Layouts and Forms

Tim Dosen PBP

Show me examples

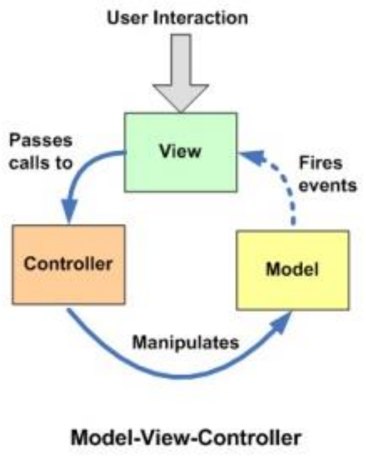
https://flutter.github.io/samples/

Review

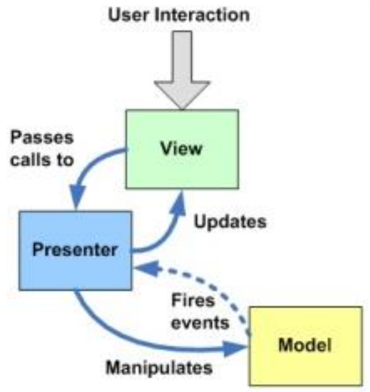
What do you know about Widget?

Widgets (Stateless & Stateful)

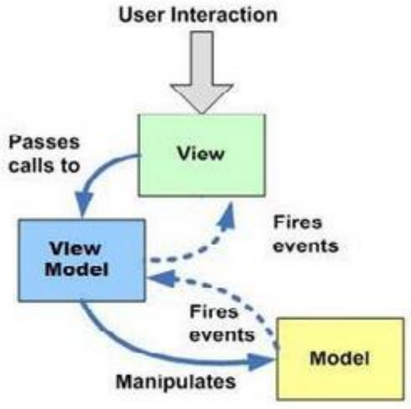
MVC vs MVP vs MVVM Architecture



Model-View-Controller



Model-View-Presenter



Model - View - ViewModel

- In MVC, every action in the View correlates with a call to a Controller along with an action.
- The controller is responsible for handling the User Input and then updating the Model or the View.

- When the view notifies the presenter that the user has done something (for example, clicked a button), the presenter will then update the model and synchronize any changes between the Model and the View.
- One important thing to mention is that the Presenter doesn't communicate directly to the view. Instead, it communicates through an interface. This way, the presenter and the model can be tested in isolation.

- The term ViewModel means "Model of a View", and can be thought of as abstraction of the view, but it also provides a specialization of the Model that the View can use for data-binding.

What is Widget?

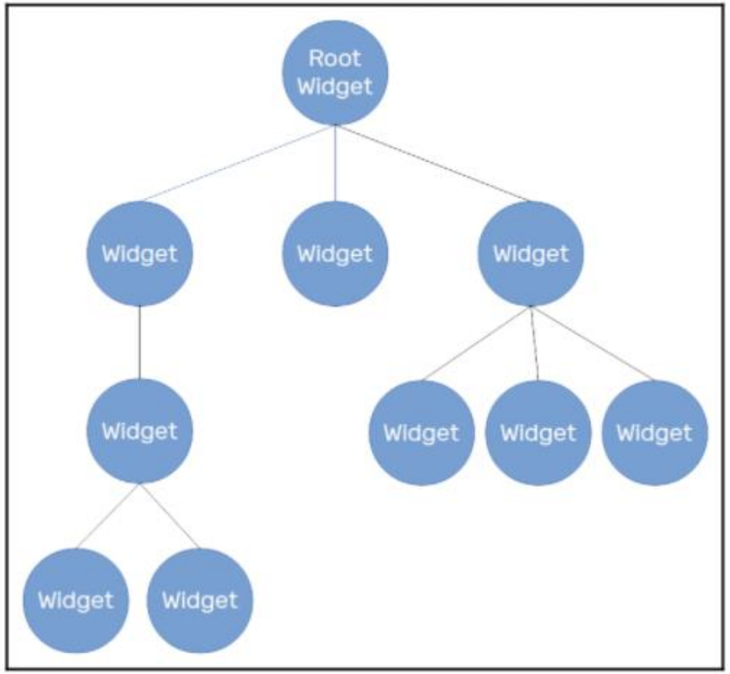
- It’s a reusable component
- Remember MTV?
- Flutter is component based => MVVM => Model, View, View-Model
- Widget is a component.

    - **Widgets** can be understood as the visual representation of parts of the application. And, also handle its own the behaviour.
    - Widgets are put together to compose the UI of an application.
    - Imagine it as a puzzle in which you define the pieces.

Everything is Widget!

- A visual/structural element that is a basic structural element, such as the Button or Text widgets
- A layout specific element that may define the position, margins, or padding, such as the Padding widget
- A style element that may help to colorize and theme a visual/structural element, such as the Theme widget
- An interaction element that helps to respond to interactions in different ways, such as the GestureDetector widget

Widget Tree



The widget tree is the logical representation of all the UIs widgets. It is computed during layout (measurements and structural info) and used during rendering (frame to screen) and hit testing (touch interactions), and this is the things Flutter does best.

Widgets are represented in the tree as nodes. It may have a state associated with it; every change to its state results in rebuilding the widget and the child involved.

