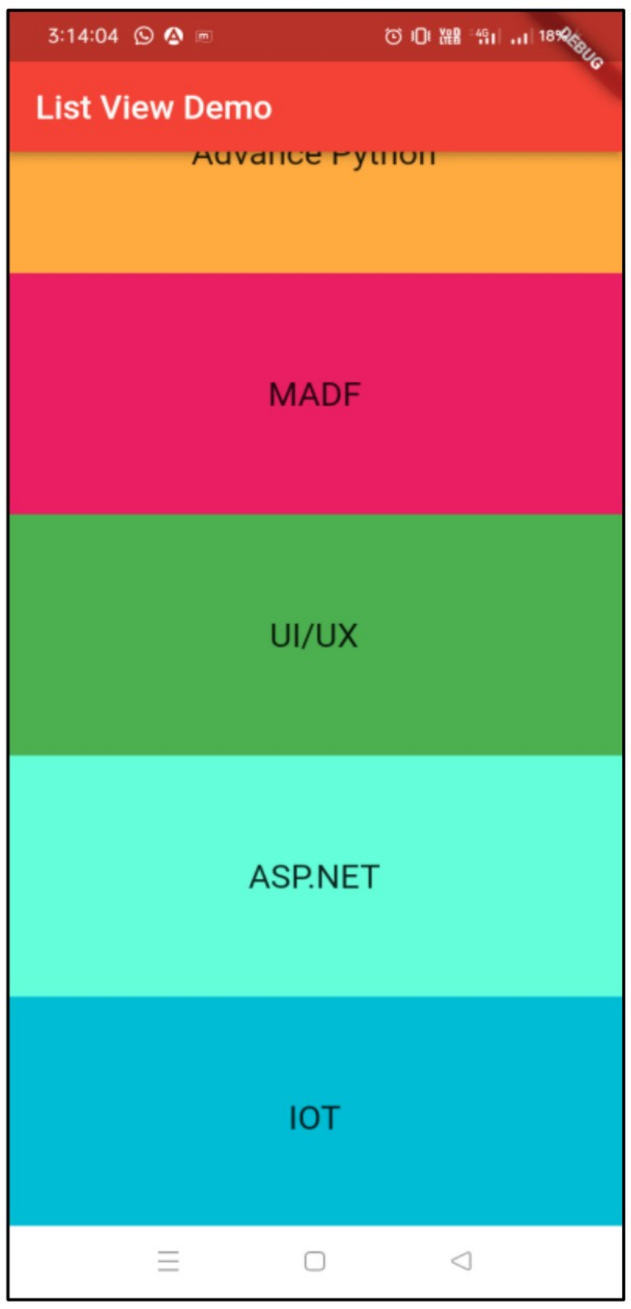
Example:

```
body: ListView(
        scrollDirection: Axis.vertical
      //scrollDirection: Axis.horizontal, you can set Axis.horizontal value for scrollDirection
       children: [
      Container(
        height: 150,
        child: Center(child: Text("Advance Python",style: TextStyle(fontSize: 20),)),
        color: Colors.orangeAccent,
      ),
      Container(
        height: 150,
        child: Center(child: Text("MADF",style: TextStyle(fontSize: 20))),
        color: Colors.pink,
      ),
      Container(
        height: 150,
        child: Center(child: Text("UI/UX",style: TextStyle(fontSize: 20))),
        color: Colors.green,
      ),
      Container(
        height: 150,
        child: Center(child: Text("ASP.NET",style: TextStyle(fontSize: 20))),
        color: Colors.tealAccent,
      ),
       Container(
        height: 150,
```

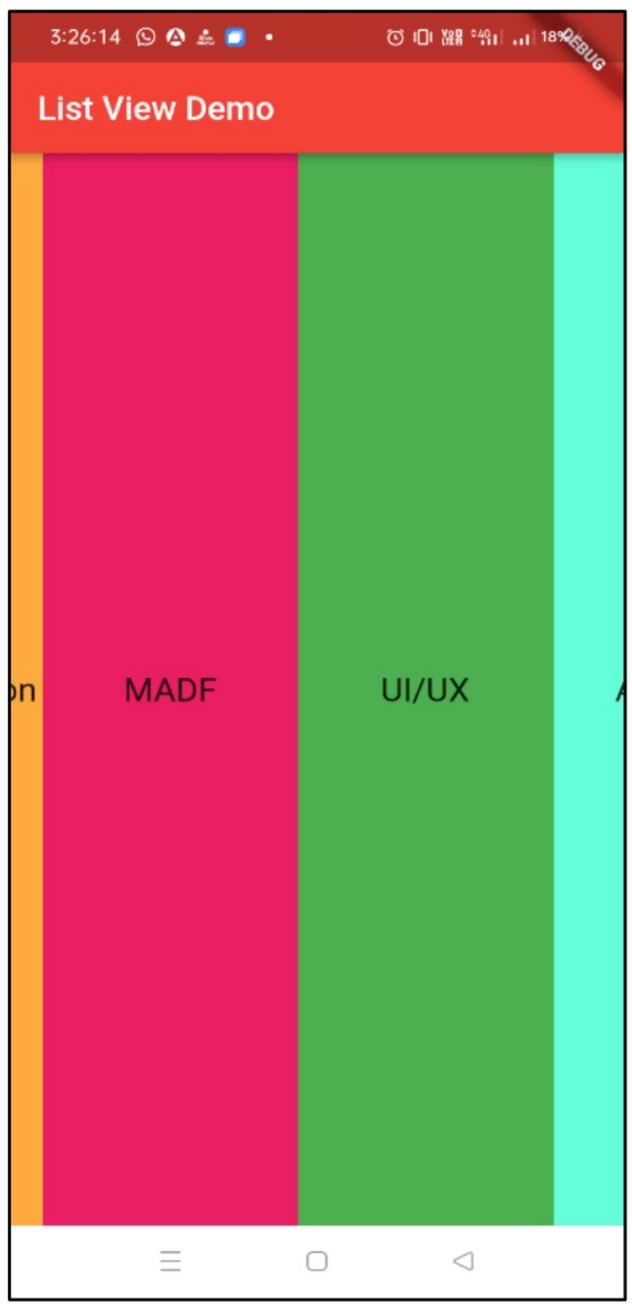
```
  child: Center(child: Text("IOT",style: TextStyle(fontSize: 20))),
  color: Colors.cyan,
),
Container(
  height: 150,
  child: Center(child: Text("Project",style: TextStyle(fontSize: 20))),
  color: Colors.lightGreen,
),
],
),
```

Output:

Vertical



Horizontal



ListView.builder

- It is used to Generate dynamic List.
- The ListView.builder constructor takes an IndexedWidgetBuilder.
- It builds the children on demand. This constructor is appropriate for list views with a large (or infinite) number of children.
- ListView.builder() is used to render long or infinite lists, especially lists of data fetched from APIs like products, news, messages, search results etc.
- This constructor takes two main parameters:
  1. itemCount for the number of List for the widget to be constructed (not compulsory).
  2. itemBuilder for constructing the widget which will be generated 'itemCount' times (compulsory).

