Stateful widgets and dynamic UI elements

- Update texts in your app when user taps a button.
- Learn that stateless widgets are not updated.
- Be able to change stateless widgets into stateful widget
- Learn to use setState in stateful widgets.
- Use Sliders, TextFields, Checkboxes and Switches in your app.
- Use ListTiles and CheckboxListTiles as part of your app.

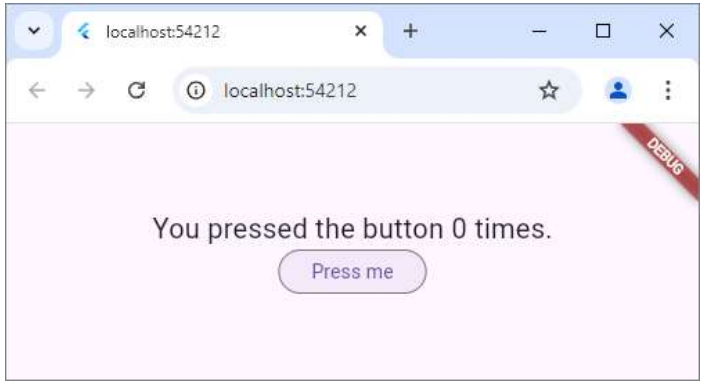
Stateless widgets do not react on user action

```
Run | Debug | Profile
void main() {
  runApp(const MainApp());
}

int counter = 0;

class MainApp extends StatelessWidget {
  const MainApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text("You pressed the button $counter times.",
                  style: const TextStyle(fontSize: 20)), // Text
              OutlinedButton(
                  onPressed: () {
                    counter++;
                    print("onPressed: counter is $counter");
                  },
                  child: const Text("Press me")) // OutlinedButton
            ],
          ), // Column
        ), // Center
      ), // Scaffold
    ); // MaterialApp
  }
}
```




```
Launching lib\main.dart on Chrome in debug mode...
This app is linked to the debug service: ws://127.0.
Debug service listening on ws://127.0.0.1:54258/9szE
Connecting to VM Service at ws://127.0.0.1:54258/9sz
Connected to the VM Service.
onPressed: counter is 1
onPressed: counter is 2
onPressed: counter is 3
onPressed: counter is 4
```

Although variable counter is increased, the UI is not updated!

Replace a StatelessWidget by a StatefulWidget

Set cursor on “StatelessWidget” and use the 

Select “Convert to StatefulWidget”.

Compare old and new code:

```
Run | Debug | Profile
void main() {
  runApp(const MainApp());
}

int counter = 0;

class MainApp extends StatelessWidget {
  const MainApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text("You pressed the button $counter times.",
                  style: const TextStyle(fontSize: 20)), // Text
              OutlinedButton(
                  onPressed: () {
                    counter++;
                    print("onPressed: counter is $counter");
                  },
                  child: const Text("Press me")) // OutlinedButton
            ],
```

=

```
Run | Debug | Profile
void main() {
  runApp(const MainApp());
}

int counter = 0;

class MainApp extends StatefulWidget {
  const MainApp({super.key});

  @override
  State<MainApp> createState() => _MainAppState();
}

class _MainAppState extends State<MainApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text("You pressed the button $counter times.",
                  style: const TextStyle(fontSize: 20)), // Text
              OutlinedButton(
                  onPressed: () {
                    counter++;
                    print("onPressed: counter is $counter");
                  },
                  child: const Text("Press me")) // OutlinedButton
            ],
```

“setState” triggers a redraw of the widget

```
OutlinedButton(
    onPressed: () {
      setState(() {
        counter++;
        print("onPressed: counter is $counter");
      });
    },
    child: const Text("Press me")) // OutlinedButton
```

You pressed the button 5 times.

Press me

Notify the framework that the internal state of this object has changed.

Whenever you change the internal state of a [State] object, make the change in a function that you pass to [setState]:

```
setState(() { _myState = newValue; });
```

The provided callback is immediately called synchronously. It must not return a future (the callback cannot be `async`), since then it would be unclear when the state was actually being set.

