502071

Cross-Platform Mobile Application Development

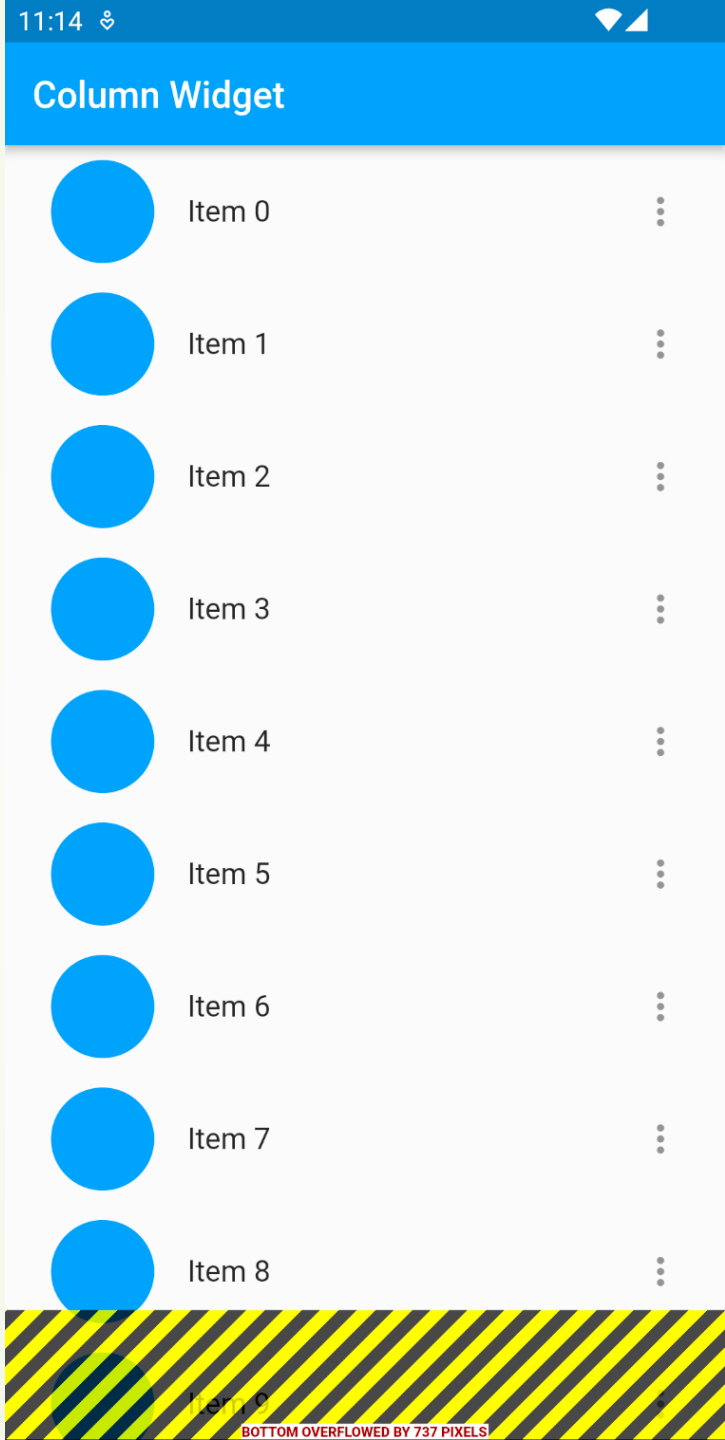
List-based Widgets

1

Listview

Drawbacks of Row/Column

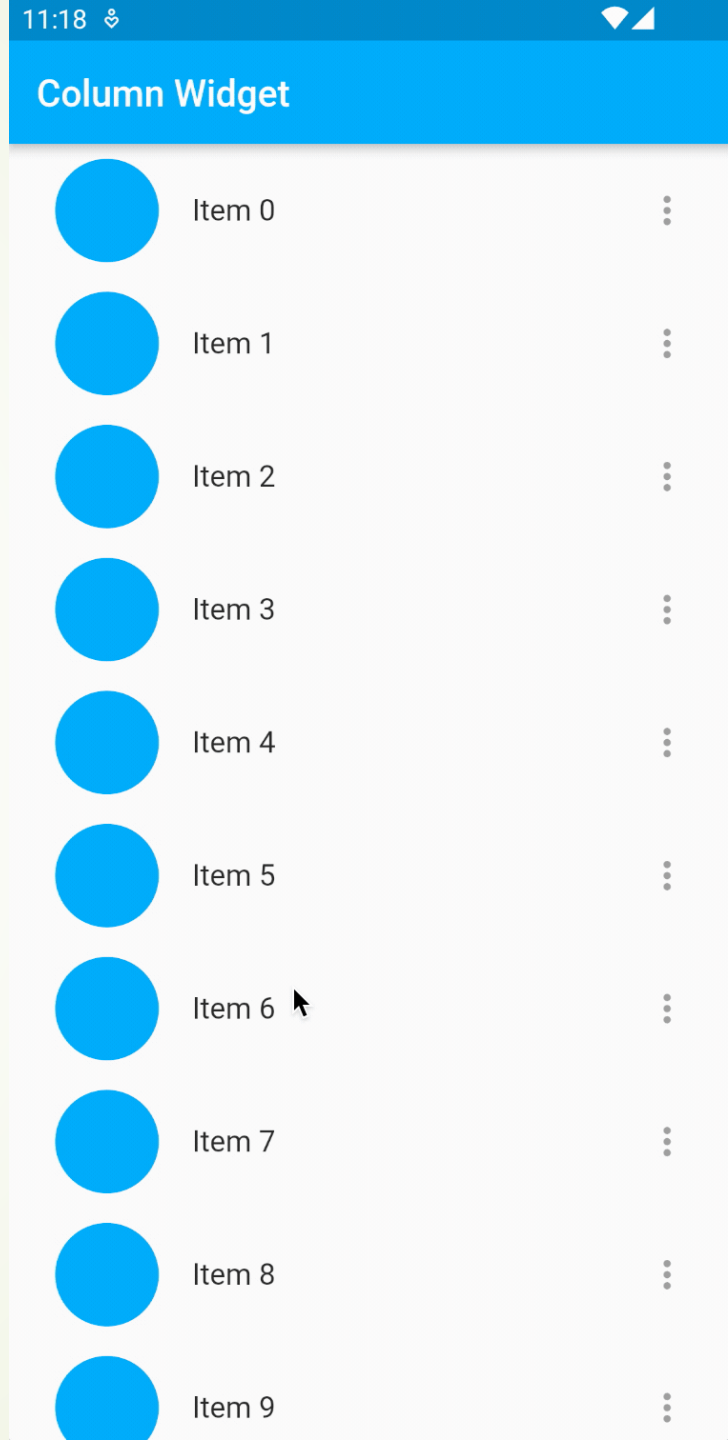
- We can use Row or Colum widget to create a vertical/horizontal list of widgets.
- The Row/Column widget in Flutter can overflow if it contains too many children or if the children have too much content to fit within the available space.
- That is because the Column/Row widget is not scrollable.



Drawbacks of Row/Column

- To fix the overflow error in a Column widget, we can wrap the Column in a SingleChildScrollView.
- A SingleChildScrollView is a widget that allows scrolling when its content overflows the available space.
- By wrapping the Column in a SingleChildScrollView, you can make the column scrollable and prevent the overflow error.

```
SingleChildScrollView(
    child: Column(
        children: [],
    ),
)
```



