World of Flutter widgets!

Understanding the Structure of Each Widget Explanation

For each widget, I will follow this format:

- **Widget Name**
  - **Purpose:** What the widget is used for.
  - **Key Properties & Customization:** A detailed explanation of its most common and important properties, including how they customize the widget's appearance or behavior.
  - **Example Implementation:** A self-contained, runnable Flutter code snippet. This snippet will be a `StatelessWidget` or `StatefulWidget` that you can directly use as the `home` property of a `MaterialApp`.

Let's begin!

I. Application Structure & Root Widgets

These widgets form the very foundation and overall structure of your Flutter application.

---

1. MaterialApp

- **Purpose:** The root widget for a Material Design application. It sets up the basic app structure, theme, navigation, and other global configurations. You typically have only one `MaterialApp` in your entire app.

- **Key Properties & Customization:**

  - `home`: The widget that appears first on the screen when the app starts. This is usually a `Scaffold`.
  - `title`: A description of the application for the device's task switcher (e.g., when you see a list of open apps).
  - `theme`: Defines the visual theme (colors, fonts, etc.) for the entire app. It takes a `ThemeData` object.
    - `primarySwatch`: A `MaterialColor` that is used to generate a set of 10 shades of the color, ranging from light to dark. This is often the main brand color.
    - `appBarTheme`: Customizes the appearance of `AppBar` widgets throughout the app.
    - `textTheme`: Defines the text styles for different text elements in the app.
  - `debugShowCheckedModeBanner`: A boolean that controls whether the "DEBUG" banner is shown in debug mode. Set to `false` for release builds or cleaner development views.
  - `routes`: A table of named routes for navigating within the application.
  - `initialRoute`: The name of the first route to show.

- **Example Implementation:**

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

```

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // title: A description for the device's task switcher
      title: 'Flutter Widgets Demo',
      // theme: Defines the overall visual theme of the app
      theme: ThemeData(
        primarySwatch: Colors.blue, // Sets a primary color and its shades
        appBarTheme: const AppBarTheme(
          backgroundColor: Colors.indigo, // Custom AppBar background
          foregroundColor: Colors.white, // Custom AppBar text/icon color
        ),
        textTheme: const TextTheme(
          bodyMedium: TextStyle(fontSize: 16.0, color: Colors.black87),
          headlineSmall: TextStyle(fontSize: 24.0, fontWeight:
FontWeight.bold),
        ),
        useMaterial3: true, // Opt-in to Material 3 design
      ),
      // home: The first widget displayed when the app starts
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Home Screen'),
        ),
        body: const Center(
          child: Text(
            'Welcome to the Flutter Widgets App!',
            style: TextStyle(fontSize: 20),
          ),
        ),
      ),
      // debugShowCheckedModeBanner: Hides the debug banner in the top right
corner
      debugShowCheckedModeBanner: false,
    );
  }
}
```

---

2. Scaffold

- **Purpose:** Implements the basic Material Design visual structure. It provides slots for common UI elements like an `AppBar`, `body`, `FloatingActionButton`, `BottomNavigationBar`, `Drawer`, etc.
- **Key Properties & Customization:**

  - `appBar`: A horizontal bar at the top of the screen. Takes an `AppBar` widget.
  - `body`: The primary content of the screen. Typically a `Center`, `Column`, `Row`, `ListView`, etc.
  - `backgroundColor`: The background color of the scaffold itself.
  - `floatingActionButton`: A button that floats above the content, usually in the bottom-right corner. Takes a `FloatingActionButton` widget.
  - `floatingActionButtonLocation`: Defines where the `floatingActionButton` should be placed (e.g., `FloatingActionButtonLocation.endFloat`).
  - `bottomNavigationBar`: A bar at the bottom of the screen for navigating between a small number of views (2-5). Takes a `BottomNavigationBar` widget.
  - `drawer`: A panel that slides in from the left, typically containing navigation links. Takes a `Drawer` widget.
  - `endDrawer`: Similar to `drawer`, but slides in from the right.
  - `snackBar`: A lightweight message displayed at the bottom of the screen. Managed via `ScaffoldMessenger.of(context).showSnackBar()`.

- **Example Implementation:**

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

class ScaffoldExample extends StatefulWidget {
  const ScaffoldExample({super.key});

  @override
  State<ScaffoldExample> createState() => _ScaffoldExampleState();
}

class _ScaffoldExampleState extends State<ScaffoldExample> {
  int _selectedIndex = 0; // For BottomNavigationBar

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // appBar: The top bar of the screen
      appBar: AppBar(
        title: const Text('Scaffold Widget Example'),
        leading: IconButton( // A leading widget (e.g., menu icon)
          icon: const Icon(Icons.menu),
          onPressed: () {
            // Open drawer
            Scaffold.of(context).openDrawer();
          },
        ),
        actions: [ // Widgets on the right side of the AppBar
          IconButton(
            icon: const Icon(Icons.search),
            onPressed: () {
              // Handle search action
            },
          ),
          IconButton(
            icon: const Icon(Icons.more_vert),
            onPressed: () {
              // Handle more options
            },
          ),
        ],
      ),
      // body: The primary content area of the screen
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              'Selected Index: $_selectedIndex',
              style: Theme.of(context).textTheme.headlineSmall,
            ),
            const SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                // Show a SnackBar
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: const Text('Hello from SnackBar!'),
                    action: SnackBarAction(
                      label: 'Dismiss',
                      onPressed: () {
                        // Code to undo the change or dismiss
                      },
                    ),
                  ),
                );
              },
              child: const Text('Show SnackBar'),
            ),
          ],
        ),
      ),
      // floatingActionButton: A button that floats above the content
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          // Action to perform when FAB is pressed
          print('FAB Pressed!');
        },
        backgroundColor: Colors.green,
        child: const Icon(Icons.add),
      ),
      // floatingActionButtonLocation: Position of the FAB
      floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,
      // bottomNavigationBar: A bar at the bottom for navigation
```

```
      bottomNavigationBar: BottomNavigationBar(
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: 'Home',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.business),
            label: 'Business',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.school),
            label: 'School',
          ),
        ],
        currentIndex: _selectedIndex,
        selectedItemColor: Colors.amber[800],
        onTap: _onItemTapped,
      ),
      // drawer: A panel that slides in from the left
      drawer: Drawer(
        child: ListView(
          padding: EdgeInsets.zero,
          children: <Widget>[
            const DrawerHeader(
              decoration: BoxDecoration(
                color: Colors.blue,
              ),
              child: Text(
                'Drawer Header',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: 24,
                ),
              ),
            ),
            ListTile(
              leading: const Icon(Icons.message),
              title: const Text('Messages'),
              onTap: () {
                Navigator.pop(context); // Close the drawer
              },
            ),
            ListTile(
              leading: const Icon(Icons.account_circle),
              title: const Text('Profile'),
              onTap: () {
                Navigator.pop(context);
              },
            ),
          ],
        ),
      ),
      // backgroundColor: Custom background color for the Scaffold
      backgroundColor: Colors.grey[100],
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ScaffoldExample()));
// }
```

---

3. AppBar

- **Purpose:** A Material Design app bar. Typically displayed at the top of a `Scaffold`. It provides a consistent place for the screen's title, navigation buttons, and actions.
- **Key Properties & Customization:**
  - `title`: The primary widget displayed in the app bar (usually a `Text` widget).
  - `backgroundColor`: The background color of the app bar. Overrides `ThemeData.appBarTheme.backgroundColor`.
  - `foregroundColor`: The color for the app bar's contents (icons, text).
  - `leading`: A widget to display before the title (e.g., a menu icon, back button).
  - `actions`: A list of widgets to display after the title (e.g., action buttons, search icon). Takes a `List<Widget>`.
  - `elevation`: The z-coordinate at which to place this app bar relative to its parent. Controls the shadow beneath the app bar.
  - `centerTitle`: A boolean that controls whether the title is centered. Defaults to `false` for Android and `true` for iOS.
  - `toolbarHeight`: The height of the toolbar component of the AppBar.
  - `bottom`: A widget displayed at the bottom of the app bar, usually a `TabBar`.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('AppBar Example'),
        backgroundColor: Colors.deepPurple, // Custom background color
        foregroundColor: Colors.white,       // Custom foreground color
(text, icons)
        leading: IconButton(                 // Leading widget
          icon: const Icon(Icons.arrow_back),
          onPressed: () {
            Navigator.of(context).pop(); // Example: navigate back
          },
        ),
        actions: [                        // List of action widgets
          IconButton(
            icon: const Icon(Icons.settings),
            onPressed: () {
              print('Settings tapped!');
            },
          ),
          IconButton(
            icon: const Icon(Icons.share),
            onPressed: () {
              print('Share tapped!');
            },
          ),
        ],
        elevation: 8.0,                    // Shadow beneath the AppBar
        centerTitle: true,                // Centers the title
        toolbarHeight: 60.0,               // Custom height
        // bottom: const PreferredSize( // Example of a bottom widget (like
a TabBar)
        //   preferredSize: Size.fromHeight(48.0),
        //   child: TabBar(
        //     tabs: [
        //       Tab(text: 'Tab 1'),
        //       Tab(text: 'Tab 2'),
        //     ],
        //   ),
        // ),
      ),
      body: const Center(
        child: Text('Content goes here...'),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: AppBarExample()));
// }
```