Calling [setState] notifies the framework that the internal state of this object has changed in a way that might impact the user interface in this subtree, which causes the framework to schedule a [build] for this [State] object.

If you just change the state directly without calling [setState], the framework might not schedule a [build] and the user interface for this subtree might not be updated to reflect the new state.

IntelliSense helps to use setState

Select the code lines you want to wrap with setState, then press on  :

```
OutlinedButton(
    onPressed: () {
      counter++;
      print("onPressed: counter is $counter");
    },
    child: const Text("Press me")) // OutlinedButton
```

```
onPressed: () {
  counter++;
  onPressed: counter is $counter");

Extract
Extract Method
Surround With
[···] Surround with block
[···] Surround with 'if'
[···] Surround with 'while'
[···] Surround with 'for-in'
[···] Surround with 'for'
[···] Surround with 'do-while'
[···] Surround with 'setState'
[···] Surround
Enter to Apply
[···] Surround with 'try-finally'
}
```

```
OutlinedButton(
    onPressed: () {
      setState(() {
  counter++;
  print("onPressed: counter is $counter");
});
    },
    child: const Text("Press me")) // OutlinedButton
```

Enter
“Shift + Alt + F”
(“Format
Document”)

```
OutlinedButton(
    onPressed: () {
      setState(() {
        counter++;
        print("onPressed: counter is $counter");
      });
    },
    child: const Text("Press me")) // OutlinedButton
```

Alternatives how to call “setState”

💡 Tip 4: setState() and setState(...) are equal

It doesn’t matter if you use `setState` like this

```
setState(){
  _text = “Hello”;
});
```

or like this

```
_text = “Hello”;
setState(){});
```

The outcome is the same.

Copied from https://quickcoder.org/flutter-set-state/

Define a Slider

IntelliSense of VS Code helps us:

```
sl
  sliderValue
  SlideTransition(…)
  Slider(…)        ({required double value, required void Fu…
  Slider.adaptive(…)
  SliderTheme(…)
```

```
Slider(value: value, onChanged: onChanged)
```

```
Slider(
  value: sliderValue,
  onChanged: (value) {
    print(value);
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

Define the sliderValue variable inside the State class:

```
class _MainPageState extends State<MainPage> {
  double sliderValue = 0;
```

Define a Slider (continued)

When onChanged is null, the slider is disabled:

```
Slider(value: sliderValue, onChanged: null),
```

Without calling setState, the slider does not move:

```
Slider(
  value: sliderValue,
  onChanged: (value) {
    sliderValue = value;
    print(sliderValue);
  },
), // Slider
```

From Intellisense about “Slider”:

The slider itself does not maintain any state. Instead, when the state of the slider changes, the widget calls the [onChanged] callback. Most widgets that use a slider will listen for the [onChanged] callback and rebuild the slider with a new [value] to update the visual appearance of the slider.

- [value] determines currently selected value for this slider.
- [onChanged] is called while the user is selecting a new value for the slider.
- [onChangeStart] is called when the user starts to select a new value for the slider.
- [onChangeEnd] is called when the user is done selecting a new value for the slider.

More properties of a Slider

IntelliSense shows the c-tor parameters -> use **min** and **max** or **label** and **divisions**:

```
(new) Slider Slider({
  Key? key,
  required double value,
  double? secondaryTrackValue,
  required void Function(double)? onChanged,
  void Function(double)? onChangeStart,
  void Function(double)? onChangeEnd,
  double min = 0.0,
  double max = 1.0,
  int? divisions,
  String? label,
  Color? activeColor,
  Color? inactiveColor,
  Color? secondaryActiveColor,
  Color? thumbColor,
  MaterialStateProperty<Color?>? overlayColor,
  MouseCursor? mouseCursor,
  String Function(double)? semanticFormatterCallback,
  FocusNode? focusNode,
  bool autofocus = false,
  SliderInteraction? allowedInteraction,
})
```

```
Slider(
  value: sliderValue,
  min: -10,
  max: 10,
  // label is only shown during movement of the
  // slider and when divisions is defined !
  label: "$sliderValue",
  divisions: 10,
  onChanged: (value) {
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

