Application Development

Lecture 3

Widgets I

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 Widgets in Flutter?

- Flutter is Google's UI toolkit for crafting beautiful, natively compiled iOS and Android apps from a single code base.
- To build any application we start with widgets - The building block of Flutter applications.
- Widgets describe what their view should look like given their current configuration and state. It includes a text widget, row widget, column widget, container widget, and many more.

What are Widgets?

Each element on the screen of the Flutter app is a widget. The view of the screen completely depends upon the choice and sequence of the widgets used to build the apps. The structure of the code of apps is a tree of widgets.

Category of Widgets

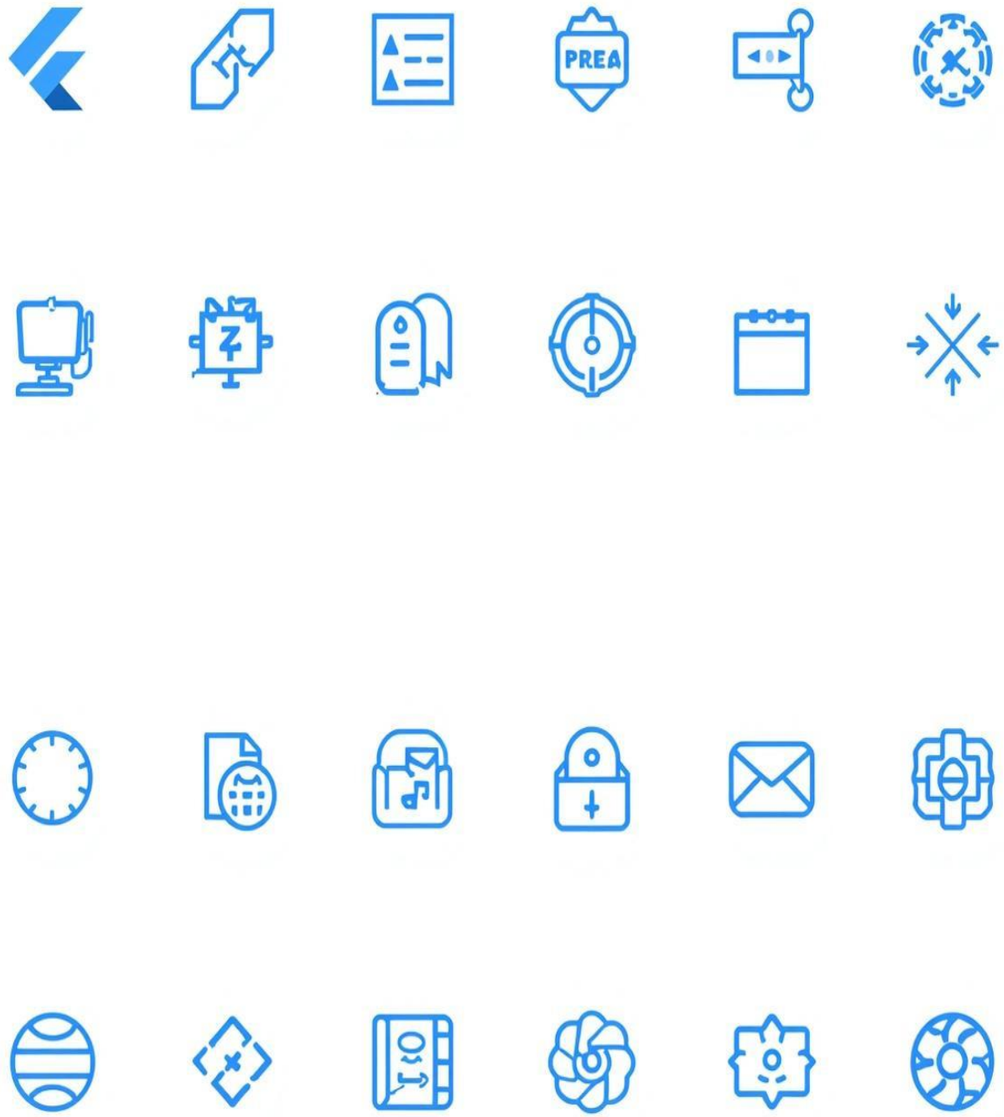
There are mainly 14 categories into which the flutter widgets are divided. They are mainly segregated on the basis of the functionality they provide in a flutter application.

- **Design systems**
- **Base widgets**

Design systems

Widgets	Description
Cupertino	These are the iOS-designed widgets.
Material Components	This is a set of widgets that mainly follow the material design by Google.

Base Widgets



Base Widgets

Accessibility

Accessibility widgets ensure that your app is usable for everyone, providing essential features for users with disabilities.

Async

Async widgets handle asynchronous tasks seamlessly, allowing your app to perform multiple operations without blocking the user interface.

Animation & Motion

Animation and motion widgets create dynamic visual experiences, enhancing user engagement through transitions and animated elements.

Basics

Basic widgets form the foundational elements of Flutter applications, including buttons, text fields, and containers essential for UI design.

Assets, Images & Icons

This category includes widgets for managing visual assets, enabling the effective display of images and icons within your application.

Input

Input widgets facilitate user interaction, capturing data through elements like text fields, checkboxes, and dropdown menus in a user-friendly manner.