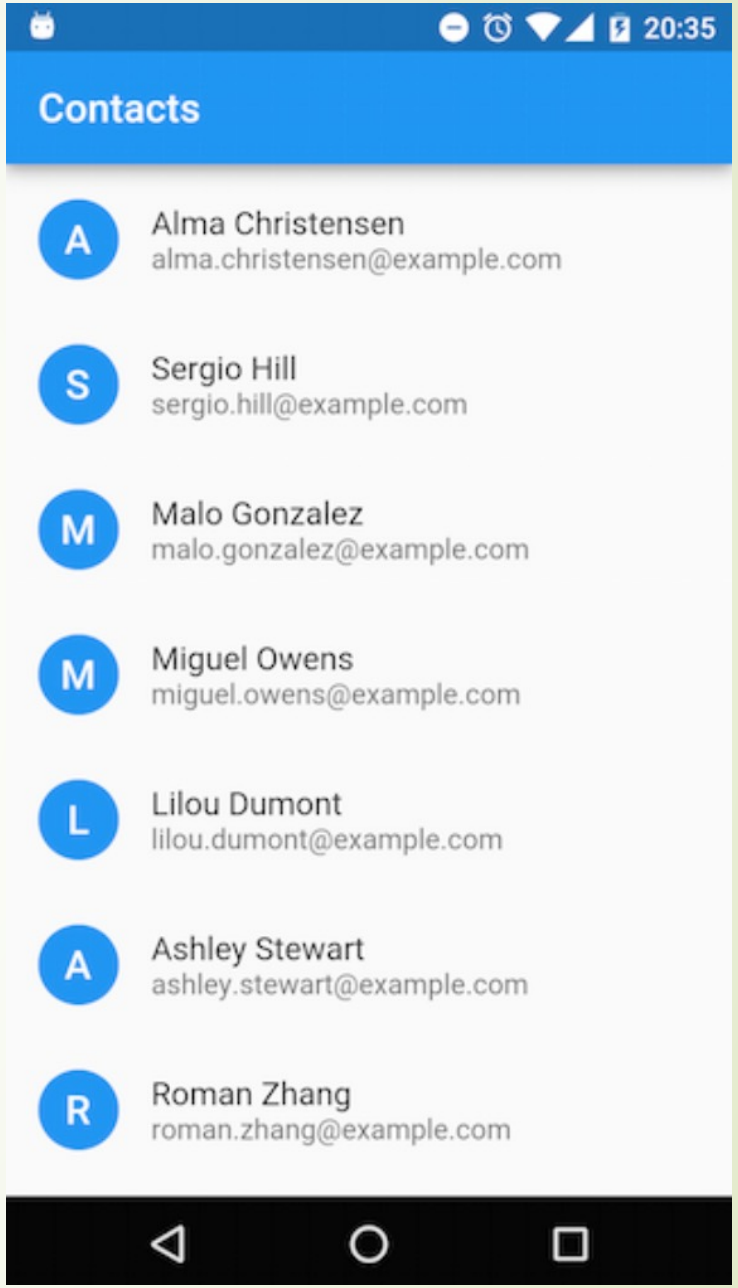
Drawbacks of Row/Column

- Wrapping a Column inside a SingleChildScrollView can be a simple and effective way to make a column scrollable and prevent overflow errors. However, it can also have some drawbacks, particularly when dealing with a large number of items.
- The main drawback of using a SingleChildScrollView with a Column is performance. When a SingleChildScrollView is used with a Column, all the children of the column are built and laid out, even if they are not visible on the screen. This can result in a significant performance hit when dealing with a large number of items or complex widgets.
- Another drawback of using a SingleChildScrollView with a Column is that it can be difficult to control the layout of the children. Since the Column lays out its children vertically, it can be challenging to control the horizontal alignment or spacing of the items.

```
SingleChildScrollView(
    child: Column(
        children: [],
    ),
)
```

Listview

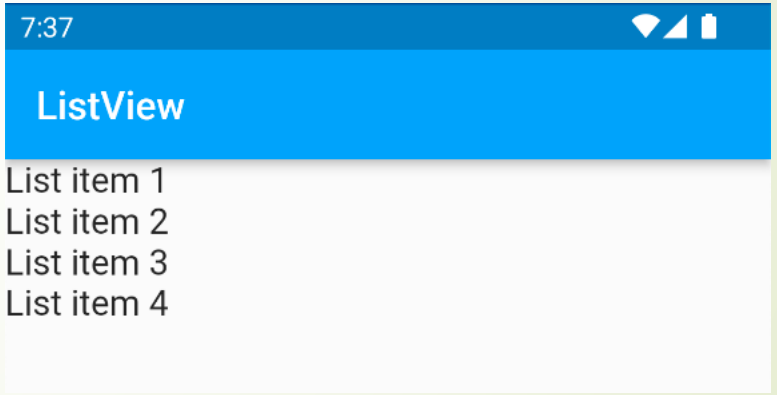
- ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction. In the cross axis, the children are required to fill the ListView.
- ListView is a highly customizable widget that can be used for many different types of lists, such as linear lists, grid lists, and more.



Listview

- The default constructor takes an explicit List<Widget> of children. This constructor is appropriate for list views with a small number of children because constructing the List requires doing work for every child that could possibly be displayed in the list view instead of just those children that are actually visible.

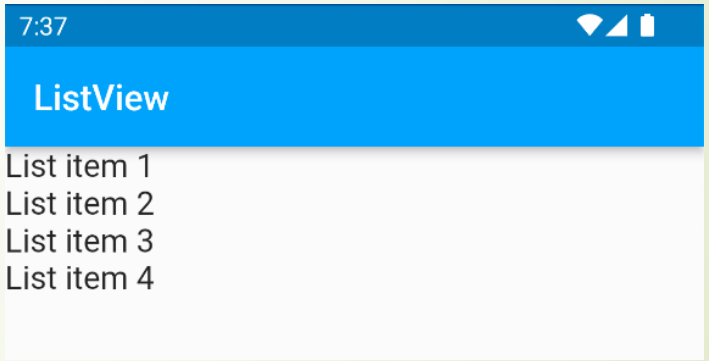
```
ListView(
    children: [
        Text('List item 1’),
        Text('List item 2’),
        Text('List item 3’),
        Text('List item 4’),
    ],
)
```



Listview

- The ListView.builder constructor takes an IndexedWidgetBuilder, which builds the children on demand. This constructor is appropriate for list views with a large (or infinite) number of children because the builder is called only for those children that are actually visible.

```
var items = ['List item 1', 'List item 2',
    'List item 3', 'List item 4', 'List item 5'
];
ListView.builder(
    itemBuilder: (context, index) => Text(items[index]),
    itemCount: items.length,
)
```



ListView vs SingleChildScrollView

- In contrast to using SingleChildScrollView and Column, when using ListView.builder, Flutter only builds and lays out the items that are visible on the screen, which can help to optimize performance, especially when dealing with a large number of items.

```
SingleChildScrollView(
    child: Column(
        children: [],
    ),
)
```

```
ListView.builder(
    itemCount: 10000,
    itemBuilder: (ct, index) => _buildItemWidget(index),
)
```

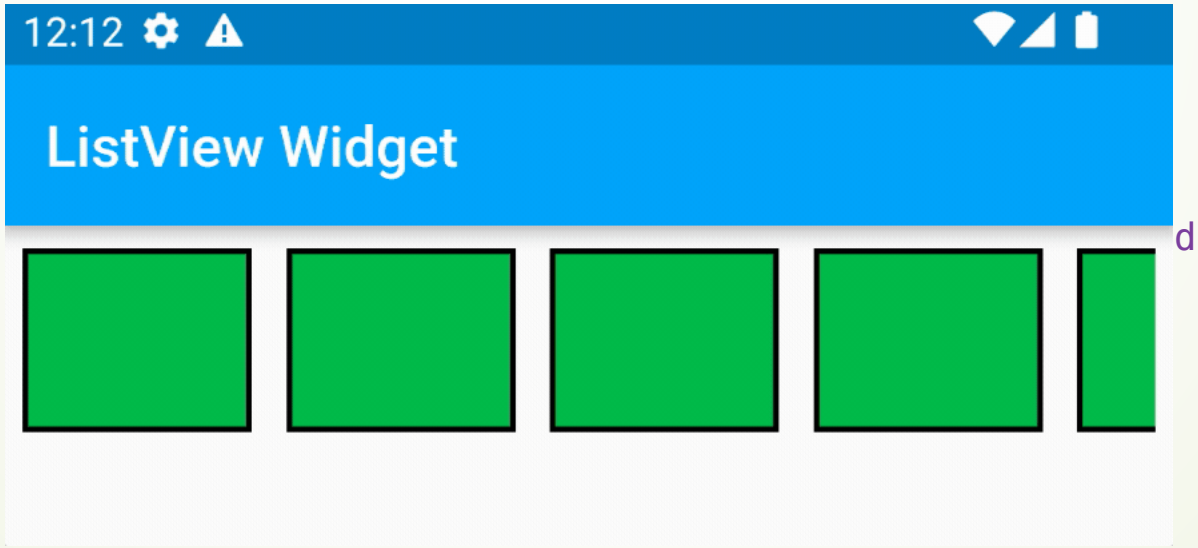
Listview

- The ListView.separated constructor takes two IndexedWidgetBuilders: itemBuilder builds child items on demand, and separatorBuilder similarly builds separator children which appear in between the child items. This constructor is appropriate for list views with a fixed number of children.

```
ListView.separated(
      itemBuilder: (context, index) => Text(items[index]),
      separatorBuilder: (context, index) =>  const Divider(height:1),
      itemCount: items.length
)
var items = ['List item 1', 'List item 2',
      'List item 3', 'List item 4', 'List item 5'];
```

Horizontal Listview

- The ListView widget supports horizontal lists.
- Use the standard ListView constructor, passing in a horizontal scrollDirection, which overrides the default vertical direction.




```
dth: 12,),
```

Special handling for an empty list

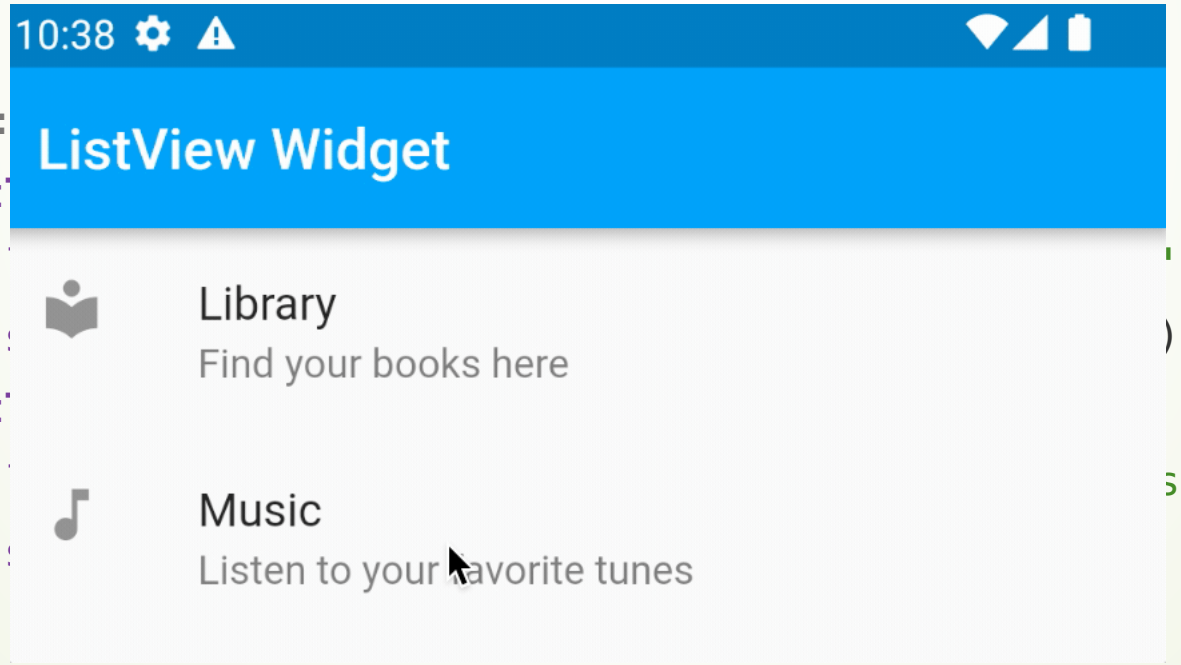
- A common design pattern is to have a custom UI for an empty list. The best way to achieve this in Flutter is just conditionally replacing the ListView at build time with whatever widgets you need to show for the empty list state.

```
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(title: const Text('Empty List Test')),
        body: (itemCount > 0) ? ListView(…) : Text('No items'),
    );
}
```