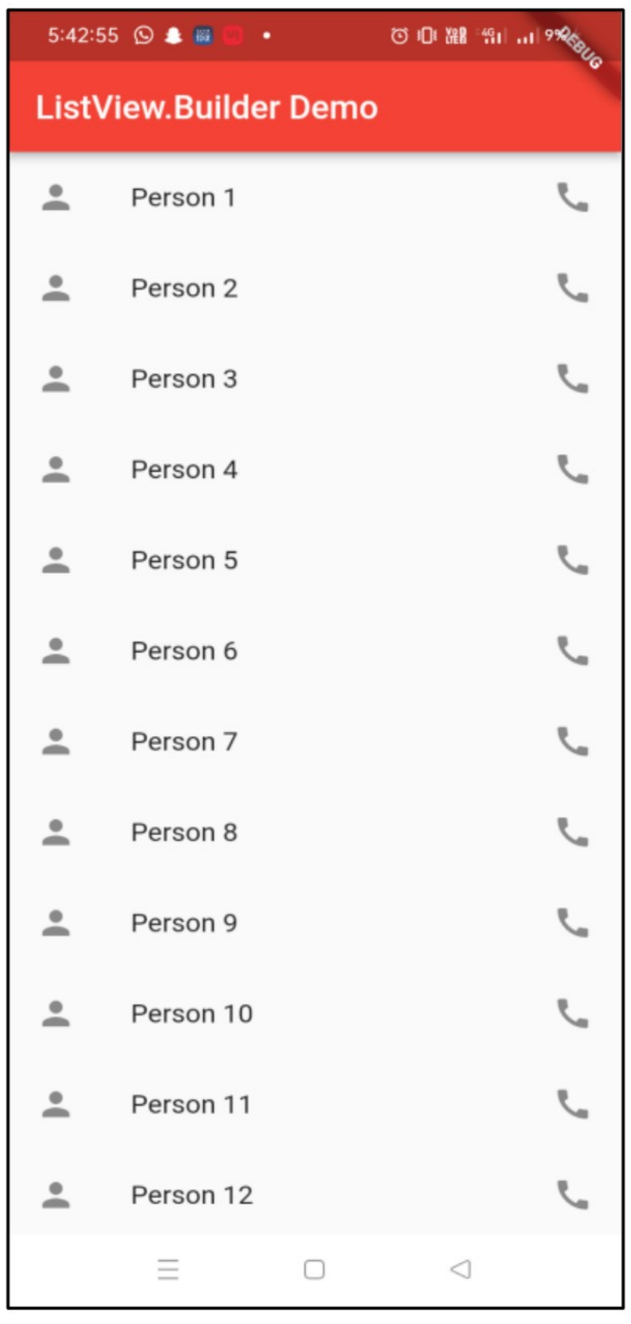
ListTile:

- A single fixed-height row that typically contains some text as well as a leading and trailing icon.

Example:

```
body: ListView.builder(
  itemCount: 40,
  itemBuilder:(BuildContext context,int index){
  return ListTile(
      leading: Icon(Icons.person),
      trailing: Icon(Icons.call),
      title: Text("Person ${index+1}"),
      );
  }),
 ),
```

Output:



ListView.separated

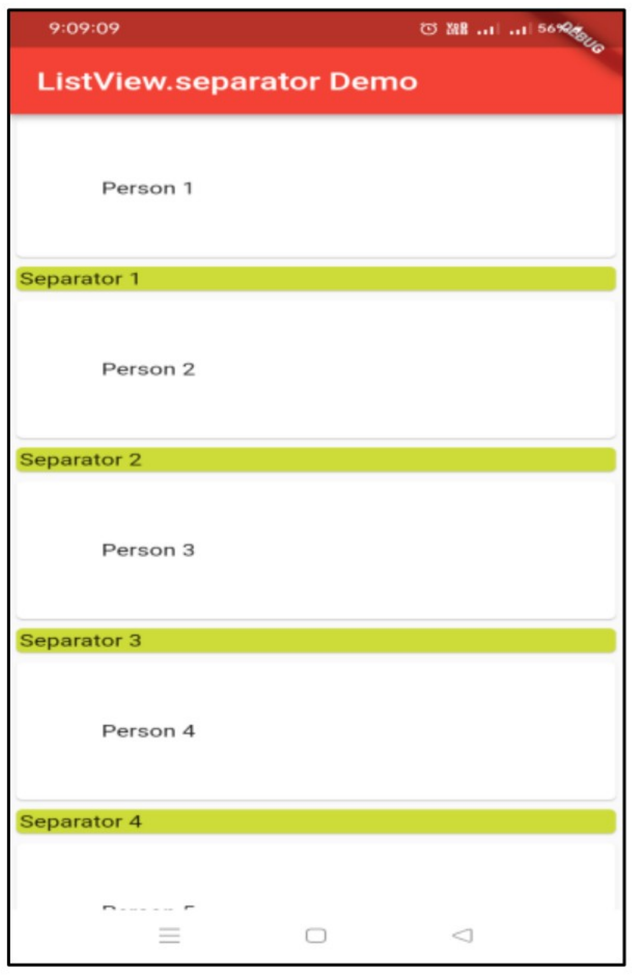
- ListView.separated() constructor is used to generate a list of widgets, but in addition, a separator widget can also be generated to separate the widgets.
- In the builder() constructor, the itemCount parameter is compulsory.

Example:

```
body: ListView.separated(
        itemCount:30,
        itemBuilder: (BuildContext context,int index)
        {
              return Card(
               child: Padding(
```

```
            padding: EdgeInsets.all(50),
            child: Text('Person ${index+1}'),
          ),
        );
},
separatorBuilder: (BuildContext context,int index){
  return Card(
    color: Colors.lime,
    child: Padding(
      padding: EdgeInsets.all(2),
      child: Text("Separator ${index+1}"),
    ),
  );
}),
```

Output:



