Application Development

Lecture 4

Widgets II : Layouts and Navigation in Flutter

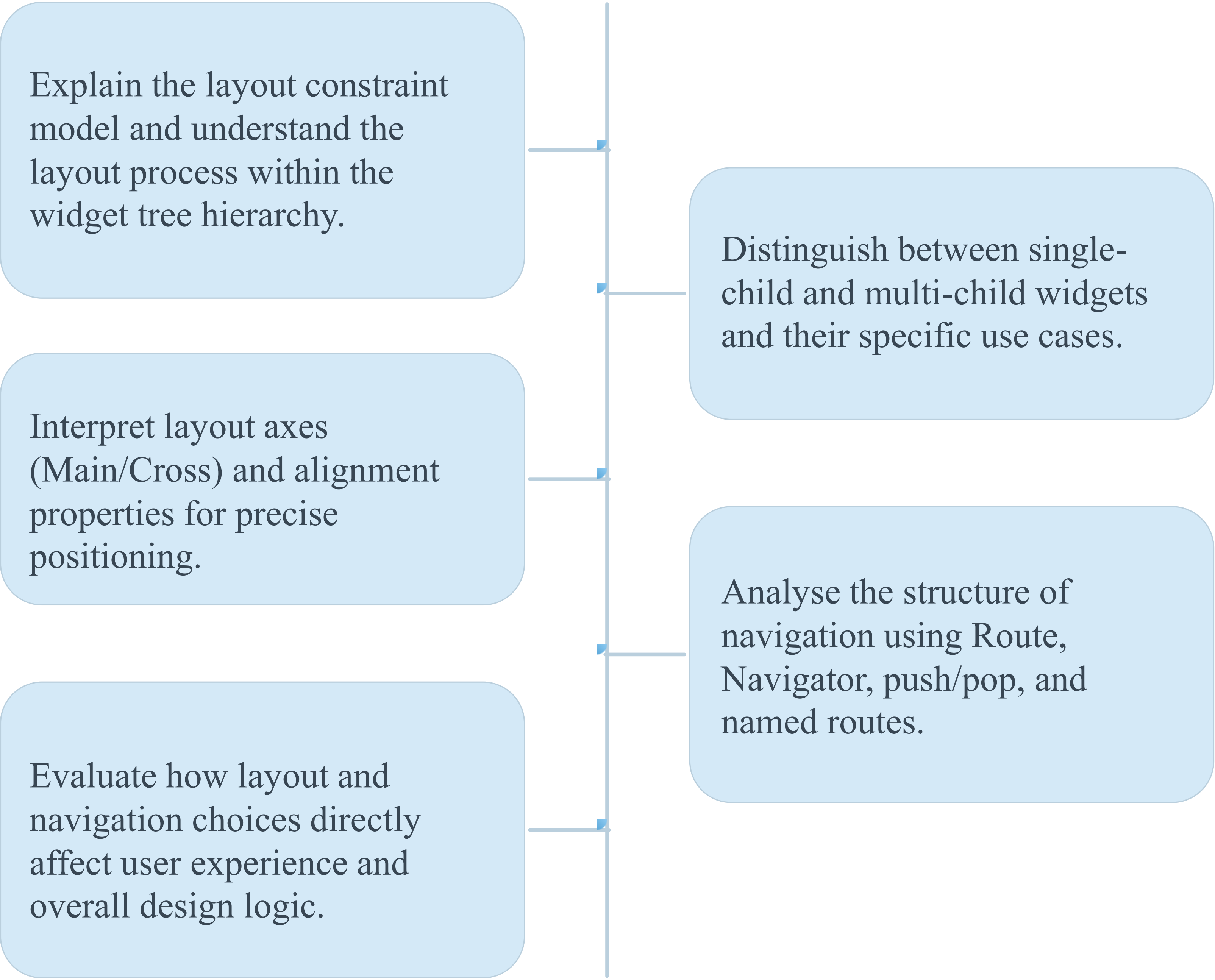
Asst. Lect. Ali Al-khawaja

Class Room

General Objective

To develop a deep theoretical understanding of Flutter layout mechanisms (Row, Column, Stack) and navigation principles (Navigator and Routes).

Behavioural Objectives



Explain the layout constraint model and understand the layout process within the widget tree hierarchy.

Distinguish between single-child and multi-child widgets and their specific use cases.

Interpret layout axes (Main/Cross) and alignment properties for precise positioning.

Analyse the structure of navigation using Route, Navigator, push/pop, and named routes.

Evaluate how layout and navigation choices directly affect user experience and overall design logic.

Lecture Outline

1 **Introduction to Layout and Navigation**

Understanding the fundamental concepts and their importance in Flutter development.

2 **The Layout System in Flutter**

Exploring the constraint-based layout model and widget tree hierarchy.

3 **Row, Column, and Stack Widgets**

Mastering the core layout widgets for arranging UI elements.

4 **Fundamentals of Navigation**

Learning Navigator, Routes, and screen management principles.

5 **Linking Layout with Navigation**

Connecting visual design with user flow and interaction patterns.

6 **Summary and Key Insights**

Consolidating knowledge and preparing for practical implementation.

Why Layout and Navigation?

Layout Defines Appearance

Layout determines **how widgets are organised and sized** on the screen. It establishes visual hierarchy, spacing, and the overall aesthetic structure of your application.

Navigation Defines Flow

Navigation defines **how users move between screens or pages**. It creates the logical pathways and transitions that guide users through your application's features.

Understanding both concepts theoretically helps you design structured, intuitive, and user-friendly applications. The interplay between these two elements forms the foundation of exceptional mobile experiences.

Activity 1: Padlet

Scan the QR code to answer the question

The Layout Constraint Model

- ## Parent Provides Constraints

  The **parent widget** passes layout constraints to its child, specifying minimum and maximum width and height boundaries.

- ## Child Chooses Size

  The **child widget** selects a size that fits within those constraints, respecting the boundaries provided by its parent.

- ## Parent Positions Child

  The **parent** then positions the child within its own coordinate space, completing the layout cycle.