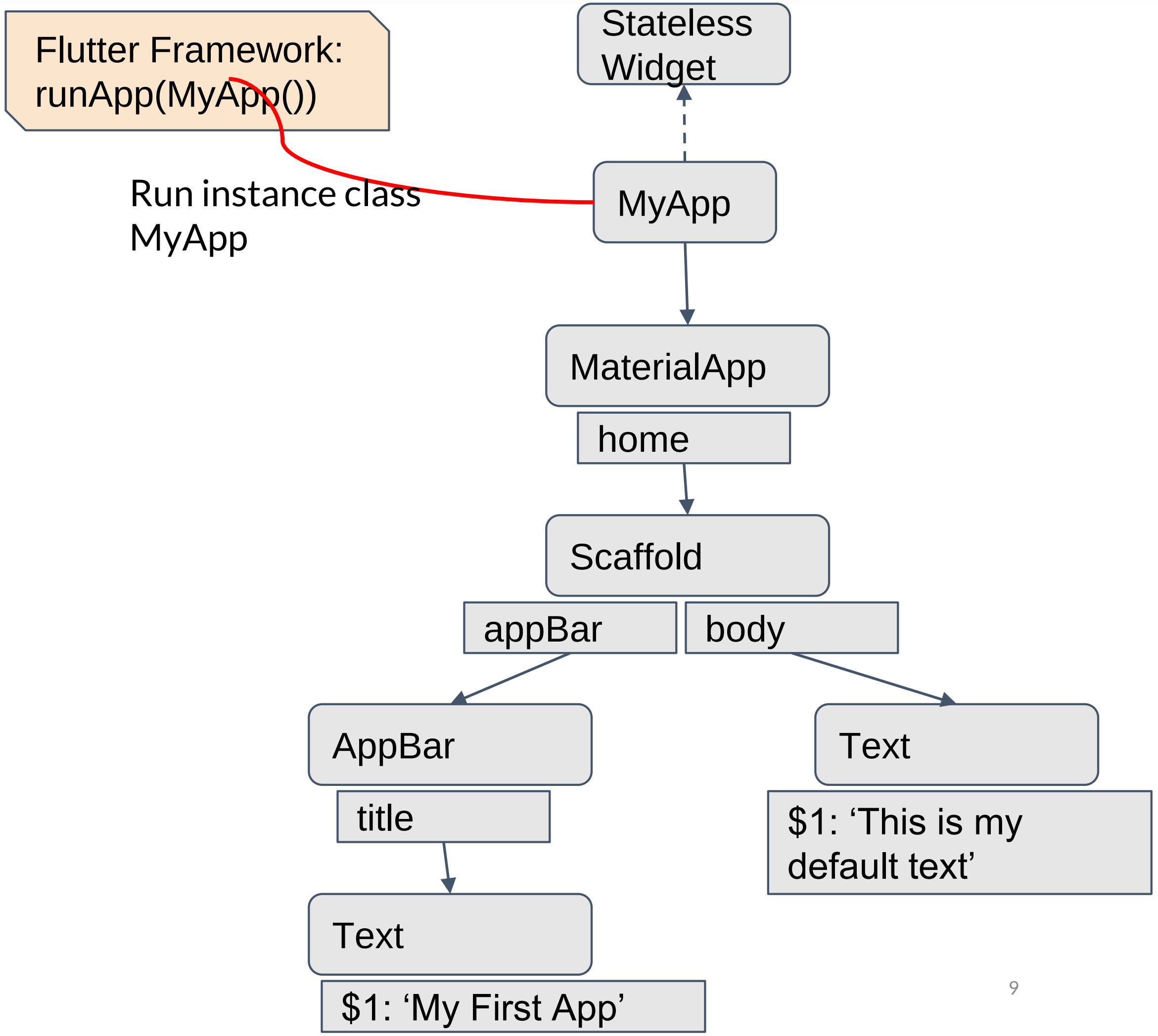
Widget Tree

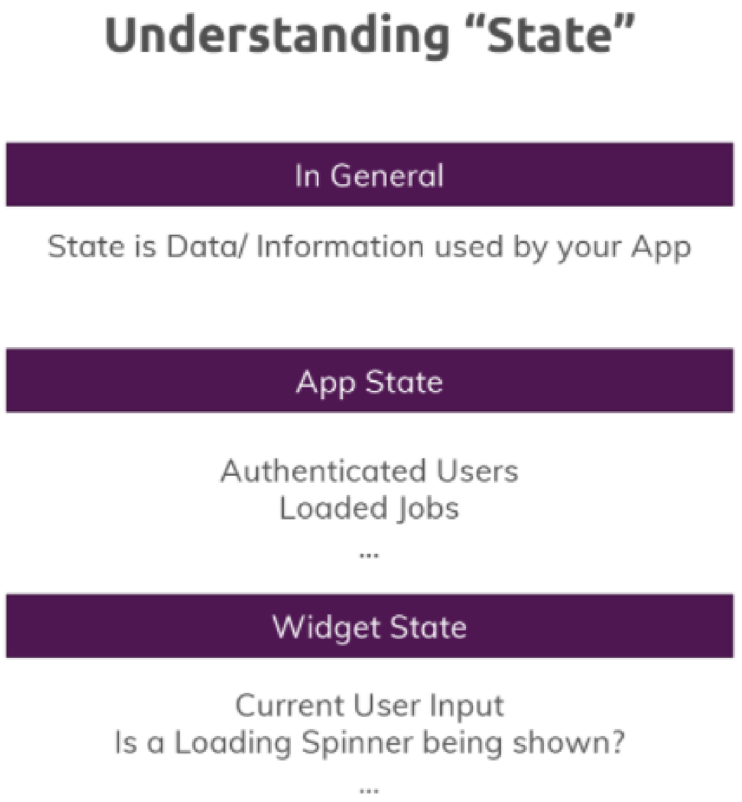
```
import 'package:flutter/material.dart';

// void main() {
//   runApp(MyApp());
// }

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('My First App'),
        ),
        body: Text('This is my default text!'),
      ),
    );
  }
}
```





Understanding "State"

In General

State is Data/ Information used by your App

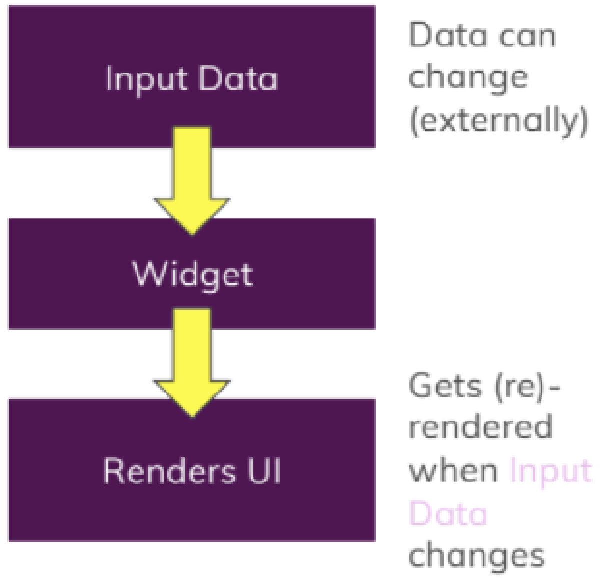
App State

Authenticated Users
Loaded Jobs
...

Widget State

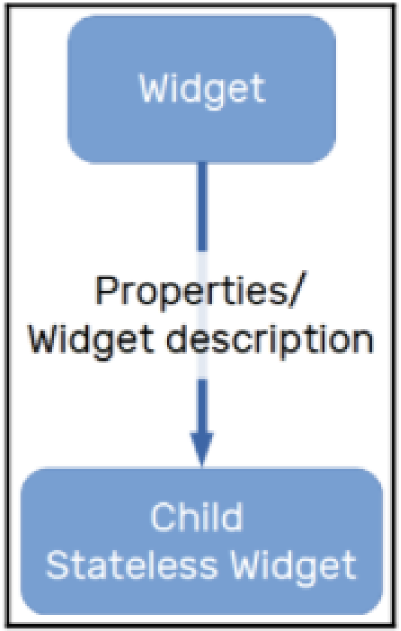
Current User Input
Is a Loading Spinner being shown?
...