ListView.custom

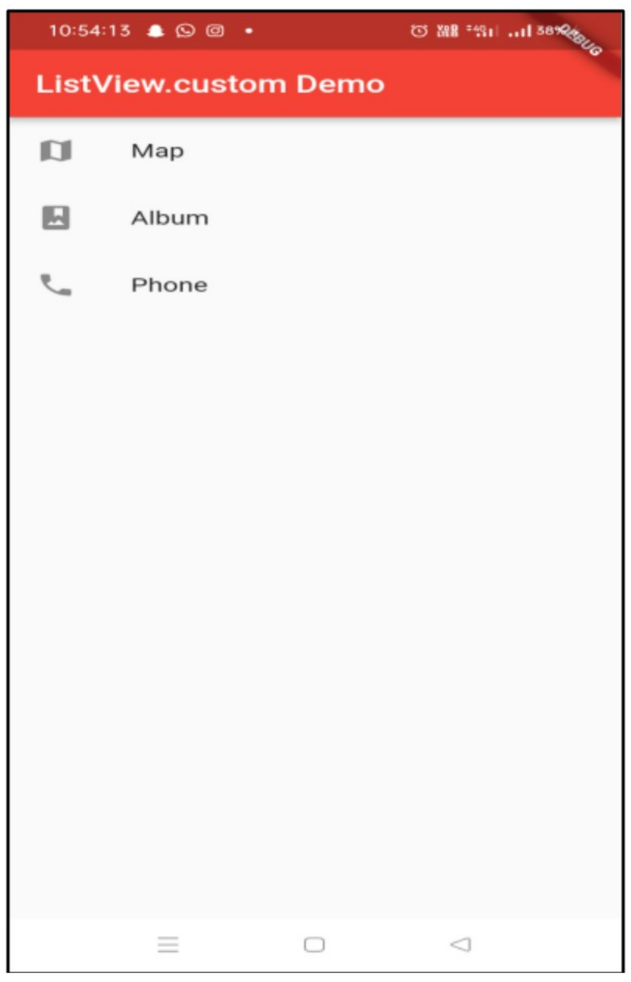
- ListView.custom() constructor build ListViews with custom functionality.
- The main parameter of this constructor is a **SliverChildDelegate** which builds the items.
- The types of SliverChildDelegates is:
  1. SliverChildListDelegate
  2. SliverChildBuilderDelegate
- SliverChildListDelegate accepts a list of children widgets.
- SliverChildBuilderDelegate accepts an IndexedWidgetBuilder.

Example:

```
        body: ListView.custom(
            childrenDelegate: SliverChildListDelegate(
              [
                ListTile(
```

```
  leading: Icon(Icons.map),
  title: Text('Map'),
 ),
 ListTile(
  leading: Icon(Icons.photo_album),
  title: Text('Album'),
 ),
          ListTile(
            leading: Icon(Icons.phone),
            title: Text('Phone'),
          ),
        ],
      ),
    ),
```

Output:



GridView

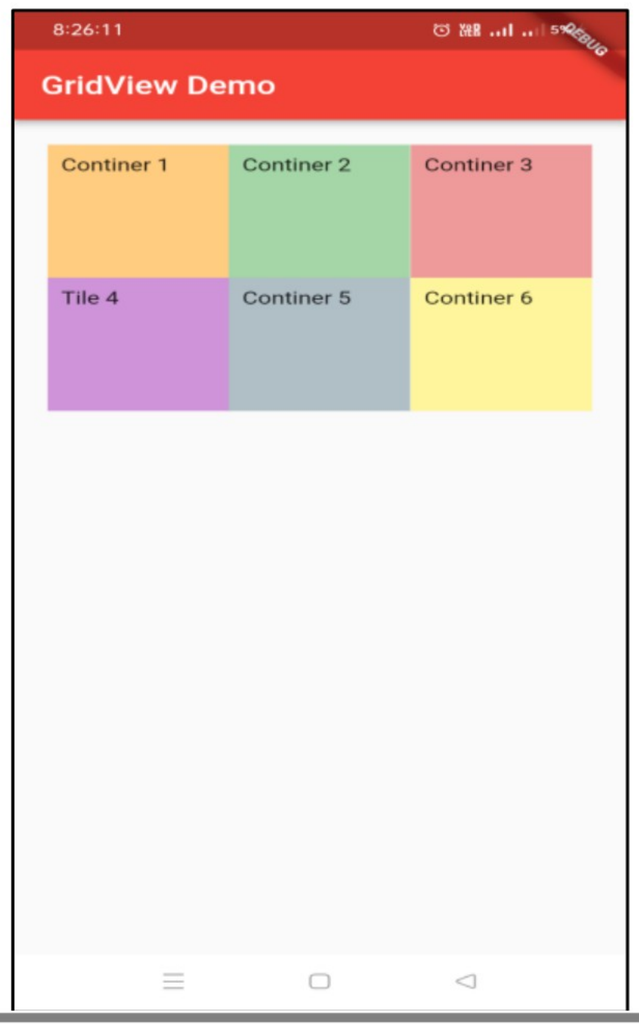
- Gridview is a graphical control element used to show items in the tabular format.
- GridView is a widget in Flutter that displays the items in a 2-D array (two-dimensional rows and columns).
- We can select the desired item from the grid list by tapping on them.
- This widget can contain text, images, icons, etc. to display in a grid layout depending on the user requirement.
- We can specify the direction in which it scrolls.

Example:

```
child: GridView(
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
     crossAxisCount: 3,
    ),
    padding: EdgeInsets.all(20),
```

```
children: [
  Container(
    padding: EdgeInsets.all(8),
    child: Text("Container 1"),
    color: Colors.orange[200],
  ),
  Container(
    padding:EdgeInsets.all(8),
    child: Text("Container 2"),
    color: Colors.green[200],
  ),
  Container(
    padding:EdgeInsets.all(8),
    child: Text("Container 3"),
    color: Colors.red[200],
  ),
  Container(
    padding:EdgeInsets.all(8),
    child: Text("Container 4"),
    color: Colors.purple[200],
  ),
  Container(
    padding:EdgeInsets.all(8),
    child: Text("Container 5"),
    color: Colors.blueGrey[200],
  ),
  Container(
    padding: EdgeInsets.all(8),
    child: Text("Container 6"),
    color: Colors.yellow[200],
  ),
```

Output:



Types of GridViews:

1. GridView.count()
2. GridView.builder()

GridView.Count()

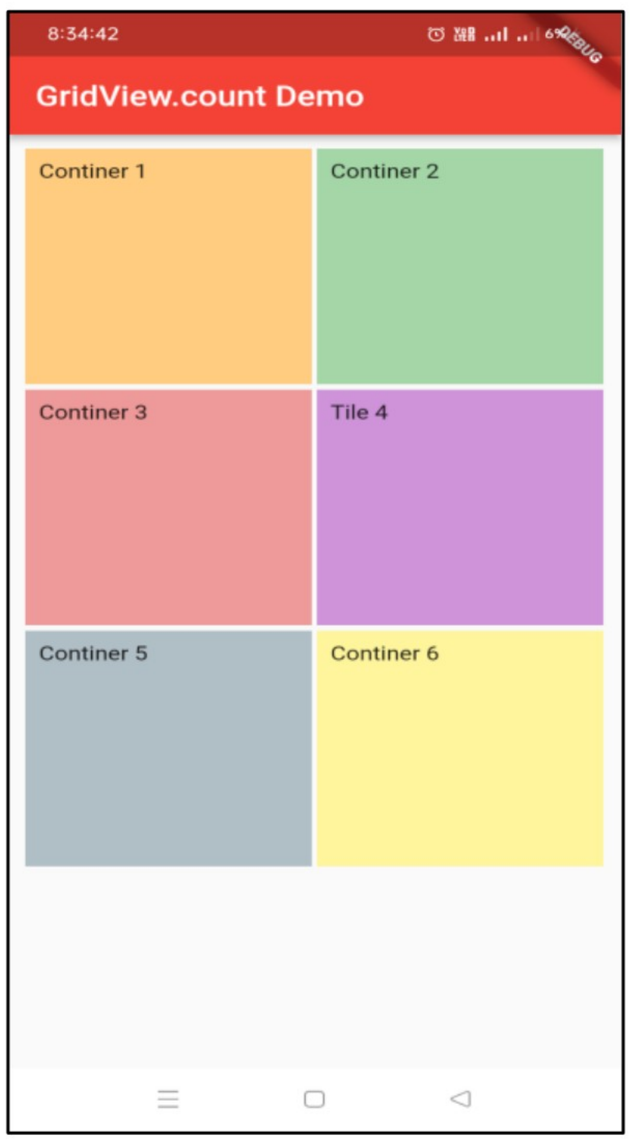
- It is the most frequently used grid layout in Flutter.
- It is used when we already know the grid's size. It allows developers to specify the fixed number of rows and columns.