Base Widgets

Interaction Models

Interaction models enable effective user engagement by managing gestures and navigation within the app environment for smooth experiences.

Layout

The layout widget organizes the placement of child widgets, ensuring a structured and responsive design across different screen sizes.

Animation & Motion

Animation and motion widgets enhance user experience by adding dynamic transitions and effects that bring the application to life.

Painting & Effects

Painting and effects widgets apply visual changes, allowing developers to implement custom graphics and enhance the overall app aesthetics.

Scrolling

Scrolling widgets facilitate the presentation of extensive content by enabling users to smoothly navigate through long lists or views.

Styling

Styling widgets manage themes, colors, and typography, ensuring a cohesive and visually appealing interface tailored to user preferences.

Enhancing App Accessibility

Screen Readers

Facilitates navigation for visually impaired users.

Text Scaling

Adapts text size for better readability.

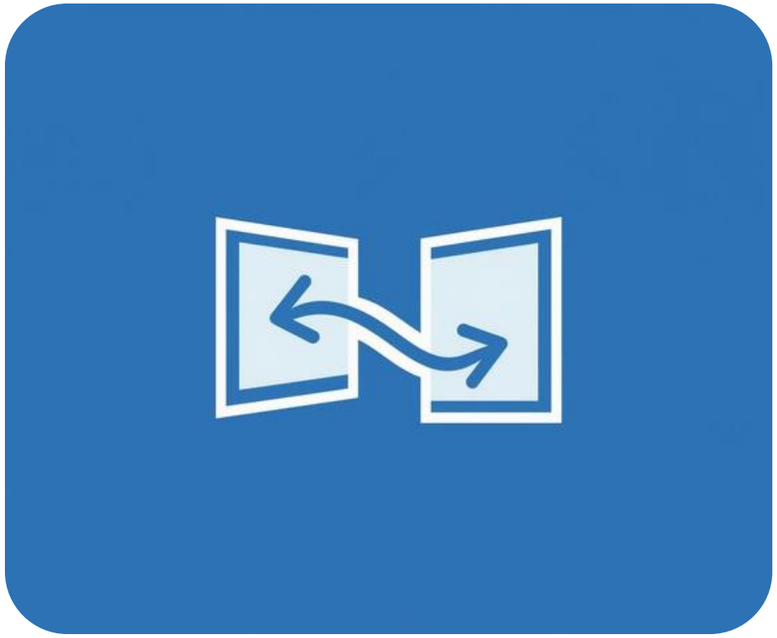
Contrast Modes

Improves visibility with adjustable color contrasts.

Animation and Motion Widgets

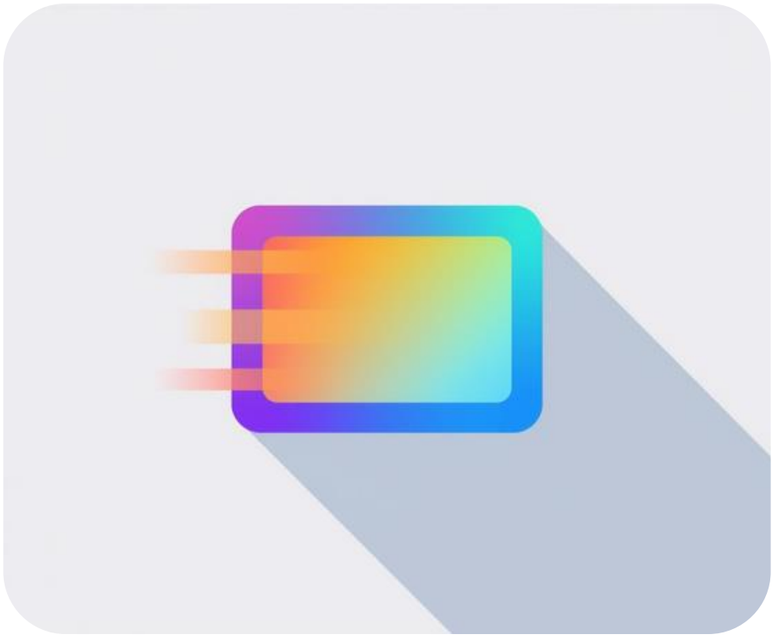
Hero

Enables smooth transitions between screens and elements.



AnimatedContainer

Creates animated changes based on properties over time.



AnimatedOpacity

Adjusts the transparency of widgets dynamically and smoothly.

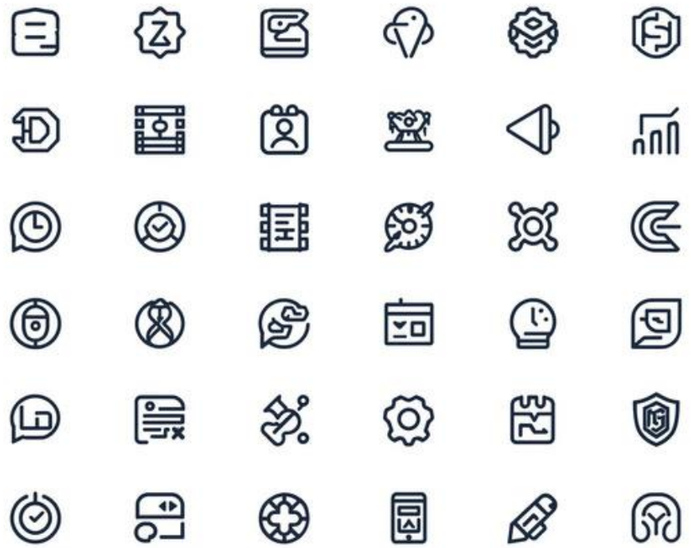
Managing Visual Elements in Flutter

Images

Display images seamlessly in your app.