Stateless Widget


Input Data
Widget
Renders UI
Data can change (externally)
Gets (re)-rendered when Input Data changes

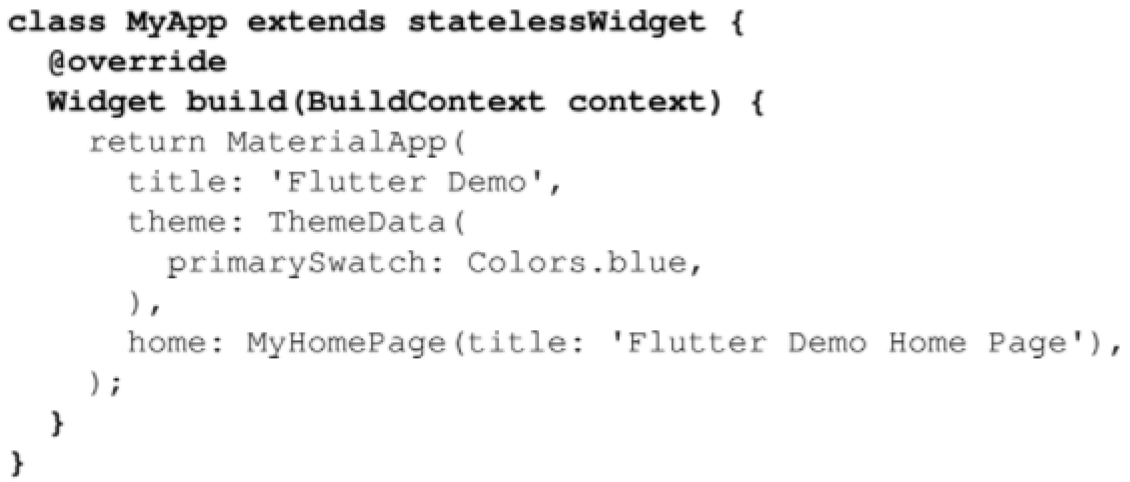
- They do not have a state; they do not change by themselves through some internal action or behavior
- They are changed by external events on parent widgets in the widgets tree.
- Give control of how they are built to some parent widget in the tree.

Stateless Widget



- The child widget will receive its description from the parent widget and will not change it by itself.
- Stateless widgets have only final properties defined during construction, and that's the only thing that needs to be built on the device screen.

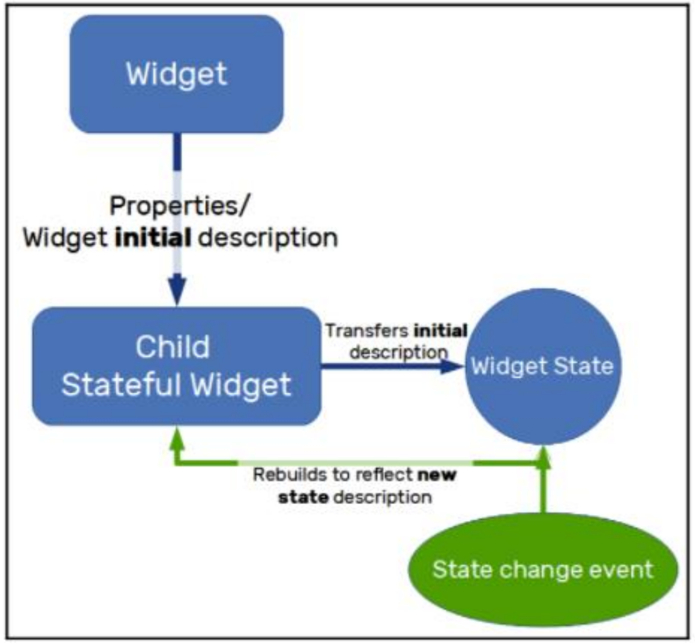
In Code

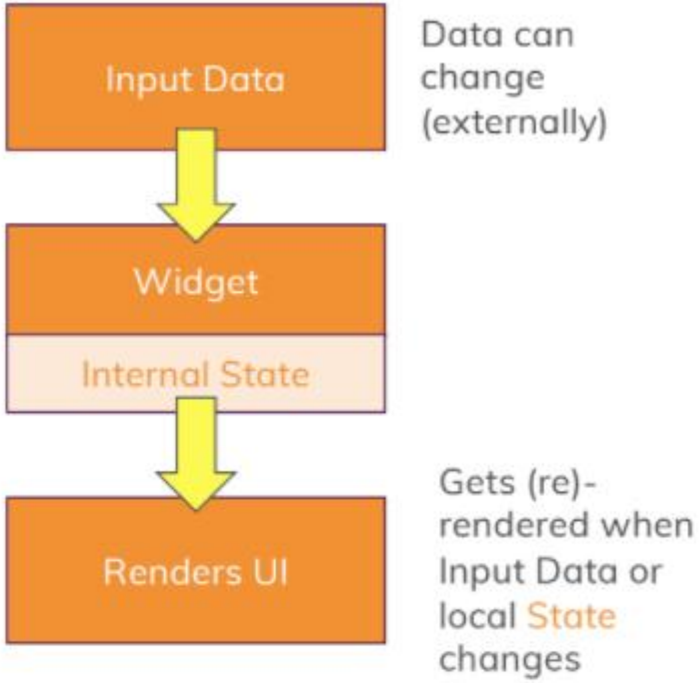


```
class MyApp extends statelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

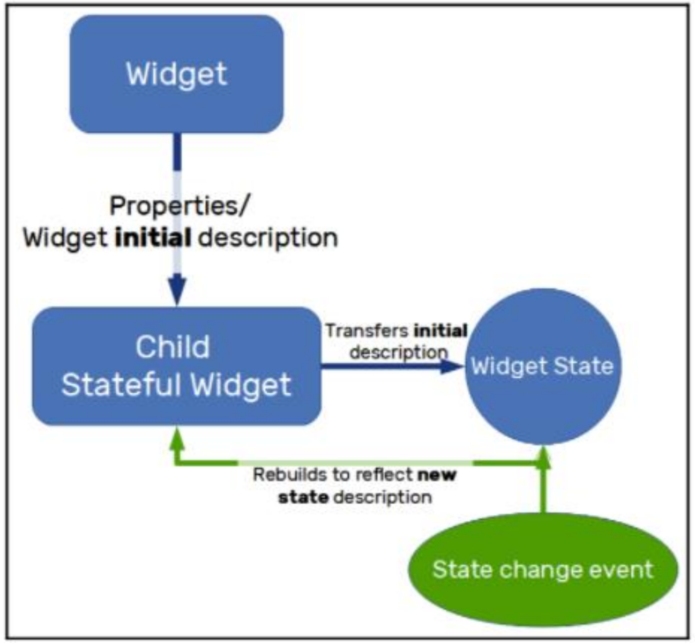
Another example: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

Stateful Widget


Widget
Properties/
Widget **initial** description
Child
Stateful Widget
Transfers **initial** description
Widget State
Rebuilds to reflect **new state** description
State change event

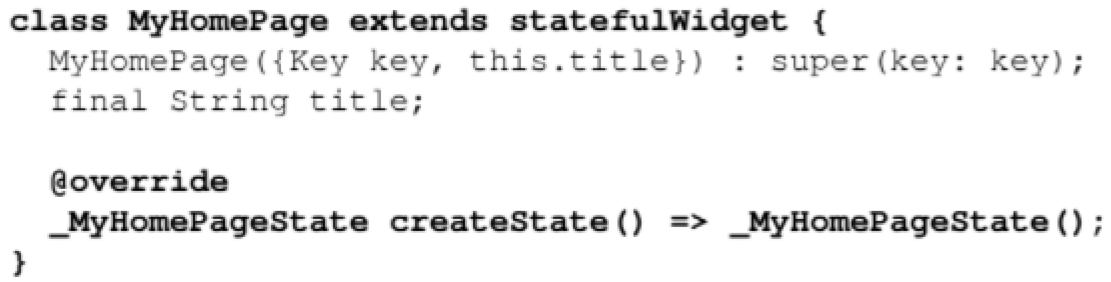

Input Data
Data can change (externally)
Widget
Internal State
Renders UI
Gets (re)-rendered when Input Data or local State changes

Stateful Widget


Widget
Properties/
Widget **initial** description
Child
Stateful Widget
Transfers **initial** description
Widget State
Rebuilds to reflect **new state** description
State change event

- Change their descriptions dynamically during their lifetimes.
- Immutable, but they have a company State class that represents the current state of the widget.

In Code



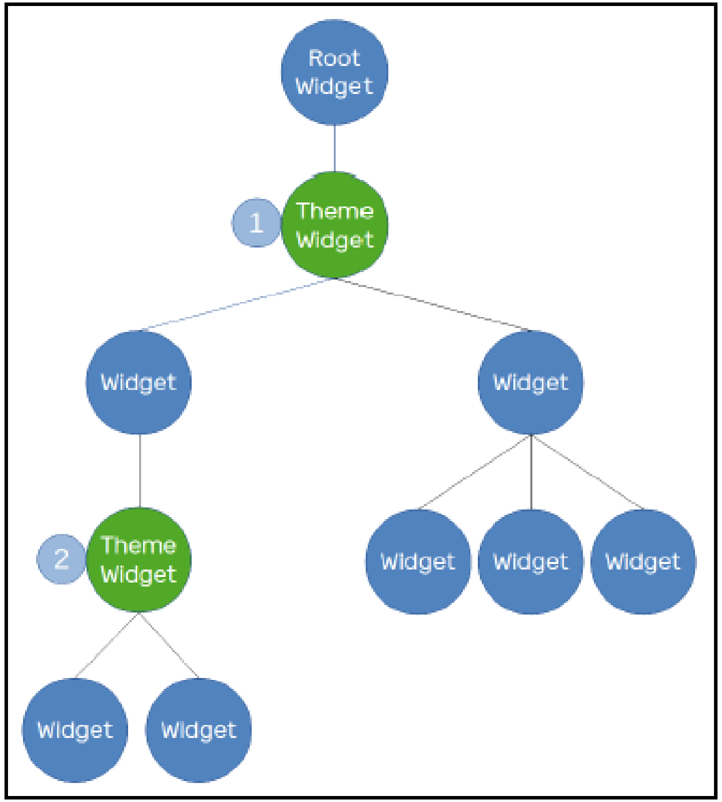
```
class MyHomePage extends statefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