Example:

```
child: GridView.count(
        crossAxisCount: 2,
        crossAxisSpacing: 3,
        mainAxisSpacing: 4,
        padding: EdgeInsets.all(10),
        children: [
         Container(
          padding: EdgeInsets.all(8),
          child: Text("Continer 1"),
          color: Colors.orange[200],
         ),
         Container(
          padding:EdgeInsets.all(8),
          child: Text("Continer 2"),
          color: Colors.green[200],
        ),
        Container(
          padding:EdgeInsets.all(8),
          child: Text("Continer 3"),
          color: Colors.red[200],
        ),
        Container(
          padding:EdgeInsets.all(8),
          child: Text("Tile 4"),
          color: Colors.purple[200],
        ),
        Container(
          padding:EdgeInsets.all(8),
          child: Text("Continer 5"),
          color: Colors.blueGrey[200],
        ),
        Container(
          padding: EdgeInsets.all(8),
          child: Text("Continer 6"),
```

```
        color: Colors.yellow[200],
      ),
    ],
  )
```

Output:



GridView.builder()

- This property is used when we want to display data dynamically or on-demand.
- If user wants to create a grid with a large (infinite) number of children, then they can use the GridView.builder() constructor with either a SliverGridDelegateWithFixedCrossAxisCount or a SliverGridDelegateWithMaxCrossAxisExtent.

  Example:

```
body: GridView.builder(gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 2,
  crossAxisSpacing: 2,
  mainAxisSpacing: 3),
    padding: EdgeInsets.all(10),
    itemCount: items.length,
   itemBuilder: (context,index){
return Image.network(items[index],
height: 100,
width: 100,
fit: BoxFit.fill,);
  }),
```

Output:

PageView

- PageView is a widget that generates the pages of the screen that are scrollable.
- PageView widget allows the user to transition between different screens in their flutter application.
- We need to set a **PageViewController** and a **PageView**.
- It works in both horizontal and vertical direction.
- Using builder function we can set fixed page or repeated page in a PageView.
- By default scrollDirection property is horizontal.
- We can also set scrollDirection property vertical to change page scrolling direction.

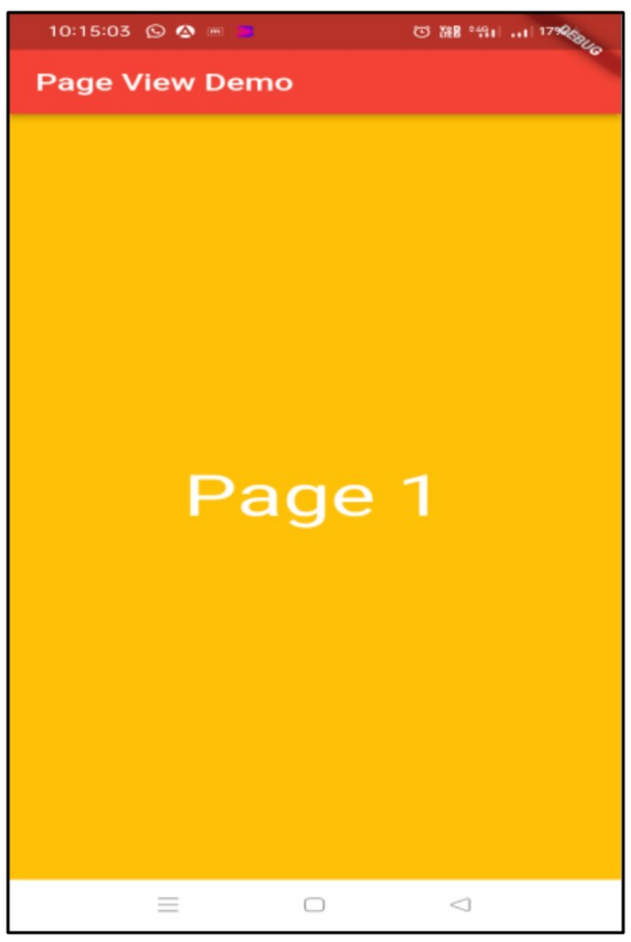
PageView

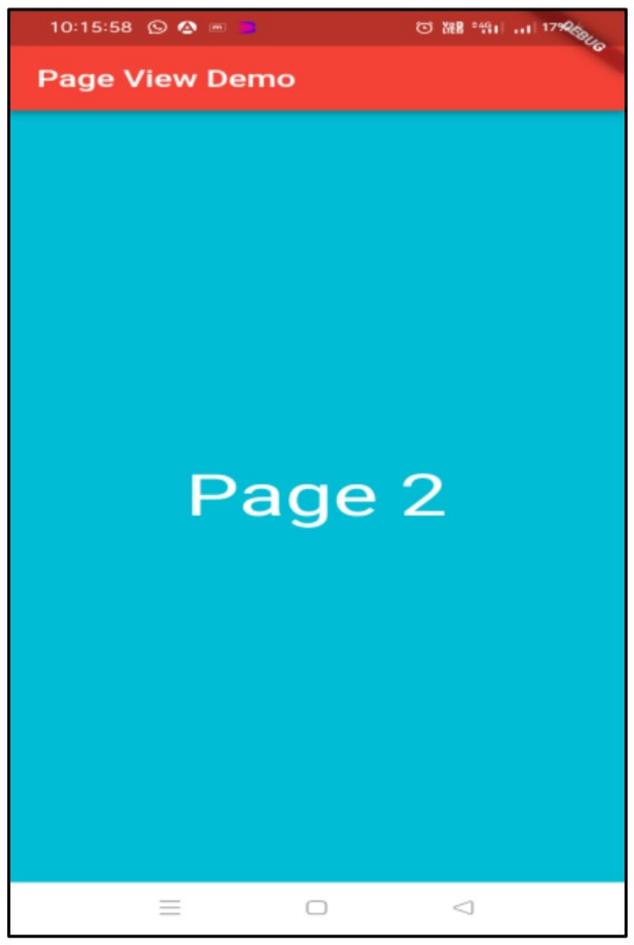
Example:

```
body: PageView(
        //  scrollDirection: Axis.vertical,
       children: [
        Container(
          color: Colors.amber,
          child: Center(child: Text("Page 1",style:TextStyle(color: Colors.white,fontSize: 50),),),
        ),
        Container(
```

```
  color: Colors.cyan,
  child: Center(child: Text("Page 2",style:TextStyle(color: Colors.white,fontSize: 50),),),
),
Container(
  color: Colors.pink,
  child: Center(child: Text("Page 3",style:TextStyle(color: Colors.white,fontSize: 50),),),
),
    Container(
      color: Colors.yellowAccent,
      child: Center(child: Text("Page 4",style:TextStyle(color: Colors.white,fontSize: 50),),),
    ),
  ],
),
```