ListTile

- ListTile is a convenient widget in Flutter that allows you to easily create a list item with options for leading and trailing icons, title, subtitle, and other widgets. It is commonly used in the ListView widget to create a scrollable list of items.

```
ListView(
    children:
        List
        List
    ],
)
```

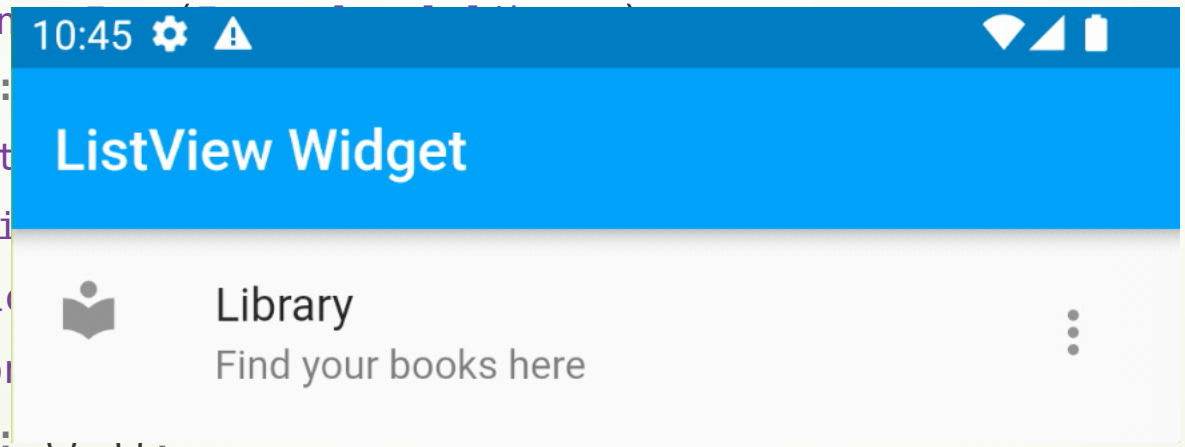



```
'Library'),
) {}),
sic'),
onTap: () {}),
```

ListTile

- You can customize the appearance and behavior of ListTile by passing different arguments to its constructor. For example, you can set the trailing argument to add a trailing icon, the contentPadding argument to set the padding around the content, and the dense argument to make the tile smaller and more compact.

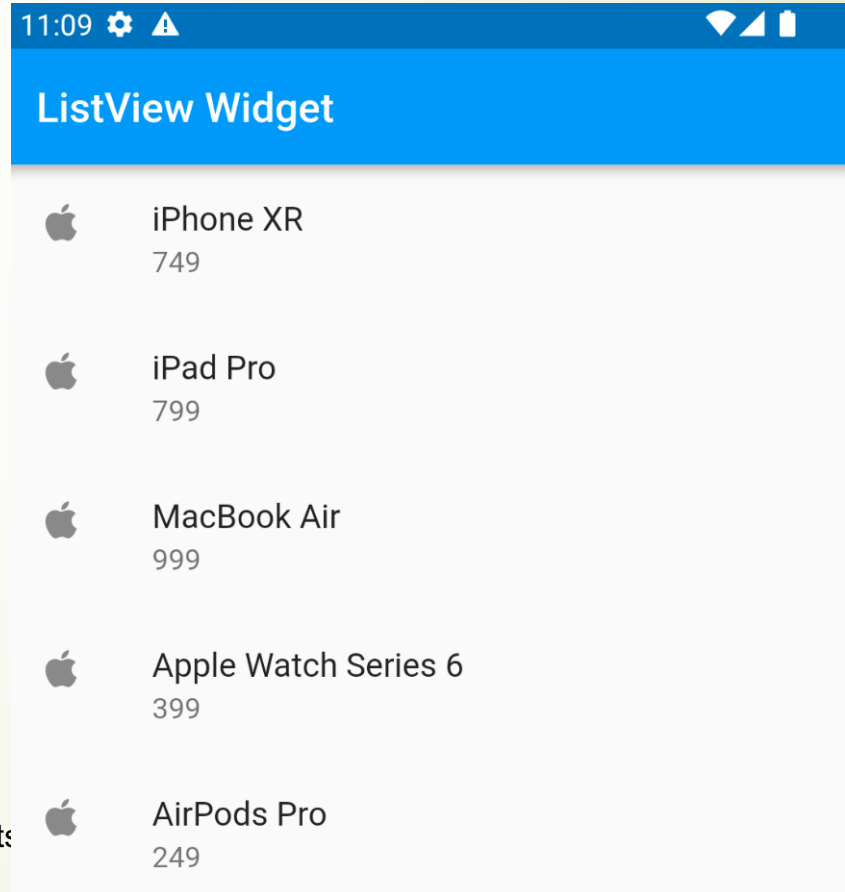
```
ListTile(
    leadin
    title:
    subtit
    traili
        i
        o
    onTap:
)
```



Dynamic ListView from dataset

- We can also create a listview from a dataset in Dart list by using the map() method.
- **Syntax**: `items.map(e => newItem(e)).toList()`

```
var products = [{'name': 'iPhone XR', 'price': 749}, …];
ListView(
    children: products.map((e) => ListTile(
        onTap: () {},
        leading: Icon(Icons.apple),
        title: Text(e['name'].toString()),
        subtitle: Text(e['price'].toString()))).toList(),
)
```



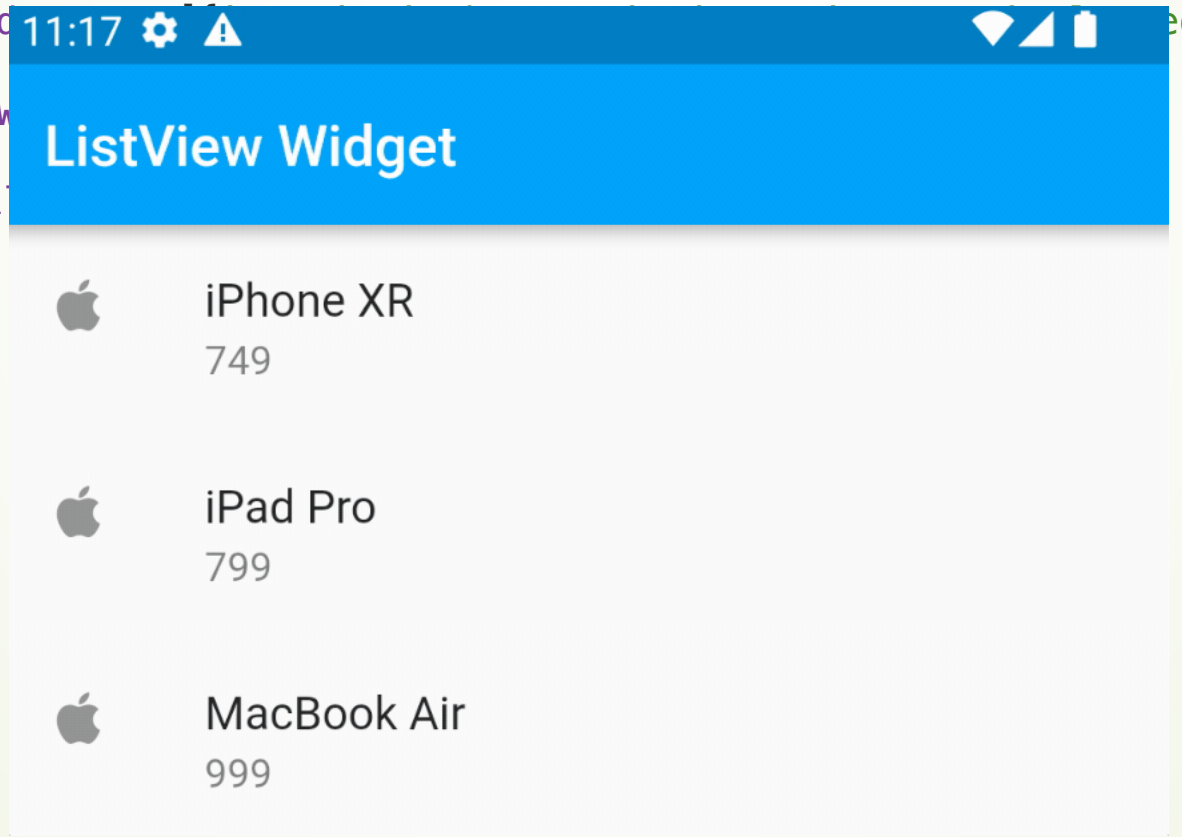
Selection of list items

- ListView has no built-in notion of a selected item or items. For a small example of how a caller might wire up basic item selection, see ListTile.selected.

```
var prod                                                    ed': false}, …];

ListView

    chi
```