By extending statefulWidget, MyHomePage must return a valid State object in its createState() method. In our example, it returns an instance of _MyHomePageState.

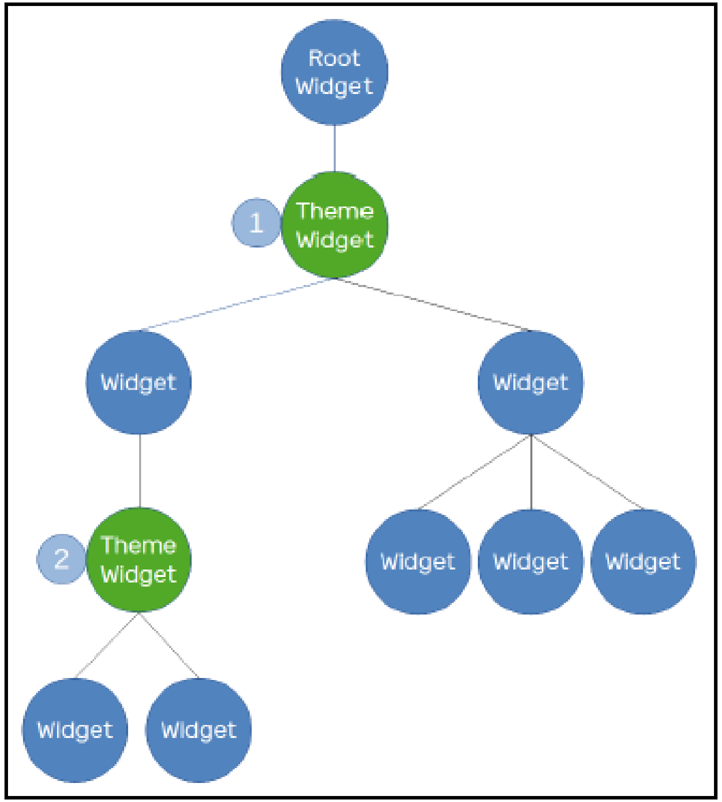
Another example: https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html

Inherited Widget



- Besides statelessWidget and statefulWidget, there is one more type of widget in the Flutter framework, InheritedWidget.
- Sometimes, one widget may need to have access to data up the tree, and in such a case, we would need to replicate the information down to the interested widget. To address this problem, Flutter provides the InheritedWidget class, an auxiliary kind of widget that helps to propagate information down the tree.
- There are some very common appearances of the usage of InheritedWidget in Flutter. One of the most common uses is from the Theme class, which helps to describe colors for a whole application.

Inherited Widget



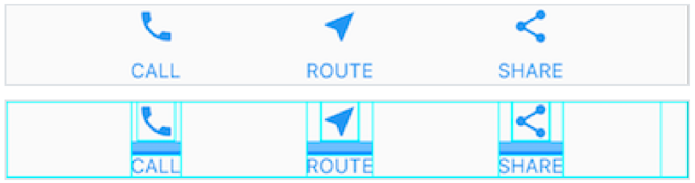
- The Theme widget also works with the InheritedWidget technique, so every descending widget can access it by using Theme.of(context), which internally makes a call to the helper inheritFromWidgetOfExactType method from the BuildContext class.
- The theme data is applied to descending widgets but can be overridden in local parts of the widget tree. In the preceding diagram, the theme with the number 2 will override the theme with the number 1 defined at the very beginning of the tree. The number 2 subtree will have different theme from the rest of the tree.
- https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html

Layouts

Layouts in Flutter

- The core of Flutter's layout mechanism is "widgets."
- The idea is composing widgets to build more complex widgets