Activity 2: Mentimeter

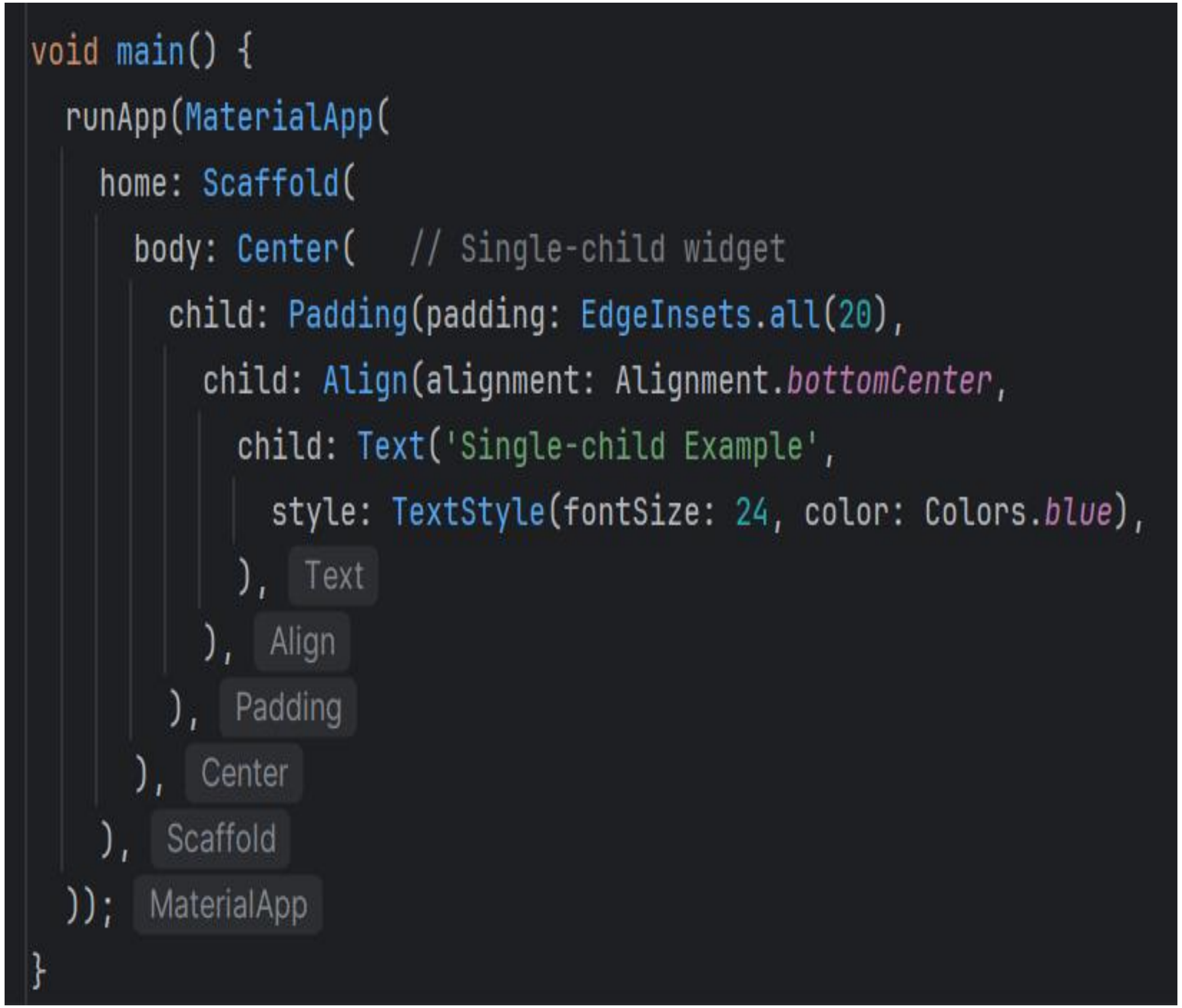
Scan the QR code to answer the question

Single-child vs Multi-child Widgets

Single-child Widgets

Handle **only one element** at a time. These widgets focus on positioning, padding, and alignment of individual components.

- **Center:** Centres its child within itself
- **Padding:** Adds space around its child
- **Align:** Positions child at specific alignment

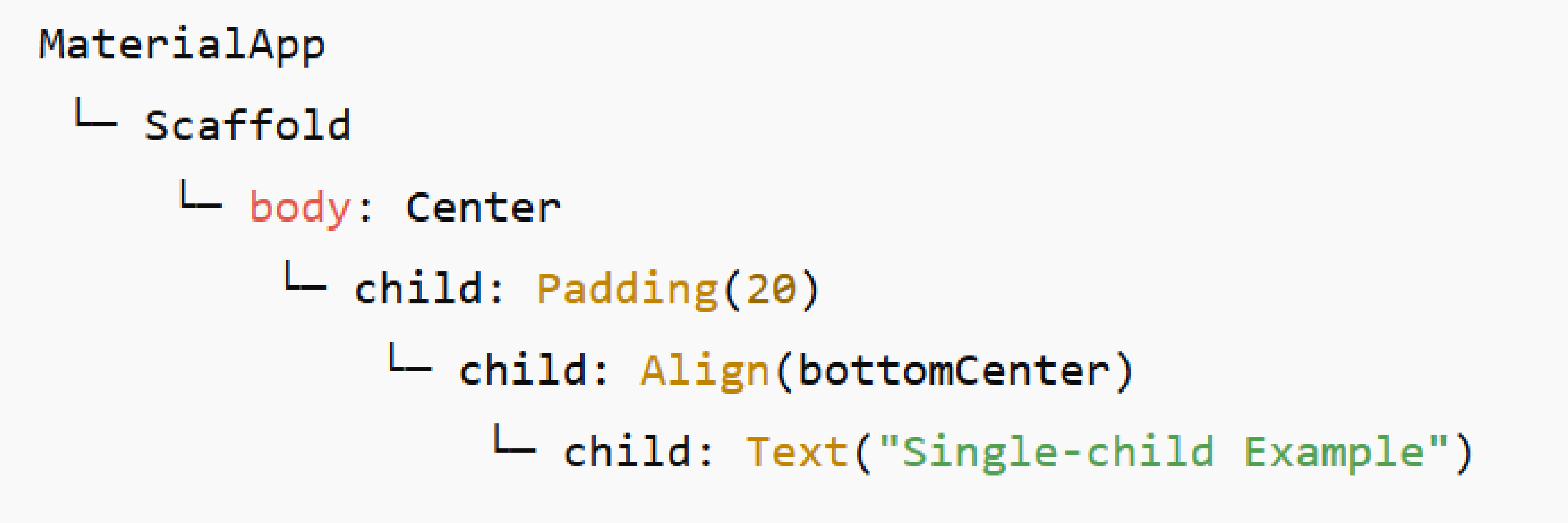


```
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      body: Center(   // Single-child widget
        child: Padding(padding: EdgeInsets.all(20),
          child: Align(alignment: Alignment.bottomCenter,
            child: Text('Single-child Example',
              style: TextStyle(fontSize: 24, color: Colors.blue),
            ), Text
          ), Align
        ), Padding
      ), Center
    ), Scaffold
  )); MaterialApp
}
```

