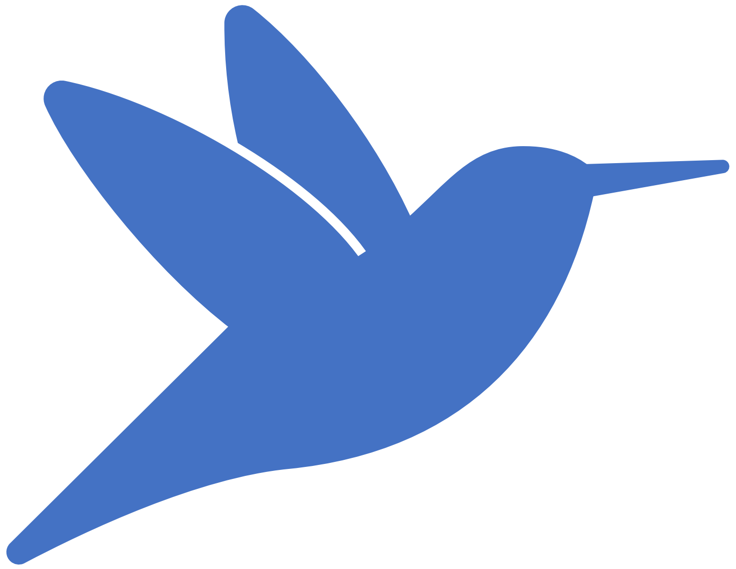
By Minase L.

Flutter Widgets Fundamentals



Scaffold Widget

The Scaffold widget is a fundamental building block for creating the basic structure of a Material Design application.

It provides a top-level container that holds the structure and common elements of an app's visual interface, such as an app bar, a floating action button, a drawer, and a bottom navigation bar.

The Scaffold widget simplifies the process of creating typical app layouts by offering a standardized structure.

Image widget

In Flutter, the Image widget is used to display images within your application.

It supports various types of images, including local assets, network images, and memory images.

The Image widget is quite versatile and can be customized to fit different layout and loading scenarios.

Container Widget

The Container widget in Flutter is a versatile and commonly used layout widget.

It allows you to create a box model that can contain other widgets, providing control over its dimensions, padding, margin, decoration, and more.

The Container widget is useful for creating simple or complex layouts within your Flutter application.

Column and Row Widgets

In Flutter, the Column and Row widgets are used to arrange child widgets vertically and horizontally, respectively.

They are both part of the Flex widget family and provide a flexible way to create complex layouts.

Icon Widget

The Icon widget is used to display icons within your application.

Flutter includes a set of built-in material icons that can be easily used with the Icon widget.

You can also use custom icons from different icon packs or create your own custom icons.

Card widget

In Flutter, the Card widget is used to create a material design card—a container with a shadow, rounded corners, and an elevation.

It's a convenient way to present information and actions in a consistent and visually appealing manner.

GridView Widget

- GridView is used for displaying a 2D array of widgets. It can be used for both scrolling and non-scrolling grids.

-
  ```
  GridView.count(
      crossAxisCount: 2,
      children:[
          Card(child: Text('Item 1')),
          Card(child: Text('Item 2')),
          Card(child: Text('Item 3')), ], );
  ```

Expanded widget

- The Expanded widget in Flutter is used to make a child widget take up the available space along the main axis within a Row or Column.
- It is particularly useful when you want a widget to expand and fill the remaining space within its parent.

Flexible widget

- The Flexible widget in Flutter is used to control the ratio of space that a child widget occupies within a Row, Column, or Flex widget.
- It provides more flexibility than the Expanded widget, allowing you to define a flex factor for each child widget, determining how space is distributed among them.

Stateless Widget

- A StatelessWidget is immutable, meaning that its properties cannot change once they are set.
- These widgets don't store any mutable state and are typically used for UI elements that don't change dynamically.
- Stateless widgets are simple and efficient.

Stateful Widget

- A StatefulWidget, on the other hand, can hold mutable state that can change over time.
- These widgets are used when you need to manage and update the state of a part of your UI dynamically.

Notice

- Understanding when to use StatelessWidget and StatefulWidget is crucial for effective Flutter development.
- Stateless widgets are suitable for static UI elements, while stateful widgets are used when you need to manage and update the state of a dynamic part of your UI.

Navigator widget

- Navigator is a widget that manages a stack of pages or routes in an app.
- It allows you to navigate between different screens or pages and maintain a history of the navigation stack.
- The Navigator widget is part of the Flutter framework and is commonly used for implementing navigation in mobile applications.
- Here are some key concepts and examples related to using Navigator in Flutter: -

Basic Navigation

- The basic navigation in Flutter involves pushing and popping routes onto and off the navigation stack. You can use the Navigator to navigate between screens using the push and pop methods.
- ElevatedButton(

```
        onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(builder: (context) => SecondScreen()),);},
          child: Text('Go to Second Screen'),)
```

Named Routes

- Using named routes is a cleaner and more maintainable way to navigate between screens.
- Define named routes in the app, and then use the Navigator to navigate to those routes.

Example

-
  ```
  MaterialApp(
      routes: {
        '/': (context) => HomeScreen(),
        '/second': (context) => SecondScreen(),
      },
    )
  ```

-
  ```
  ElevatedButton(
      onPressed: () {
        Navigator.pushNamed(context, '/second');
      },
      child: Text('Go to Second Screen'),
    )
  ```

Passing Data

- You can pass data between screens using the Navigator by providing arguments to the push method.
- ElevatedButton(

```
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => SecondScreen(data:  'Hello from First Screen'),
        ),
      );
    },
    child: Text('Go to Second Screen'),
  )
```

Cont.

- In the SecondScreen, you can access the passed data:

-
  ```
  class SecondScreen extends StatelessWidget {
      final String data;
      SecondScreen({required this.data});
      @override
      Widget build(BuildContext context) {
        return Scaffold(
          appBar: AppBar(title: Text('Second Screen')),
          body: Center(child: Text(data)),
        );
      }
    }
  ```

Pop and Navigate Back

- To navigate back, you can use the pop method. This removes the current route from the stack and goes back to the previous route.

-
  ```
  ElevatedButton(
    onPressed: () {
      Navigator.pop(context);
    },
    child: Text('Go Back'),
  )
  ```