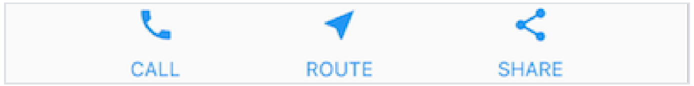
Consider the ROWS and COLUMNS

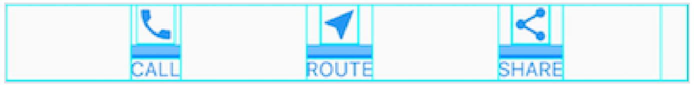


https://docs.flutter.dev/development/ui/layout

Layouts in Flutter

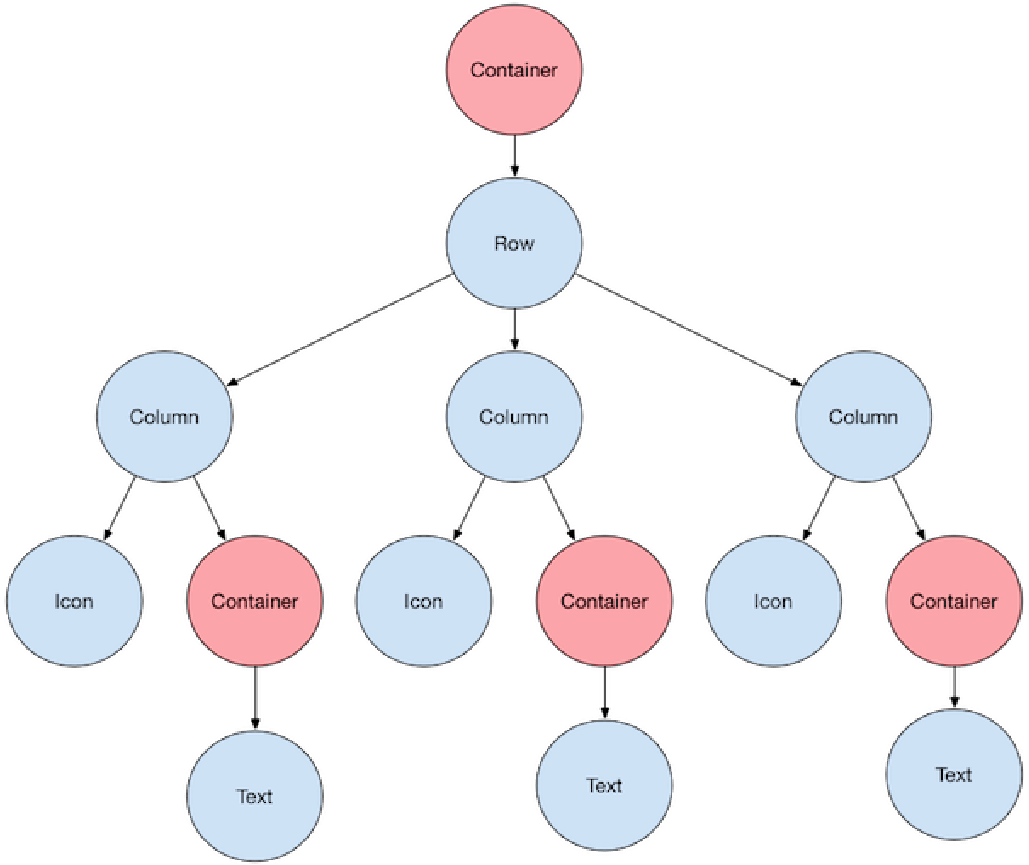
- The most common containers in Flutter are the Row and Column widgets.
- The first screenshot shows 3 icons with a label under each one.





- The second screenshot displays the visual layout, showing a row of 3 columns where each column contains an icon and a label.

Widget Tree Structure



Container

- Container is a widget class that allows you to customize its child widget.
- wrapping another widgets -> parent widget
- combines common painting, positioning, and sizing of the child widgets.
- has some properties which include:
  - padding
  - margin
  - borders
  - color
- https://api.flutter.dev/flutter/widgets/Container-class.html

Constraints

- The Flutter layout rule:

  Constraints go down. Sizes go up. Parent sets position.

  - A widget gets its own **constraints** from its **parent**. A *constraint* is just a set of 4 doubles: a minimum and maximum width, and a minimum and maximum height.
  - Then the widget goes through its own list of **children**. One by one, the widget tells its children what their **constraints** are (which can be different for each child), and then asks each child what size it wants to be.
  - Then, the widget positions its **children** (horizontally in the x axis, and vertically in the y axis), one by one.
  - And, finally, the widget tells its parent about its own **size** (within the original constraints, of course).

- https://docs.flutter.dev/development/ui/layout/constraints

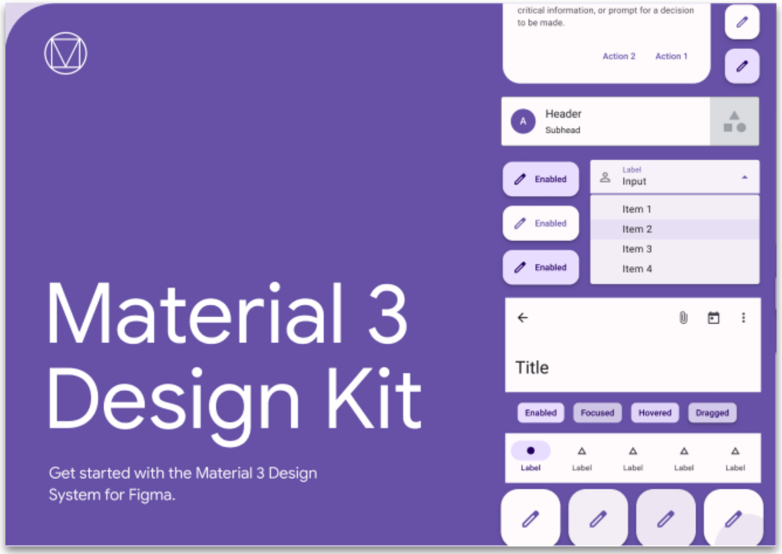
Limitations

- Flutter’s layout engine has a few important limitations:
    - A widget can decide its own size only within the constraints given to it by its parent. This means a widget usually **can’t have any size it wants**.
    - A widget **can’t know and doesn’t decide its own position in the screen**, since it’s the widget’s parent who decides the position of the widget.
    - Since the parent’s size and position, in its turn, also depends on its own parent, it’s impossible to precisely define the size and position of any widget without taking into consideration the tree as a whole.
    - If a child wants a different size from its parent and the parent doesn’t have enough information to align it, then the child’s size might be ignored. **Be specific when defining alignment.**
- https://docs.flutter.dev/development/ui/layout/constraints

Choosing your layout widgets...

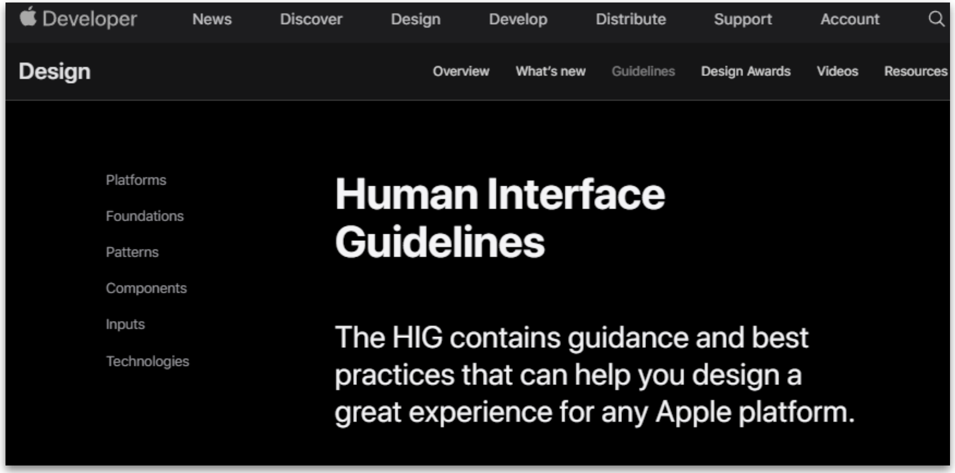
Material & Non-Material Apps

- What is the difference between material apps and non-material apps?
- https://docs.flutter.dev/development/ui/layout#material-apps



Material 3 Design Kit

Get started with the Material 3 Design System for Figma.