Single-child vs Multi-child Widgets

Single-child Widgets

Widget Tree



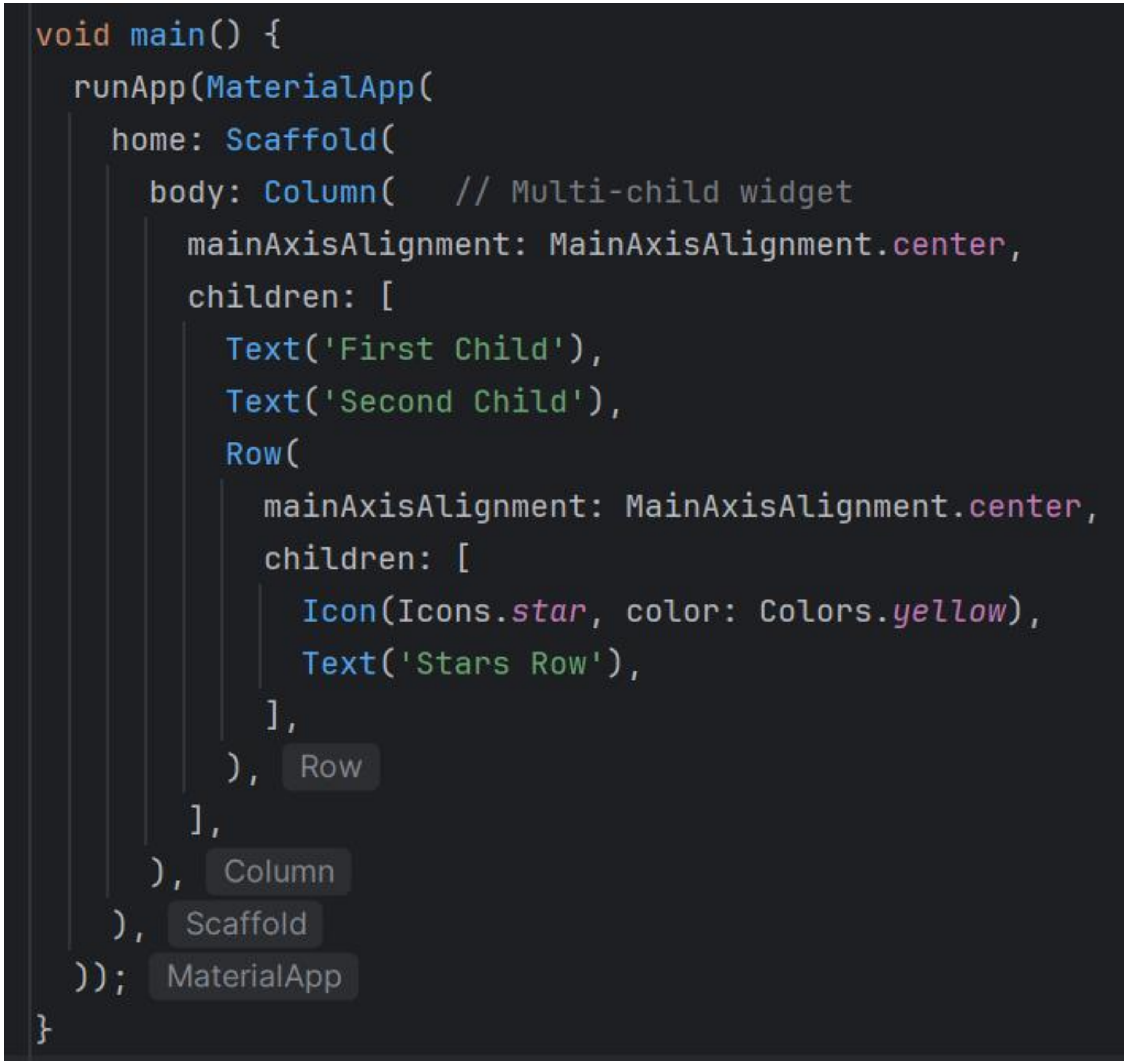
```
MaterialApp
 └─ Scaffold
     └─ body: Center
         └─ child: Padding(20)
             └─ child: Align(bottomCenter)
                 └─ child: Text("Single-child Example")
```

Single-child vs Multi-child Widgets

Multi-child Widgets

Handle **multiple elements** simultaneously, defining layout rules and arrangement patterns for all children.

- **Row:** Horizontal arrangement
- **Column:** Vertical arrangement
- **Stack:** Overlapping layers

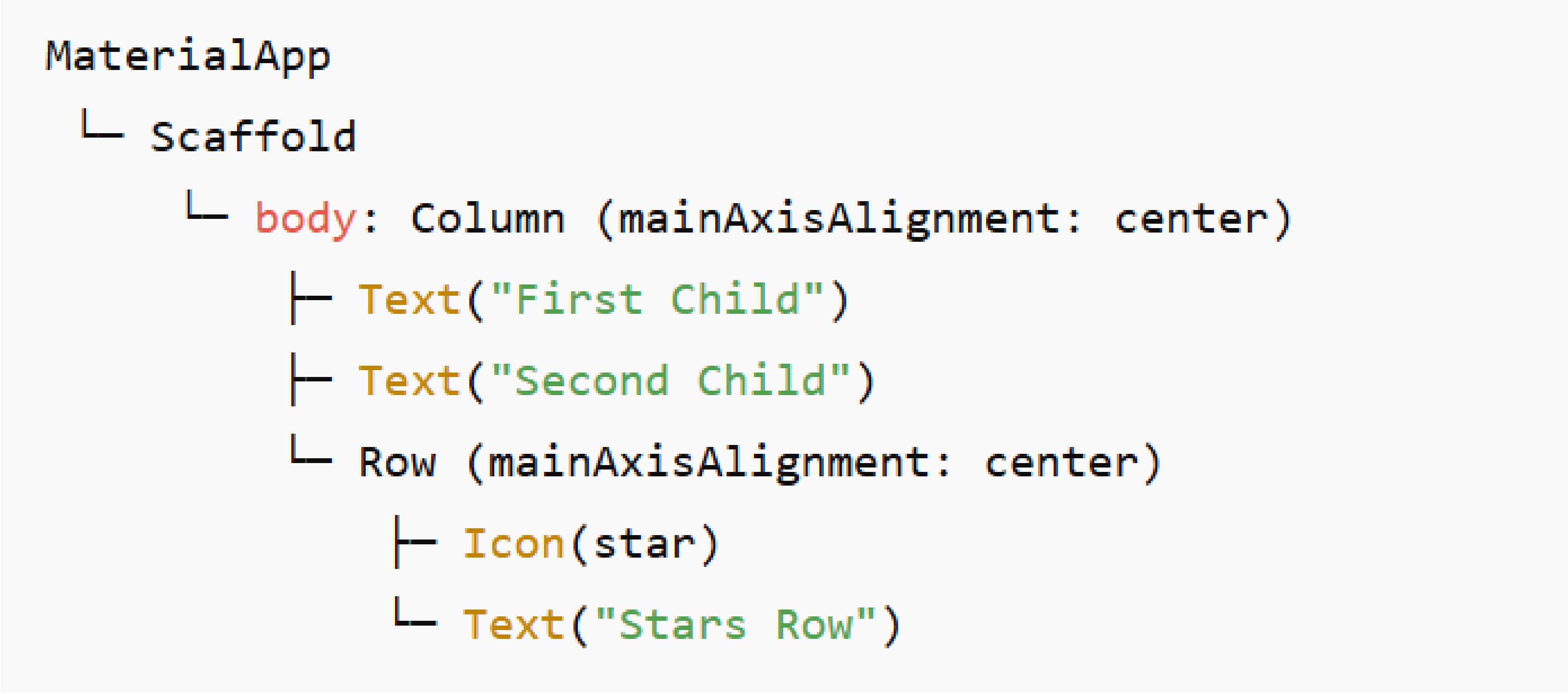


```
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      body: Column(   // Multi-child widget
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text('First Child'),
          Text('Second Child'),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Icon(Icons.star, color: Colors.yellow),
              Text('Stars Row'),
            ],
          ), Row
        ],
      ), Column
    ), Scaffold
  )); MaterialApp
}
```