II. Layout Widgets

These widgets help you arrange other widgets on the screen. They are crucial for creating responsive and organized UIs.

---

4. Container

- **Purpose:** A versatile widget that can hold a single child. It allows you to decorate, position, and size its child. Often used for styling, spacing, or adding backgrounds. It's like a `div` in web development.
- **Key Properties & Customization:**
    - `child`: The single widget contained within the container.
    - `alignment`: Aligns the child within the container. Takes an `AlignmentGeometry` (e.g., `Alignment.center`, `Alignment.topLeft`, `Alignment.bottomRight`).
    - `padding`: Empty space inserted inward from the container's edges to its child. Takes `EdgeInsetsGeometry` (e.g., `EdgeInsets.all(10.0)`, `EdgeInsets.symmetric(horizontal: 20.0)`, `EdgeInsets.only(top: 50.0)`).
    - `margin`: Empty space inserted outward from the container's edges. Takes `EdgeInsetsGeometry`.
    - `color`: The background color of the container. Cannot be used with `decoration` if `decoration` also specifies a color.
    - `decoration`: More complex visual decoration (e.g., gradients, borders, box shadows, rounded corners). Takes a `BoxDecoration` object.
        - `color`: Background color for the decoration (can be used here).
        - `borderRadius`: Rounded corners. Takes `BorderRadiusGeometry` (e.g., `BorderRadius.circular(10.0)`).
        - `border`: A border around the box. Takes `Border.all(color: Colors.black, width: 2.0)`.
        - `boxShadow`: A list of shadows cast by the box. Takes `List<BoxShadow>`.
        - `gradient`: A linear or radial gradient for the background.
        - `image`: An image to paint into the background.
    - `width`, `height`: Explicit dimensions for the container. If not set, the container will try to be as large as its parent or as small as its child, depending on its parent's constraints.
    - `constraints`: Additional constraints to apply to the child. Takes `BoxConstraints`.
    - `transform`: A 3D transformation to apply to the container. Takes a `Matrix4`.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Container Widget Example'),
      ),
      body: Center(
        child: Container(
          // width and height: Explicit dimensions for the container
          width: 200,
          height: 200,
          // padding: Inner spacing from container edges to child
          padding: const EdgeInsets.all(20.0),
          // margin: Outer spacing from container edges to surrounding
elements
          margin: const EdgeInsets.all(30.0),
          // decoration: Provides rich visual styling (cannot use `color`
directly if `decoration` has a color)
          decoration: BoxDecoration(
            color: Colors.blueAccent, // Background color of the decoration
            borderRadius: BorderRadius.circular(15.0), // Rounded corners
            border: Border.all(
              color: Colors.purple,
              width: 3.0,
            ),
            boxShadow: const [ // List of shadows
              BoxShadow(
                color: Colors.black26,
                offset: Offset(5, 5),
                blurRadius: 10.0,
              ),
            ],
            gradient: const LinearGradient( // Gradient background
              colors: [Colors.blueAccent, Colors.lightBlue],
              begin: Alignment.topLeft,
              end: Alignment.bottomRight,
            ),
          ),
          // alignment: Aligns the child within the container
          alignment: Alignment.center,
          child: const Text(
            'Hello Container!',
            style: TextStyle(
              color: Colors.white,
              fontSize: 20,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ContainerExample()));
// }
```

---

5. Row

- **Purpose:** Lays out its children in a horizontal array. It's a fundamental widget for horizontal arrangement.

- **Key Properties & Customization:**
  - `children`: A `List<Widget>` to be laid out horizontally.
  - `mainAxisAlignment`: How children are placed along the main axis (horizontal for `Row`). Takes a `MainAxisAlignment` enum.
    - `start`: Children are placed at the beginning of the row.
    - `end`: Children are placed at the end of the row.
    - `center`: Children are centered along the row.
    - `spaceBetween`: Evenly distribute space between children. The first child is at the start, the last at the end.
    - `spaceAround`: Evenly distribute space around children. The space before the first and after the last child is half the space between children.
    - `spaceEvenly`: Evenly distribute space both around and between children, resulting in equal spacing everywhere.
  - `crossAxisAlignment`: How children are placed along the cross axis (vertical for `Row`). Takes a `CrossAxisAlignment` enum.
    - `start`: Children are aligned to the top.
    - `end`: Children are aligned to the bottom.
    - `center`: Children are centered vertically.
    - `stretch`: Children are stretched to fill the available height (if they don't have explicit height constraints).
    - `baseline`: Aligns children along their text baseline (useful when mixing text of different sizes).
  - `mainAxisSize`: How much space the `Row` should occupy along its main axis.
    - `MainAxisSize.max`: The row will try to take up as much horizontal space as possible.
    - `MainAxisSize.min`: The row will try to take up as little horizontal space as possible, just enough to contain its children.
  - `textDirection`: How children are ordered along the main axis. Defaults to `TextDirection.ltr` (left-to-right). Can be `TextDirection.rtl` (right-to-left).
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Row Widget Example'),
      ),
      body: Center(
        child: Container(
          color: Colors.grey[200], // Visual aid for the Row's boundaries
          height: 100, // Give the row a fixed height to see
crossAxisAlignment clearly
          child: Row(
            // mainAxisAlignment: How children are spaced horizontally
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            // crossAxisAlignment: How children are aligned vertically
within the row's height
            crossAxisAlignment: CrossAxisAlignment.center,
            // mainAxisSize: How much horizontal space the Row occupies
            mainAxisSize: MainAxisSize.max,
            children: <Widget>[
              Container(
                width: 50,
                height: 50,
                color: Colors.red,
                child: const Center(child: Text('1', style: TextStyle(color:
Colors.white))),
              ),
              Container(
                width: 70,
                height: 70,
                color: Colors.green,
                child: const Center(child: Text('2', style: TextStyle(color:
Colors.white))),
              ),
              Container(
                width: 60,
                height: 60,
                color: Colors.blue,
                child: const Center(child: Text('3', style: TextStyle(color:
Colors.white))),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: RowExample()));
// }
```

---

6. Column

- **Purpose:** Lays out its children in a vertical array. It's the counterpart to `Row` for vertical arrangement.
- **Key Properties & Customization:**
  - `children`: A `List<Widget>` to be laid out vertically.
  - `mainAxisAlignment`: How children are placed along the main axis (vertical for `Column`). Takes a `MainAxisAlignment` enum (same options as `Row`).