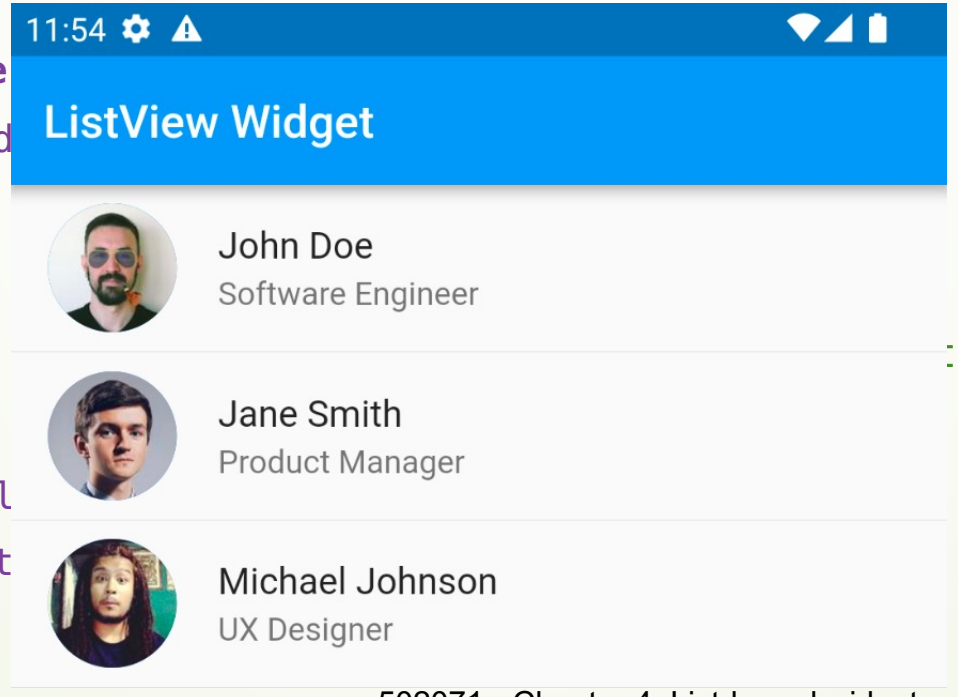



```
subtitle: Text(e['price'].toString()).toList(),
```

CircleAvatar

- The CircleAvatar widget in Flutter is a widget that displays an avatar in a circular shape. It can be used to display a user profile picture, or an image representing a particular item. The CircleAvatar widget automatically clips the image or child widget to a circular shape, which makes it useful for displaying circular profile pictures or images.

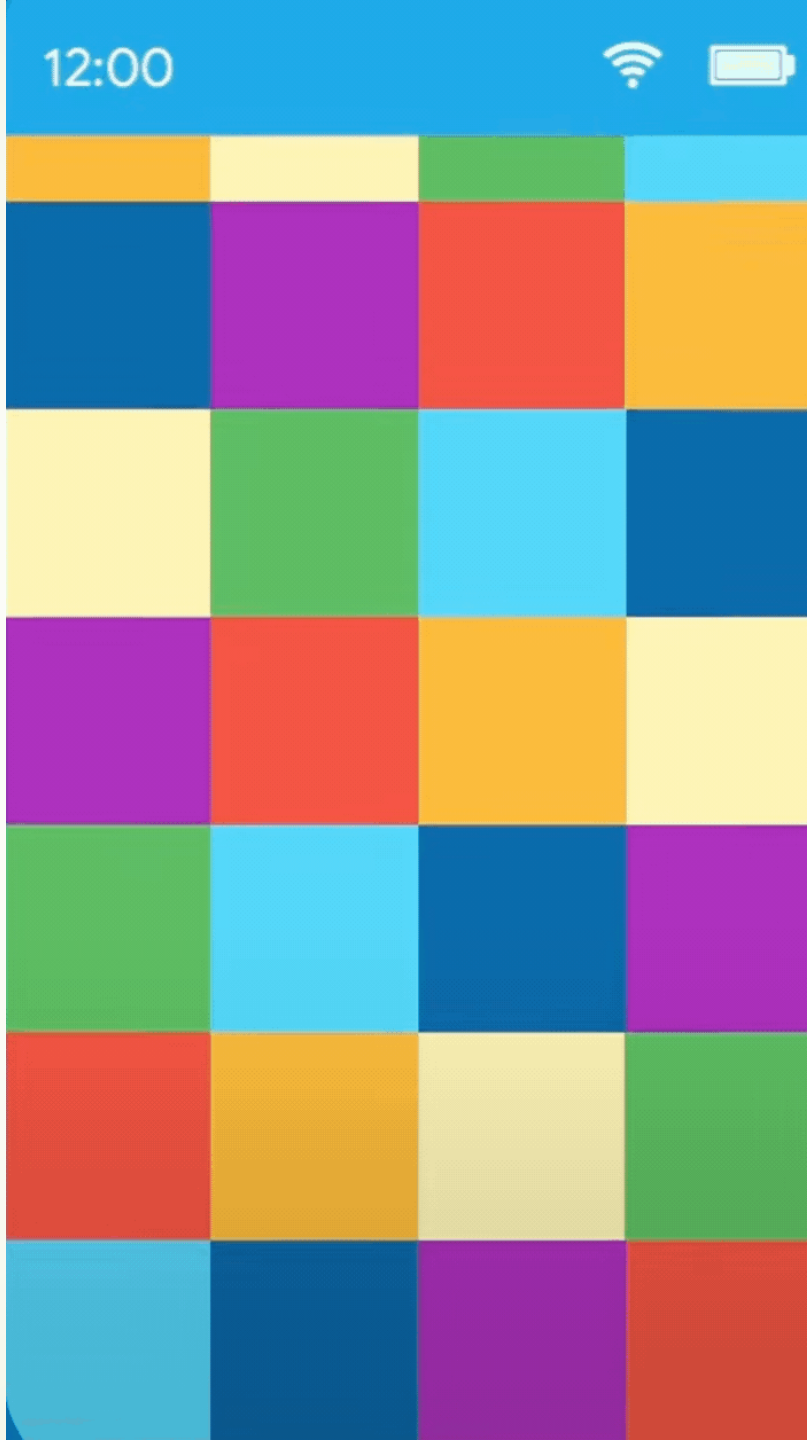
```
ListTile
    lead
                                                    :s/men/1.jpg'),
    titl
    subt                                                ,
)
```



GridView

GridView

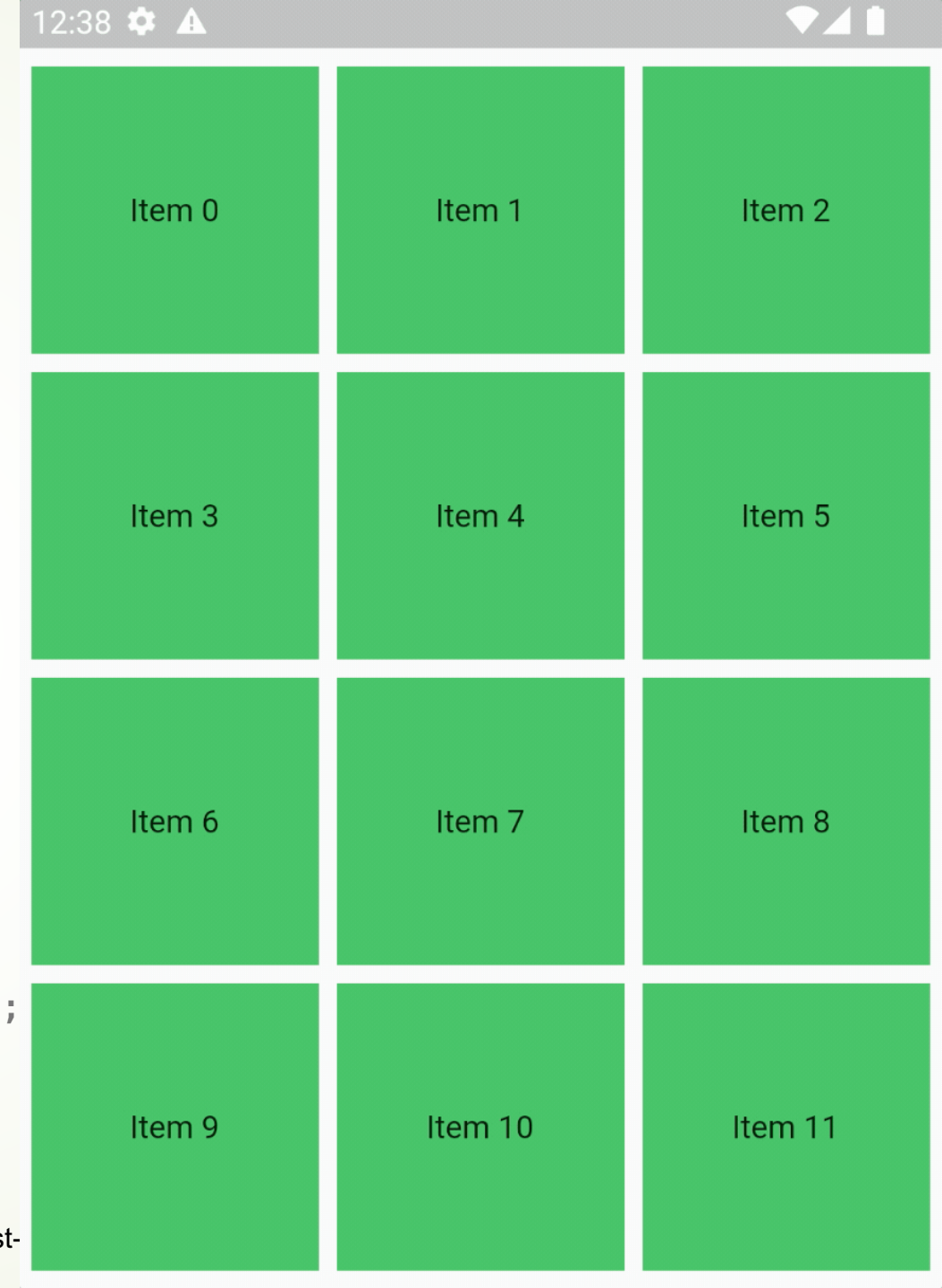
- GridView is a widget in Flutter that is used to create a scrollable grid of widgets. It's an efficient way of displaying a large number of widgets in a limited amount of space.
- You can use the GridView widget to create a grid of widgets that are evenly spaced, either horizontally or vertically, based on the size of the screen.