Single-child vs Multi-child Widgets

Multi-child Widgets

Widget Tree



```
MaterialApp
 └─ Scaffold
     └─ body: Column (mainAxisAlignment: center)
         ├─ Text("First Child")
         ├─ Text("Second Child")
         └─ Row (mainAxisAlignment: center)
             ├─ Icon(star)
             └─ Text("Stars Row")
```

Row and Column: Axes, Alignment, and Flex

Row Widget

Arranges widgets **horizontally** along the main axis (left to right or right to left).

Column Widget

Arranges widgets **vertically** along the main axis (top to bottom).

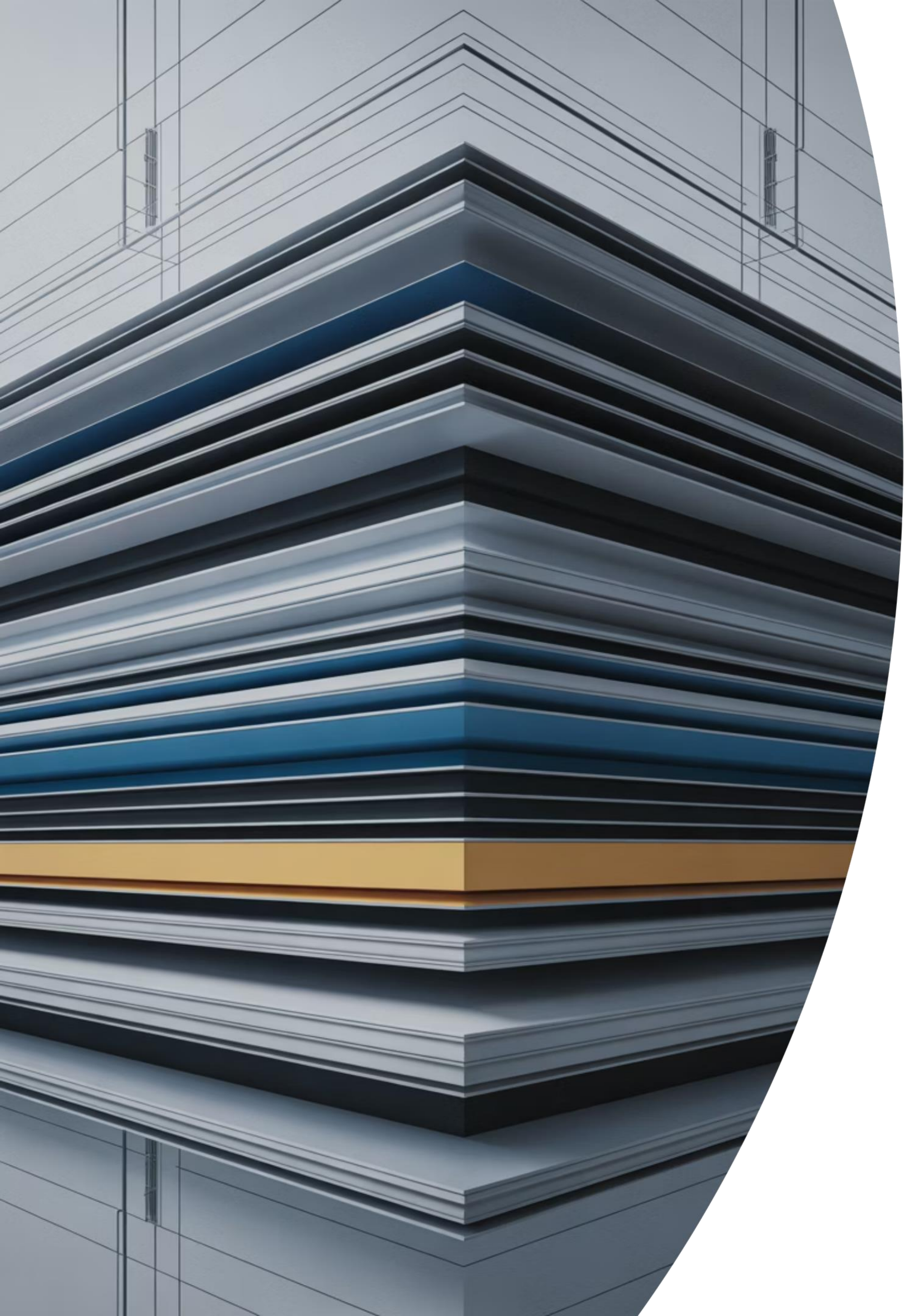
Important Properties

- **mainAxisAlignment:** Controls spacing along the primary axis (start, centre, spaceBetween, spaceAround, spaceEvenly, end)
- **crossAxisAlignment:** Controls alignment perpendicular to main axis (start, centre, stretch, end)
- **Expanded / Flexible:** Distribute remaining space proportionally amongst children using flex values

Example: If text overflows in a narrow Row, solutions include wrapping the text in Flexible, shortening the content, or enabling scrolling functionality.

Activity 3: Hand-Raising

If you want to place text on top of an image, which widget would you use and why?



Stack: Overlapping and Design Considerations

What is Stack?