Test more slider properties

A “German flag slider” (but take care: too many colors might confuse the user):

```
Slider(
  activeColor: Colors.black,
  thumbColor: Colors.red,
  inactiveColor: Colors.amber,
  value: sliderValue,
  onChanged: (value) {
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

A slider with a secondary tag:

```
Slider(
  secondaryTrackValue: sliderValue * 1.2,
  secondaryActiveColor: Colors.green,
  value: sliderValue,
  onChanged: (value) {
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

The secondary track value for this slider.

If not null, a secondary track using [Slider.secondaryActiveColor] color is drawn between the thumb and this value, over the inactive track.

If less than [Slider.value], then the secondary track is not shown.

It can be ideal for media scenarios such as showing the buffering progress while the [Slider.value] shows the play progress.

Alternative to anonymous onChanged method

Use Intellisense to create a method:

```
Slider(value: value, onChanged: onChanged)
Quick Fix
Create getter 'onChanged'
Create local variable 'onChanged'
Create method 'onChanged'
Create mixin 'onChanged'
Create field 'onChanged'
Enter to apply
```

```
void onChanged(double value) {
}
```

Fill it:

```
void onChanged(double value) {
  setState(() {
    sliderValue = value;
  });
}
```

and then rename it with F2:

```
void onChanged(double value) {
  set
    s
  });
}
onSliderValueChanged
Enter to Rename, Shift+Enter to Preview
```

```
void onSliderValueChanged(double value) {
  setState(() {
    sliderValue = value;
  });
}
```

"**Golden rule**": use an own method when quite a lot of code is needed in onChanged.

Exercise

- Define a Switch in your App. A Switch has 2 states: 

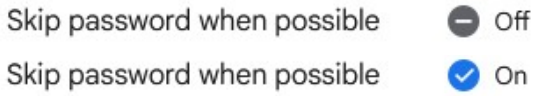
  BTW: the Flutter version used in the training last year showed here: 

  Hint: Let IntelliSense help you by entering “Sw” in a new line of your code.

- Use properties of the Switch to display it like that: 

- Display the state as a text right of the Switch: 

Remark: I saw such a combination of Switch and Text in my Google account settings:



Possible solution

```
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Switch(
      value: switchValue,
      activeColor: Colors.green,
      inactiveThumbColor: Colors.red,
      onChanged: (value) {
        setState(() {
          switchValue = value;
        });
      },
    ), // Switch
    const SizedBox(width: 20),
    Text(
      switchValue ? "ON" : "OFF",
      style: const TextStyle(fontSize: 22),
    ), // Text
  ],
) // Row
```

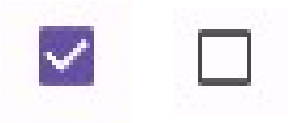
Alternatives for the text display:

```
const SizedBox(width: 20),
switchValue
    ? const Text("ON", style: TextStyle(fontSize: 22))
    : const Text("OFF", style: TextStyle(fontSize: 22))
```

With collection if's:

```
const SizedBox(width: 20),
if (switchValue)
  const Text("ON", style: TextStyle(fontSize: 22)),
if (!switchValue)
  const Text("OFF", style: TextStyle(fontSize: 22))
```

Define a Checkbox

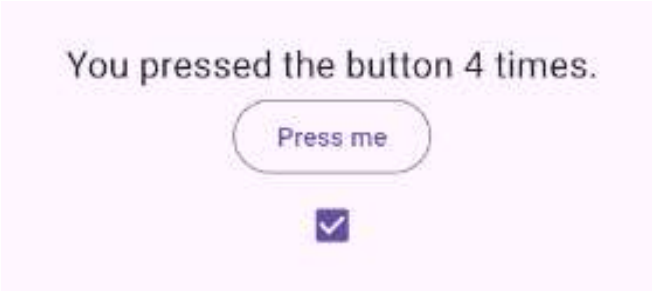


IntelliSense of VS Code helps as usual:

```
Che
    Checkbox(…)      ({required bool? value, required void Fun…
    Checkbox.adaptive(…)
    CheckboxListTile(…)
```

```
Checkbox(value: value, onChanged: onChanged)
```