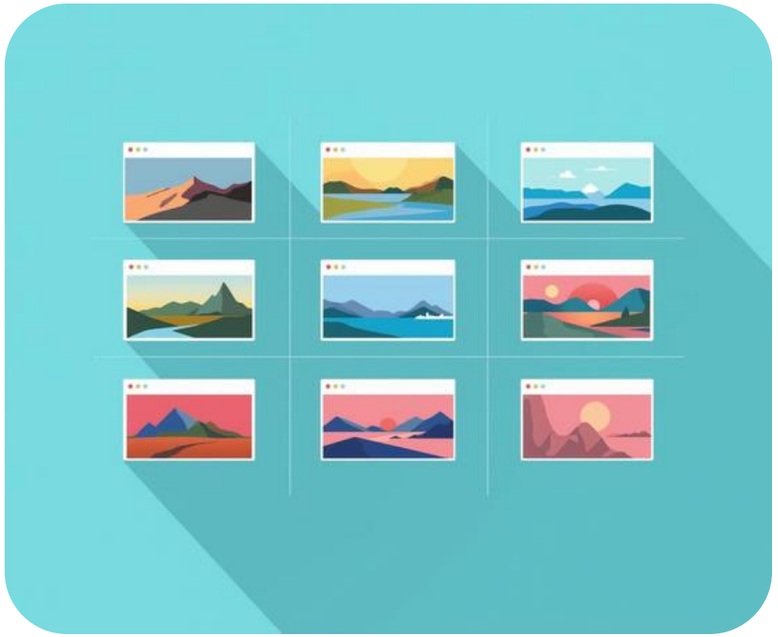
Icons

Use icons to enhance user navigation.



AssetImage

Load images from assets for efficiency.



Understanding Async Widgets

FutureBuilder

Handles asynchronous data, providing a flexible UI.

StreamBuilder

Updates UI in real-time as data streams.

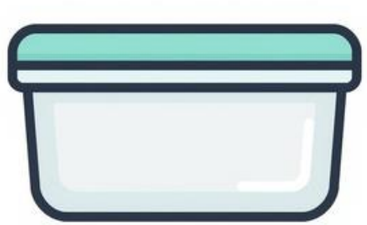
AsyncSnapshot

Represents the state of asynchronous operations.

Essential Base Widgets in Flutter

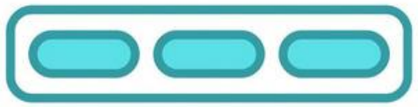
Container

The **Container** widget is crucial for layout.



Row

Use the **Row** widget for horizontal alignment.



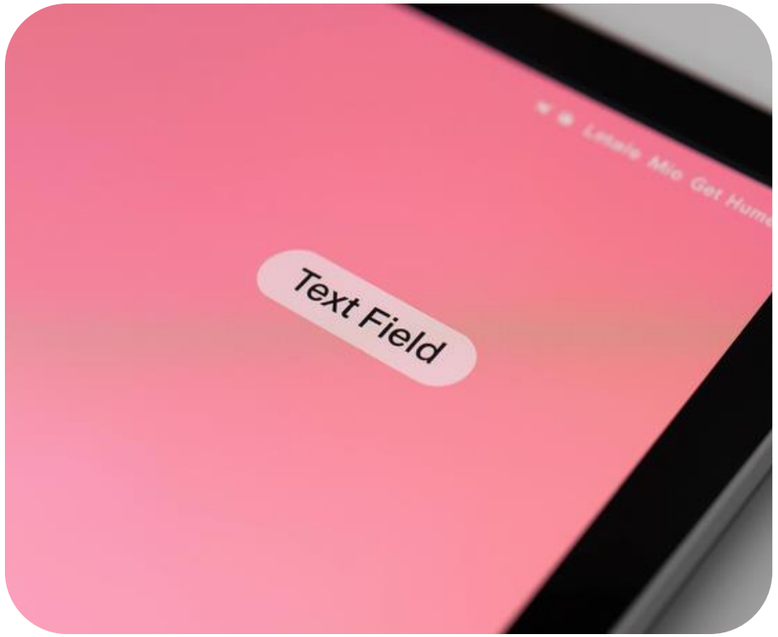
Column

The **Column** widget organizes children vertically.

Essential Input Widgets

TextField

A **TextField** allows users to enter text data.