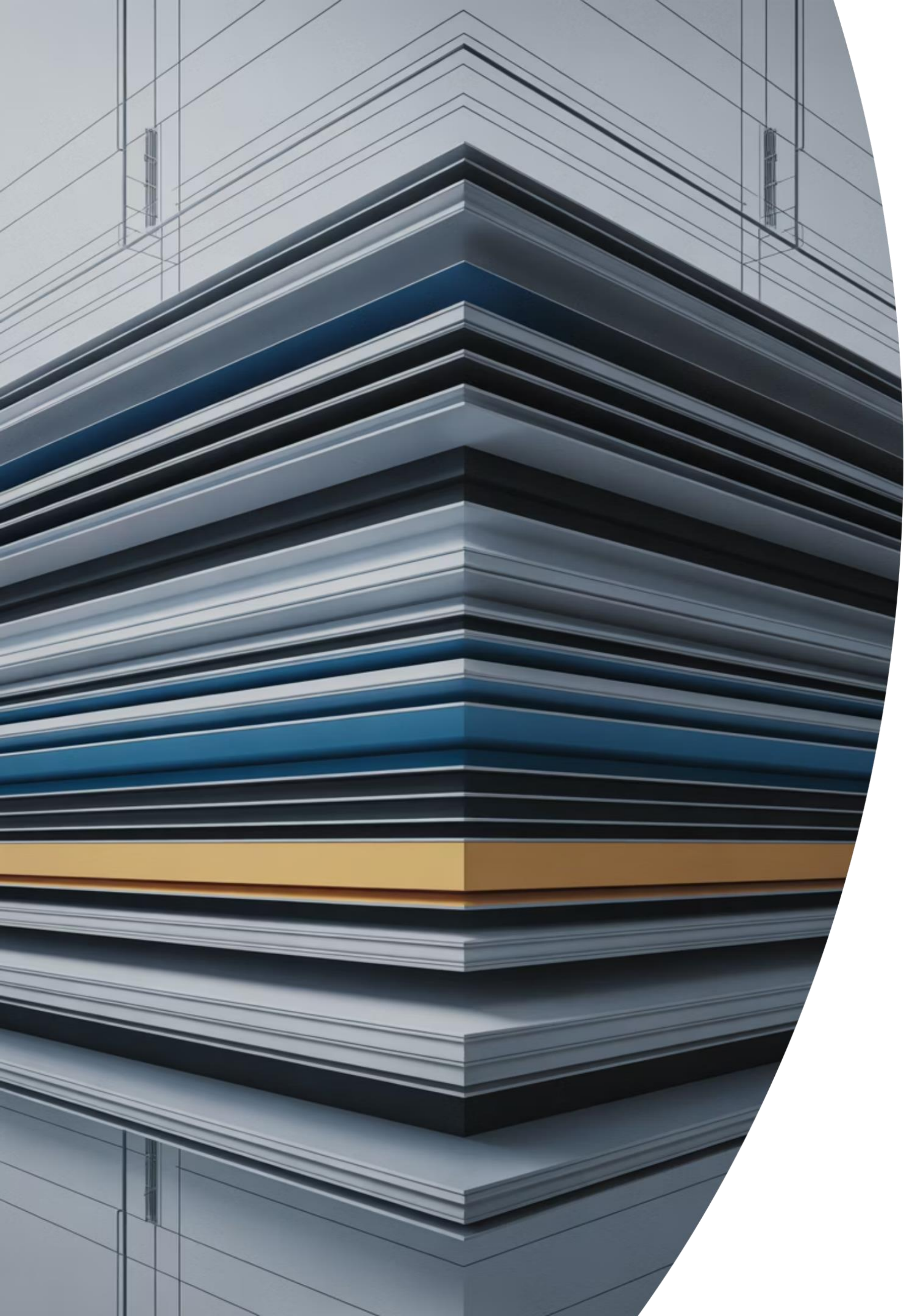
Stack arranges widgets *on top of each other* along the Z-axis, creating layers. The first child appears at the bottom, with subsequent children stacked above.

Common Use Cases

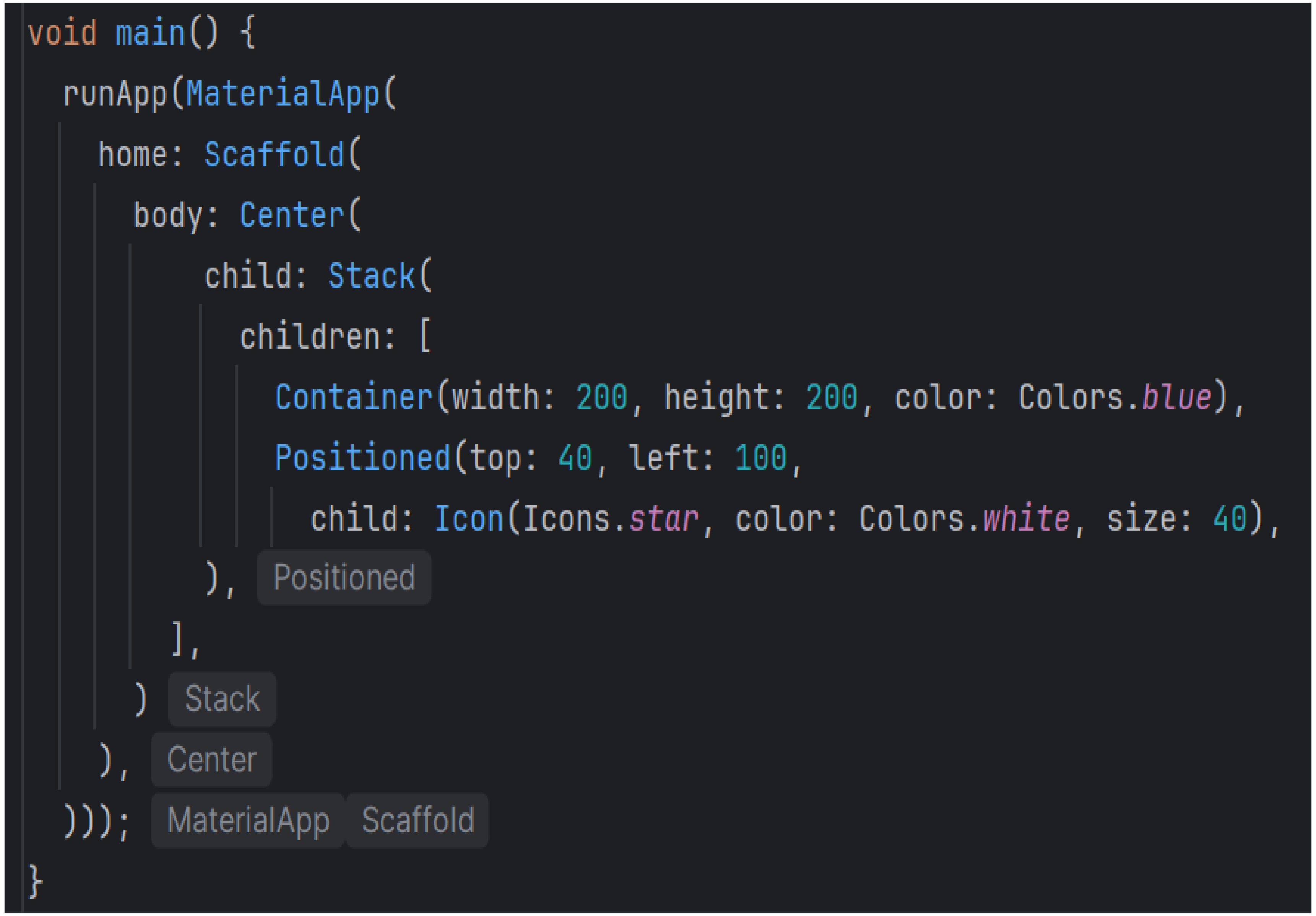
- Displaying text overlays on images
- Creating notification badges or status indicators
- Building custom buttons with complex backgrounds
- Implementing card-based designs with overlapping elements