GridView

- The most commonly used grid layouts are GridView.count, which creates a layout with a fixed number of tiles in the cross axis.

```
GridView.count(
    crossAxisCount: 3,
    crossAxisSpacing: 8,
    mainAxisSpacing: 8,
    children: List.generate(30, (index) {
        return Container(
            color: Colors.green[400],
            child: Center(child: Text("Item $index")),);
    }),
)
```



GridView

- The GridView widget also provides several constructors for creating grids with more advanced layouts, including the GridView.builder constructor for creating grids with a large number of items, and the GridView.extent constructor for creating grids with items of a fixed size.

```
GridView.builder(
    itemCount: 100,
    gridDelegate: ...,
    itemBuilder: ...
)
```

GridView

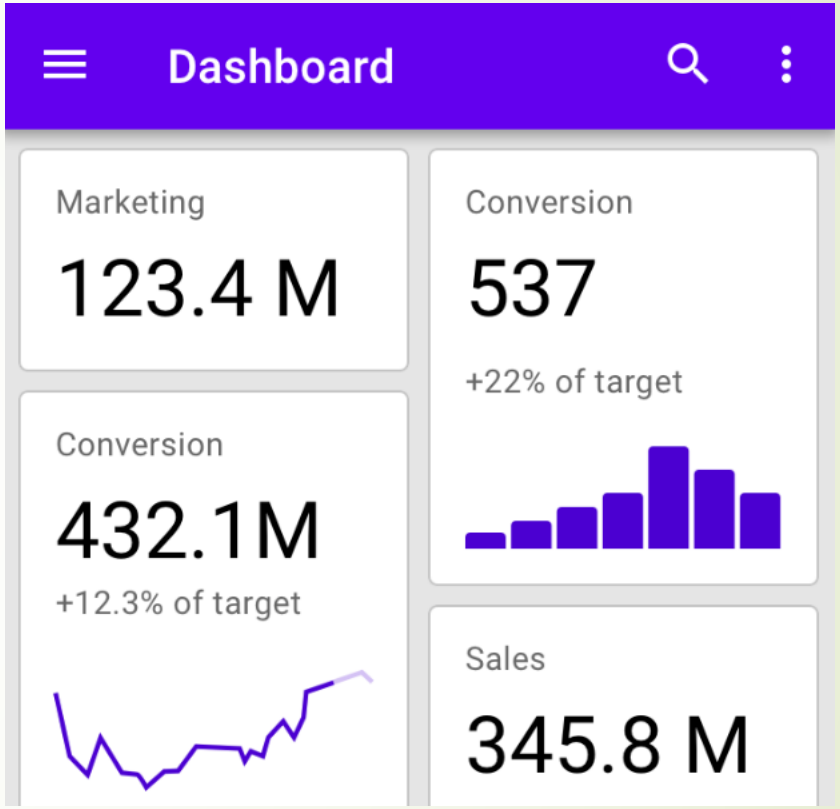
- To create a grid with a large (or infinite) number of children, use the GridView.builder constructor with either a SliverGridDelegateWithFixedCrossAxisCount or a SliverGridDelegateWithMaxCrossAxisExtent for the gridDelegate.

```
GridView.builder(
    itemCount: 100,
    gridDelegate: const
    SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 3,
        crossAxisSpacing: 8,
        mainAxisSpacing: 8,
    ),
    itemBuilder: (context, index) {
        return Container(
            color: Colors.green[400],
            child: Center(child: Text("Item $index")),
        );
    })
```

Card widget

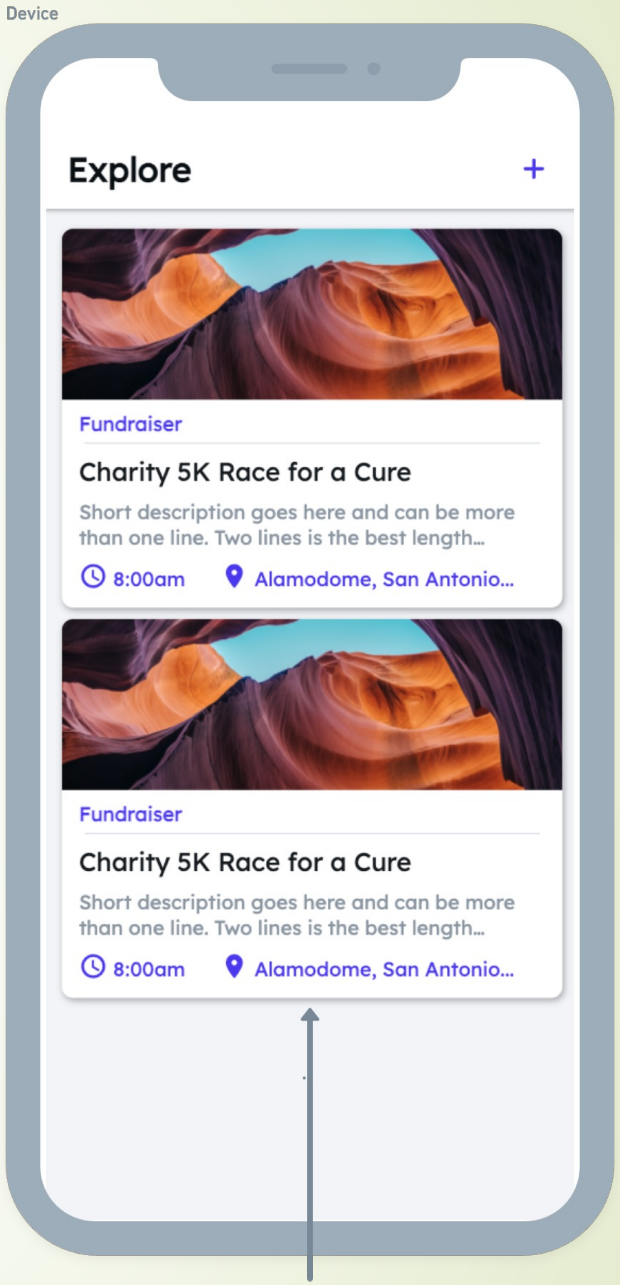
Card widget

- The Card widget in Flutter is a container that provides a visually appealing and stylized interface for displaying information, such as images, text, and other widgets. It is often used to create a tile-based user interface, which can be useful for displaying content in a grid format or for creating a list view.



Card widget

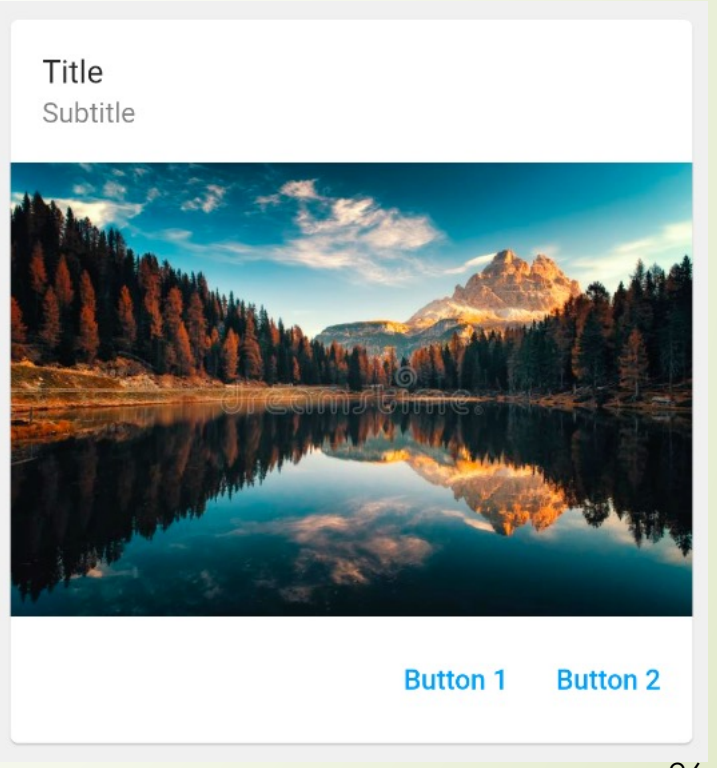
- The basic structure of a Card widget consists of a container that contains a child widget, and optionally, a header and footer. The container has a background color, elevation, and border radius, giving the card a 3D effect.