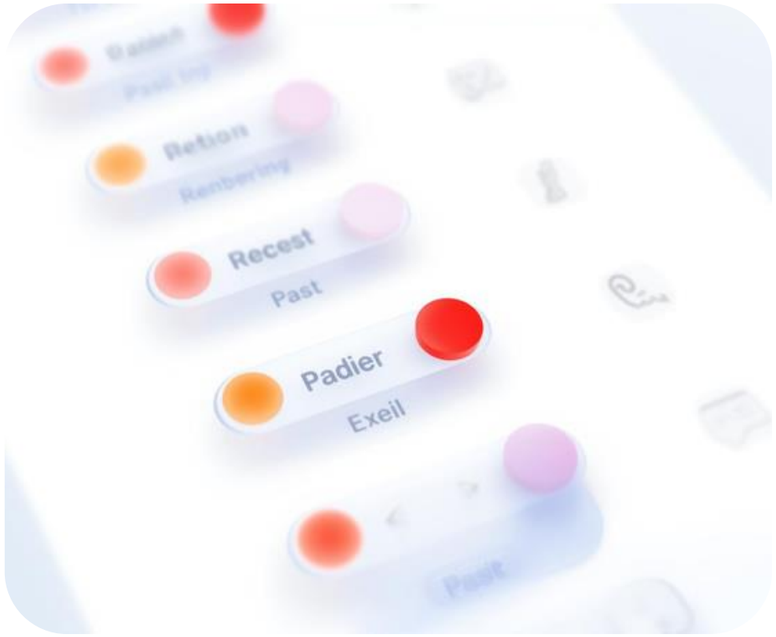


Checkbox

A **Checkbox** enables users to make binary choices.

Radio

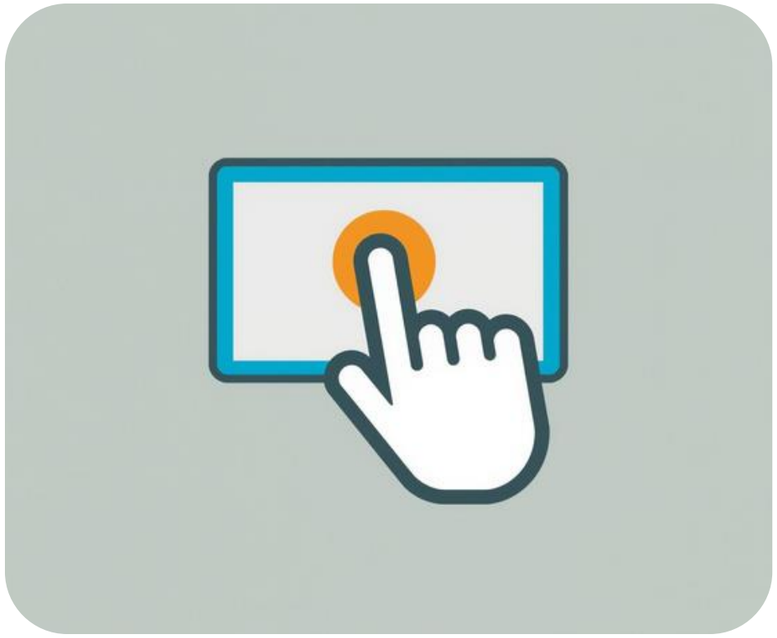
The **Radio** widget presents mutually exclusive options.



Interaction Models in Flutter

GestureDetector

Detects gestures like taps or swipes.



InkWell

Creates a ripple effect on touch.

Navigator

Manages app navigation between screens.

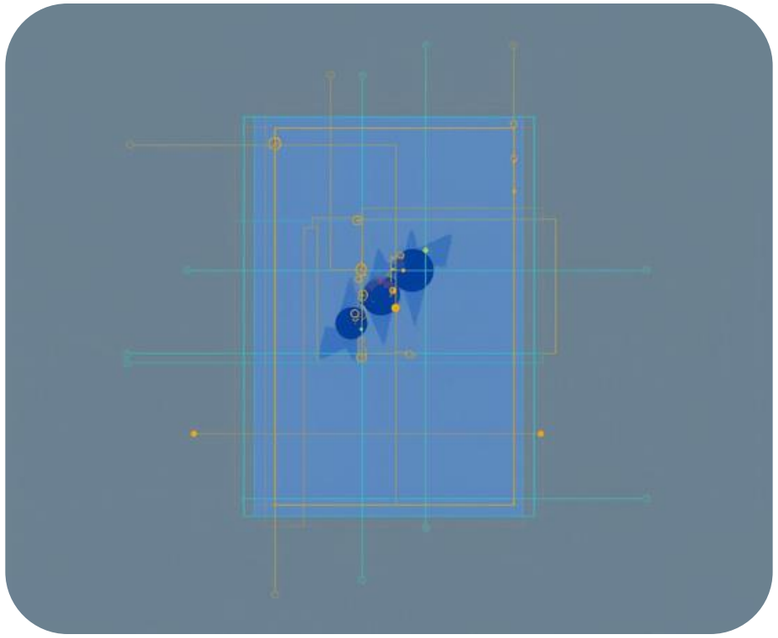
Exploring Layout Widgets

Stack

The Stack widget allows overlapping elements easily.

Align

Align positions widgets according to specified coordinates.



Expanded

The Expanded widget stretches children to fill space.

Understanding Painting & Effects Widgets

Opacity

Adjusts the transparency of a widget.

ClipRRect

Clips the child widget to a rounded rectangle.