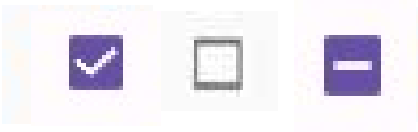
```
Checkbox(
    value: checkboxValue,
    onChanged: (value) {
      setState(() {
        print(value);
        checkboxValue = value;
      });
    }), // Checkbox
```




```
class _MainPageState extends State<MainPage> {
  bool? checkboxValue = true;
```

Type “bool**?**” is needed for **tristate checkboxes** with values true, false, null. See next slide.

Tristate checkbox



IntelliSense shows the c-tor arguments:

```
(new) Checkbox Checkbox({
  Key? key,
  required bool? value,
  bool tristate = false,
  required void Function(bool?)? onChanged,
  MouseCursor? mouseCursor,
  Color? activeColor,
  MaterialStateProperty<Color?>? fillColor,
  Color? checkColor,
  Color? focusColor,
  Color? hoverColor,
  MaterialStateProperty<Color?>? overlayColor,
  double? splashRadius,
  MaterialTapTargetSize? materialTapTargetSize,
  VisualDensity? visualDensity,
  FocusNode? focusNode,
  bool autofocus = false,
  OutlinedBorder? shape,
  BorderSide? side,
  bool isError = false,
  String? semanticLabel,
})
```

```
Checkbox(
  tristate: true,
  activeColor: Colors.amber,
  checkColor: Colors.green,
  splashRadius: 50, // effect size when pressed
  value: checkboxValue,
  onChanged: (value) {
    setState(() {
      checkboxValue = value;
      print("checkboxValue is $checkboxValue");
    });
  },
), // Checkbox
```

checkboxValue:

true

false

null

Tristate checkbox with text


```
Row(
  children: [
    Checkbox(
      tristate: true,
      value: checkboxValue,
      onChanged: (value) {
        setState(() {
          checkboxValue = value;
          print("checkboxValue is $checkboxValue");
        });
      },
    ), // Checkbox

    if (checkboxValue == true) const Text("SELECTED") ,
    if (checkboxValue == false) const Text("UNSELECTED"),
    if (checkboxValue == null) const Text("UNDEF")
  ],
), // Row
```

```
Row(
  children: [
    Checkbox(
      tristate: true,
      value: checkboxValue,
      onChanged: (value) {
        setState(() {
          checkboxValue = value;
          print("checkboxValue is $checkboxValue");
        });
      },
    ), // Checkbox
    getCheckboxTextWidget()
  ],
), // Row
```

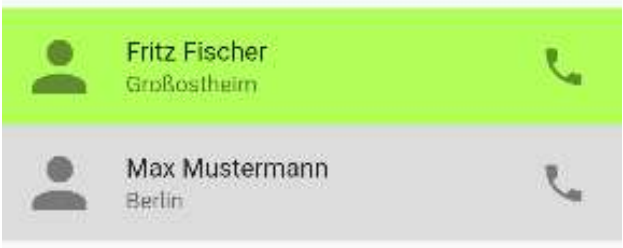
```
Widget getCheckboxTextWidget() {
  var result = const Text("undef");
  if (checkboxValue == true) {
    result = const Text("on");
  } else if (checkboxValue == false) {
    result = const Text("off");
  }
  return result;
}
```

Next goal

Show Switches and Checkboxes with some context:

Define ListTiles*) as preparation for CheckboxListTiles

```
ListT
    ListTile(…)      ({Color? textColor, Widget? leading, Widg…
    ListTileTheme(…)
```




```
(new) ListTile ListTile({
  Key? key,
  Widget? leading,
  Widget? title,
  Widget? subtitle,
  Widget? trailing,
  bool isThreeLine = false,
  bool? dense,
  VisualDensity? visualDensity,
  ShapeBorder? shape,
  ListTileStyle? style,
  Color? selectedColor,
  Color? iconColor,
ListTile()
```

```
ListTile(
  title: Text("Fritz Fischer"),
  subtitle: Text("Großostheim"),
  leading: Icon(Icons.person, size: 50),
  trailing: IconButton(
    icon: Icon(Icons.phone),
    iconSize: 30,
    onPressed: () {},
  ), // IconButton
  tileColor: Colors.lightGreenAccent,
), // ListTile
```

