Class Activity 4: Building Interactive Flutter Apps

Topic: Stateful Widgets & User Interaction
Date: In-Class Workshop

⏱️ **Duration: 2 Hours**

🎯 Learning Objectives:

- Understand the difference between Stateless and Stateful widgets.
- Implement user interaction using Buttons, Sliders, and Text Fields.
- Master the use of `setState()` to update the UI dynamically.
- Apply logical conditions to control widget state (limits, colors, resets).

🚀 Quick Overview

What we are building: A dynamic Counter Application that reacts to your input. It starts as a simple number, but you will transform it into a full dashboard with buttons, sliders, color-changing alerts, and custom inputs.

Prerequisites:

- ✅ Flutter SDK installed and running
- ✅ Basic understanding of Dart syntax
- ✅ A simulator/emulator or physical device connected

Part 1: Understanding the Basics

🕒 30 Minutes

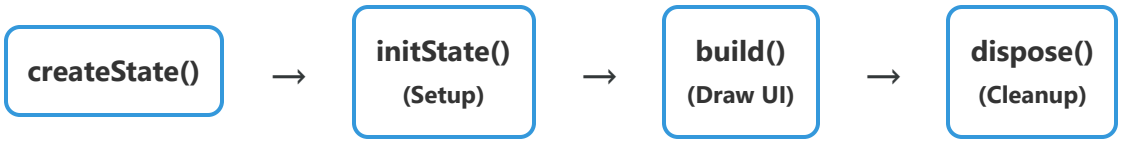
1. What is "State"?

State is simply the **data** that determines how your app looks at any moment.

- **StatelessWidget:** Like a printed photo. Once created, it cannot change. (Examples: Icons, Static Text).
- **StatefulWidget:** Like a video or game. It can change dynamically when users interact with it. (Examples: Checkboxes, Sliders, Counters).

2. The Widget Lifecycle

Unlike basic widgets, a StatefulWidget has a "lifecycle" - a series of steps it goes through.



Part 2: Building the Base App

🕑 30 Minutes

Step 1: Create Project

Open your terminal and run: `flutter create stateful_lab`

Step 2: The Skeleton Code

Replace the contents of `lib/main.dart` with this code. This gives us a basic app with a Slider.

Copy

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Stateful Lab',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: CounterWidget(),
    );
  }
```

```
}

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

class _CounterWidgetState extends State<CounterWidget> {
  int _counter = 0; // This is our STATE

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Interactive Counter')),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Center(
            child: Container(
              color: Colors.blue.shade100,
              padding: EdgeInsets.all(20),
              child: Text(
                '$_counter',
                style: TextStyle(fontSize: 50.0),
              ),
            ),
          ),
          SizedBox(height: 20),
          Slider(
            min: 0, max: 100,
            value: _counter.toDouble(),
            onChanged: (double value) {
              // 👇 This triggers the UI rebuild
              setState(() {
                _counter = value.toInt();
              });
            },
          ),
        ],
      ),
    );
  }
}
```

> 💡 **Key Concept:** The `setState()` method is the engine. If you update `_counter` without it, the variable changes, but the screen stays the same!

Part 3: Hands-On Challenges

🕒 **50 Minutes**

Complete these levels to master State management. Raise your hand if you get stuck!

LEVEL 1: The Basics (Easy)

Add Buttons

Task: Add two buttons below the slider: one to increment (+1) and one to reset the counter to 0.

Hint: Use `ElevatedButton` inside a `Row` widget.

Success Criteria:

- [ ] Button labeled "+" increases counter by 1.
- [ ] Button labeled "Reset" sets counter to 0.
- [ ] UI updates immediately when tapped.

LEVEL 2: Logic & Feedback (Medium)

Colors & Limits

Task: Add visual feedback logic to the Text widget display.

1. Add a Decrement Button (-1). Ensure it **cannot go below 0**.
2. Change the Text color based on the value:
    - Red if value is 0.
    - Green if value is > 50.
    - Black otherwise.

Success Criteria:

- [ ] Counter stops at 0 (no negative numbers).

- [ ] Text turns red at 0 and green above 50.

LEVEL 3: Advanced Input (Hard)

Custom Increment

Task: Allow users to type a custom number to jump the counter.

1. Add a `TextField` widget (requires a `TextEditingController`).
2. Add a button "Set Value" that reads the text field and updates the counter.
3. **Constraint:** The counter cannot exceed 100. If user tries, show a `SnackBar` saying "Limit Reached!".

Success Criteria:

- [ ] User can type a number and set the counter.
- [ ] App does not crash if non-numbers are typed (handle errors).
- [ ] Counter refuses to go above 100.

🏆 **BONUS CHALLENGE**

For Fast Finishers: Implement an "Undo" button that reverts the counter to its previous value. You will need a `List<int>` to store history!

Part 4: Skeleton Code

🕒 **Reference Implementation**

> 📘 **Understanding the Basics:** This skeleton code provides a foundation for building interactive Flutter applications using stateful widgets and user interactions.

Replace the contents of `lib/main.dart` with the following code:

📋 Copy

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

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

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // Application name
      title: 'Stateful Widget',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      // A widget that will be started on the application startup
      home: CounterWidget(),
    );
  }
}

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

class _CounterWidgetState extends State<CounterWidget> {
  //initial counter value
  int _counter = 0;

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