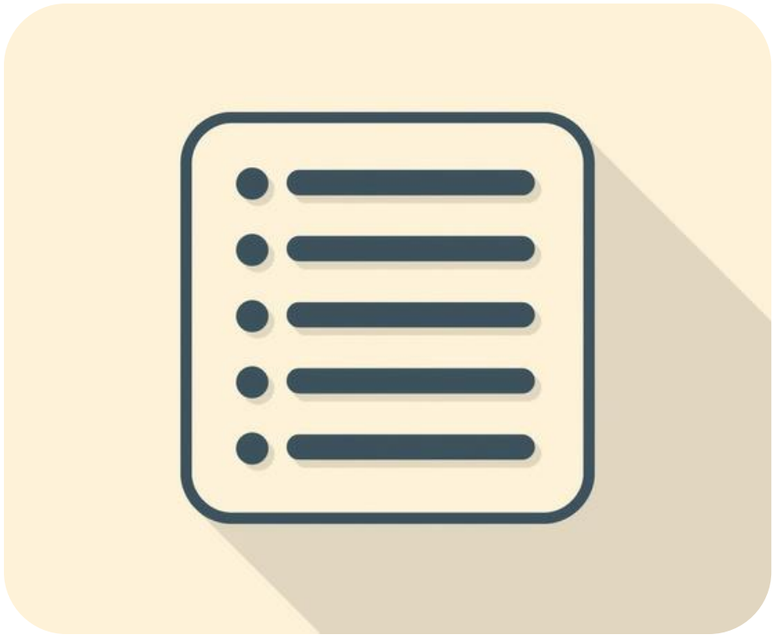
DecoratedBox

Adds visual decoration to a box widget.

Understanding Scrolling Widgets

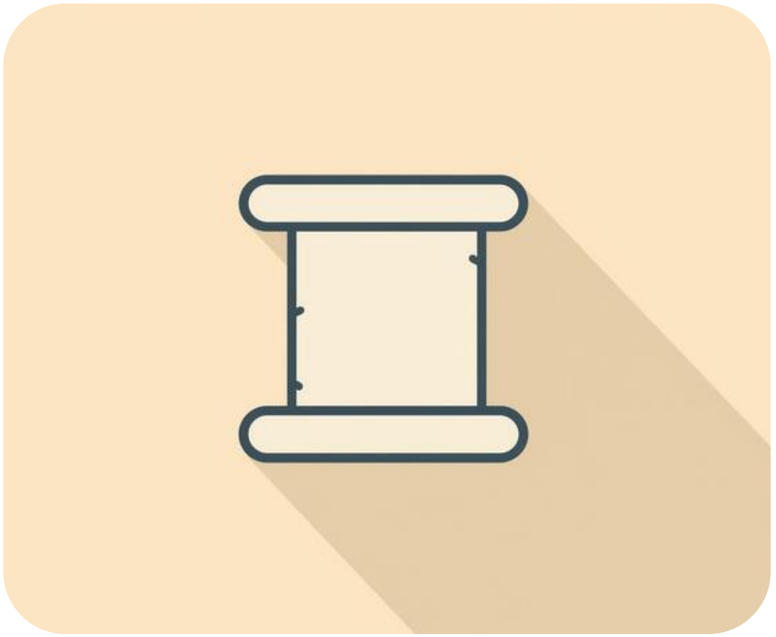
ListView

A scrollable list displaying multiple items efficiently.



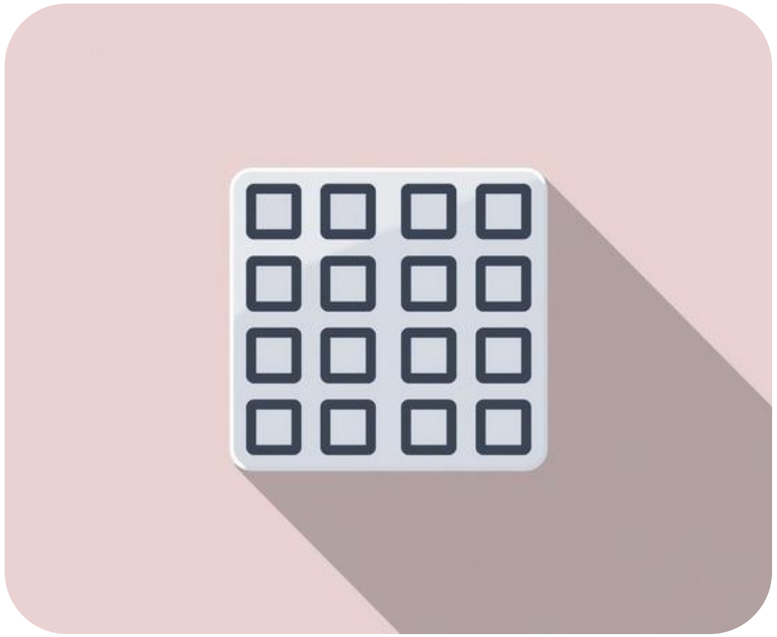
SingleChildScrollView

Enables scrolling for a single widget, enhancing usability.



GridView

Arranges multiple items in a grid layout for accessibility.



