Lecture 4 - Flutter Widget

Introduction 🚀

- **Widgets:** These are the `building blocks` of user interfaces in Flutter. They are used to compose the UI elements.
- **Composition:** The process of putting widgets together to `create` the `UI` is known as `composition`.
- **Widget Tree:** Widgets are arranged in a `tree structure`, with each widget having parent-child relationships. 🌳

Stateful and Stateless Widgets 🎭

- **Stateful Widgets:** 🔄 These are the shape-shifters of Flutter, capable of changing appearance based on user interactions or data updates.
  - Examples: `Checkbox`, `Radio`, `Slider`, `Form`, `TextField`.
- **Stateless Widgets:** 🗿 These are the steadfast sentinels, maintaining their form without change.
  - Examples: `Icon`, `IconButton`.

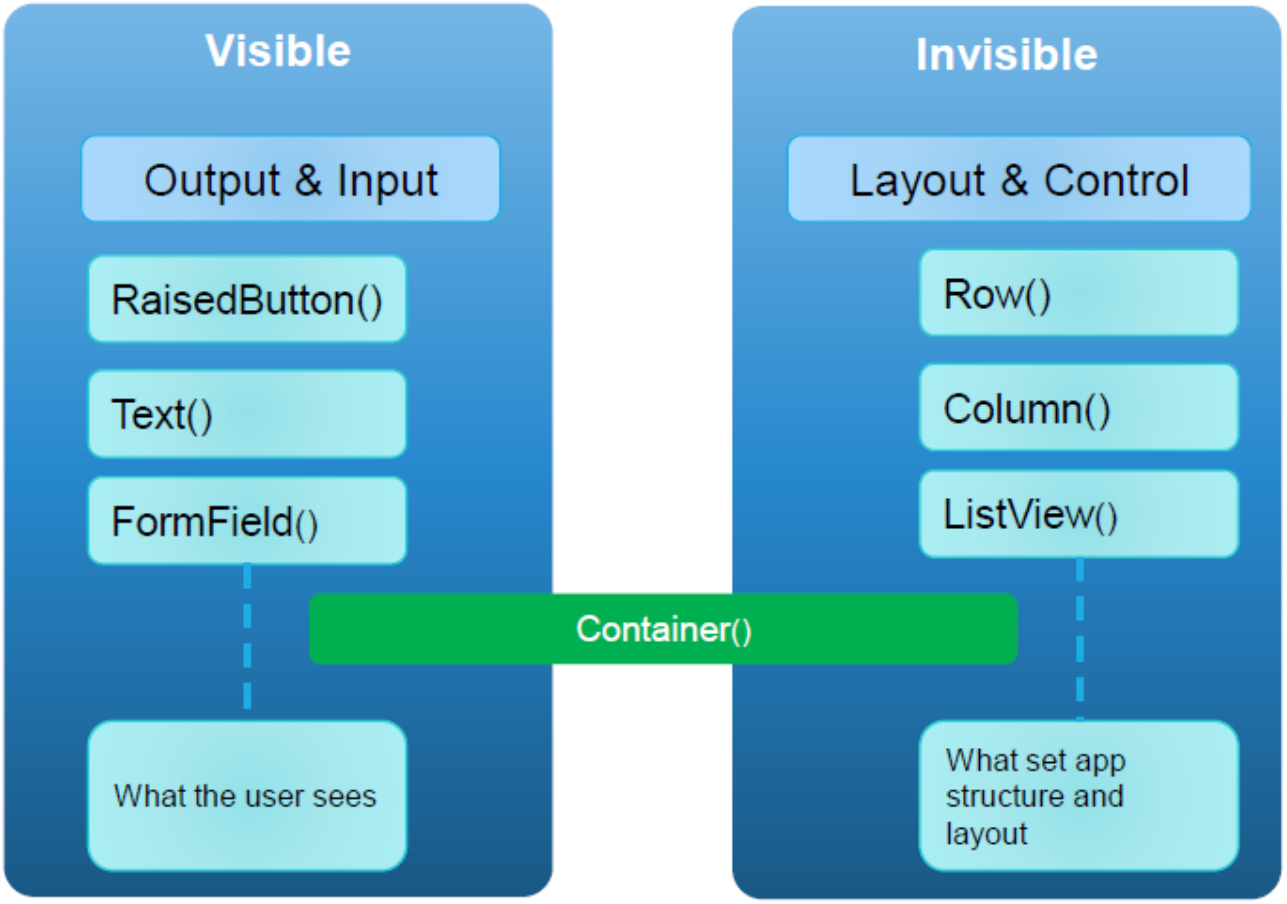
Common Widgets 🎨

Flutter offers a treasure trove of widgets for every need:

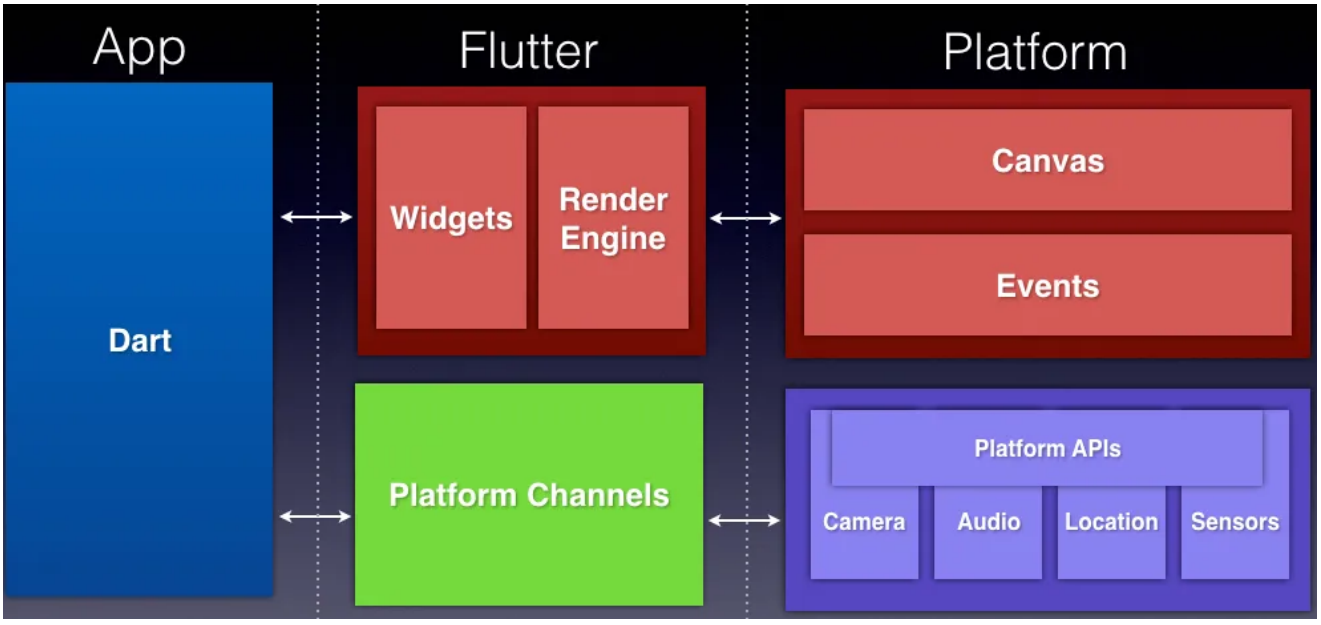
- **Layout Widgets:** 📐 `Scaffold`, `Column`, `Row`, `Stack`.
- **Structure Widgets:** 📐 `Button`, `Text`, `Image`, `RaisedButton`.
- **Style Widgets:** 🎩 `TextStyle`, `Color`.
- **Animation Widgets:** 🌀 `FadeInPhoto`, `Transformations`.
- **Positioning and Alignment Widgets:** 🎯 `Center`, `Padding`.