  - `crossAxisAlignment`: How children are placed along the cross axis (horizontal for `Column`). Takes a `CrossAxisAlignment` enum (same options as `Row`).
  - `mainAxisSize`: How much space the `Column` should occupy along its main axis.
    - `MainAxisSize.max`: The column will try to take up as much vertical space as possible.
    - `MainAxisSize.min`: The column will try to take up as little vertical space as possible, just enough to contain its children.
  - `textDirection`: How children are ordered along the main axis. Defaults to `TextDirection.ltr` (left-to-right). Can be `TextDirection.rtl` (right-to-left).

- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Column Widget Example'),
      ),
      body: Center(
        child: Container(
          color: Colors.grey[200], // Visual aid for the Column's boundaries
          width: 150, // Give the column a fixed width to see
crossAxisAlignment clearly
          child: Column(
            // mainAxisAlignment: How children are spaced vertically
            mainAxisAlignment: MainAxisAlignment.spaceAround,
            // crossAxisAlignment: How children are aligned horizontally
within the column's width
            crossAxisAlignment: CrossAxisAlignment.stretch, // Children
stretch to fill width
            // mainAxisSize: How much vertical space the Column occupies
            mainAxisSize: MainAxisSize.min, // Column only takes space
needed by children
            children: <Widget>[
              Container(
                height: 50,
                color: Colors.orange,
                child: const Center(child: Text('Item A', style:
TextStyle(color: Colors.white))),
              ),
              Container(
                height: 70,
                color: Colors.purple,
                child: const Center(child: Text('Item B', style:
TextStyle(color: Colors.white))),
              ),
              Container(
                height: 60,
                color: Colors.teal,
                child: const Center(child: Text('Item C', style:
TextStyle(color: Colors.white))),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ColumnExample()));
// }
```

---

7. Center

- **Purpose:** Centers its single child within itself. It expands to fill all available space and then centers its child within that space.
- **Key Properties & Customization:**
  - `child`: The single widget to be centered.
  - `widthFactor`: If non-null, the center will be exactly `widthFactor` times its child's width.
  - `heightFactor`: If non-null, the center will be exactly `heightFactor` times its child's height.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Center Widget Example'),
      ),
      body: Container( // Using a Container to visualize the Center's
boundaries
        color: Colors.lightBlue[50],
        width: double.infinity, // Ensure the container takes full width
        height: double.infinity, // Ensure the container takes full height
        child: Center(
          // child: The widget to be centered
          child: Container(
            width: 150,
            height: 150,
            color: Colors.deepOrange,
            child: const Center( // Even this inner Center is just for
emphasis,
                                 // the outer Center would center its child
(this Container)
              child: Text(
                'Centered Content',
                style: TextStyle(color: Colors.white, fontSize: 18),
                textAlign: TextAlign.center,
              ),
            ),
          ),
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: CenterExample()));
// }
```

---

8. Padding

- **Purpose:** Inserts empty space around its single child. It's used to create spacing between a widget's content and its own borders.
- **Key Properties & Customization:**
  - `padding`: An `EdgeInsetsGeometry` that defines the amount of padding.
    - `EdgeInsets.all(double value)`: Applies the same padding to all four sides.
    - `EdgeInsets.symmetric({double vertical, double horizontal})`: Applies padding to vertical and/or horizontal sides.
    - `EdgeInsets.only({double left, double top, double right, double bottom})`: Applies padding to specific sides.
    - `EdgeInsets.fromLTRB(double left, double top, double right, double bottom)`: Defines padding from each side individually.
  - `child`: The single widget inside the padding.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Padding Widget Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Example 1: Padding all around
            Container(
              color: Colors.red[100],
              child: Padding(
                padding: const EdgeInsets.all(20.0), // Padding on all sides
                child: Container(
                  color: Colors.red,
                  width: 100,
                  height: 50,
                  child: const Center(child: Text('All', style:
TextStyle(color: Colors.white))),
                ),
              ),
            ),
            const SizedBox(height: 20), // Spacer
            // Example 2: Horizontal and Vertical padding
            Container(
              color: Colors.green[100],
              child: Padding(
                padding: const EdgeInsets.symmetric(horizontal: 40.0,
vertical: 10.0),
                child: Container(
                  color: Colors.green,
                  width: 100,
                  height: 50,
                  child: const Center(child: Text('Symmetric', style:
TextStyle(color: Colors.white))),
                ),
              ),
            ),
            const SizedBox(height: 20), // Spacer
            // Example 3: Specific side padding
            Container(
              color: Colors.blue[100],
              child: Padding(
                padding: const EdgeInsets.only(left: 50.0, top: 10.0, right:
10.0, bottom: 50.0),
                child: Container(
                  color: Colors.blue,
                  width: 100,
                  height: 50,
                  child: const Center(child: Text('Only', style:
TextStyle(color: Colors.white))),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: PaddingExample()));
// }
```

---

9. SizedBox

- **Purpose:** Creates a box with a specified width and/or height. Often used to create empty space between widgets (as a spacer) or to enforce a specific size on a child.
- **Key Properties & Customization:**
  - `width`: The explicit width of the box.
  - `height`: The explicit height of the box.
  - `child`: An optional child to force its size. If a `child` is provided, the `SizedBox` will attempt to size the child to its `width` and `height`.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('SizedBox Widget Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Container(
              color: Colors.purple[100],
              child: const Text('Above SizedBox'),
            ),
            // SizedBox as a spacer (common use case)
            const SizedBox(height: 30.0), // Vertical spacing
            Container(
              color: Colors.purple[200],
              child: const Text('Below a vertical SizedBox'),
            ),
            const SizedBox(height: 30.0),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Container(
                  color: Colors.orange[100],
                  child: const Text('Left'),
                ),
                // SizedBox as a spacer (common use case)
                const SizedBox(width: 50.0), // Horizontal spacing
                Container(
                  color: Colors.orange[200],
                  child: const Text('Right of horizontal SizedBox'),
                ),
              ],
            ),
            const SizedBox(height: 30.0),
            // SizedBox to enforce a specific size on a child
            SizedBox(
              width: 200.0,
              height: 100.0,
              child: Container(
                color: Colors.teal,
                child: const Center(
                  child: Text(
                    'Fixed Size Box',
                    style: TextStyle(color: Colors.white, fontSize: 18),
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: SizedBoxExample()));
// }
```

---

10. Expanded / Flexible (within Row or Column)

- **Purpose:** These widgets allow children of `Row` or `Column` to expand or flex to fill available space along the main axis. They are crucial for creating responsive layouts.

    - `Expanded`: Forces its child to fill all available space along the main axis. If multiple `Expanded` widgets are present, the space is divided according to their `flex` factors.
    - `Flexible`: Allows its child to fill available space, but it can also be smaller than the available space if its content dictates. It's "flexible" in sizing.

- **Key Properties & Customization:**

    - `flex`: An integer that determines the proportion of available space the widget should occupy. If `flex` is 1 for all children, they share space equally. If one has `flex: 2` and another `flex: 1`, the first takes twice as much space as the second. Default is 1.
    - `fit`: (Only for `Flexible`) How the child should fit the available space.
        - `FlexFit.tight` (default for `Expanded`): The child must fill the space.
        - `FlexFit.loose` (default for `Flexible`): The child can be as large as the space allows, but not necessarily fill it.
    - `child`: The widget that will expand or flex.

- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Expanded/Flexible Example'),
      ),
      body: Column(
        children: <Widget>[
          // Example 1: Expanded in a Row
          Container(
            height: 100,
            color: Colors.grey[200],
            child: Row(
              children: <Widget>[
                Container(width: 80, color: Colors.red, child: const
Center(child: Text('Fixed', style: TextStyle(color: Colors.white)))),
                Expanded( // Takes up all remaining space
                  child: Container(
                    color: Colors.green,
                    child: const Center(child: Text('Expanded (flex:1)',
style: TextStyle(color: Colors.white))),
                  ),
                ),
              ],
            ),
          ),
          const SizedBox(height: 20),
          // Example 2: Multiple Expanded with different flex factors
          Container(
            height: 100,
            color: Colors.grey[300],
            child: Row(
              children: <Widget>[
                Expanded(
                  flex: 2, // Takes 2/3 of available space
                  child: Container(
                    color: Colors.blue,
                    child: const Center(child: Text('Expanded (flex:2)',
style: TextStyle(color: Colors.white))),
                  ),
                ),
                Expanded(
                  flex: 1, // Takes 1/3 of available space
                  child: Container(
                    color: Colors.purple,
                    child: const Center(child: Text('Expanded (flex:1)',
style: TextStyle(color: Colors.white))),
                  ),
                ),
              ],
            ),
          ),
          const SizedBox(height: 20),
          // Example 3: Flexible with FlexFit.tight (behaves like Expanded)
          Container(
            height: 100,
            color: Colors.grey[200],
            child: Row(
              children: <Widget>[
                Flexible(
                  flex: 1,
                  fit: FlexFit.tight, // Behaves like Expanded
                  child: Container(
                    color: Colors.orange,
                    child: const Center(child: Text('Flexible (tight)',
style: TextStyle(color: Colors.white))),
                  ),
                ),
                Flexible(
                  flex: 1,
                  fit: FlexFit.loose, // Child can be smaller than space
                  child: Container(
                    color: Colors.cyan,
                    // This text widget might not fill the entire allocated
space, demonstrating 'loose' fit
                    child: const Center(child: Text('Flexible (loose) - will
not expand if content is small')),
                  ),
                ),
              ],
            ),
          ),
          const SizedBox(height: 20),
          // Example 4: Flexible with content dictating size (when there's
no fixed width/height constraint)
          Container(
            height: 100,
            color: Colors.grey[300],
            child: Row(
              children: <Widget>[
                Flexible(
                  child: Container(
                    color: Colors.brown,
                    padding: const EdgeInsets.all(8.0),
                    child: const Text('Short text here', style:
TextStyle(color: Colors.white)),
                  ),
                ),
                Flexible(
                  child: Container(
                    color: Colors.indigo,
                    padding: const EdgeInsets.all(8.0),
                    child: const Text('A much longer piece of text that will
cause this flexible widget to take up more space.', style: TextStyle(color:
Colors.white)),
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ExpandedFlexibleExample()));
// }
```

---

11. Stack

- **Purpose:** Allows you to layer multiple widgets on top of each other. Useful for overlays, custom positioning, or creating complex UIs where elements overlap. Widgets are painted in the order they appear in the `children` list, with the last child being on top.
- **Key Properties & Customization:**
  - `children`: A `List<Widget>` to be stacked.
  - `alignment`: How non-positioned children (those not wrapped in `Positioned`) are aligned within the stack. Takes an `AlignmentGeometry` (e.g., `Alignment.center`, `Alignment.topLeft`).
  - `fit`: How the non-positioned children should be sized.

        - `StackFit.loose`: The stack tries to size itself to its largest non-positioned child. Non-positioned children can be smaller than the stack.
        - `StackFit.expand`: The stack and its non-positioned children expand to fill the available space.
    - `overflow`: How to handle children that overflow the stack's bounds.
        - `Overflow.clip`: Clips the overflowing children.
        - `Overflow.visible`: Allows overflowing children to be visible (can cause layout issues if not handled carefully).
- **Example Implementation:** (Combined with `Positioned` in the next section for a more practical example)

---

12. Positioned (within Stack)

- **Purpose:** Controls where a child of a `Stack` is placed using precise pixel offsets from the stack's edges. Only works as a child of a `Stack`.

- **Key Properties & Customization:**

    - `top`, `bottom`, `left`, `right`: Distances from the respective edges in logical pixels. You must provide at least one horizontal (`left` or `right`) and one vertical (`top` or `bottom`) constraint, or `width`/`height`. If you provide `left` and `right`, the widget's width is determined by the stack's width minus these offsets.
    - `width`, `height`: Explicit dimensions for the positioned child. If both `left` and `right` are specified, `width` is ignored. If both `top` and `bottom` are specified, `height` is ignored.
    - `child`: The single widget to be positioned.

- **Example Implementation (Stack & Positioned):**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Stack & Positioned Example'),
      ),
      body: Center(
        child: Container(
          width: 300,
          height: 300,
          color: Colors.blueGrey[100], // Background for stack area
          child: Stack(
            // alignment: Aligns non-positioned children within the stack
            alignment: Alignment.center,
            children: <Widget>[
              // Base Layer: A large container
```

```
                Container(
                  width: 250,
                  height: 250,
                  color: Colors.deepPurple,
                  child: const Center(
                    child: Text(
                      'Base Layer',
                      style: TextStyle(color: Colors.white, fontSize: 22),
                    ),
                  ),
                ),

                // Layer 2: Positioned relative to top-left
                Positioned(
                  top: 20,
                  left: 20,
                  child: Container(
                    width: 100,
                    height: 100,
                    color: Colors.redAccent,
                    child: const Center(
                      child: Text(
                        'Top Left',
                        style: TextStyle(color: Colors.white, fontSize: 14),
                      ),
                    ),
                  ),
                ),

                // Layer 3: Positioned relative to bottom-right
                Positioned(
                  bottom: 30,
                  right: 30,
                  child: Container(
                    width: 120,
                    height: 80,
                    color: Colors.greenAccent,
                    child: const Center(
                      child: Text(
                        'Bottom Right',
                        style: TextStyle(color: Colors.white, fontSize: 14),
                        textAlign: TextAlign.center,
                      ),
                    ),
                  ),
                ),

                // Layer 4: Positioned with width and height
                Positioned(
                  top: 80,
                  left: 80,
                  width: 140, // Explicit width
                  height: 140, // Explicit height
                  child: Container(
                    color: Colors.yellow[700],
                    child: const Center(
                      child: Text(
                        'Middle (Fixed Size)',
                        style: TextStyle(color: Colors.black, fontSize: 14),
                        textAlign: TextAlign.center,
                      ),
                    ),
                  ),
                ),

                // Layer 5: A non-positioned child (aligned by stack's
  alignment)
                const Text(
                  'Stack Overlay Text',
                  style: TextStyle(
                      color: Colors.black,
                      fontSize: 16,
                      fontWeight: FontWeight.bold),
                ),
              ],
            ),
          ),
        ),
      );
    }
  }

  // To Run:
  // void main() {
  //   runApp(const MaterialApp(home: StackPositionedExample()));
  // }
```

III. Basic UI Elements

These are the common visible components you'll put on your screen.

---

13. Text

- **Purpose:** Displays a string of text. One of the most fundamental widgets.
- **Key Properties & Customization:**
    - `(unnamed argument)`: The string literal to be displayed. This is the `data` property.
    - `style`: Takes a `TextStyle` object to control appearance.
        - `color`: `Color` of the text.
        - `fontSize`: `double` for the size of the font.
        - `fontWeight`: `FontWeight` (e.g., `FontWeight.bold`, `FontWeight.w700`).
        - `fontStyle`: `FontStyle` (e.g., `FontStyle.italic`).
        - `letterSpacing`: `double` for the space between letters.