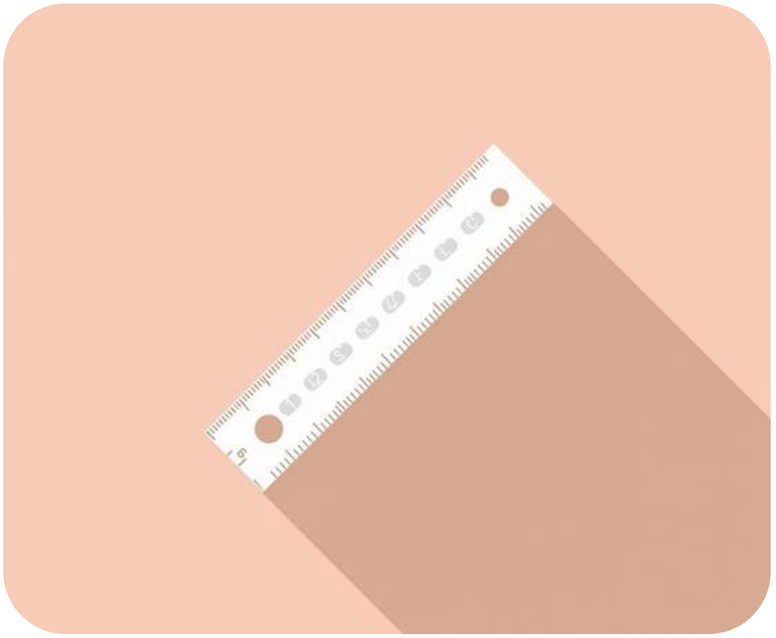
Understanding Styling Widgets

Theme

Theme widget provides consistent styling across the app.

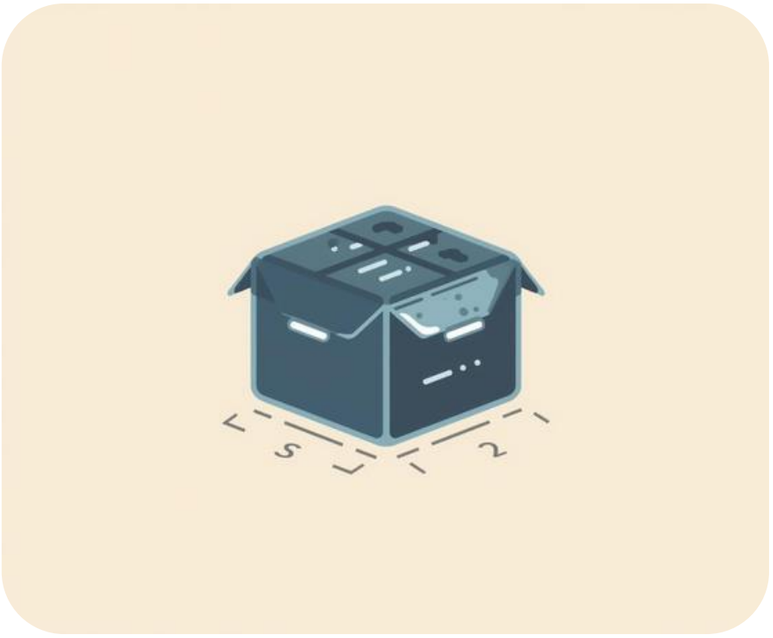
MediaQuery

MediaQuery allows responsive design based on screen size.



SizedBox

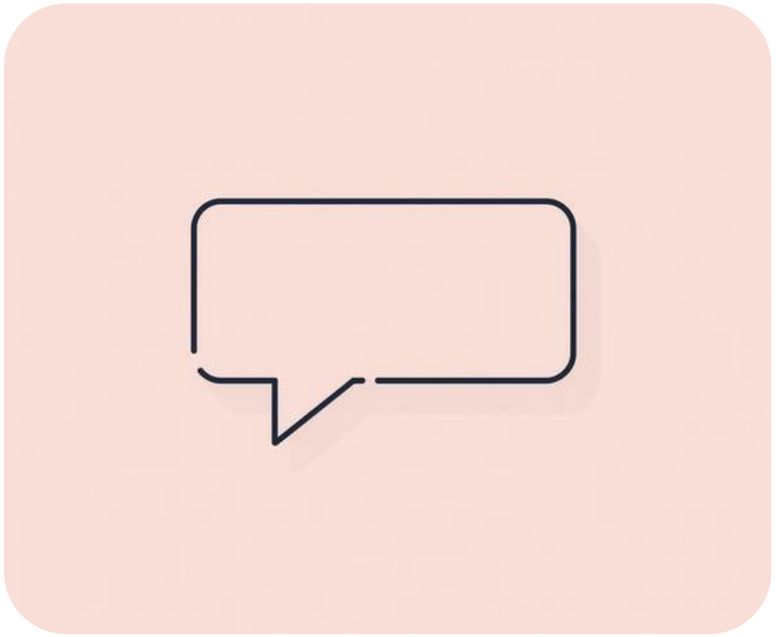
SizedBox is used for adding spacing between elements.



Exploring Text Widgets in Flutter

Text

Displays simple strings, fundamental for presenting content.