The Positioned Widget

Use **Positioned** to define exact placement of children within the Stack using properties like top, bottom, left, and right.



Stack: Overlapping and Design Considerations (Example)



```
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      body: Center(
          child: Stack(
            children: [
              Container(width: 200, height: 200, color: Colors.blue),
              Positioned(top: 40, left: 100,
                child: Icon(Icons.star, color: Colors.white, size: 40),
          ), Positioned
        ],
      ) Stack
    ), Center
  ))); MaterialApp Scaffold
}
```

Activity 4: using Paper & pen

Design Challenge

Grab a piece of paper and sketch a layout that includes:

1. A title at the top
2. An image in the centre
3. A button at the bottom

Then explain: Would you use a Column, Stack, or combination (Column + Expanded)? Why is your choice the most appropriate solution?

Fundamentals of Navigation

Navigator

Manages a **stack of screens (Routes)**. Think of it as a pile of cards where only the top card is visible to the user at any given moment.

push() Method

Adds a new screen to the top of the stack. The new screen slides into view, covering the previous screen whilst keeping it in memory.

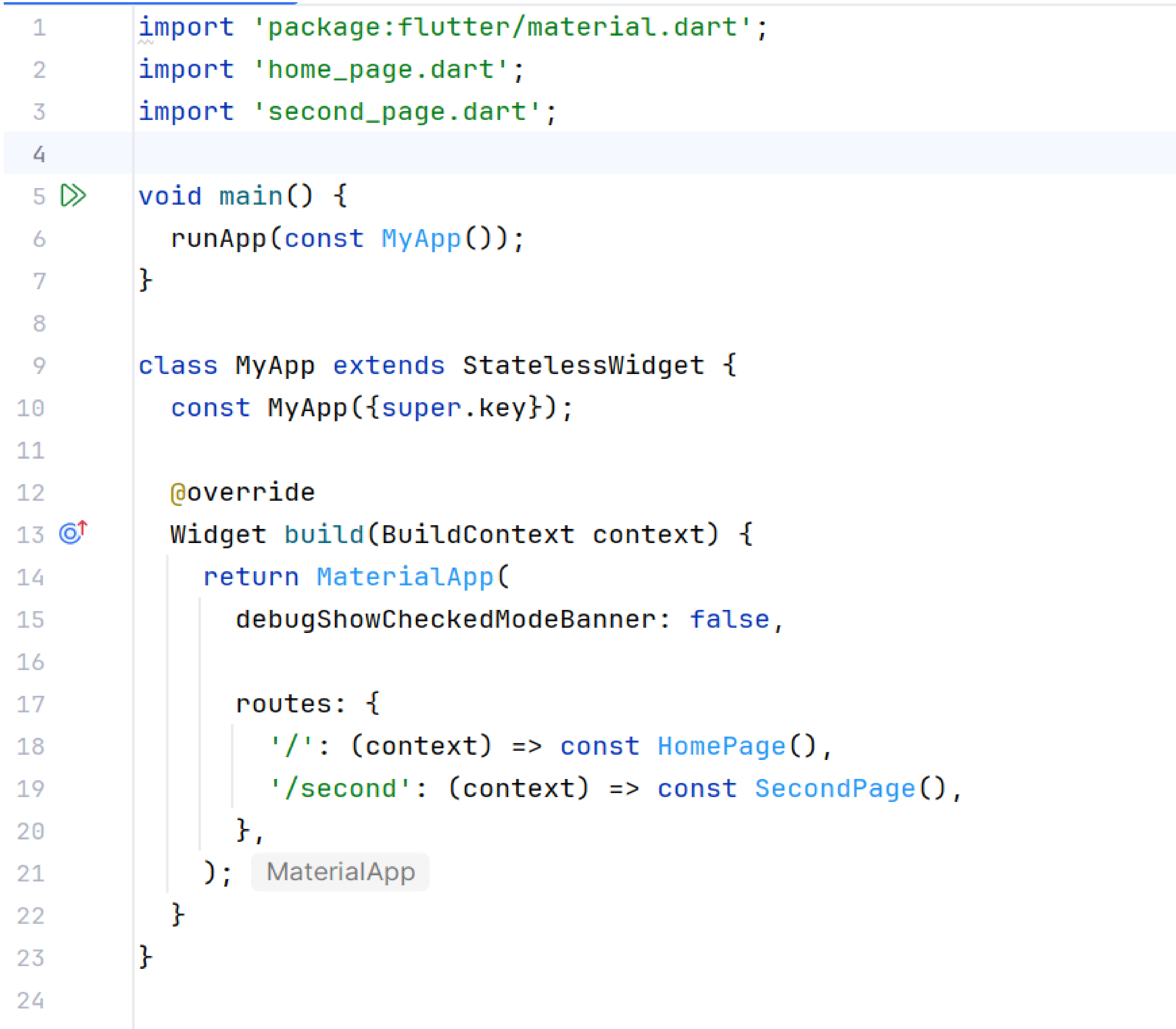
pop() Method

Removes the current screen and returns to the previous one. The top screen slides away, revealing the screen beneath it.