        - `wordSpacing`: `double` for the space between words.
        - `decoration`: `TextDecoration` (e.g., `TextDecoration.underline`, `TextDecoration.lineThrough`).
        - `decorationColor`: `Color` of the text decoration.
        - `decorationStyle`: `TextDecorationStyle` (e.g., `TextDecorationStyle.dashed`).
        - `fontFamily`: `String` name of the font family.
    - `textAlign`: How to align the text within its own bounding box. `TextAlign` enum (e.g., `TextAlign.left`, `TextAlign.right`, `TextAlign.center`, `TextAlign.justify`).
    - `overflow`: How visual overflow should be handled. `TextOverflow` enum (e.g., `TextOverflow.ellipsis` for `...`, `TextOverflow.clip`).
    - `maxLines`: An integer specifying the maximum number of lines for the text. Useful with `overflow`.
    - `softWrap`: Whether the text should break at soft line breaks (e.g., word wraps). Defaults to `true`.

- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Text Widget Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Basic Text
            const Text('Hello, Flutter!'),
            const SizedBox(height: 20),

            // Text with custom style
            const Text(
              'Styled Text',
              style: TextStyle(
                color: Colors.blue,
                fontSize: 24.0,
                fontWeight: FontWeight.bold,
                fontStyle: FontStyle.italic,
                letterSpacing: 2.0,
                decoration: TextDecoration.underline,
                decorationColor: Colors.red,
                decorationStyle: TextDecorationStyle.wavy,
              ),
            ),
            const SizedBox(height: 20),
            // Long text with overflow handling and maxLines
            const Padding(
              padding: EdgeInsets.symmetric(horizontal: 20.0),
              child: Text(
                'This is a very long piece of text that demonstrates how
text overflow and max lines work in Flutter. When the text is too long to
fit, it will either clip or show an ellipsis.',
                textAlign: TextAlign.justify, // Justify alignment
                maxLines: 3,                  // Limit to 3 lines
                overflow: TextOverflow.ellipsis, // Show "..." if overflows
                style: TextStyle(fontSize: 16),
              ),
            ),
            const SizedBox(height: 20),

            // Text with predefined theme style (e.g., from MaterialApp's
theme)
            Text(
              'Text using App Theme Style',
              style: Theme.of(context).textTheme.headlineSmall, // Accessing
theme's text styles
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(MaterialApp(
//     home: const TextExample(),
//     theme: ThemeData( // Adding a simple theme for the example
//       textTheme: const TextTheme(
//         headlineSmall: TextStyle(fontSize: 28.0, fontWeight:
FontWeight.bold, color: Colors.deepOrange),
//       ),
//     ),
//   ));
// }
```

---

14. Icon

- **Purpose:** Displays a graphical icon from the Material Design icon set (or custom font icon sets).
- **Key Properties & Customization:**
  - `(unnamed argument)`: The specific icon from `Icons` (e.g., `Icons.phone`, `Icons.email`, `Icons.settings`).
  - `color`: The `Color` of the icon.

    - `size`: The `double` size of the icon in logical pixels.
    - `semanticLabel`: A semantic description of the icon for accessibility tools.
    - `textDirection`: Controls the directionality of the icon (e.g., arrow pointing left or right based on LTR/RTL).

- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Icon Widget Example'),
      ),
      body: const Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Basic Icon
            Icon(Icons.star),
            SizedBox(height: 20),

            // Icon with custom color and size
            Icon(
              Icons.favorite,
              color: Colors.pink,
              size: 60.0,
              semanticLabel: 'A heart icon indicating favorite', // For
accessibility
            ),
            SizedBox(height: 20),

            // Another icon example
            Icon(
              Icons.thumb_up_alt,
              color: Colors.blue,
              size: 40.0,
            ),
            SizedBox(height: 20),

            // Icon used within a Button
            ElevatedButton.icon(
              onPressed: null, // Disabled button for example
              icon: Icon(Icons.mail_outline),
              label: Text('Send Email'),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: IconExample()));
// }
```

---

15. Image

- **Purpose:** Displays an image. Can load from various sources: assets (local files bundled with the app), network URLs, or device files.
- **Key Properties & Customization:**
  - `image`: The `ImageProvider` that specifies the image source.
    - `AssetImage('assets/my_image.png')`: For images bundled in your `pubspec.yaml` `assets` section.
    - `NetworkImage('https://example.com/image.jpg')`: For images loaded from a URL.
    - `FileImage(File('/path/to/image.jpg'))`: For images loaded from a device's file system.
    - `MemoryImage(Uint8List bytes)`: For images loaded from a raw byte list.
  - `width`, `height`: `double` dimensions of the image.
  - `fit`: How the image should be inscribed into the box. Takes a `BoxFit` enum.
    - `BoxFit.fill`: Fill the box, potentially distorting the image.
    - `BoxFit.contain`: As large as possible while still containing the image completely within the box. Aspect ratio is preserved.
    - `BoxFit.cover`: As small as possible while still covering the box. Aspect ratio is preserved. Some parts of the image may be clipped.
    - `BoxFit.fitWidth`: Ensures the image fills the width, possibly clipping top/bottom.
    - `BoxFit.fitHeight`: Ensures the image fills the height, possibly clipping left/right.
    - `BoxFit.none`: Do not scale the image.
    - `BoxFit.scaleDown`: Scales down the image if it is larger than the box, otherwise leaves it at its original size.
  - `alignment`: How to align the image within its bounds when `fit` is not `fill`.
  - `repeat`: How the image should be repeated if it doesn't fill the box and `fit` is not `fill`.
  - `color`: A `Color` to blend with the image.
  - `colorBlendMode`: How the `color` is mixed with the image.
  - `loadingBuilder`: A callback to build a placeholder widget while the image is loading.
  - `errorBuilder`: A callback to build a widget to display if the image fails to load.
- **Example Implementation:**

  (Note: For `AssetImage`, you need to add an `assets` section to your `pubspec.yaml` file and place an image there. For `NetworkImage`, ensure you have internet access.)

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Image Widget Example'),
      ),
      body: Center(
        child: SingleChildScrollView( // Allow scrolling if content
overflows
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              const Text('Image from Network:'),
              const SizedBox(height: 10),
              // Image from network with custom size and fit
              Image.network(
                'https://flutter.dev/images/flutter-logo-sharing.png', //
Example URL
                width: 200,
                height: 200,
                fit: BoxFit.contain, // Contain within the box, preserving
aspect ratio
                loadingBuilder: (context, child, loadingProgress) {
                  if (loadingProgress == null) return child;
                  return Center(
                    child: CircularProgressIndicator(
                      value: loadingProgress.expectedTotalBytes != null
                          ? loadingProgress.cumulativeBytesLoaded /
                              loadingProgress.expectedTotalBytes!
                          : null,
                    ),
                  );
                },
                errorBuilder: (context, error, stackTrace) {
                  return const Text('Failed to load image!');
                },
              ),
              const SizedBox(height: 30),

              const Text('Image from Assets:'),
              const SizedBox(height: 10),
              // Image from assets (requires setup in pubspec.yaml)
              // Uncomment and replace 'assets/flutter_logo.png' with your
asset path
              // Image.asset(
              //   'assets/flutter_logo.png', // Make sure this asset exists
and is declared in pubspec.yaml
              //   width: 150,
              //   height: 150,
              //   fit: BoxFit.cover, // Cover the box, potentially clipping
              // ),
              // const SizedBox(height: 30),

              const Text('Image with Color Blend:'),
              const SizedBox(height: 10),
              Image.network(
                'https://picsum.photos/id/237/200/200', // A random image
                width: 150,
                height: 150,
                color: Colors.purple.withOpacity(0.5), // Apply a purple
tint
                colorBlendMode: BlendMode.darken, // How the color is
blended
                fit: BoxFit.cover,
              ),
            ],
          ),
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ImageExample()));
// }