*) “Tile” in German: Fliese, Ziegel, Plättchen

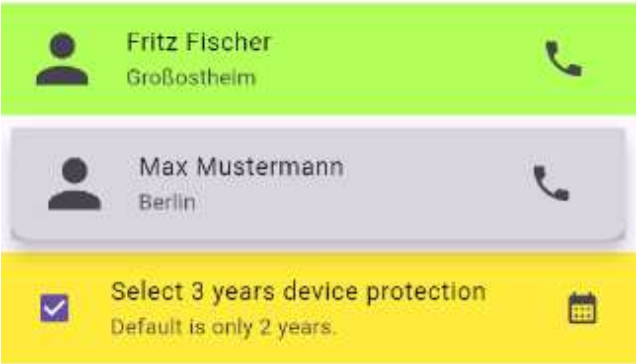
CheckboxListTile

```
CheckboxListTile(
  title: const Text("Select 3 years device protection"),
  subtitle: const Text("Default is only 2 years."),
  value: checkboxValue,
  onChanged: (value) {
    setState(() {
      checkboxValue = value;
    });
  },
  tileColor: Colors.yellow,
  secondary: const Icon(Icons.calendar_month),
  // with next line the checkbox is displayed on the left
  controlAffinity: ListTileControlAffinity.leading,
), // CheckboxListTile
```

Tapping anywhere in the tile is modifying the Checkbox !

Surrounding ListTile with a Card widget

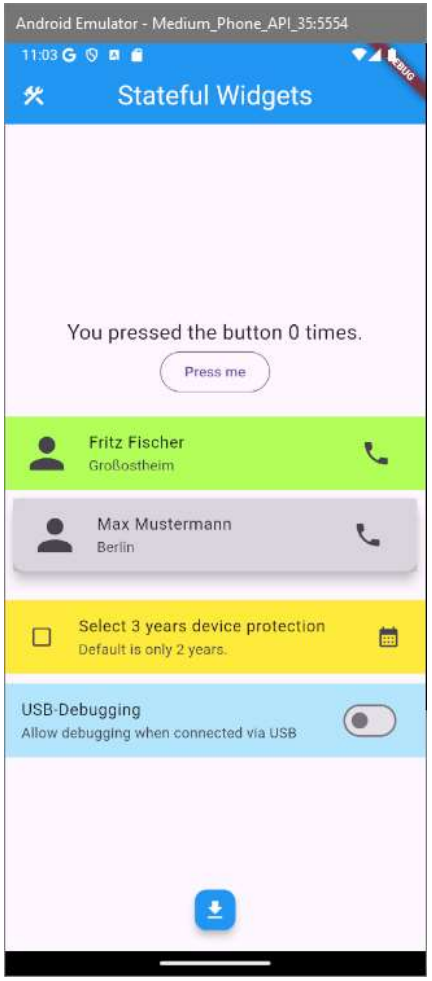
```
Card(
  margin: const EdgeInsets.all(8),
  elevation: 10,
  child: ListTile(
    title: const Text("Max Mustermann"),
    subtitle: const Text("Berlin"),
    leading: const Icon(Icons.person, size: 50),
    trailing: IconButton(
      icon: const Icon(Icons.phone),
      iconSize: 30,
      onPressed: () {},
    ), // IconButton
    tileColor: Colors.black12,
  ), // ListTile
), // Card
```

