Application Development

Lecture 3

Core Widgets I – Stateless & Stateful Widgets, Widget Tree, and UI Composition

Asst. Lect. Ali Al-khawaja

Class Room

Lecture Contents Table

General & Behavioral Objectives

General Goal:

Provide students with a deep understanding of Flutter's widget system and the fundamental difference between Stateless and Stateful widgets, enabling them to design interactive, maintainable UIs.

Behavioral Objectives:

By the end of this lecture, students will be able to:

1. **Define** the role of widgets as the core building blocks of Flutter UIs.
2. **Differentiate** between Stateless and Stateful widgets in terms of structure and use cases.
3. **Illustrate** the hierarchy of a widget tree and explain how composition creates complex UIs.
4. **Analyze** UI requirements and decide whether a widget should be stateless or stateful.
5. **Demonstrate** understanding through class activities and group discussion.

Introduction

- Flutter apps are entirely composed of **widgets**—from the root app to every text label and layout container.
- Understanding widgets is essential for building any Flutter UI, whether simple or complex.
- Today we focus on the **two core widget types**—Stateless and Stateful—and the way they form a **widget tree** that defines the interface.

What Is a Widget?

Definition:

A widget is a **declarative description** of part of a user interface.

Everything is a widget:

buttons, text, layout structures, even the entire app.

Declarative approach:

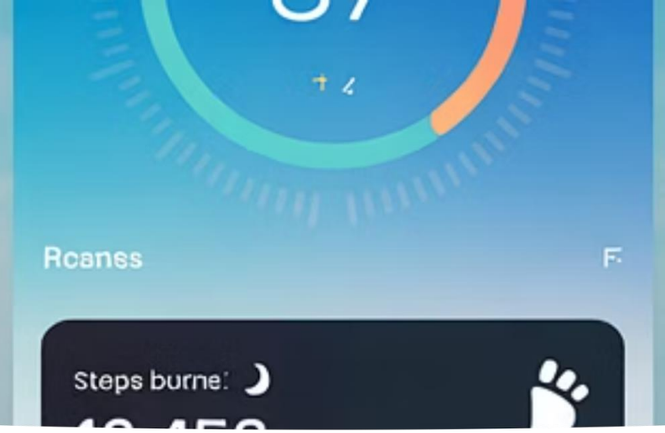
Widgets describe **how the UI should look**, not how to draw each pixel.

Rendering:

Flutter's rendering engine converts these descriptions into actual pixels.

Activity 1 – Brainstorming

"List as many UI elements as you can that could be represented as widgets in a mobile application."



Stateless Widgets: Concept

Immutable UI Elements

Represent parts of the UI that **do not change** once built.

Final Properties

Immutable: properties are final after construction.

Common Examples

Examples: static text labels, icons, decorative images.

Stateless Widget Structure

Typical pattern:

```
class MyTitle extends StatelessWidget {  @override
Widget build(BuildContext context) {    return
Text('Welcome to Flutter!');  }}
```

Build method runs **only when the widget is inserted** or **its parent changes**.

Expanded Stateless Examples

Product Descriptions

Display static product descriptions.

Design Elements

Constant design elements like headers, logos, or background graphics.

Performance Benefits

Efficient because Flutter **skips unnecessary rebuilds**.

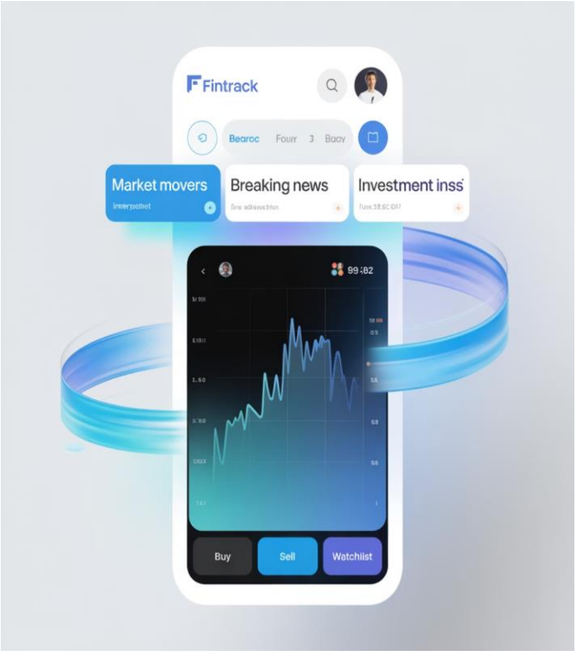
Activity 2 – Paper & Pen

Write down one real-life screen from any app that can be built entirely with Stateless widgets and explain why state changes are unnecessary.