Visible & Invisible Widgets 👁️

- **Visible Widgets:** 👀 These directly shape what the user sees, forming the UI structure.
  - Examples: `RaisedButton`, `Text()`, `FormField`.
- **Invisible Widgets:** 🌌 These control the layout and behavior behind the scenes.
  - Examples: `Row()`, `Column()`, `ListView`.



Flutter Application Render



MyApp and MaterialApp Widget 📱

- **MyApp:** 🏠 A blueprint for the app's structure, setting the `stage` for all the widgets to come.
- **MaterialApp:** 🛠️ The `container for your app`, providing essential services like `navigation` and `theming`.

User Interface: Material & Cupertino 🎨 🍏

- **Material Design:** 🎉 Google's design language for creating visually appealing digital products with consistent styling and interactions.
- **Cupertino:** 📱 Apple's UI design system. Flutter's got you covered with iOS-styled widgets for those sleek iPhone apps.

Building Widgets 🛠️

- **Build Method:** 🏗️ This method is where the magic happens! Flutter calls `build()` to learn how to render a widget.
- **Configuration Information:** 📝 The `build()` method returns a Widget object that provides Flutter with the necessary details on how to bring the widget to life.

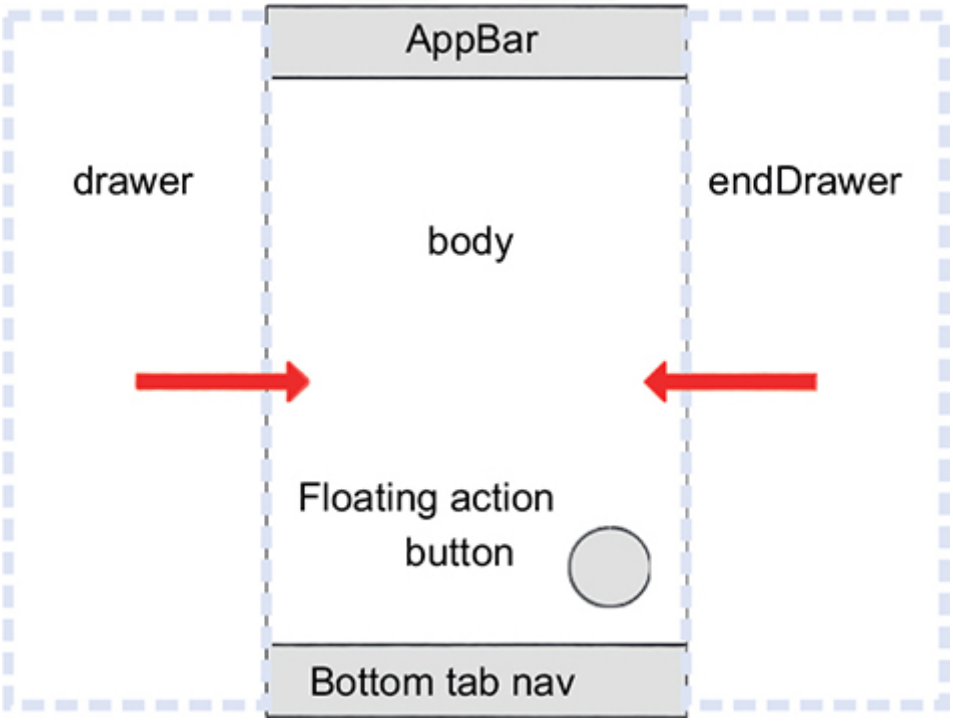
```
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: "First App",
      home: Text("Hello World"),
    );
  }
}
```