Card widget

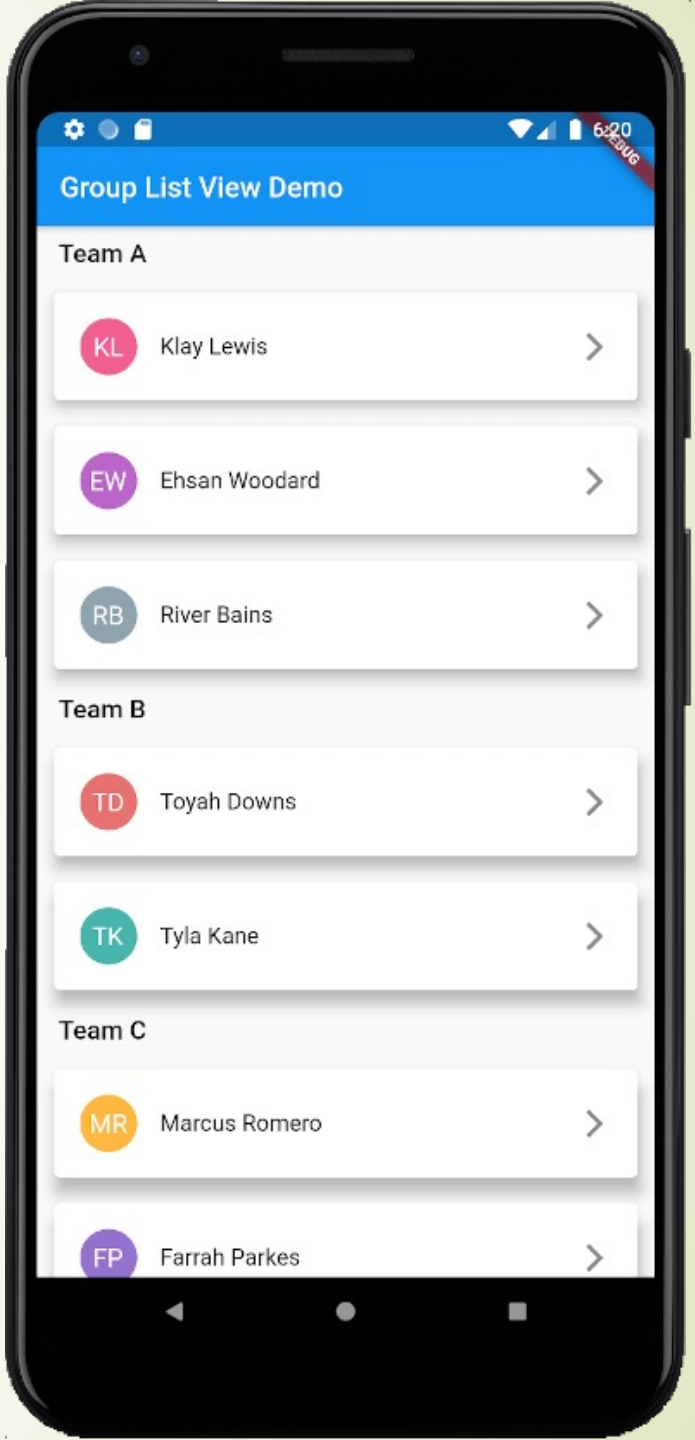
- Here is a basic example of Card with an image and two text buttons

```
Card(child: Column(
    mainAxisSize: MainAxisSize.min,
    children: [
        ListTile(title: Text('Title'),subtitle: Text('Subtitle'),),
        Image.network('https://image.jpg'),
        ButtonBar(
            children: [
                TextButton(child: const Text('Button 1'), onPressed
                TextButton(child: const Text('Button 2'), onPressed
            ],
        ),
    ]),
)
```



Card widget

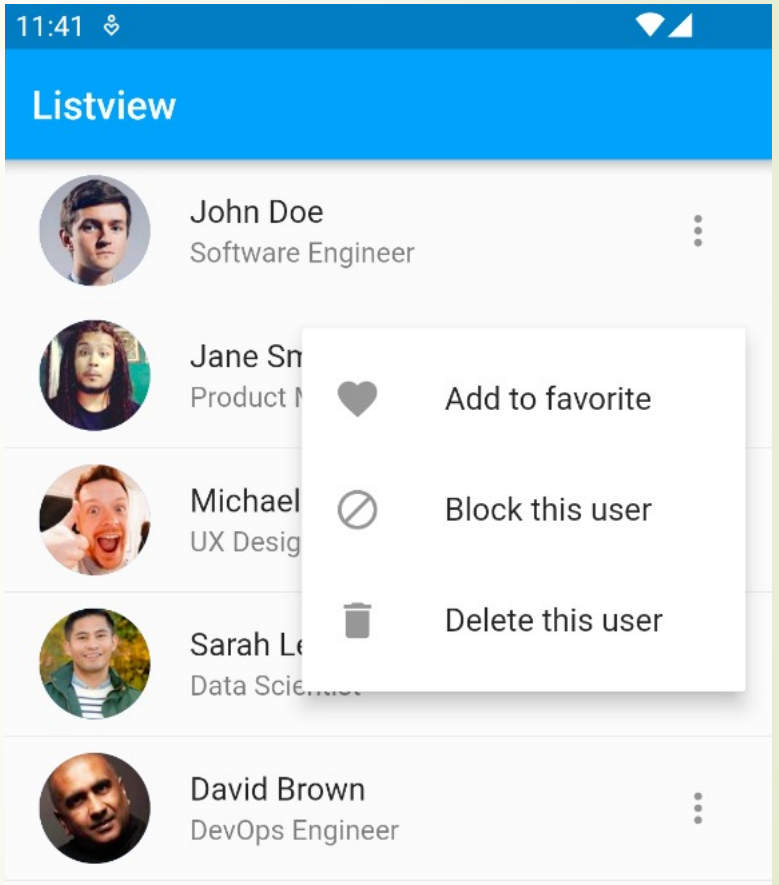
- Card widgets are also commonly used inside GridView or ListView as placeholders for grid/list items. Instead of using the widget container, the widget card already supports properties like rounded corners, drop shadows, and so on.



Popup Menu

Popup Menu

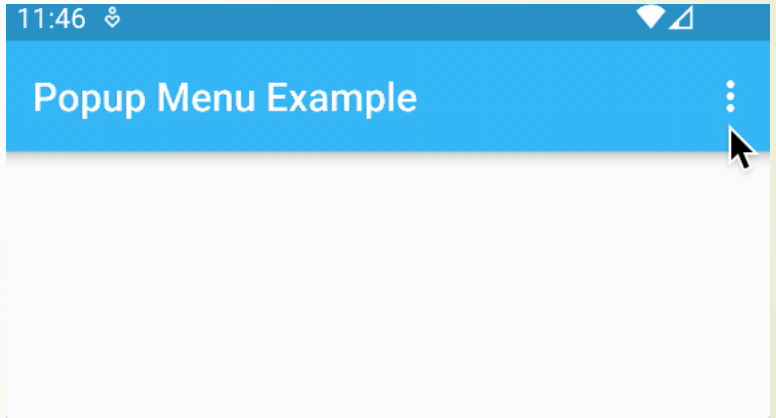
- The PopupMenu widget in Flutter is a widget that displays a list of options in a pop-up menu when the user taps on a button or other widget. It is similar to the context menu in desktop applications or the "three dots" menu in mobile apps.
- The PopupMenu widget is useful for providing additional options or actions that are not immediately visible on the screen.



Popup Menu

- In this example, we're using the PopupMenuButton widget to create a button that displays a pop-up menu when tapping on the AppBar action item.

```
PopupMenuButton<String>(
    onSelected: (value) { print('Selected: $value'); },
    itemBuilder: (context) => [
        const PopupMenuItem(
            value: 'option 1',
            child: Text('Option 1'),
        ),
        const PopupMenuItem(
            value: 'option 2',
            child: Text('Option 2'),
        ),
    ]},
),
```



Popup Menu

- There are several properties of the PopupMenuButton widget that we can use to customize its behavior:
  - onSelected: This property is a callback that is called when the user selects an option from the menu.
  - itemBuilder: This property is a callback that returns a list of PopupMenuItem widgets that make up the menu.
  - child: This property is the widget that the user taps to display the pop-up menu.
  - icon: This property is the icon that is displayed on the PopupMenuButton widget.
  - offset: This property is the offset of the menu from the button.
  - enabled: This property determines whether the button is enabled or disabled.

Popup Menu

- The PopupMenuButton widget has an onSelected callback that is called when the user selects an item from the menu, while the PopupMenuItem widget has an onTap callback that is called when the user taps on the item.

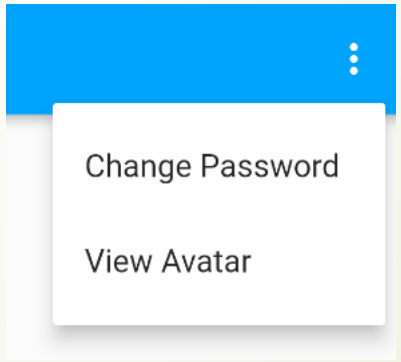
- The onSelected callback of the PopupMenuButton widget is a function that takes a single argument, which is the value of the selected item.
- On the other hand, the onTap callback of the PopupMenuItem widget is a function that is called when the user taps on the item.