Output:





PageViewController

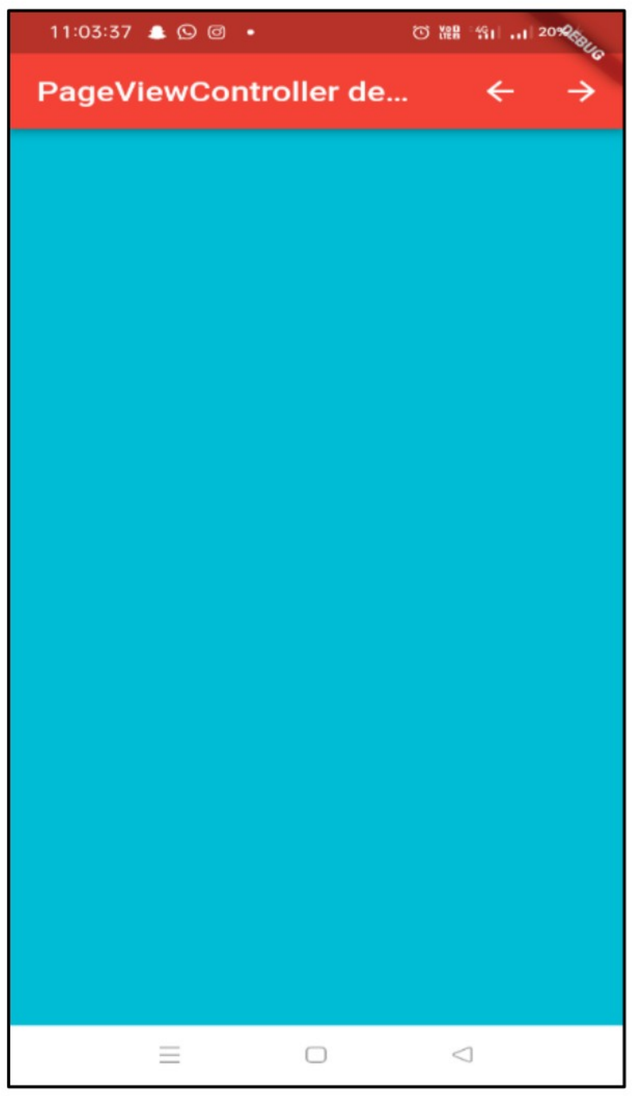
- A controller for PageView.
- It includes which page should be shown first, how each page should occupy the viewport, as well as whether to keep the page number when the scrollable is recreated.

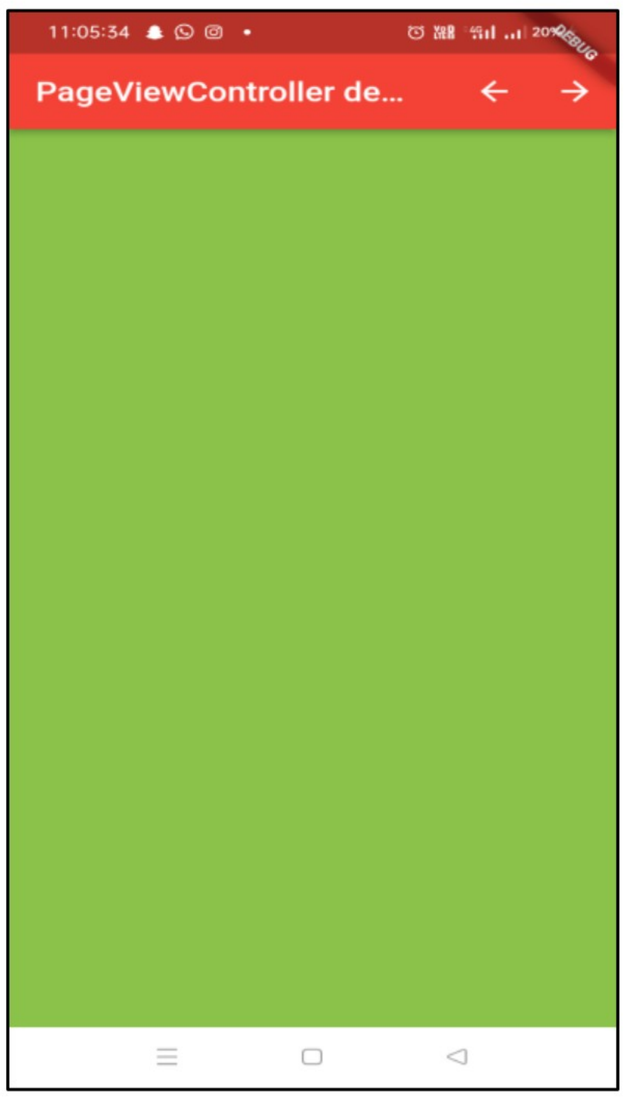
  Example:

```
home: Scaffold(
  appBar: AppBar(title: Text("PageViewController demo"),
backgroundColor: Colors.red,actions: [
  IconButton(onPressed:(){
  pageController.previousPage(duration: Duration(milliseconds: 300), curve:
Curves.easeInOut);
  }, icon: Icon(Icons.arrow_back)),

   IconButton(onPressed: (){
  pageController.nextPage(duration: Duration(milliseconds: 300), curve:
```

```
Curves.easeInOut);
                    }, icon: Icon(Icons.arrow_forward)),
                  ],),

                   body: PageView(
                   controller: pageController,
                   children: [
                    Container(color: Colors.cyan,),
                     Container(color: Colors.lightGreen,),
                     Container(color: Colors.amber,)
                   ],
                  ),
                ),
```

Output:





SingleChildScrollView

- SingleChildScrollView Widget is a box that can scroll a single widget.
- This widget is useful when you have a single box that will normally be entirely visible, for example a clock face in a time picker, but you need to make sure it can be scrolled if the container gets too small in one axis.
- It is also useful if we need to shrink-wrap in both axes (the main scrolling direction as well as the cross axis), as one might see in a dialog or pop-up menu.

