Flutter State Management: setState, Stateful Widgets & Basics

Subtitle: Learn how to manage dynamic data and update UI in Flutter apps effectively.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Static UI is useless in real apps. You need dynamic behavior—user input, data changes, UI updates. That's where **state management** comes in. This guide focuses on the most important beginner concept: **setState and StatefulWidget**, which you must master before moving to advanced tools.

Step 1: Stateless vs Stateful Widgets

StatelessWidget

- UI does **not change** after build
- Used for static content

StatefulWidget

- UI **changes dynamically**
- Uses **state** to update UI

Example:

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text("Static Text");
  }
}
```

```
class MyWidget extends StatefulWidget {
  @override
  _MyWidgetState createState() => _MyWidgetState();
}
```

```
class _MyWidgetState extends State<MyWidget> {
  @override
  Widget build(BuildContext context) {
    return Text("Dynamic Text");
  }
}
```

Step 2: Understanding setState()

`setState()` tells Flutter:
👉 “UI needs to update”

Whenever state changes, you must call `setState()`

Example: Counter App

```
int count = 0;

ElevatedButton(
  onPressed: () {
    setState(() {
      count++;
    });
  },
  child: Text("Increase"),
)
```

Exercise: Build a counter with **Increase and Decrease buttons**.

Step 3: Managing User Input

Use `TextField` with state to handle input.

Example:

```
TextEditingController controller = TextEditingController();

TextField(
  controller: controller,
)
```

```
ElevatedButton(
```

```
  onPressed: () {
    setState(() {
      name = controller.text;
    });
  },
  child: Text("Submit"),
)
```

Exercise: Create an input field that displays entered text below it.

Step 4: Updating UI Dynamically

- Store values in state variables
- Update them using `setState()`
- UI rebuilds automatically

Example:

```
String message = "Hello";

ElevatedButton(
  onPressed: () {
    setState(() {
      message = "Welcome to Flutter";
    });
  },
  child: Text("Change Text"),
)
```

Step 5: Best Practices

- Keep state **local and minimal**
- Avoid unnecessary `setState()` calls
- Break UI into smaller widgets
- Use StatefulWidget only when needed
- Prepare to move to advanced state tools later (Provider, Riverpod, etc.)

Step 6: Mini Project

Build a **Simple Counter App with Input**:

Features:

- Counter with increment/decrement
- Input field to enter name
- Display greeting: “Hello, [Name]”
- Button to reset counter

Key Takeaways

- Use **StatefulWidget** for dynamic UI
- `setState()` is the core of Flutter state updates
- UI rebuilds automatically when state changes
- Keep state simple and organized
- Master this before learning advanced state management

Most beginners skip mastering `setState()` and jump to advanced tools too early. That’s a mistake. If you don’t understand this deeply, everything later becomes confusing and messy.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
