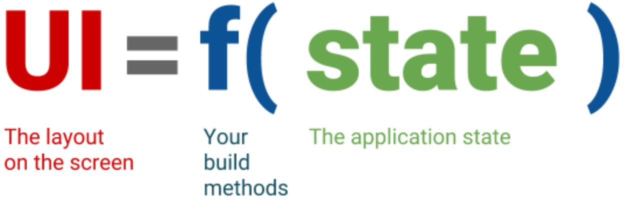
LUT
University

Stateful widgets

UI = build(state)



Flutter is declarative: The framework builds the user interface to reflect the current state of the app. (versus being built by function calls)

Flutter is fast. Rebuild instead of modify.

You cannot modify widgets. You can only edit state and trigger rebuilds.

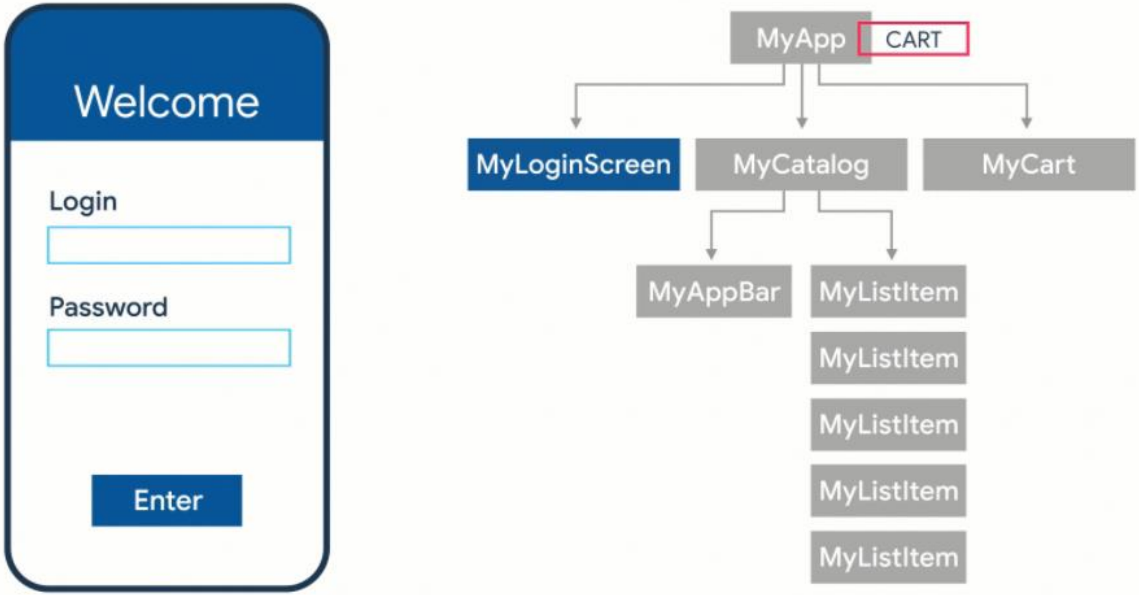
It might feel unintuitive at first. However, the major advantage is that there is only one code path for building the UI.

Overview

User interacts with widgets (listeners).

Widgets update embedded state and state above the widget.

Stateful widgets trigger rebuilds of widgets below them.