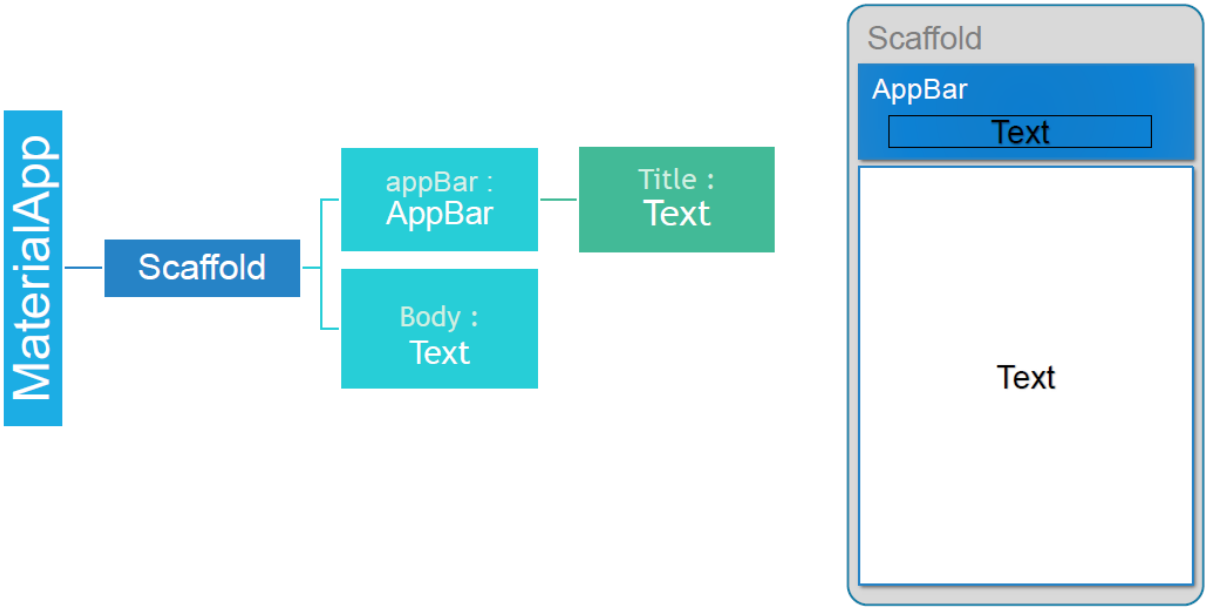
Scaffold Widget 🏗️

Layout: Scaffold Widget

- The `Scaffold` widget serves as the primary container for a `MaterialApp`.
- It automatically fills the entire device screen.

- Standard elements like `AppBar` and `Drawer` can be added easily.

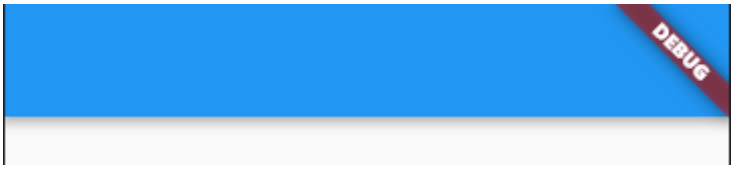





```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(
          title: Text('First App'),
        ),
        body: Text('Home Page'),
      ),
    );
  }
}
```

Layout: AppBar Widget

- Positioned at the top of the screen, the `AppBar` can contain various widgets.
- It's commonly used for branding elements like `logos` and `titles`, and for `user interaction` components like buttons or search fields.
- Comprises three components: `Leading`, `Title`, and `Actions`.




```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(
          title: Text('First App'),
        ),
        body: Text('Home Page'),
      ),
    );
  }
}
```

Layout: FloatingActionButton Widget

- The `FloatingActionButton` is a button that floats at the bottom right corner of the screen by default.
- It remains fixed in its position even when the page is scrolled.
- Commonly used for prominent actions like adding a photo or creating a new item.

```
  home: Scaffold(
        appBar: AppBar(
          title: Text('First App'),
        ),
        floatingActionButton: FloatingActionButton(
          elevation: 10.0,
          child: Icon(Icons.add_a_photo),
          onPressed: () {
                print("Photo added successfully");
          },
        ),
        body: Center(
```

```
        heightFactor: 3,
        child: Text('Home page'),
      ),
  )
```