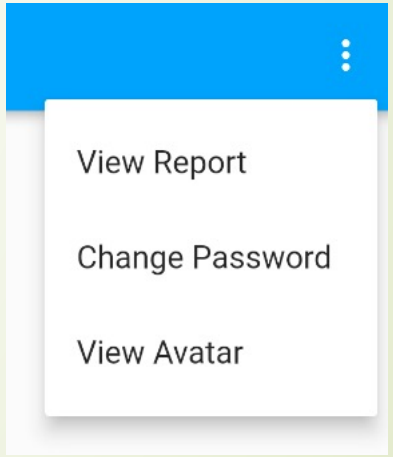
```
PopupMenuButton(
    onSelected: (value) {
        print('Selected: $value');
    },
    itemBuilder: (context) => [
        PopupMenuItem(
            onTap: () {},
            value: 'option 1',
            child: Text('Option 1'),
        ),
    ];
)
```

Popup Menu

- You can decide to show/not show a certain menu item based on a certain condition by using a conditional if expression.

```
PopupMenuButton(
    onSelected: (value) { print('Selected: $value'); },
    itemBuilder: (context) => [
        if (isAdmin)
            const PopupMenuItem(child: Text('View Report')),
        const PopupMenuItem(child: Text('Change Password')),
        const PopupMenuItem(child: Text('View Avatar')),
    ],
)
```

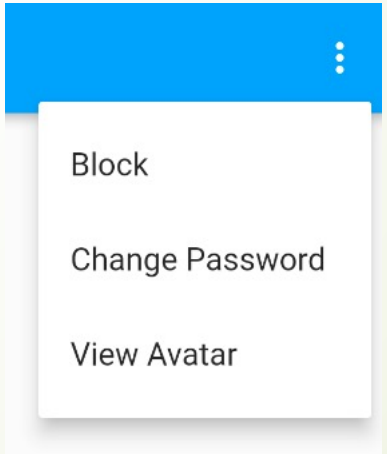


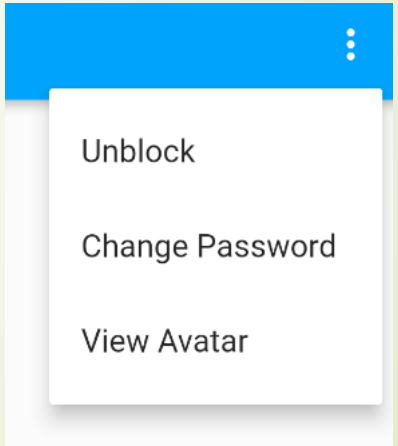


Popup Menu

- You can also display different text or icons for each specific case using the ternary operator directly inside the widgets.

```
PopupMenuItem(
    child: Text(isBlocked ? 'Unblock' : 'Block'),
),
```

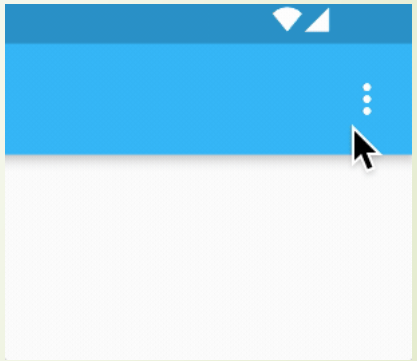




Popup Menu

- You can customize the appearance of the menu items by wrapping them in other widgets, such as ListTile, Card, or Container.
- You can also use the PopupMenuItemDivider widget to add a divider between the menu items.

```
itemBuilder: (context) => <PopupMenuEntry>[
    const PopupMenuItem(value: 'item1',
        child: ListTile(leading: Icon(Icons.delete),title: Text('Delete')),
    ),
    const PopupMenuDivider(), // <PopupMenuEntry<String>>
    const PopupMenuItem(value: 'item2',
        child: ListTile(leading: Icon(Icons.edit), title: Text('Edit')),
    ),
]
```

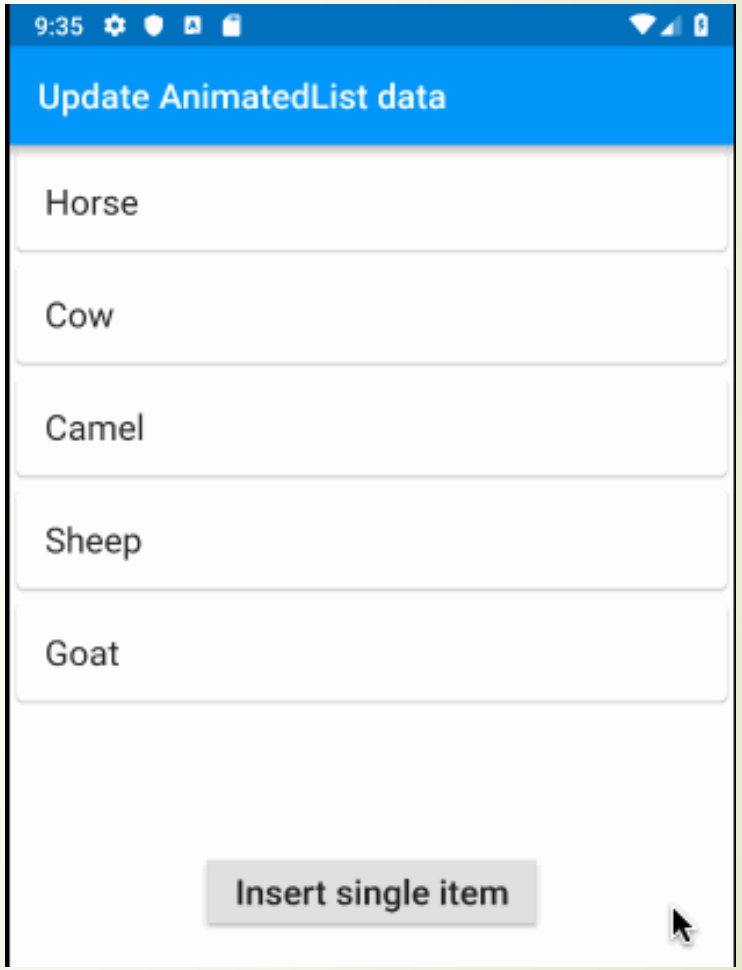


Advanced ListView

AnimatedList widget

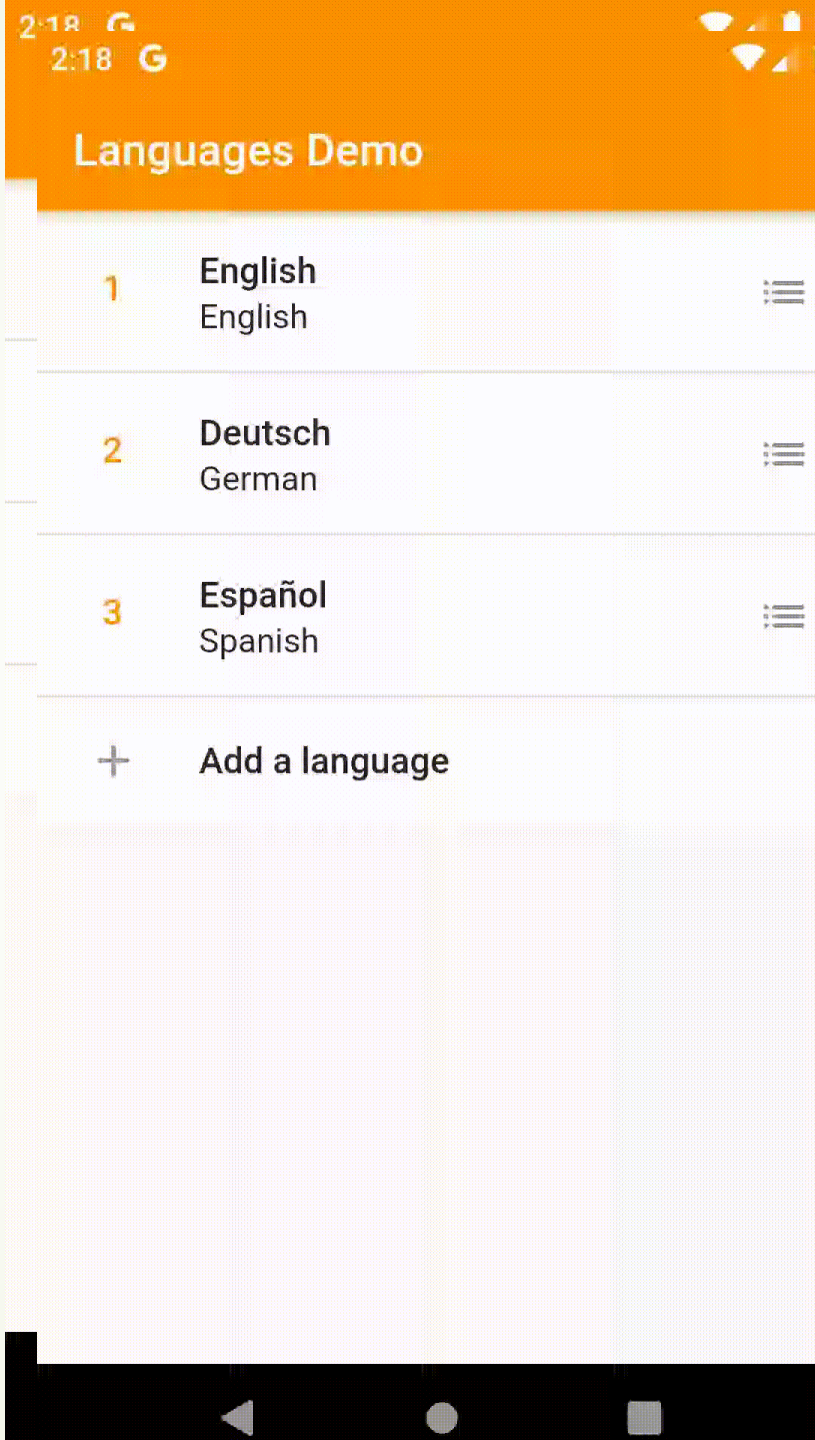
- AnimatedList widget is a specialized type of list view that provides built-in animations for adding, removing, and changing items. It allows you to create a list that animates the appearance and disappearance of items in response to changes in the list's contents.

- The AnimatedList widget maintains a list of items, each of which is represented by a widget. When you add, remove, or modify an item in the list, the AnimatedList widget animates the changes to give a smooth and polished look to the user interface.