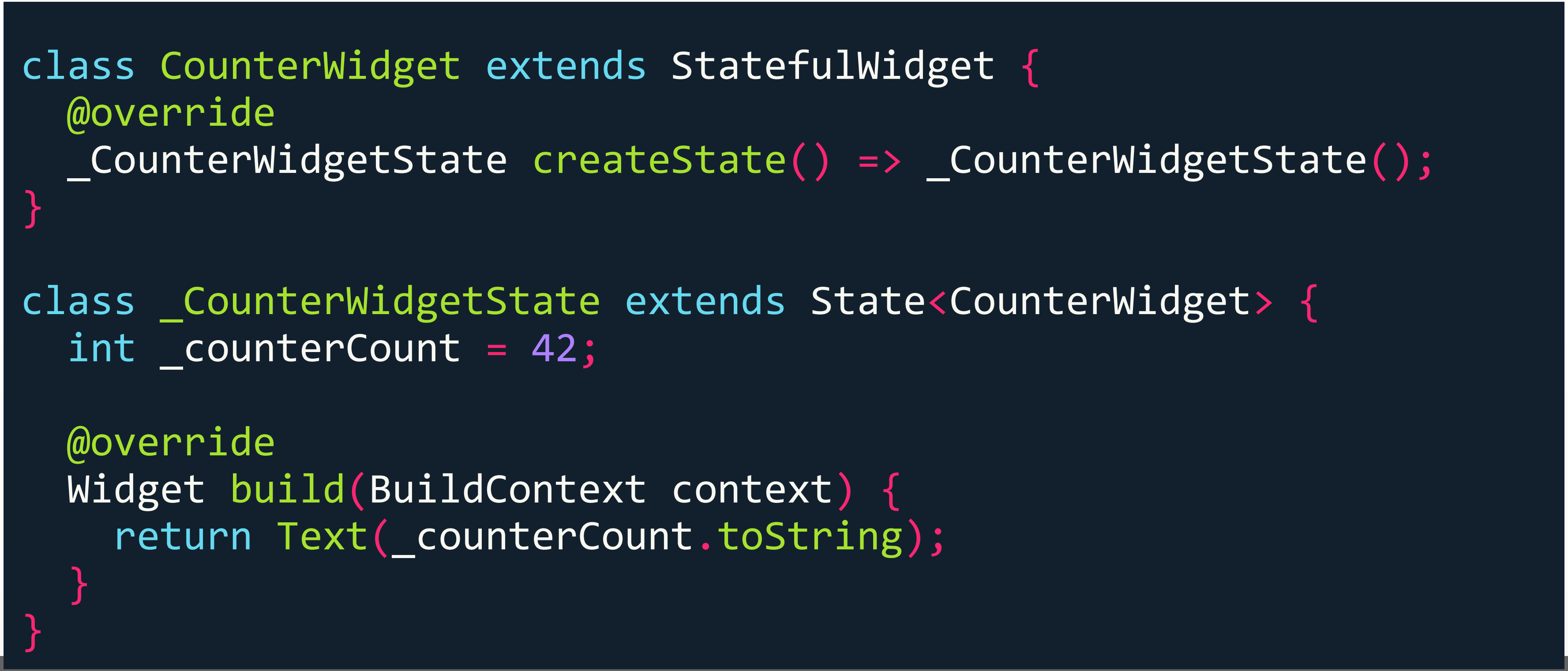
How does it happen in practise?

You need

- Subclass of StatefulWidget that defines the widget
- A subclass of State that contains the widget’s state and defines the build method

State contains.. the widget state! StatefulWidget can get rebuilt over and over, but will find the correct state and rebuild based on that data.

Stateful widget (minimal example)



```
class CounterWidget extends StatefulWidget {
  @override
  _CounterWidgetState createState() => _CounterWidgetState();
}

class _CounterWidgetState extends State<CounterWidget> {
  int _counterCount = 42;

  @override
  Widget build(BuildContext context) {
    return Text(_counterCount.toString);
  }
}
```

What was the point?

That was not very exciting, since the state was hardcoded into the State object.

Let’s review recommended reading and go into a more complicated demo.

Let’s have a look at a demo with dynamic content

More complicated, but with more interactivity.

Key features to look for:

setState() triggers a rebuild
onPressed() property: defines which function gets executed when the widget is pressed

Further reading

Stateful widgets and interactivity
https://flutter.dev/docs/development/ui/interactive

State management
https://flutter.dev/docs/development/data-and-backend/state-mgmt/intro

Pragmatic state management in Flutter
https://www.youtube.com/watch?v=d_m5csmrf7I