Stateful Widgets: Concept

Dynamic UI Elements

- Represent UI elements that **change dynamically** in response to user actions or data updates.
- Contain a **State object** separate from the widget configuration.



Anatomy of a Stateful Widget

Two classes required:

```
class Counter extends StatefulWidget {
  @override
  _CounterState createState() => _CounterState();
}

class _CounterState extends State<Counter> {
  int count = 0;
  @override
  Widget build(BuildContext context) {
    return Text('Count: $count');
  }
}
```

1

StatefulWidget

defines configuration

2

State

holds mutable data and the build() method

Stateful Lifecycle Methods

initState()

initialise data/resources

setState()

request UI rebuild when state changes

didChangeDependencies()

respond to changes in inherited widgets

dispose()

clean up resources

Activity 3 – Class Discussion

"Why must the UI update immediately when user interaction occurs, and how does setState() enable this?"

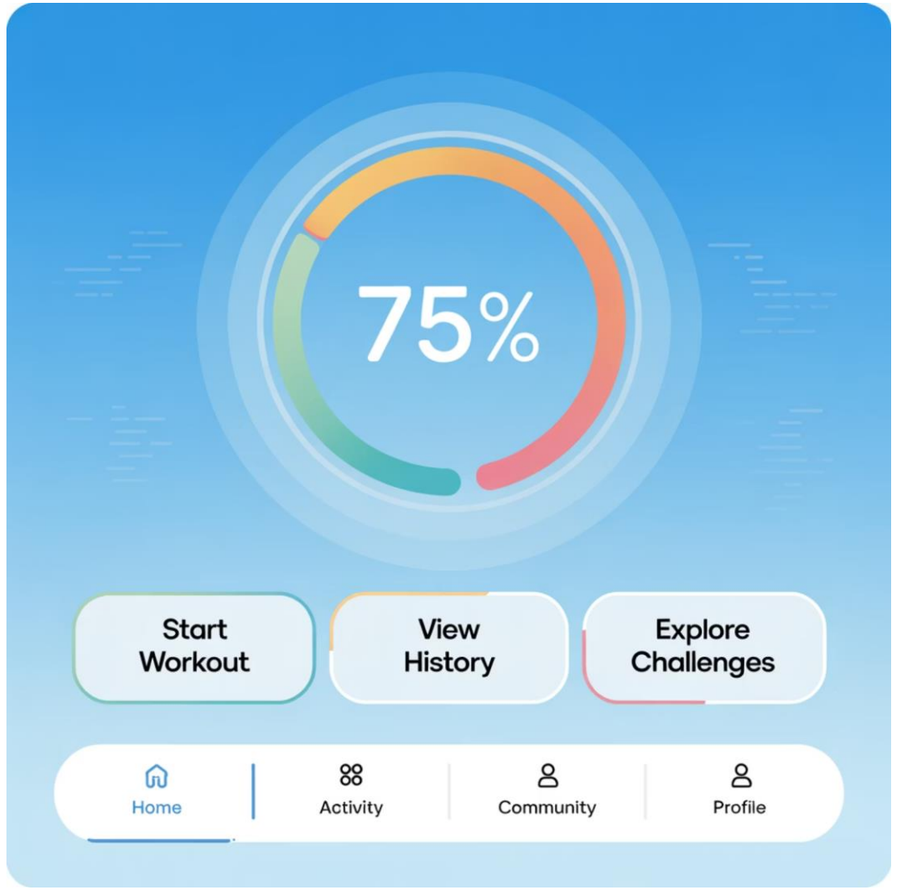
Comparing Stateless & Stateful

Feature	Stateless	Stateful
Data Changes	No	Yes
Rebuild Trigger	Parent change only	setState / external triggers
Complexity	Simple	More complex, needs State object
Performance	Very fast	Slight overhead

When to Choose Which

Use Stateless when:

UI depends solely on final parameters.



Use Stateful when:

UI depends on dynamic data, animations, or user input.

Activity 4 – Raise-Hand Quick Quiz

Question:

"Would a login form with live validation be Stateless or Stateful? Why?"

Widget Tree Fundamentals

Hierarchical Structure

Widgets are arranged in a **hierarchical tree**.

Root Widget

The **root widget** is typically MaterialApp or CupertinoApp.

Composition

Each child widget can contain its own children, forming complex UIs through **composition**.

Build Process

Widget Tree Construction

Flutter builds the widget tree from top to bottom.

Layout Constraints

Layout constraints are passed down; sizes flow up.

Painting

The final render tree is drawn to the screen.

Activity 5 – Group Work

Groups of 4–5:

Design a widget tree for a simple "Profile Page" that includes a profile image, name, and list of settings. Identify which nodes are Stateless and which should be Stateful.