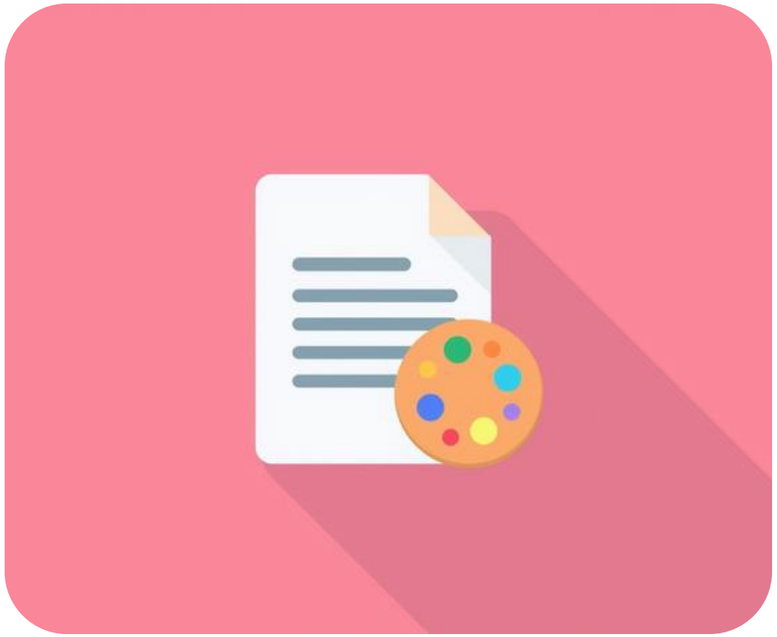


RichText

Allows complex text formatting, integrating multiple styles easily.

DefaultTextStyle

Defines default styling for text widgets in the hierarchy.



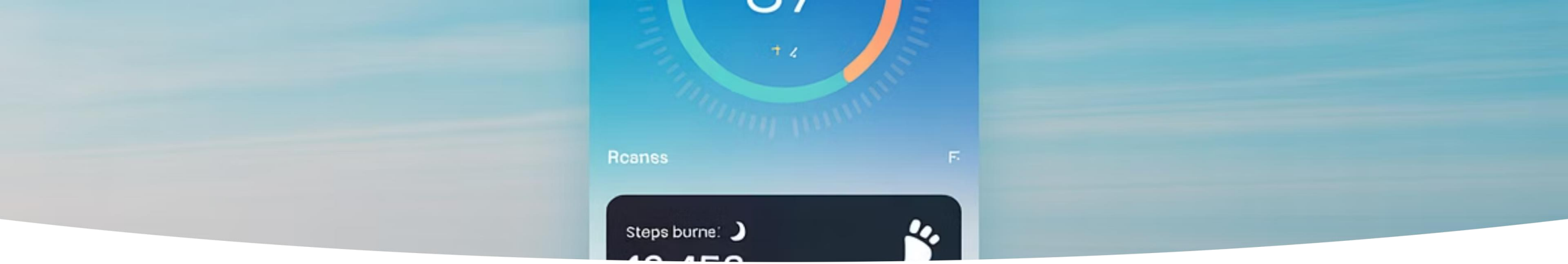
Activity 1 – Brainstorming

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

Core Widget Types

There are broadly two types of widgets in the flutter:

- **Stateless Widget**
- **Stateful Widget**



Stateless Widgets

- Stateless Widget is a type of widget which once built , then it's properties and state can't be changed. These widgets are immutable, once created can't be modified.
- These are used for static content or UI content that don't need a change after time.
- Key Characterstics of Stateless Widgets are: **Immutable , No State and Lightweight.**
- **Examples:** Display Text, Icons, Images, etc.

Stateless Widget Example

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('My First Flutter App')),
        body: Center(
          child: Text('Hello World',style: TextStyle(fontSize: 28)),
        ),
      ),
    );
  }
}
```

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**.

Stateful Widgets

- Stateful Widgets is a type of widget that can change state. It can maintain and update the appearance in the response to change in state.
- These are used for dynamic change in the properties and appearance over the time.
- Key Characterstics of Stateful Widgets are: ***Mutable State*** , ***State Lifecycle*** and ***Dynamic Updates***.
- **Examples**: Buttons, Sliders, Text Fields, etc.

Stateful Widget Example

```
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?"