Named Routes

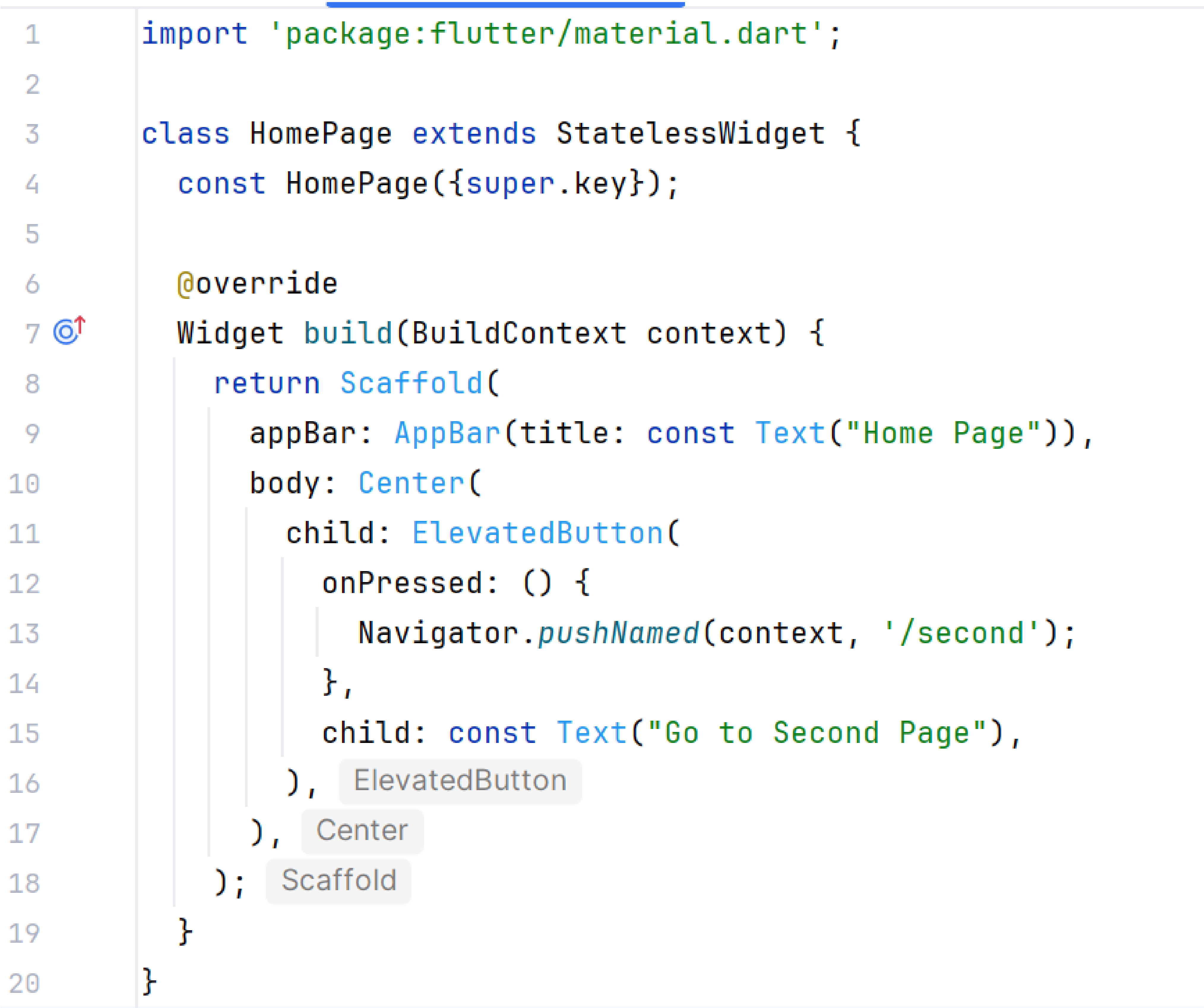
Predefined identifiers for screens that enable **cleaner, more maintainable navigation**. Like street addresses for your app screens.

Navigator.dart file



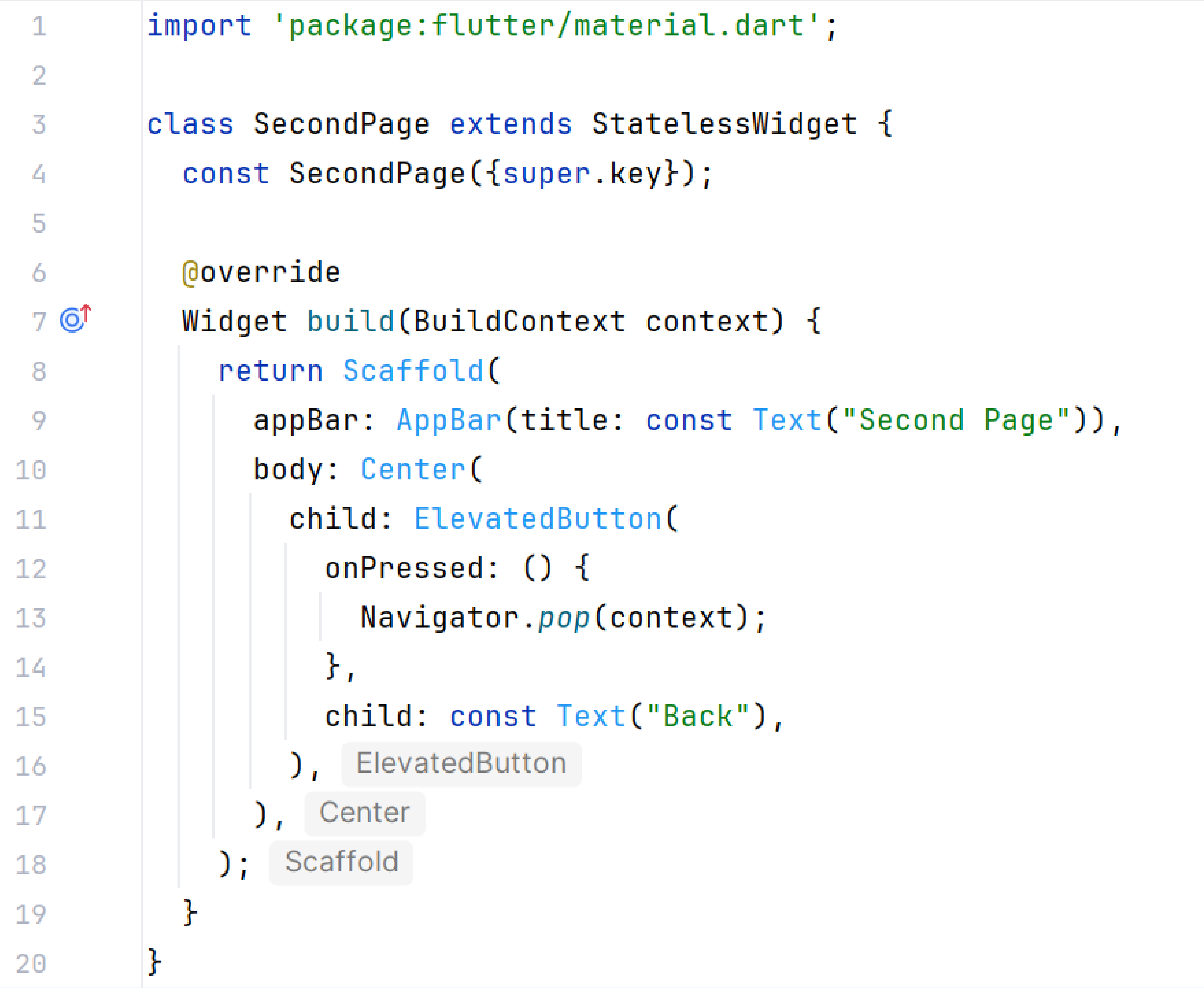
```dart
import 'package:flutter/material.dart';
import 'home_page.dart';
import 'second_page.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,

      routes: {
        '/': (context) => const HomePage(),
        '/second': (context) => const SecondPage(),
      },
    );  MaterialApp
  }
}