/*
To use AssetImage:
1. Create a folder (e.g., `assets`) in your project root.
2. Put your image file (e.g., `flutter_logo.png`) inside it.
3. Add this to your pubspec.yaml:

   flutter:
     uses-material-design: true
     assets:
       - assets/flutter_logo.png # Make sure the path matches your file
*/
```

---

16. Card

- **Purpose:** A Material Design card. Cards are surfaces that display content and actions on a single topic, with a slight elevation and rounded corners. They are a common way to group related information in a visually distinct block.
- **Key Properties & Customization:**
  - `child`: The single widget contained within the card.

    - `color`: Background color of the card.
    - `margin`: Empty space around the outside of the card. Defaults to `EdgeInsets.all(4.0)`.
    - `elevation`: The z-coordinate at which to place this card relative to its parent. Controls the size of the shadow beneath the card. Higher elevation means a larger, softer shadow.
    - `shape`: The shape of the card's border and corners. Takes a `ShapeBorder` (e.g., `RoundedRectangleBorder`, `BeveledRectangleBorder`).
        - `RoundedRectangleBorder(borderRadius: BorderRadius.circular(10.0))`: For custom rounded corners.
    - `shadowColor`: The color of the card's shadow.
    - `surfaceTintColor`: The color to blend with the card's surface. (Material 3 property)

- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Card Widget Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Card(
              // elevation: Controls the shadow depth
              elevation: 8.0,
              // margin: Outer spacing around the card
              margin: const EdgeInsets.all(16.0),
              // color: Background color of the card
              color: Colors.amber[50],
              // shape: Customizes the card's border and corners
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(15.0), // More rounded
corners
                side: const BorderSide(color: Colors.orange, width: 2.0), //
Add a border
              ),
              shadowColor: Colors.deepOrange.withOpacity(0.8), // Custom
shadow color
              child: const Padding(
                padding: EdgeInsets.all(20.0),
                child: Column(
                  mainAxisSize: MainAxisSize.min, // Make column as small as
its children
                  children: <Widget>[
                    ListTile(
                      leading: Icon(Icons.album, size: 50),
                      title: Text('The Essential Flutter Card'),
                      subtitle: Text('Music by Dart & Widgets'),
                    ),
                    SizedBox(height: 10),
                    Row(
                      mainAxisAlignment: MainAxisAlignment.end,
                      children: <Widget>[
                        TextButton(
                          onPressed: null,
                          child: Text('BUY TICKETS'),
                        ),
                        SizedBox(width: 8),
                        TextButton(
                          onPressed: null,
                          child: Text('LISTEN'),
                        ),
                        SizedBox(width: 8),
                      ],
                    ),
                  ],
                ),
              ),
            ),
            const SizedBox(height: 20),
            Card(
              elevation: 4.0,
              color: Colors.blue[50],
              margin: const EdgeInsets.symmetric(horizontal: 24.0, vertical:
8.0),
              child: const Padding(
                padding: EdgeInsets.all(16.0),
                child: Text(
                  'Another Card with less elevation.',
                  style: TextStyle(fontSize: 16),
                  textAlign: TextAlign.center,
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: CardExample()));
// }
```

17. Divider

- **Purpose:** A thin horizontal line, often used to separate content visually.
- **Key Properties & Customization:**
  - `height`: The height of the box containing the line. The line itself will be centered vertically within this height. Often, this is set to be slightly larger than `thickness` to provide some visual spacing.
  - `thickness`: The thickness of the line itself.
  - `color`: The `Color` of the line.
  - `indent`: The amount of empty space from the leading edge of the divider.
  - `endIndent`: The amount of empty space from the trailing edge of the divider.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Divider Widget Example'),
      ),
      body: Column(
        children: <Widget>[
          const ListTile(
            title: Text('Item 1'),
            subtitle: Text('Description for Item 1'),
          ),
          // Basic Divider
          const Divider(), // Default height, thickness, color

          const ListTile(
            title: Text('Item 2'),
            subtitle: Text('Description for Item 2'),
          ),
          // Custom Divider: thicker, colored, with height for spacing
          Divider(
            height: 30, // The height of the box containing the line
            thickness: 2, // The actual thickness of the line
            color: Colors.red,
          ),

          const ListTile(
            title: Text('Item 3'),
            subtitle: Text('Description for Item 3'),
          ),
          // Custom Divider with indents
          Divider(
            height: 50,
            thickness: 1,
            color: Colors.blue,
            indent: 50, // Space from the left
            endIndent: 50, // Space from the right
          ),

          const ListTile(
            title: Text('Item 4'),
            subtitle: Text('Description for Item 4'),
          ),
        ],
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: DividerExample()));
// }
```

IV. Input & Interaction Widgets

These widgets allow users to interact with your application.

---

18. ElevatedButton, TextButton, OutlinedButton

- **Purpose:** Different styles of Material Design buttons for user interaction, each with a distinct visual emphasis.

  - **`ElevatedButton`**: A filled button with a shadow. Used for primary actions, adding prominence.
  - **`TextButton`**: A button with just text, no fill or shadow. Used for less prominent actions, often within dialogs or cards.
  - **`OutlinedButton`**: A button with a thin border. Used for medium-emphasis actions, often as an alternative to `ElevatedButton` for secondary actions.

- **Key Properties (common to all, with some variations):**

  - `onPressed`: A `VoidCallback` function that is called when the button is tapped. If `null`, the button is disabled and visually greyed out.
  - `child`: The widget displayed inside the button (e.g., `Text`, `Icon`, `Row` with `Icon` and `Text`).
  - `style`: Takes a `ButtonStyle` object to customize the button's appearance (colors, padding, shape, elevation, text style, etc.). This is the primary way to customize.
    - `backgroundColor`: `MaterialStateProperty.all(Colors.blue)`.
    - `foregroundColor`: `MaterialStateProperty.all(Colors.white)` (text/icon color).
    - `padding`: `MaterialStateProperty.all(EdgeInsets.all(16.0))`.
    - `shape`: `MaterialStateProperty.all(RoundedRectangleBorder(borderRadius: BorderRadius.circular(10.0)))`.

    - `elevation`: `MaterialStateProperty.all(5.0)` (for `ElevatedButton`).
    - `side`: `MaterialStateProperty.all(BorderSide(color: Colors.blue, width: 2.0))` (for `OutlinedButton`).

- **Example Implementation:**

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

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

  void _showSnackBar(BuildContext context, String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Button Widgets Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // ElevatedButton
            ElevatedButton(
              onPressed: () {
                _showSnackBar(context, 'Elevated Button Pressed!');
              },
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.deepPurple, // Button background
color
                foregroundColor: Colors.white,       // Text/icon color
                padding: const EdgeInsets.symmetric(horizontal: 30,
vertical: 15),
                textStyle: const TextStyle(fontSize: 18),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(8.0),
                ),
                elevation: 5, // Shadow depth
              ),
              child: const Text('Elevated Button'),
            ),
            const SizedBox(height: 20),

            // TextButton
            TextButton(
              onPressed: () {
                _showSnackBar(context, 'Text Button Pressed!');
              },
              style: TextButton.styleFrom(
                foregroundColor: Colors.deepOrange, // Text color
                textStyle: const TextStyle(fontSize: 18),
                padding: const EdgeInsets.all(10),
              ),
              child: const Text('Text Button'),
            ),
            const SizedBox(height: 20),

            // OutlinedButton
            OutlinedButton(
              onPressed: () {
                _showSnackBar(context, 'Outlined Button Pressed!');
              },
              style: OutlinedButton.styleFrom(
                foregroundColor: Colors.teal,       // Text/icon color
                side: const BorderSide(color: Colors.teal, width: 2), //
Border color and width
                padding: const EdgeInsets.symmetric(horizontal: 25,
vertical: 12),
                textStyle: const TextStyle(fontSize: 16),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(20.0), // More rounded
corners
                ),
              ),
              child: const Text('Outlined Button'),
            ),
            const SizedBox(height: 20),

            // Disabled Button (onPressed: null)
            ElevatedButton(
              onPressed: null, // Button is disabled
              child: const Text('Disabled Button'),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ButtonExample()));
// }
```