```
    return Scaffold(
      appBar: AppBar(
        title: const Text('Stateful Widget'),
      ),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Center(
            child: Container(
              color: Colors.blue,
              child: Text(
                //displays the current number
                '$_counter',
                style: TextStyle(fontSize: 50.0),
              ),
            ),
          ),
          Slider(
            min: 0,
            max: 100,
            value: _counter.toDouble(),
            onChanged: (double value) {
              setState(() {
                _counter = value.toInt();
              });
            },
            activeColor: Colors.blue,
            inactiveColor: Colors.red,
          ),
        ],
      ),
    );
  }
}
```

📝 Quick Summary

🎯 How This Code Works:

📦 State Management:
The `_CounterWidgetState` class is responsible for managing the widget's state. It holds the data that can change over time.

🔢 Counter Variable:
The `_counter` variable stores the current value of the counter. When the user moves the slider, this value changes.

🔄 **setState() Method:**
The `setState()` method is the key to updating the UI. When called, it tells Flutter that the widget's state has changed, triggering a rebuild of the widget tree. Without `setState()`, the UI won't update even if the variable changes!

💡 **Remember:** Always wrap state changes inside `setState(() { ... })` to see updates on the screen!

🔑 Keys to the Code

📊 State Management Components

- **State Variable (_counter):** A state variable initialized to 0 that stores the current counter value. The underscore (_) makes it private to this file.
- **build Method:** This method builds and returns the UI for the CounterWidget. It's called automatically whenever `setState()` is triggered.

📐 Layout Structure Components

- **Scaffold:** Provides the basic material design structure for the visual interface, including an AppBar at the top and a body for the main content.
- **AppBar:** The blue bar at the top of the app that displays the title "Stateful Widget".
- **Column:** A layout widget that arranges its children vertically in a single column.
- **Center:** Centers its child widget both horizontally within the available space.

🎨 Display & Interaction Components

- **Container:** A box model widget that holds the counter text. It has a blue background color and acts as a visual wrapper.
- **Text:** Displays the current value of `_counter` with a font size of 50. The `'$_counter'` syntax inserts the variable value into the string.
- **Slider:** An interactive slider widget that allows users to select a value between 0 and 100. When the slider moves:
  - The `onChanged` callback is triggered
  - `setState()` is called with the new value
  - `_counter` is updated
  - The entire widget rebuilds, showing the new value

Testing & Final Polish

🕒 10 Minutes

✅ Testing Checklist

- ☐ Does the slider move smoothly?
- ☐ Does the Reset button work from any number?
- ☐ Does the color change exactly at 50?
- ☐ Does the counter stop at 0 when decrementing?
- ☐ Does the layout look good on screen rotation?

⚠️ Common Mistakes

1. **Forgetting setState:** Changing variables without it won't update the screen.
2. **Logic in Build:** Don't do heavy calculations inside `build()`.
3. **Infinite Loops:** Never call `setState` directly inside the build method.
4. **Type Errors:** Remember `TextField` gives a String, but counter needs an Int. Use `int.parse()`.

📤 Submission Instructions

1. Build and Upload APK:

Build your release APK using the following command:

```
flutter build apk --release
```

📋 Copy

Upload the generated `app-release.apk` file located in: `build/app/outputs/flutter-apk/app-release.apk`

2. GitHub Repository:

- Push all code to GitHub with regular commits
- Include repository URL in submission
- Show collaboration through commit history

3. Required Submission:

- `app-release.apk` file
- GitHub repository URL

Due Date: End of class session.

Quick Reference Card

Standard setState Syntax:

```
setState(() {
  _counter++;
});
```

Simple Button:

```
ElevatedButton(
  onPressed: () { ... },
  child: Text("Click Me"),
)
```

Conditional Color Logic:

```
style: TextStyle(
  color: _counter > 50 ? Colors.green : Colors.black
)
```