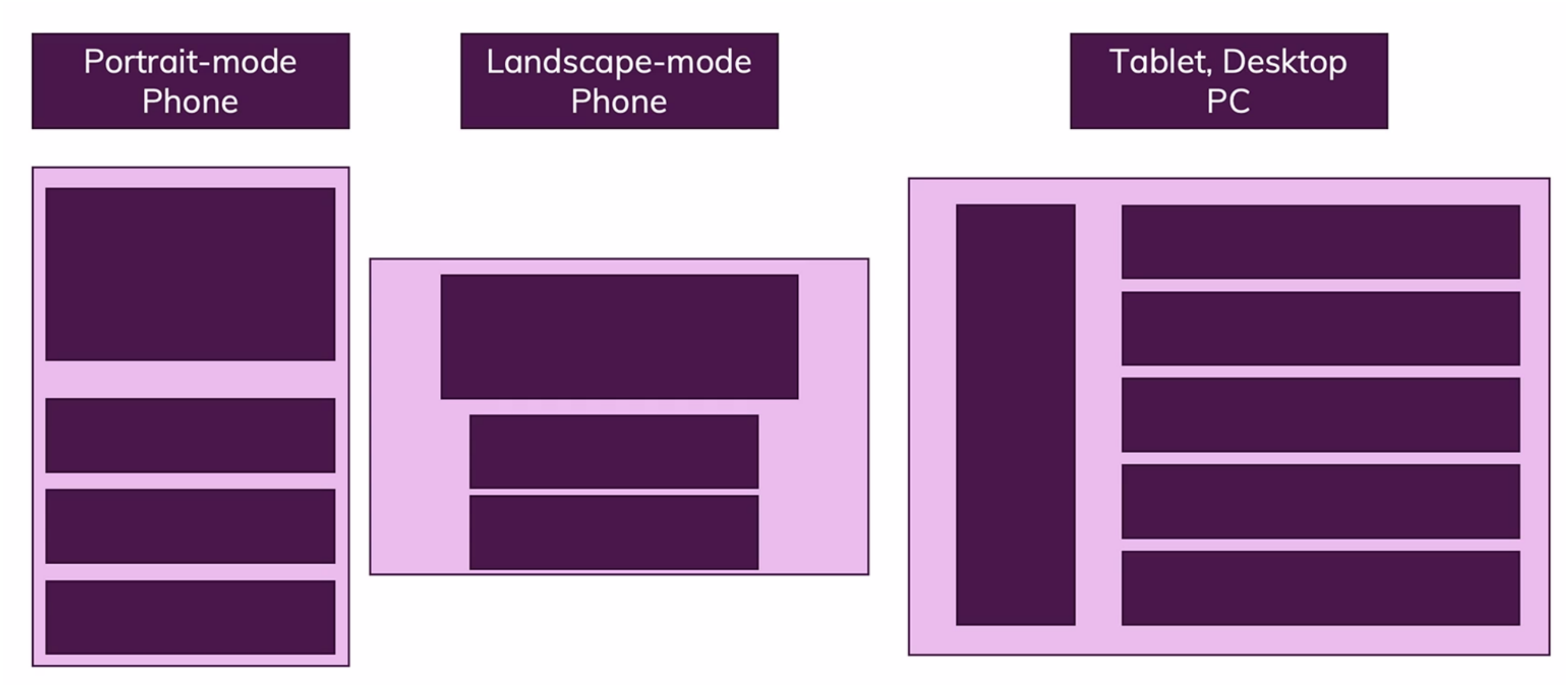
Human Interface Guidelines

The HIG contains guidance and best practices that can help you design a great experience for any Apple platform.

Adaptive & Responsive Apps

- What is the difference between an adaptive and a responsive app?
- https://docs.flutter.dev/development/ui/layout/adaptive-responsive

Responsive App



Adaptive App

Android	iOS
Material-Design Look / Style	Cupertino Look / Style
Android Animations / Route Transitions	iOS Animations / Route Transitions
Android Fonts	iOS Fonts

Responsive Flutter Apps

- There are two basic approaches to creating Flutter apps with responsive design:
  1. Use the `LayoutBuilder` class
  2. Use the `MediaQuery.of()` method in your build functions

Adaptive Flutter Apps

- There are many considerations for developing platform-adaptive apps, but they fall into three major categories:
  1. Layout
     - how to adapt to the various sizes and shapes of the screens that your app will run on
  2. Input
     - how to support varying user inputs
  3. Idioms and norms
     - each platform has its own idioms and norms; these nominal or de facto standards inform user expectations of how an application should behave

Animations

Animation

- The primary building block of the animation system:
  Animation Class

- Types of animation:
  - ❖ tween animation -> the beginning and ending points are defined
  - ❖ physics based animation -> motion is modeled to resemble real-world behavior.

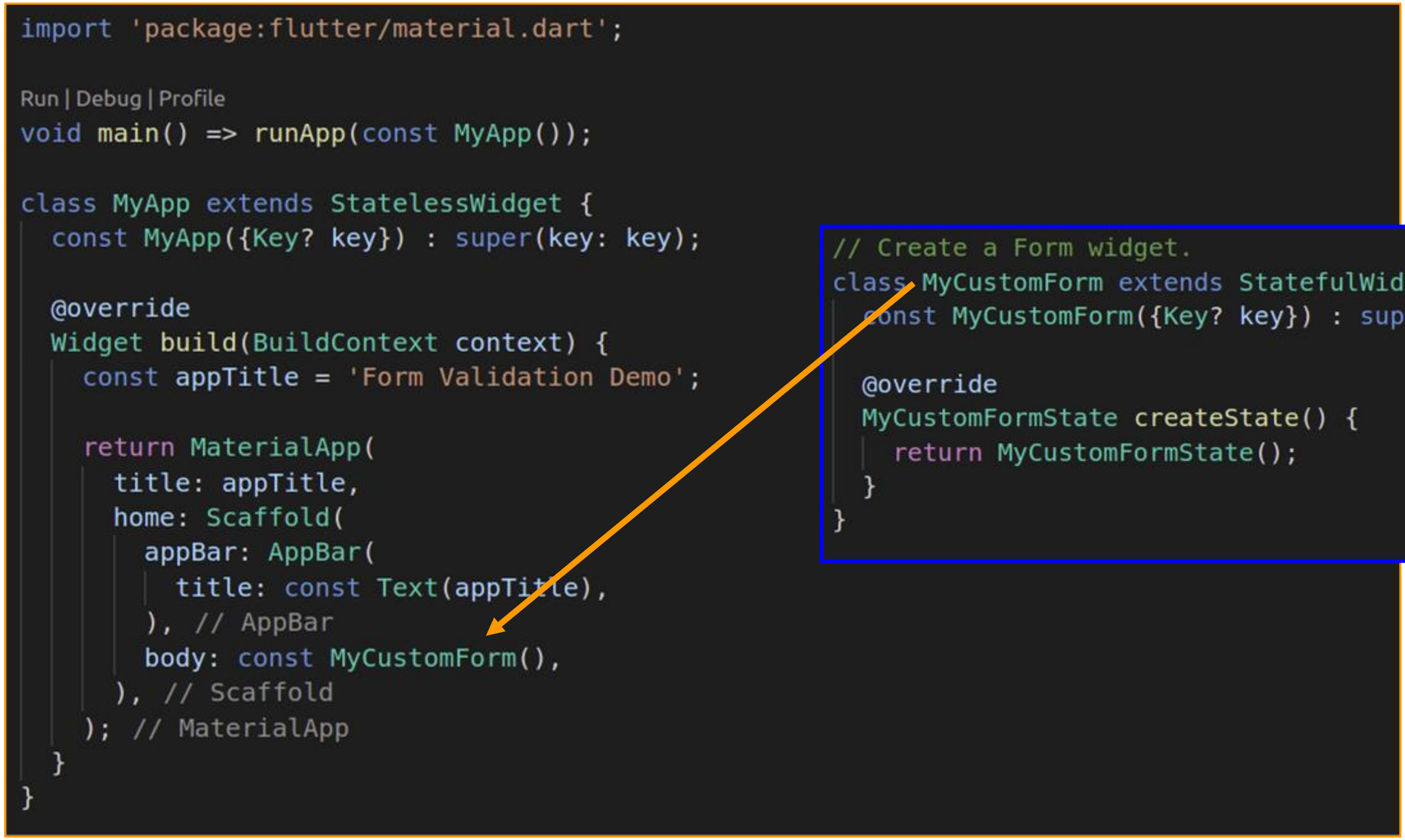
Check this out!
https://flutter.github.io/samples/web/animations/