Layout: Drawer Widget

- The `Drawer` widget displays a slider menu or panel on the side of the `Scaffold`.
- Users can swipe left or right to access the menu.
- The `AppBar` automatically includes an icon for opening the drawer.

```
  home: Scaffold(
      drawer: Drawer(
        child: ListView(
              children: <Widget>[
                ListTile(
                      title: Text('Item 1'),
                ),
                ListTile(
                      title: Text('Item 2'),
                ),
              ],
        ),
      )
```

Layout: BottomNavigationBar Widget

- The `BottomNavigationBar` widget acts as a menu at the bottom of the `Scaffold`.
- Commonly used for navigation purposes, it displays multiple icons or texts as items.

```
  home: Scaffold(
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: 0,
        fixedColor: Colors.blueAccent,
        items: [
              BottomNavigationBarItem(
                label: 'Home',
                icon: Icon(Icons.home),
              ),
              BottomNavigationBarItem(
                label: 'Search',
                icon: Icon(Icons.search),
              ),
        ],
      ),
  ),
```

Basic Material Widgets 🎨

Basic: Text Widget

- The `Text` widget displays a string of text with a single style.
- It supports multiple lines of text and various styling options.
- Essential properties include:
  - `textAlign`: Specifies horizontal text alignment.
  - `textDirection`: Determines the layout direction.
  - `overflow`: Controls text overflow behavior.
  - `textScaleFactor`: Scales the text size.

```
Scaffold(
  appBar: AppBar(title: Text('First')),
  body: Container(
    width: double.infinity,
    child: Text(
      'Home Page. This is the first lecture in this Flutter course',
      overflow: TextOverflow.clip,
      textAlign: TextAlign.end,
      textDirection: TextDirection.rtl,
      textScaleFactor: 1.5,
    ),
  ),
);
```

Essential properties:

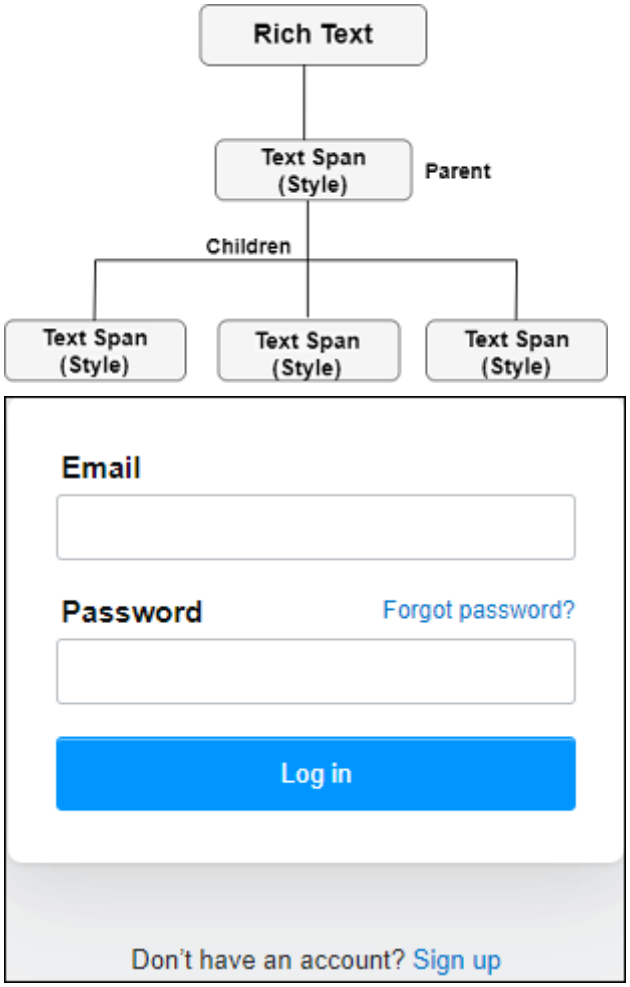
- `SoftWrap`: It determines whether to show all content when there is not enough space available.
  - If set to `true`, it will `show all content`; otherwise, it will not.
- `MaxLines`: It specifies the maximum number of lines displayed in the text widget.
- `Style`: It allows developers to style their text.

```
home: Scaffold(
  appBar: AppBar(
    title: Text('First'),
  ),
  body: Container(
    width: double.infinity,
    child: Text(
      'Home Page. This is the first lecture in this flutter course',
      overflow: TextOverflow.clip, // ellipsis
```

```
      softWrap: false,
      maxLines: 3,
      textScaleFactor: 1.0,
    ),
  ),
),
```