Example:

```
body: SingleChildScrollView(
        padding: EdgeInsets.all(20),
        child: Column(
         children: [
          Container(
```

```
  height: 400,
  color: Colors.limeAccent,
),
Container(
  height: 400,
  color: Colors.pinkAccent,
)
        ],
      ),
    ),
```

Output:

NestedScrollView

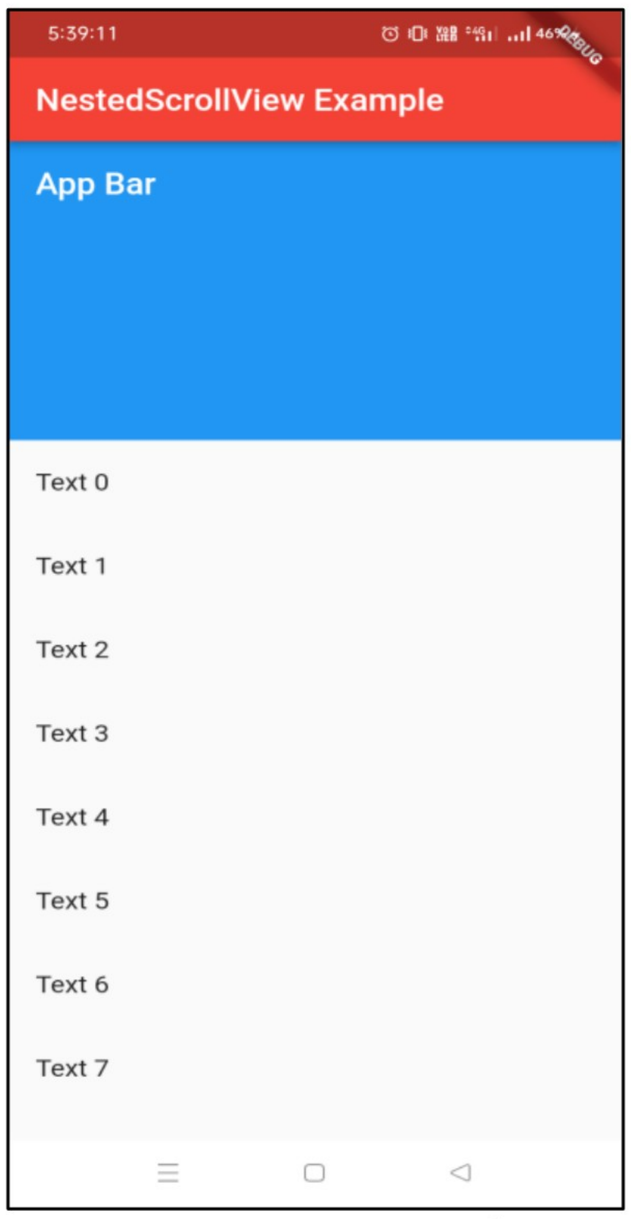
- A scrolling view inside scrolling view, is known as a NestedScrollView.
- It means that with NestedScrollView, you get two scrolling areas. One is the header part and the other is its body part.
- It connects these two parts so it behave like one consistent scrollable area. The two parts will scroll together and their scroll position will be linked.
- It is used with a SliverAppBar that has a TabBar in the header and a TabView in the body.
- The TabView scrolls horizontally and can have scrollable content that scrolls vertically.
- In many social media application NestedScrollView is available.

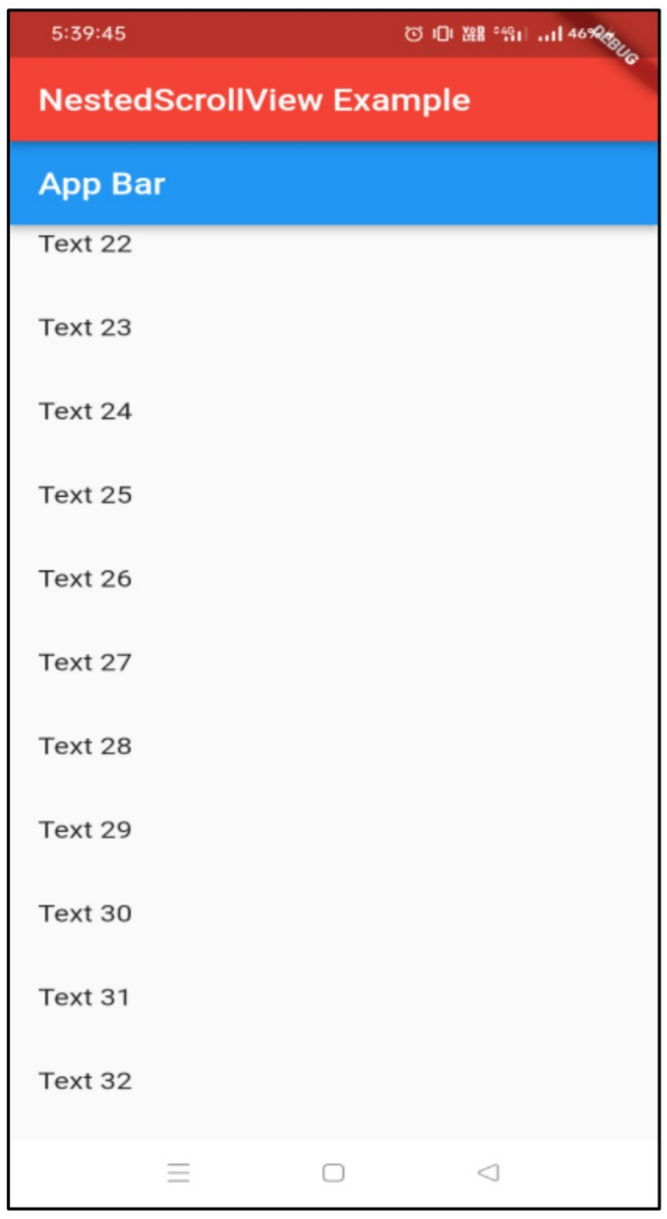
Example:

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("NestedScrollView Example"),backgroundColor: Colors.red,),
```