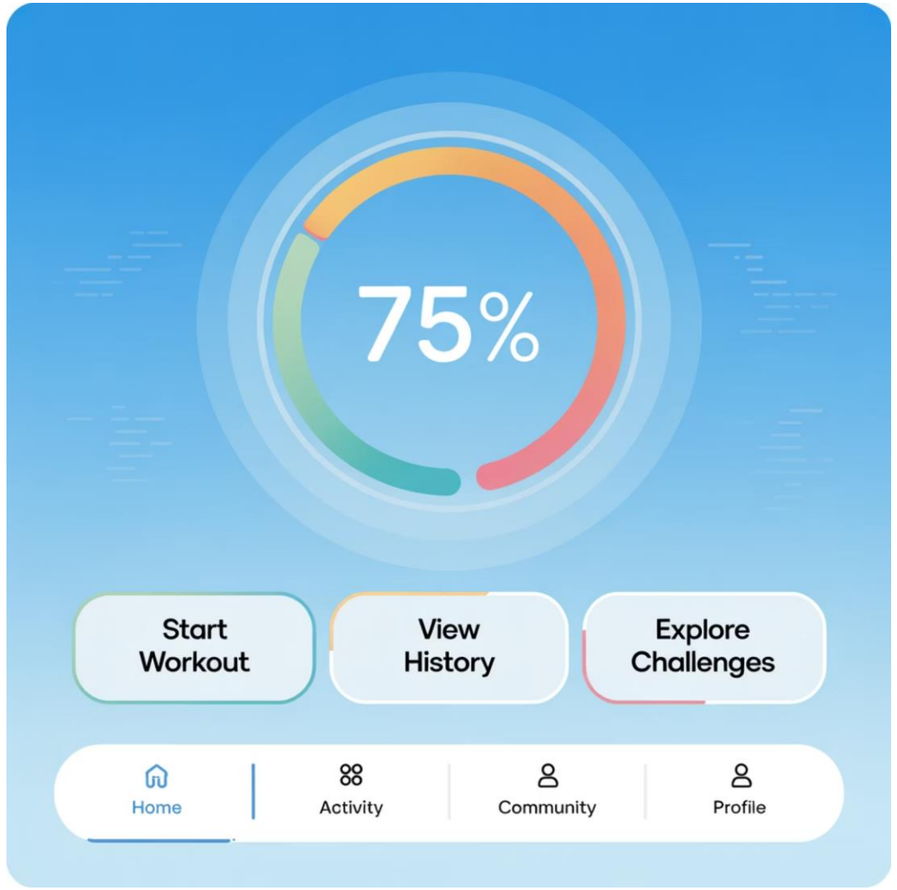
Comparing Stateless & Stateful

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**.

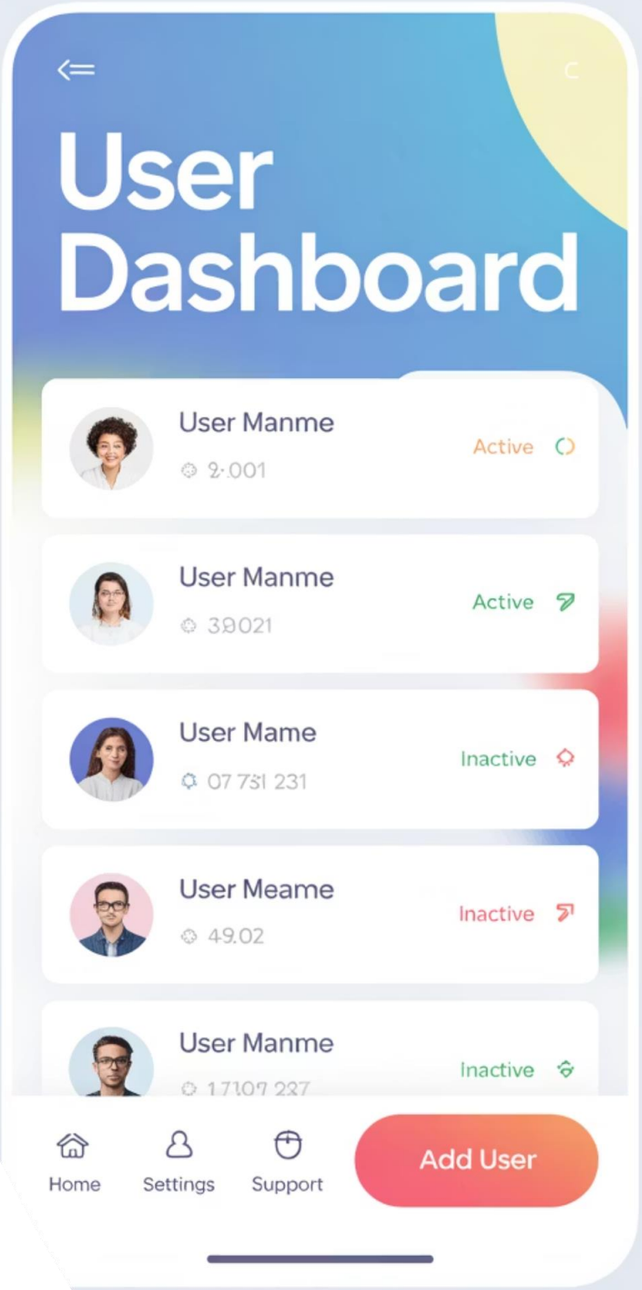
Activity 5 – Group Work

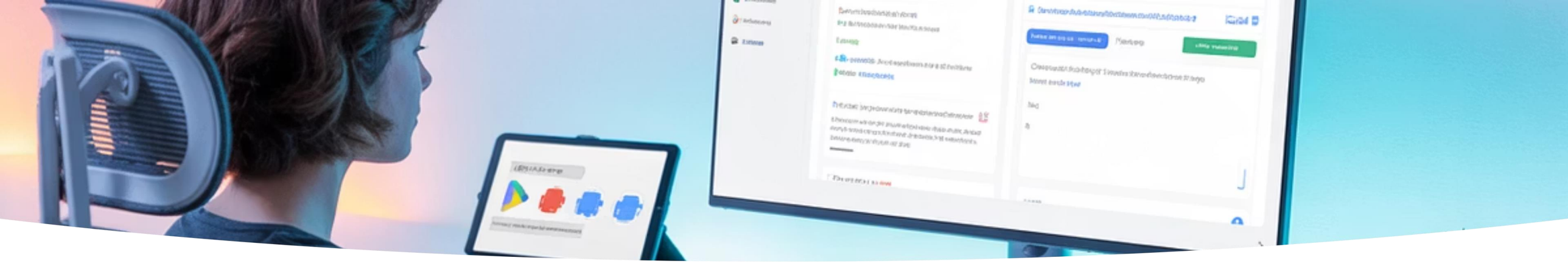
Groups of 4–5:

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.

Advanced Composition Example

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





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.

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