---

19. FloatingActionButton (FAB)

- **Purpose:** A circular button that floats above content, usually associated with the primary action on a screen (e.g., adding a new item, composing an email).
- **Key Properties & Customization:**
  - `onPressed`: A `VoidCallback` function when the button is tapped. If `null`, the button is disabled.
  - `child`: The widget displayed inside the button (e.g., an `Icon` widget).
  - `backgroundColor`: The background color of the button.
  - `foregroundColor`: The color of the child (icon/text).
  - `tooltip`: Text to display when the user long-presses the button (for accessibility).
  - `elevation`: The z-coordinate for the button's shadow.
  - `mini`: If true, the button is smaller.
  - `shape`: The shape of the button.
  - `heroTag`: A unique tag for Hero animations. Useful if you have multiple FABs or want to disable the default hero animation.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('FloatingActionButton Example'),
      ),
      body: const Center(
        child: Text(
          'Scroll down or interact to see FAB',
          style: TextStyle(fontSize: 18),
        ),
      ),
      floatingActionButton: Column(
        mainAxisSize: MainAxisSize.min, // Make column take minimum space
        children: [
          FloatingActionButton(
            onPressed: () {
              ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(content: Text('Add button pressed!')));
            },
            heroTag: 'add_button', // Required if multiple FABs in one
screen
            backgroundColor: Colors.teal,
            foregroundColor: Colors.white,
            elevation: 6.0,
            tooltip: 'Add new item', // Accessibility text
            child: const Icon(Icons.add),
          ),
          const SizedBox(height: 10), // Spacing between FABs
          FloatingActionButton.extended( // FAB with text label
            onPressed: () {
              ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(content: Text('Compose email!')));
            },
            heroTag: 'compose_email',
            label: const Text('Compose'),
            icon: const Icon(Icons.mail),
            backgroundColor: Colors.deepOrange,
            foregroundColor: Colors.white,
            elevation: 6.0,
          ),
          const SizedBox(height: 10),
          FloatingActionButton(
            onPressed: null, // Disabled FAB
            heroTag: 'disabled_fab',
            mini: true, // Smaller FAB
            child: const Icon(Icons.delete),
          ),
        ],
      ),
      // floatingActionButtonLocation: Defines where the FAB is placed
      floatingActionButtonLocation: FloatingActionButtonLocation.endFloat,
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: FABExample()));
// }
```

---

20. TextField

- **Purpose:** Allows users to input text. This is a `StatefulWidget` because its internal text content changes over time.

- **Key Properties & Customization:**

  - `controller`: A `TextEditingController` to manage and retrieve the text. Essential for getting/setting text programmatically.
  - `decoration`: An `InputDecoration` to style the input field. This is where most visual customization happens.
    - `labelText`: A floating label above the input field.
    - `hintText`: Placeholder text displayed when the input is empty.
    - `prefixIcon`, `suffixIcon`: Icons displayed at the beginning or end of the input.
    - `border`: The border around the input field (e.g., `OutlineInputBorder()`, `UnderlineInputBorder()`).
    - `filled`: If `true`, the decoration's container is filled with `fillColor`.

        - `fillColor`: The background color of the input field.
        - `helperText`: Text displayed below the input field, often for instructions.
        - `counterText`: Text displayed below the input field, typically showing character count.
        - `errorText`: Error message displayed below the input field.
    - `keyboardType`: The type of keyboard to display (e.g., `TextInputType.emailAddress`, `TextInputType.number`, `TextInputType.phone`).
    - `obscureText`: `true` for password fields (hides input with dots/asterisks).
    - `onChanged`: Callback function `(String value)` that is called whenever the text changes.
    - `onSubmitted`: Callback function `(String value)` that is called when the user submits the text (e.g., by pressing 'Done' on the keyboard).
    - `readOnly`: If `true`, the user cannot edit the text.
    - `maxLines`: Maximum number of lines for the input. Set to `null` for unlimited, `1` for a single line (default).
    - `minLines`: Minimum number of lines for the input.
    - `maxLength`: Maximum number of characters allowed.
    - `textAlign`: How to align the text within the input field.
    - `style`: `TextStyle` for the input text itself.

- **Example Implementation:**

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

class TextFieldExample extends StatefulWidget {
  const TextFieldExample({super.key});

  @override
  State<TextFieldExample> createState() => _TextFieldExampleState();
}

class _TextFieldExampleState extends State<TextFieldExample> {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  String _displayText = '';

  @override
  void dispose() {
    _nameController.dispose();
    _emailController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('TextField Widget Example'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(20.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            // Basic TextField with label and hint
            TextField(
              controller: _nameController,
              decoration: const InputDecoration(
                labelText: 'Enter your name', // Label that floats
                hintText: 'e.g., John Doe',    // Placeholder text
                border: OutlineInputBorder(),   // Outline border
                prefixIcon: Icon(Icons.person), // Icon inside the input
field
              ),
              keyboardType: TextInputType.text, // Text keyboard
              onChanged: (text) {
                setState(() {
                  _displayText = 'Name: $text';
                });
              },
            ),
            const SizedBox(height: 20),

            // Email TextField with error and custom style
            TextField(
              controller: _emailController,
              decoration: InputDecoration(
                labelText: 'Email',
                hintText: 'your_email@example.com',
                border: const UnderlineInputBorder(), // Underline border
                suffixIcon: IconButton(               // Icon at the end
                  icon: const Icon(Icons.clear),
                  onPressed: () {
                    _emailController.clear();
                    setState(() {
                      _displayText = '';
                    });
                  },
                ),
                errorText: _emailController.text.isEmpty ||
_emailController.text.contains('@')
                    ? null
                    : 'Please enter a valid email', // Dynamic error message
                filled: true,                   // Fill the background
                fillColor: Colors.blue[50],     // Background color
              ),
              keyboardType: TextInputType.emailAddress, // Email keyboard
type
              onSubmitted: (text) { // Called when user presses 'Done'
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text('Email Submitted: $text')),
                );
              },
            ),
            const SizedBox(height: 20),
            // Password TextField
            const TextField(
              obscureText: true, // Hides input for passwords
              decoration: InputDecoration(
                labelText: 'Password',
                border: OutlineInputBorder(),
                prefixIcon: Icon(Icons.lock),
                suffixIcon: Icon(Icons.visibility), // Eye icon
              ),
            ),
            const SizedBox(height: 20),

            // Read-only TextField
            const TextField(
              readOnly: true,
              decoration: InputDecoration(
                labelText: 'Read-only field',
                border: OutlineInputBorder(),
              ),
              controller: TextEditingController(text: 'This text cannot be
edited.'),
            ),
            const SizedBox(height: 30),
            Text(
              'Live Input: $_displayText',
              style: const TextStyle(fontSize: 16, color: Colors.indigo),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: TextFieldExample()));
// }
```

---

21. GestureDetector

- **Purpose:** Detects various gestures (taps, drags, long presses) on its child widget. Useful for making any widget interactive, even those that don't have built-in `onPressed` callbacks (like `Container`, `Image`, `Text`).
- **Key Properties & Customization:**
  - `onTap`: Callback when the widget is tapped.
  - `onDoubleTap`: Callback when the widget is double-tapped.
  - `onLongPress`: Callback when the widget is long-pressed.
  - `onHorizontalDragUpdate`, `onVerticalDragUpdate`: Callbacks during drag events.