```

home_page.dart file



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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Home Page")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pushNamed(context, '/second');
          },
          child: const Text("Go to Second Page"),
        ),
      ),
    );
  }
}
```

second_page.dart



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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Second Page")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text("Back"),
        ), ElevatedButton
      ), Center
    ); Scaffold
  }
}
```

Activity 5: Group Discussion

Class Division: Groups of 4 Students

> **Task: Compare Navigator.push() versus Navigator.pushNamed() from a theoretical perspective.**

1

Project Organisation

Which approach is more organised in large projects with dozens of screens?

2

Maintainability

Which method improves code maintainability and readability over time?

3

Testing and Scalability

How does each method affect testing capabilities and application scalability?

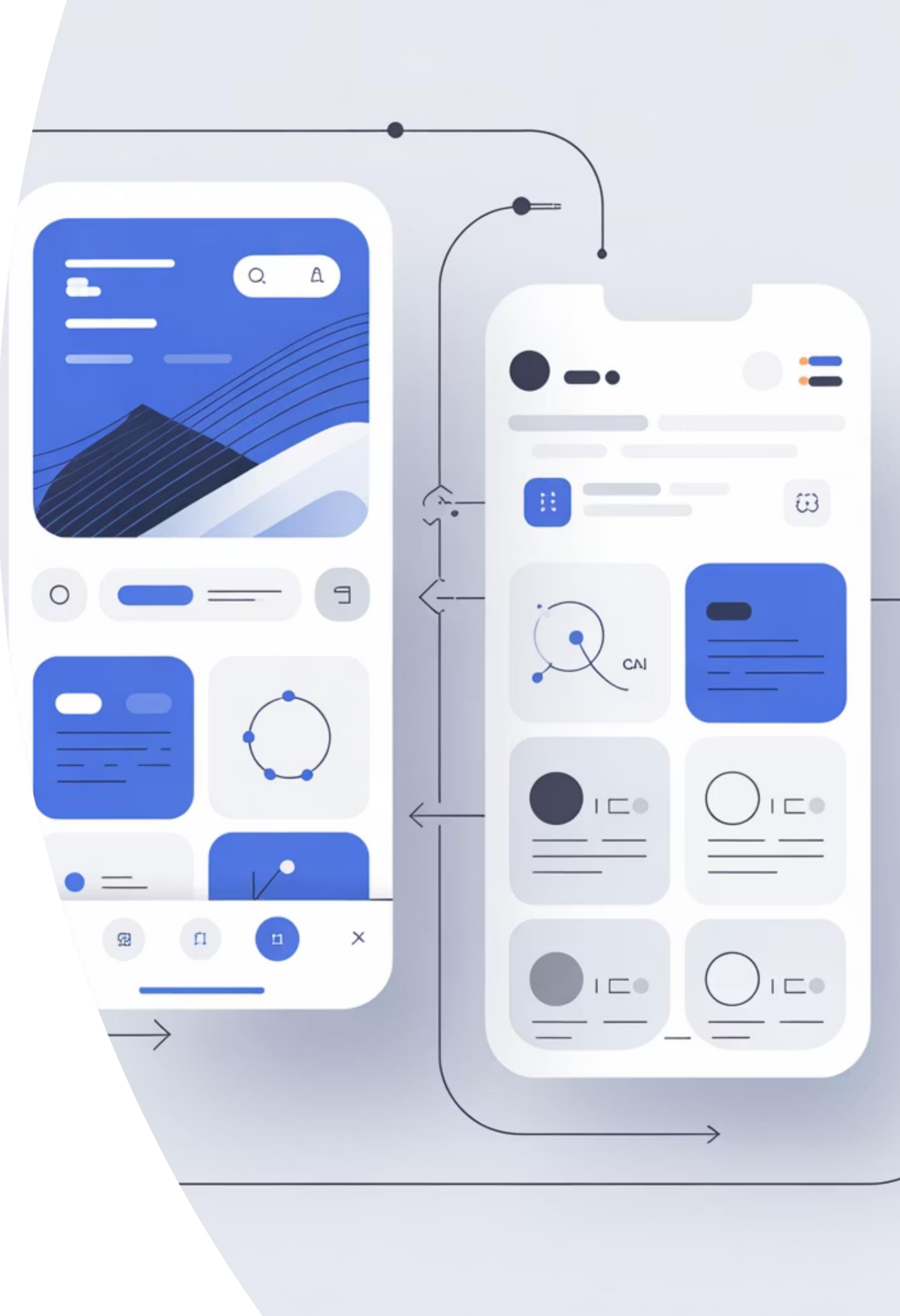
Discuss within your groups for 8-10 minutes, then we'll share insights with the entire class.

Linking Layout and Navigation

Layout determines **interaction points**—buttons, cards, lists, and gestures—that trigger navigation events. These two concepts are deeply interconnected in creating seamless user experiences.

Planning Principle

Good navigation planning requires defining route relationships *before implementation*. Map out your screens and their connections on paper first.



Activity 6: Classroom Homework

Assignment Requirements

- Write a text-based outline for a Flutter application containing three screens: Home, Details, and About.
- For each screen, specify the layout type you would use (Row, Column, or Stack) and provide a short theoretical justification for your choice.
- Describe five consecutive navigation transitions (push/pop) and explain the state of the Navigator stack after each step.
- Write a short essay (8–10 lines) comparing Expanded and Flexible from a theoretical perspective — include their roles, differences, and when each should be used.

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