Animation

- **Animation**, a core class in Flutter’s animation library, interpolates the values used to guide an animation.
- An **Animation object** knows the current state of an animation (for example, whether it’s started, stopped, or moving forward or in reverse), but doesn’t know anything about what appears onscreen.
- An **AnimationController** manages the Animation.
- A **CurvedAnimation** defines progression as a non-linear curve.
- A **Tween** interpolates between the range of data as used by the object being animated. For example, a Tween might define an interpolation from red to blue, or from 0 to 255.
- Use **Listeners** and **StatusListeners** to monitor animation state changes.

Forms, Input, and Events

Example Form



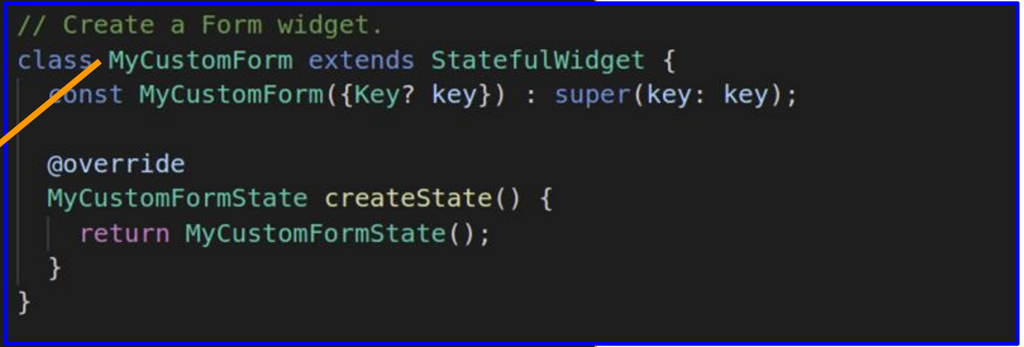
```dart
import 'package:flutter/material.dart';

Run | Debug | Profile
void main() => runApp(const MyApp());

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

  @override
  Widget build(BuildContext context) {
    const appTitle = 'Form Validation Demo';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ), // AppBar
        body: const MyCustomForm(),
      ), // Scaffold
    ); // MaterialApp
  }
}
```

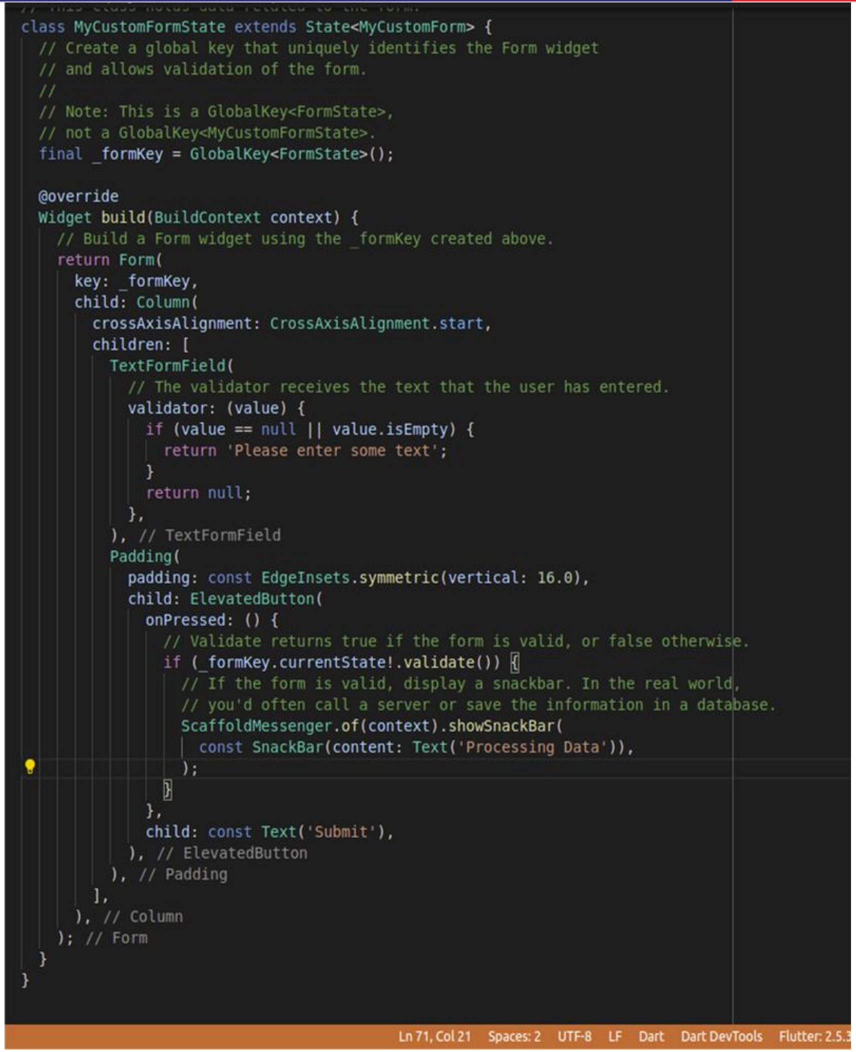


```dart
// Create a Form widget.
class MyCustomForm extends StatefulWidget {
  const MyCustomForm({Key? key}) : super(key: key);

  @override
  MyCustomFormState createState() {
    return MyCustomFormState();
  }
}
```