```
body: NestedScrollView(
  // This builds the scrollable content above the body
  headerSliverBuilder: (context, innerBoxIsScrolled) =>
  [
    SliverAppBar(
      title: const Text('App Bar'),
      expandedHeight: 200,
      pinned: true,
      forceElevated: innerBoxIsScrolled,
    ),
  ],
  // The content of the scroll view
  body: ListView.builder(
    itemBuilder: (context, index) =>
        ListTile(
          title: Text(
            'Text $index',
          ),
        ),
  ),
                    ),
                  ),
                );
              }
            }
```

Output:





Send data one page to another page

- We can pass data between screens using Navigator and the MaterialPageRoute class.
- Navigator.push user for send data one page to another page.
- Navigator widget is used to manage the navigation stack, while the MaterialPageRoute class defines the transition between two pages (screens).

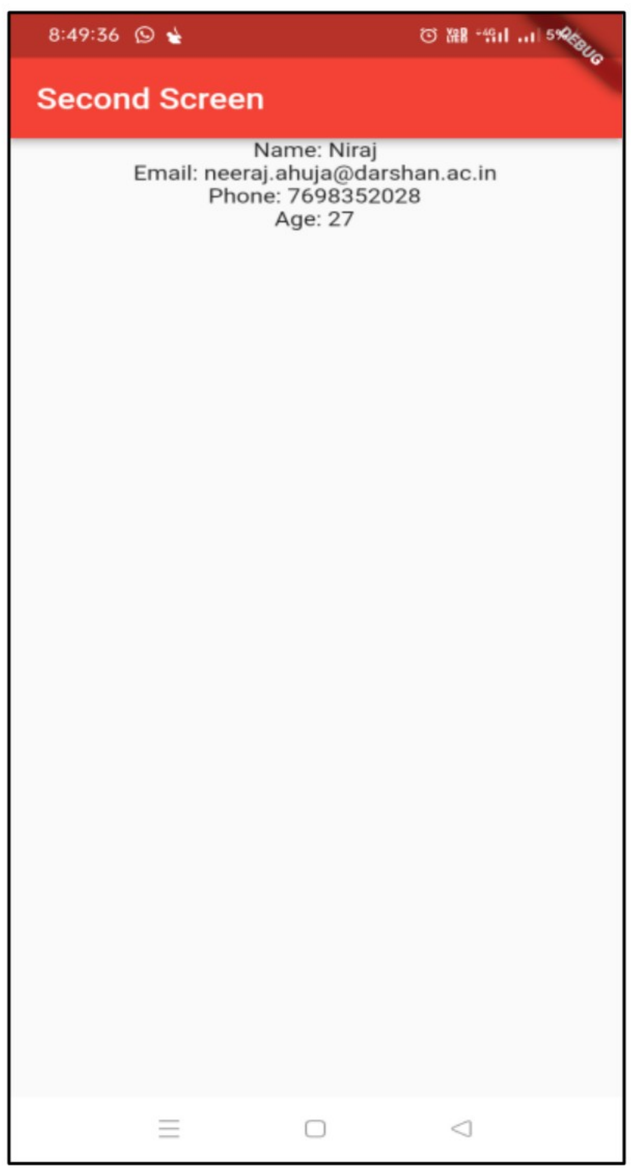
Example:

```
class data1 extends StatefulWidget {
        const data1({super.key});
    @override
        State<data1> createState() => _data1State();
                                        }
class _data1State extends State<data1> {
 final TextEditingController _name=TextEditingController();
 final TextEditingController _email=TextEditingController();
 final TextEditingController _phone=TextEditingController();
 final TextEditingController _age=TextEditingController();
@override
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(title: Text("First Screen"),backgroundColor: Colors.red,),
  body: Center(
   child: Column(
    children: [
     Padding(
      padding: const EdgeInsets.all(10.0),
```

```
child: TextField(
 controller: _name,
 decoration: InputDecoration(
  labelText: "Name",
  border: OutlineInputBorder(
   borderRadius: BorderRadius.circular(20.0)
  )
 ),
),
),
Padding(
 padding: const EdgeInsets.all(10.0),
 child: TextField(
  controller: _email,
  decoration: InputDecoration(
     labelText: "Email",
     border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(20.0)
     )
  ),
 ),
),
Padding(
 padding: const EdgeInsets.all(10.0),
 child: TextField(
  controller: _phone,
  decoration: InputDecoration(
     labelText: "Phone",
     border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(20.0)
     )
  ),
 ),
),
Padding(
 padding: const EdgeInsets.all(10.0),
 child: TextField(
  controller: _age,
  decoration: InputDecoration(
     labelText: "Age",
     border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(20.0)
```