UI Composition

Composition over inheritan

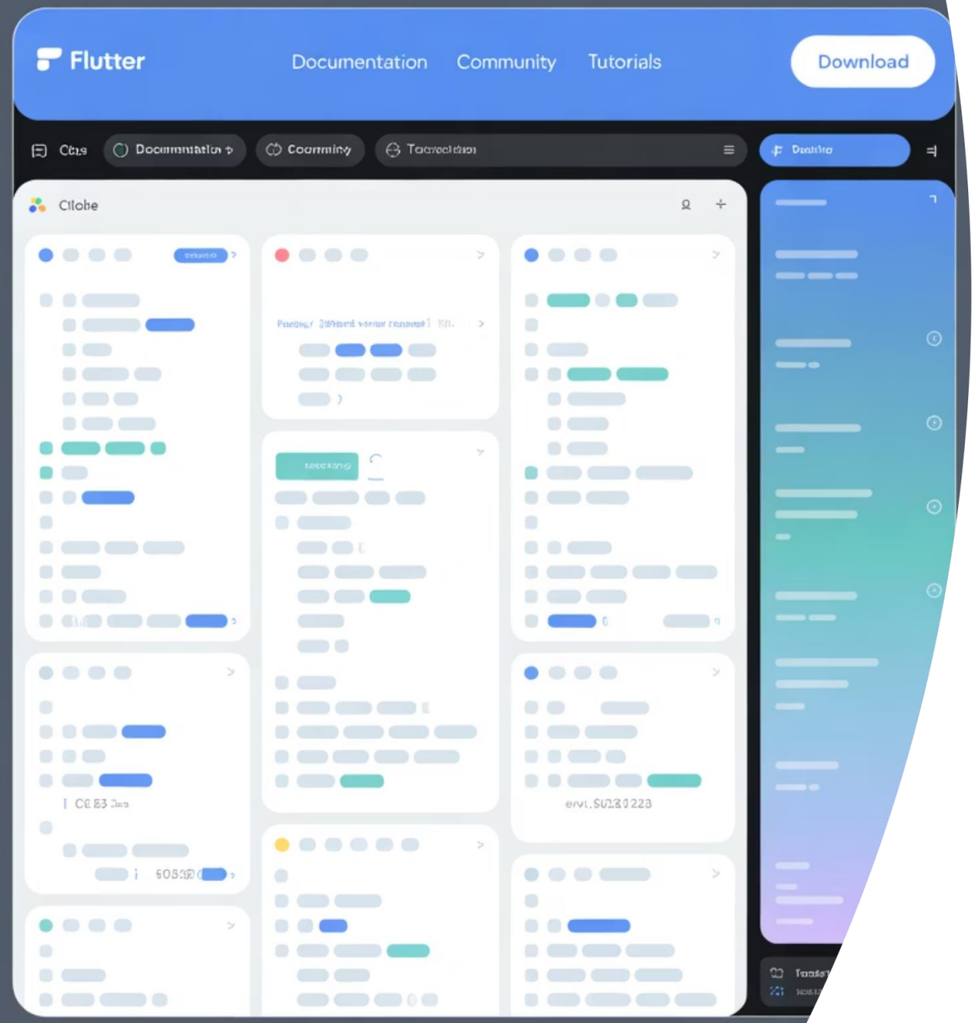
combine small widgets to create rich UIs

Reusable custom widgets

improve maintainability

Example

create a UserCard widget reused in different screens



Practical Composition Tips

Logical Components

Break UI into **logical components**.