MyCustomFormState

Form
- key
- child
  - Column widget
    - crossAxisAlignment
    - children
      - TextFormField with validator
      - Padding
        - padding:
        - child: ElevatedButton



```dart
class MyCustomFormState extends State<MyCustomForm> {
  // Create a global key that uniquely identifies the Form widget
  // and allows validation of the form.
  //
  // Note: This is a GlobalKey<FormState>,
  // not a GlobalKey<MyCustomFormState>.
  final _formKey = GlobalKey<FormState>();

  @override
  Widget build(BuildContext context) {
    // Build a Form widget using the _formKey created above.
    return Form(
      key: _formKey,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          TextFormField(
            // The validator receives the text that the user has entered.
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter some text';
              }
              return null;
            },
          ), // TextFormField
          Padding(
            padding: const EdgeInsets.symmetric(vertical: 16.0),
            child: ElevatedButton(
              onPressed: () {
                // Validate returns true if the form is valid, or false otherwise.
                if (_formKey.currentState!.validate()) {
                  // If the form is valid, display a snackbar. In the real world,
                  // you'd often call a server or save the information in a database.
                  ScaffoldMessenger.of(context).showSnackBar(
                    const SnackBar(content: Text('Processing Data')),
                  );
                }
              },
              child: const Text('Submit'),
            ), // ElevatedButton
          ), // Padding
        ],
      ), // Column
    ); // Form
  }
}
```

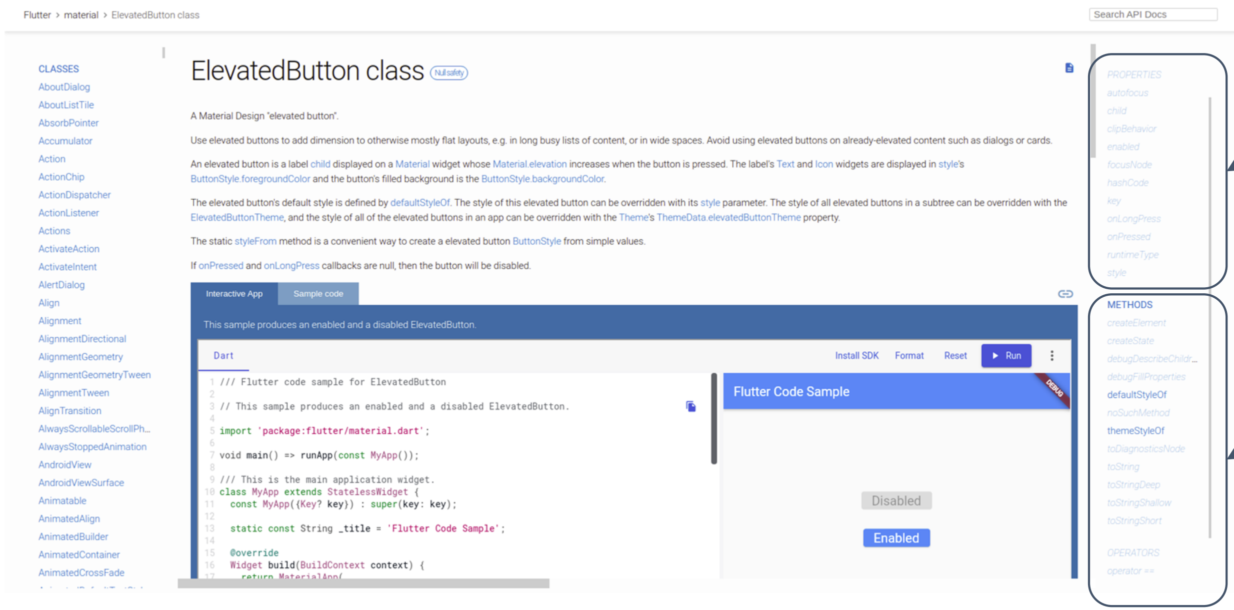
Forms & Input Validation

- Basically, it’s the same as forms in HTML with keys and input elements
- Widget validation attribute can be used to check input on executed

https://docs.flutter.dev/cookbook#forms
https://docs.flutter.dev/cookbook/forms/validation

Event onXXXXX

- Event handling using widget properties “onPressed” or “onXXXXX”
- Example: ElevatedButton
- https://api.flutter.dev/flutter/material/ElevatedButton-class.html



Flutter > material > ElevatedButton class

Search API Docs

CLASSES
AboutDialog
AboutListTile
AbsorbPointer
Accumulator
Action
ActionChip
ActionDispatcher
ActionListener
Actions
ActivateAction
ActivateIntent
AlertDialog
Align
Alignment
AlignmentDirectional
AlignmentGeometry
AlignmentGeometryTween
AlignmentTween
AlignTransition
AlwaysScrollableScrollPh...
AlwaysStoppedAnimation
AndroidView
AndroidViewSurface
Animatable
AnimatedAlign
AnimatedBuilder
AnimatedContainer
AnimatedCrossFade

ElevatedButton class (Null safety)

A Material Design "elevated button".

Use elevated buttons to add dimension to otherwise mostly flat layouts, e.g. in long busy lists of content, or in wide spaces. Avoid using elevated buttons on already-elevated content such as dialogs or cards.

An elevated button is a label child displayed on a Material widget whose Material.elevation increases when the button is pressed. The label's Text and Icon widgets are displayed in style's ButtonStyle.foregroundColor and the button's filled background is the ButtonStyle.backgroundColor.

The elevated button's default style is defined by defaultStyleOf. The style of this elevated button can be overridden with its style parameter. The style of all elevated buttons in a subtree can be overridden with the ElevatedButtonTheme, and the style of all of the elevated buttons in an app can be overridden with the Theme's ThemeData.elevatedButtonTheme property.

The static styleFrom method is a convenient way to create a elevated button ButtonStyle from simple values.

If onPressed and onLongPress callbacks are null, then the button will be disabled.

Interactive App | Sample code

This sample produces an enabled and a disabled ElevatedButton.

Dart | Install SDK | Format | Reset | Run

```dart
/// Flutter code sample for ElevatedButton

// This sample produces an enabled and a disabled ElevatedButton.

import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

/// This is the main application widget.
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  static const String _title = 'Flutter Code Sample';

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

PROPERTIES
autofocus
child
clipBehavior
enabled
focusNode
hashCode
key
onLongPress
onPressed
runtimeType
style

METHODS
createElement
createState
debugDescribeChildr...
debugFillProperties
defaultStyleOf
noSuchMethod
themeStyleOf
toDiagnosticsNode
toString
toStringDeep
toStringShallow
toStringShort

OPERATORS
operator ==

Available Properties

Available Method

References

- Flutter documentation:
  - https://docs.flutter.dev/development/ui/layout
  - https://docs.flutter.dev/development/ui/layout#material-apps
  - https://docs.flutter.dev/development/ui/layout/adaptive-responsive
  - https://docs.flutter.dev/cookbook#forms
  - https://docs.flutter.dev/cookbook/forms/validation
  - https://flutter.dev/docs/development/ui/layout/building-adaptive-apps
  - https://flutter.dev/docs/development/data-and-backend/networking
  - https://flutter.dev/docs/development/data-and-backend/json
- Dart API documentation:
  - https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html
  - https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html
  - https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html
  - https://api.flutter.dev/flutter/widgets/Container-class.html
  - https://api.flutter.dev/flutter/material/ElevatedButton-class.html
  - https://api.flutter.dev/flutter/material/MaterialApp-class.html
- https://flutter.github.io/samples/web/animations/