ReorderableList widget

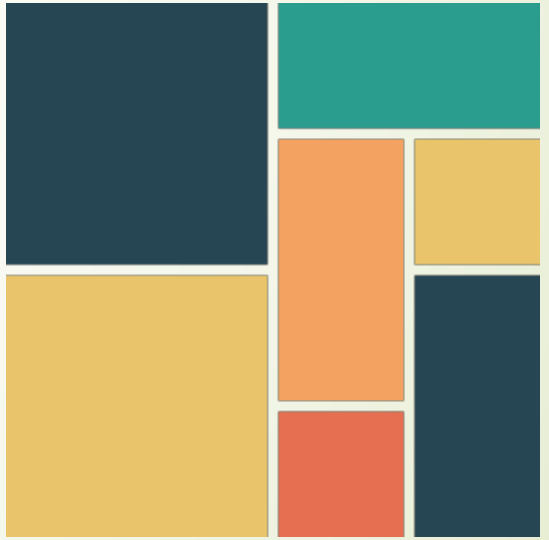
- ReorderableListView widget in Flutter is a powerful widget that allows users to reorder items in a list by dragging and dropping them. It can be used to create a wide range of user interfaces, including shopping lists, to-do lists, and many other types of list-based applications.



Helpful Packages

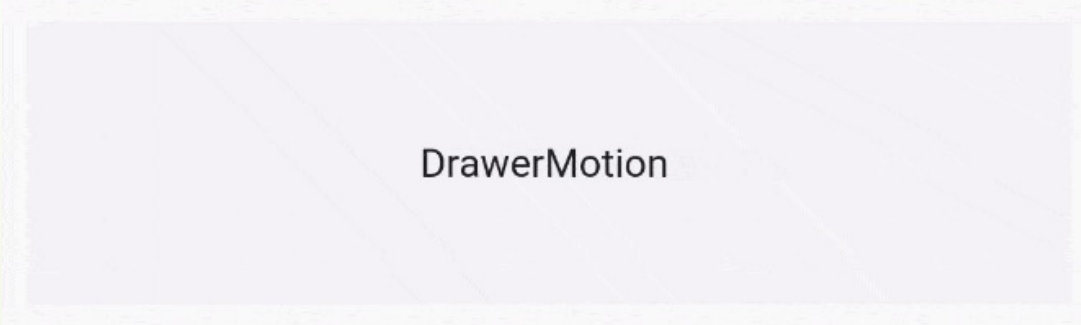
flutter_staggered_grid_view

- Flutter has a vibrant ecosystem of third-party packages that can be used to extend the functionality of its built-in widgets. Here are some popular packages that can be helpful when working with the ListView or GridView widgets:
  - **flutter_staggered_grid_view**: This package provides a StaggeredGridView widget that can be used to create a grid with items of varying sizes. It supports both vertical and horizontal scrolling, and can be customized with a variety of parameters.
  - https://pub.dev/packages/flutter_staggered_grid_view



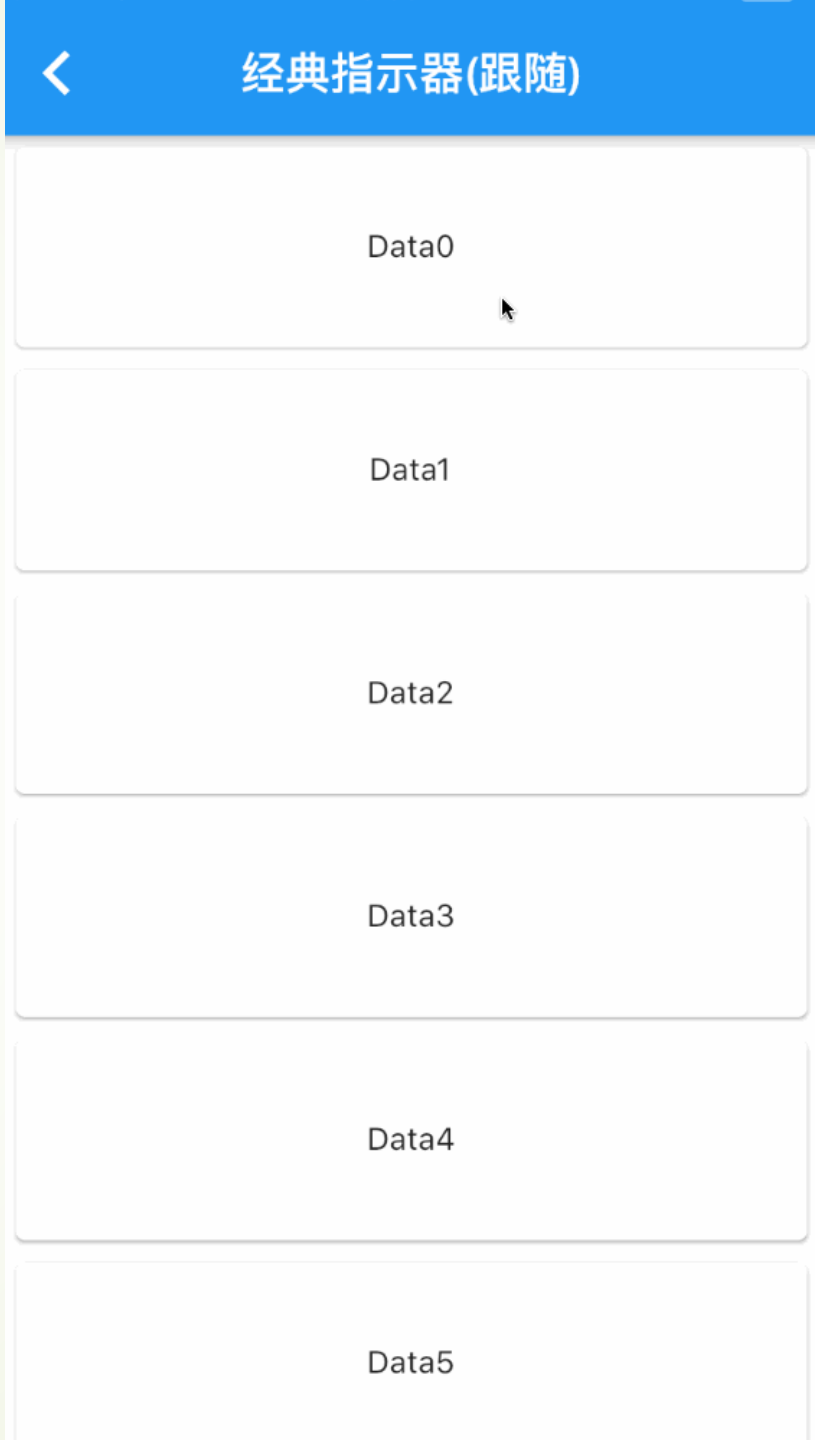
flutter_slidable

- **flutter_slidable**: This package provides a Slidable widget that can be used to create swipeable items in a ListView or GridView. It supports left and right swipes, and can be customized with various actions.
- https://pub.dev/packages/flutter_slidable



pull_to_refresh

- **pull_to_refresh:** This package provides a PullToRefresh widget that can be used to add pull-to-refresh functionality to a ListView or GridView. It supports various customization options, including loading and error states.
- https://pub.dev/packages/pull_to_refresh



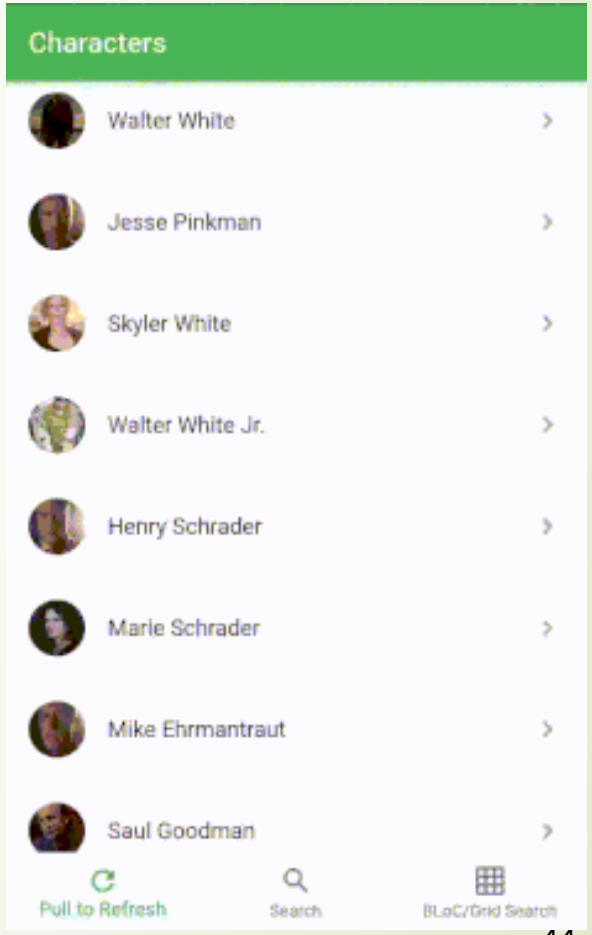
cached_network_image

- **cached_network_image:** This package provides a CachedNetworkImage widget that can be used to display images from a network source with caching capabilities. It can be used in a ListView or GridView to load images as the user scrolls, and can help reduce network requests and improve performance.
- https://pub.dev/packages/cached_network_image

```
CachedNetworkImage(
    imageUrl: "http://via.placeholder.com/350x150",
    placeholder: (context, url) =>
    CircularProgressIndicator(),
    errorWidget: (context, url, error) => Icon(Icons.error),
),
```

infinite_scroll_pagination

- **infinite_scroll_pagination:** This package provides a PagedListView widget that can be used to implement infinite scroll pagination in a ListView. It supports various customization options and can help reduce network requests and improve performance.
- https://pub.dev/packages/infinite_scroll_pagination



flutter_spinkit

- **flutter_spinkit:** This package provides a collection of loading indicators that can be used in a ListView or GridView. It includes various customizable styles, such as rotating circles, bouncing bars, and fading cubes.
- https://pub.dev/packages/flutter_spinkit