Performance Optimisation

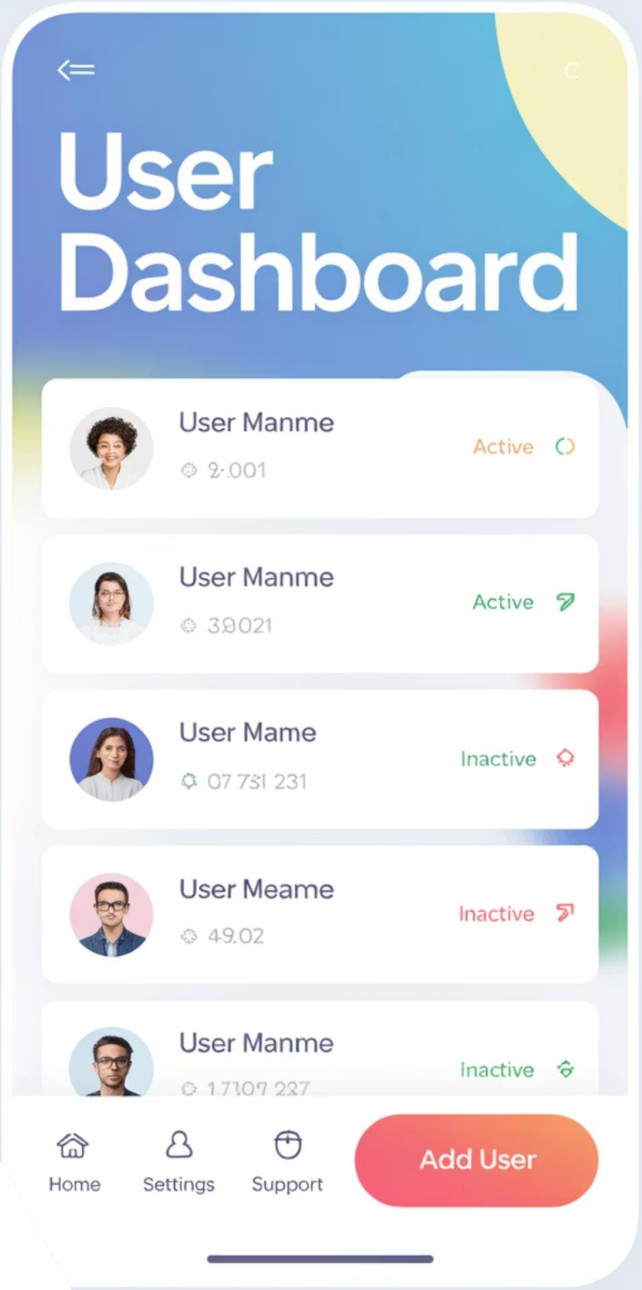
Use const constructors when possible to improve performance.

Shallow Trees

Keep widget trees shallow when feasible; extract widgets instead of deeply nesting.

Advanced Composition Example

Demonstration of a screen built from multiple custom widgets (Header, UserList, Footer) to show modular design and readability.



Common Mistakes

Overusing Stateful Widgets

Overusing Stateful widgets where Stateless is enough → performance penalty.

Deep Widget Trees

Deep widget trees without refactoring → harder maintenance.

setState() Errors

Forgetting to call setState() correctly → UI not updating.

Best Practices

Start Simple

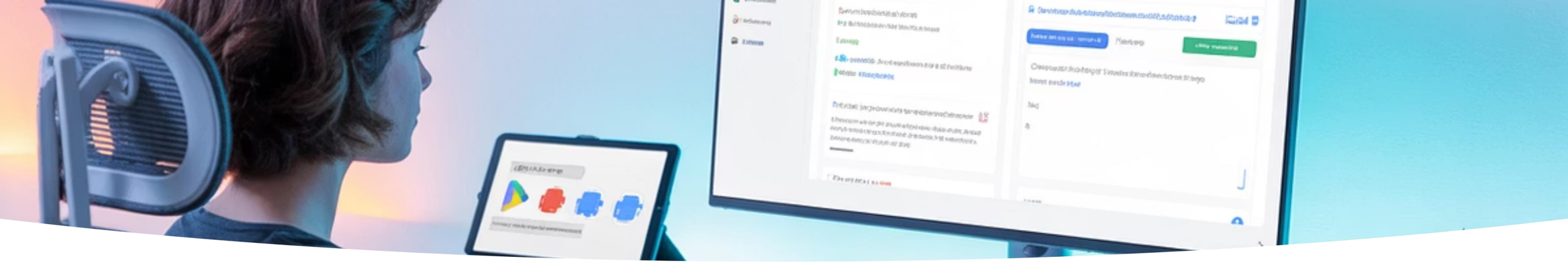
Start with Stateless, switch to Stateful only when necessary.

Extract Early

Extract widgets early to keep code readable

Use Keys Wisely

Use keys (Key) wisely when building lists or dynamic UI elements.



Activity 7 – Homework (Google Classroom)

Task:

1. Create a small Flutter UI demonstrating one Stateless and one Stateful widget.
2. Explain in ~150 words when each widget is appropriate and how they interact in the widget tree.

Summary & Key Takeaways

Everything is a widget

in Flutter.

Stateless vs Stateful:

choose based on data mutability.

The widget tree

is the foundation of UI composition.

Effective composition

ensures scalability and maintainability.

Conclusion

Today we explored:

- Core concept of widgets as Flutter's fundamental building blocks.
- Detailed differences between Stateless and Stateful widgets.
- Hierarchical widget tree and UI composition strategies.
- Activities to reinforce conceptual understanding and practical design thinking.

Thank you…

Any questions??

يرجى مسح رمز الاستجابة السريعة QR Code لتعبئة نموذج التغذية الراجعة حول المحاضرة. ملاحظاتكم مهمة لتحسين المحاضرات القادمة.

My google site