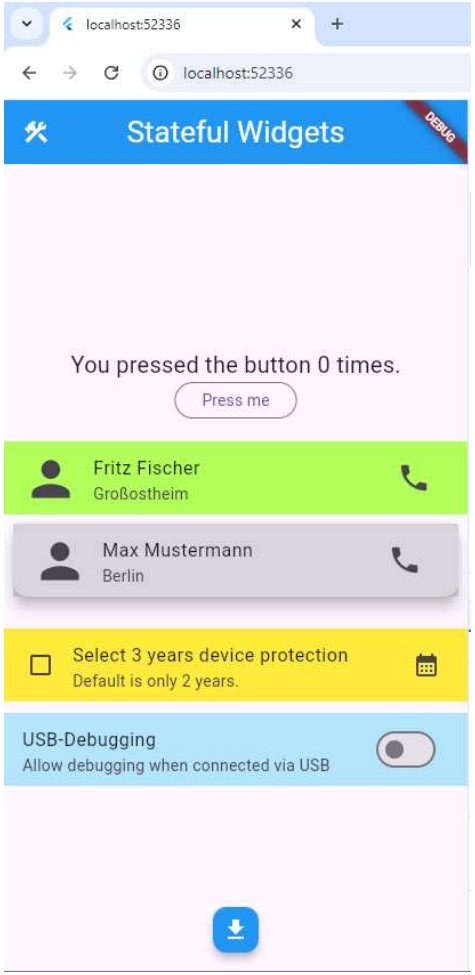


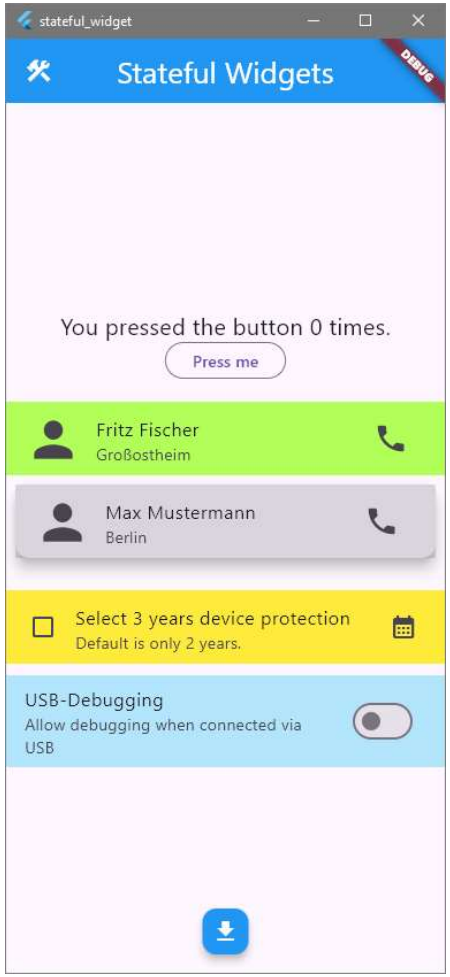
Same without Card:

Same UI on different devices

As Windows app:

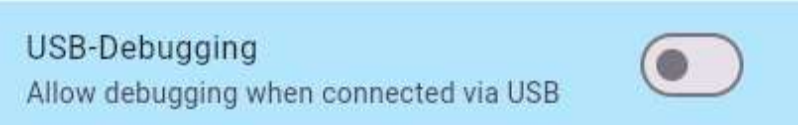






Exercise

- Define a SwitchListTile in your App which looks like the one displayed on last slide:



- Find and watch YouTube videos on Flutter Switch and Flutter SwitchListTile.

  Some samples:

  Switch: https://www.youtube.com/watch?v=MnR2xBcqgw8
  SwitchListTile: https://www.youtube.com/watch?v=J_8bZ2trhrM
  Switch- and other ListTiles: https://www.youtube.com/watch?v=0igIjvtEWNU

- Ask ChatGPT about SwitchListTile.

Possible solution

```
SwitchListTile(
  value: switchValue,
  title: const Text("USB-Debugging"),
  subtitle: const Text("Allow debugging when connected via USB"),
  tileColor: Colors.lightBlue.shade100,
  onChanged: (value) {
    setState(() {
      switchValue = value;
    });
  },
), // SwitchListTile
```