  - onPanUpdate, onScaleUpdate: Callbacks for more complex gestures.
  - child: The widget on which gestures are detected.
  - behavior: How to hit-test the child. HitTestBehavior.opaque makes the entire box respond to hits, HitTestBehavior.translucent lets hits pass through if no part of the child handles it.
- **Example Implementation:**

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

class GestureDetectorExample extends StatefulWidget {
  const GestureDetectorExample({super.key});

  @override
  State<GestureDetectorExample> createState() =>
_GestureDetectorExampleState();
}

class _GestureDetectorExampleState extends State<GestureDetectorExample> {
  String _gestureStatus = 'No gesture detected';
  Color _boxColor = Colors.blue;

  void _updateStatus(String status) {
    setState(() {
      _gestureStatus = status;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('GestureDetector Example'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            GestureDetector(
              onTap: () {
                _updateStatus('Single Tap!');
                setState(() {
                  _boxColor = Colors.green;
                });
              },
              onDoubleTap: () {
                _updateStatus('Double Tap!');
                setState(() {
                  _boxColor = Colors.purple;
                });
              },
              onLongPress: () {
                _updateStatus('Long Press!');
```

```
                  setState(() {
                    _boxColor = Colors.red;
                  });
                },
                onPanUpdate: (details) {
                  _updateStatus('Panning... Dx:
${details.delta.dx.toStringAsFixed(2)}, Dy:
${details.delta.dy.toStringAsFixed(2)}');
                },
                child: Container(
                  width: 200,
                  height: 200,
                  color: _boxColor,
                  child: const Center(
                    child: Text(
                      'Tap, Double Tap, Long Press, or Drag Me!',
                      style: TextStyle(color: Colors.white, fontSize: 16),
                      textAlign: TextAlign.center,
                    ),
                  ),
                ),
              ),
              const SizedBox(height: 30),
              Text(
                _gestureStatus,
                style: const TextStyle(fontSize: 20, fontWeight:
FontWeight.bold),
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  setState(() {
                    _boxColor = Colors.blue;
                    _gestureStatus = 'No gesture detected';
                  });
                },
                child: const Text('Reset Box'),
              ),
            ],
          ),
        ),
      );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: GestureDetectorExample()));
// }
```

V. Scrolling Widgets

These widgets are essential for displaying content that might exceed the screen's size.

---

22. ListView

- **Purpose:** A scrollable list of widgets arranged linearly (vertically by default). It's highly optimized for long lists because it only renders items that are currently visible on the screen, improving performance.
- **Key Properties & Customization:**
  - `children`: A `List<Widget>` to display. Use this for shorter, fixed lists where all items are known beforehand (e.g., a menu).
  - `itemBuilder`: (For `ListView.builder`) A `IndexedWidgetBuilder` callback that builds children on demand. This is crucial for long or infinite lists to conserve memory.
    - `itemCount`: The total number of items the builder should create. Essential for `ListView.builder`.
  - `scrollDirection`: `Axis.vertical` (default) or `Axis.horizontal`.
  - `padding`: `EdgeInsetsGeometry` for spacing around the list items.
  - `physics`: How the scroll view responds to user input (e.g., `AlwaysScrollableScrollPhysics`, `NeverScrollableScrollPhysics`, `BouncingScrollPhysics`).
  - `shrinkWrap`: If `true`, the list will take up only as much space as its children require, rather than trying to fill the available space. Useful inside `Column` or `Row` when the list's size is not constrained.
  - `controller`: A `ScrollController` to programmatically control the scroll position.
  - `separatorBuilder`: (For `ListView.separated`) A builder that adds a separator widget between items.
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('ListView Widget Example'),
      ),
      body: Column(
        children: <Widget>[
          const Padding(
            padding: EdgeInsets.all(8.0),
            child: Text(
              'ListView.builder (Dynamic List):',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
          ),
          Expanded( // ListView.builder needs to be constrained in size
            child: ListView.builder(
              itemCount: 20, // Total number of items
              itemBuilder: (BuildContext context, int index) {
                return Card(
                  margin: const EdgeInsets.symmetric(horizontal: 10,
vertical: 5),
                  elevation: 2,
                  child: ListTile(
                    leading: CircleAvatar(
                      child: Text('${index + 1}'),
                    ),
                    title: Text('List Item ${index + 1}'),
                    subtitle: Text('This is the description for item ${index
+ 1}.'),
                    trailing: const Icon(Icons.arrow_forward_ios),
                    onTap: () {
                      ScaffoldMessenger.of(context).showSnackBar(
                          SnackBar(content: Text('Tapped Item ${index +
1}')));
                    },
                  ),
                );
              },
            ),
          ),
          const Padding(
            padding: EdgeInsets.all(8.0),
            child: Text(
              'ListView (Fixed Children):',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
            ),
          ),
          SizedBox( // Constrain the height for the second ListView
            height: 200,
            child: ListView(
              padding: const EdgeInsets.all(10), // Padding around the list
              // scrollDirection: Axis.horizontal, // Uncomment for
horizontal list
              physics: const BouncingScrollPhysics(), // Custom scroll
physics
              children: <Widget>[
                Container(
                  height: 50,
                  color: Colors.amber[600],
                  child: const Center(child: Text('Entry A')),
                ),
                const Divider(), // Separator
                Container(
                  height: 50,
                  color: Colors.amber[500],
                  child: const Center(child: Text('Entry B')),
                ),
                const Divider(),
                Container(
                  height: 50,
                  color: Colors.amber[400],
                  child: const Center(child: Text('Entry C')),
                ),
                const Divider(),
                Container(
                  height: 50,
                  color: Colors.amber[300],
                  child: const Center(child: Text('Entry D')),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: ListViewExample()));
// }
```

---

23. SingleChildScrollView

- **Purpose:** Makes its single child scrollable. Useful when you have a small amount of content that might overflow the screen (e.g., a form that needs to scroll on smaller devices or different orientations) but don't need the virtualization of `ListView` (which is for many repeating items).
- **Key Properties & Customization:**
  - `child`: The single widget that becomes scrollable.
  - `scrollDirection`: `Axis.vertical` (default) or `Axis.horizontal`.
  - `padding`: `EdgeInsetsGeometry` for padding around the scrollable content.
  - `physics`: How the scroll view responds to user input (e.g., `AlwaysScrollableScrollPhysics` ensures it always scrolls, even if content fits).
  - `controller`: A `ScrollController` to programmatically control the scroll position.
  - `reverse`: If `true`, the scroll view scrolls in the opposite direction (bottom-to-top for vertical, right-to-left for horizontal).
- **Example Implementation:**

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

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('SingleChildScrollView Example'),
      ),
      body: SingleChildScrollView(
        // scrollDirection: Axis.horizontal, // Uncomment for horizontal
scrolling
        padding: const EdgeInsets.all(16.0),
        physics: const BouncingScrollPhysics(), // Provides a bouncing
effect at ends
        child: Column(
          children: <Widget>[
            Container(
              height: 200,
              color: Colors.red[100],
              child: const Center(child: Text('Header Section')),
            ),
            const SizedBox(height: 20),
            const Text(
              'This is some introductory text. Below this, there will be a
lot of content '
              'that will definitely exceed the screen height, demonstrating
the need for '
              'SingleChildScrollView to make the entire column scrollable.',
              style: TextStyle(fontSize: 16),
            ),
            const SizedBox(height: 20),
            // Lots of content to make it scrollable
            ...List.generate(
              15,
              (index) => Padding(
                padding: const EdgeInsets.symmetric(vertical: 8.0),
                child: Container(
                  height: 80,
                  color: Colors.blueGrey[index * 100 % 900 + 100], //
Varying shades
                  child: Center(
                    child: Text(
                      'Content Block ${index + 1}',
                      style: const TextStyle(color: Colors.white, fontSize:
18),
                    ),
                  ),
                ),
              ),
            ),
            const SizedBox(height: 20),
            Container(
              height: 150,
              color: Colors.green[100],
              child: const Center(child: Text('Footer Section')),
            ),
          ],
        ),
      ),
    );
  }
}

// To Run:
// void main() {
//   runApp(const MaterialApp(home: SingleChildScrollViewExample()));
// }
```