```
            )
          ),
        ),
      ),
      ElevatedButton(onPressed: (){
        Navigator.of(context).push(MaterialPageRoute(builder:
(context)=>data2(name:_name.text,email:_email.text,phone:_phone.text,age:_age.text)));
      }, child: Text("Send"))
        ],
      ),
    ),
  );
 }
}
```

Output:



Return data to the Previous page

- We might want to return data from a new screen.
- Navigator.pop the top-most route off the navigator class.
- We now need to put await to stop and wait until the second screen is poped and we get our data.
- In the second screen, we can pop the screen using the function Navigator.pop().

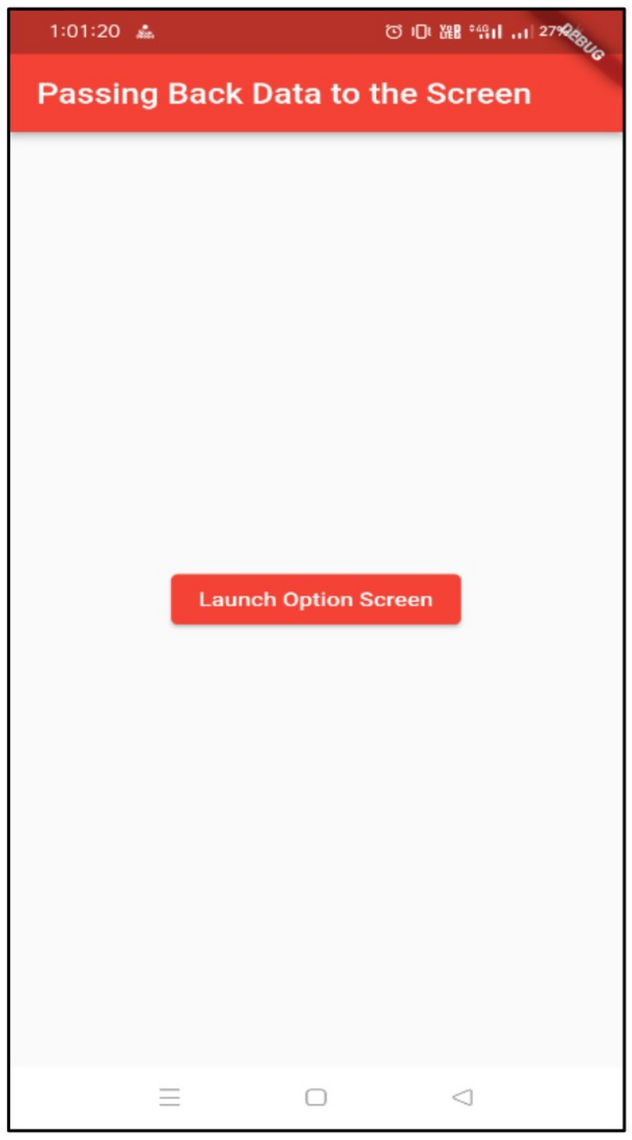
Example:

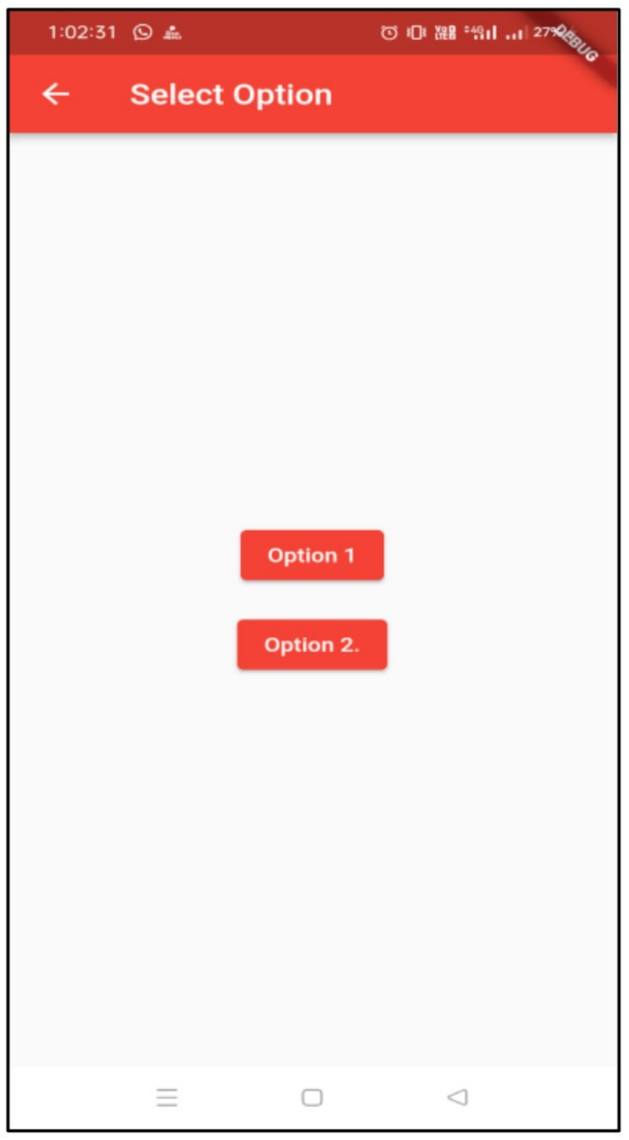
```
class Selection extends StatelessWidget {
  const Selection({Key? key}) : super(key: key);
```

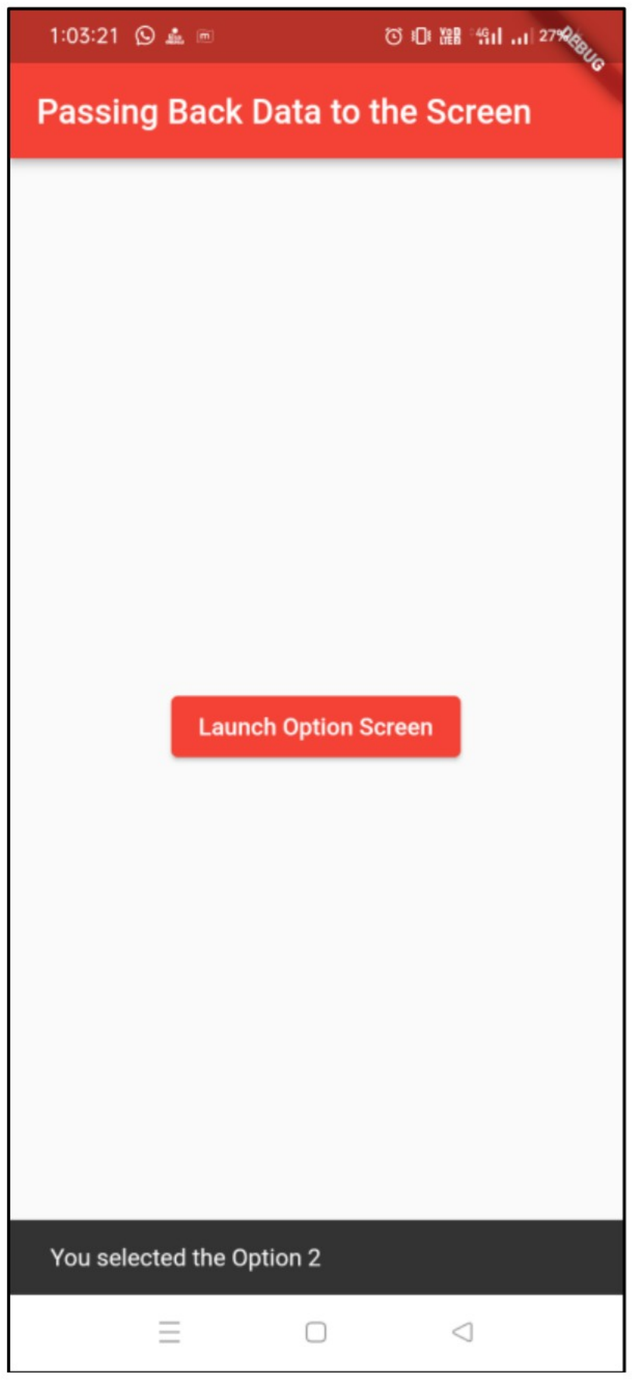
```
@override
Widget build(BuildContext context) {
 return ElevatedButton(
  style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
  onPressed: () {
   _navigate(context);
  },
   child: const Text('Launch Option Screen'),
  );
 }
 _navigate(BuildContext context) async {
  final result = await Navigator.push(
   context,
   MaterialPageRoute(builder: (context) => const SelectionScreen()),
  );
  ScaffoldMessenger.of(context)
   ..removeCurrentSnackBar()
   ..showSnackBar(SnackBar(content: Text("$result")));
 }
}
class SelectionScreen extends StatelessWidget {
 const SelectionScreen({Key? key}) : super(key: key);
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   appBar: AppBar(
    title: const Text('Select Option'),
    backgroundColor: Colors.red,
   ),
   body: Center(
    child: Column(
     mainAxisAlignment: MainAxisAlignment.center,
     children: <Widget>[
Padding(
 padding: const EdgeInsets.all(8.0),
 child: ElevatedButton(
  style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
  onPressed: () {
   Navigator.pop(context, ' You selected the Option 1');
  },
  child: const Text('Option 1'),
 ),
```

```
),
Padding(
  padding: const EdgeInsets.all(8.0),
  child: ElevatedButton(
    style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
    onPressed: () {
      Navigator.pop(context, 'You selected the Option 2');
                    },
                    child: const Text('Option 2.'),
                  ),
                )
              ],
            ),
          ),
        );
      }
    }
```

Output:






1:03:21
DEBUG
Passing Back Data to the Screen
Launch Option Screen
You selected the Option 2