Basic: RichText Widget

- The `RichText` widget displays a paragraph with multiple styles such as bold, underlined, or colored text.
- Useful for scenarios like login screens or creating user account options.




```
RichText(
  text: TextSpan(
    text: "Don't have an account? ",
    style: TextStyle(color: Colors.black, fontSize: 20),
    children: [
      TextSpan(
        text: 'Sign up',
        style: TextStyle(color: Colors.blueAccent, fontSize: 20),
      ),
      TextSpan(text: ' Go back to home page', style: TextStyle(color:
Colors.black, fontSize: 20)),
```

```
      TextSpan(text: ' Home', style: TextStyle(color: Colors.black, fontSize:
20)),
    ],
  ),
);
```

Basic: TextField Widget

- An input element to hold alphanumeric data, such as a name or password.
- By default, Flutter decorates the TextField with an underline.
- We can customize its appearance by adding attributes such as label, icon, and error text using an InputDecoration.
- If we want to remove all decoration properties, we can set the decoration to null.
- Common `attributes` used with the TextField widget:
  - `decoration` : Shows the decoration around the TextField.
  - `border` : Creates a default rounded rectangle border.
  - `labelText` : Displays the label text when the TextField is selected.
  - `hintText` : Displays hint text inside the TextField.
  - `icon` : Adds icons directly to the TextField.
  - `obscureText` : Makes the field not easily readable.

```
Container(
  margin: EdgeInsets.only(top: 10, bottom: 10),
  child: TextField(
    decoration: InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'Username',
      hintText: 'Enter Your Username',
    ),
  ),
),

Container(
  margin: EdgeInsets.only(top: 10, bottom: 10),
  child: TextField(
    obscureText: true,
    decoration: InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'Password',
      hintText: 'Enter Your Password',
      suffixIcon: IconButton(
        onPressed: () {
          print('Password Visibility Toggled');
        },
        icon: Icon(Icons.visibility),
      ),
```

```
      ),
    ),
  ),
```

Basic: Buttons Widget

- Buttons provide user interaction options like triggering actions or navigating.
- Flutter offers various types of buttons like `TextButton`, `ElevatedButton`, `OutlinedButton`, and `MaterialButton`.

```
TextButton(
  onPressed: () {},
  child: Text('Sign Up', style: TextStyle(fontSize: 20)),
),
ElevatedButton(
  onPressed: () {},
  child: Text('Click', style: TextStyle(fontSize: 20)),
),
OutlinedButton(
  onPressed: () {},
  child: Text('Outline', style: TextStyle(fontSize: 20)),
),
MaterialButton(
  onPressed: () {},
  color: Colors.blue,
  height: 60,
  child: Text('Submit', style: TextStyle(fontSize: 20, color: Colors.white)),
);
```

Basic: Image Widget

- Used to `display an image`, where the image source can be specified in various ways:
  - **Image Provider**
  - **Asset**
  - **Network**
  - **File**
  - **Memory**
- **Steps to Add an Image:**
  1. Create a new folder named "assets" (or any desired name) in the root of the project.
  2. Manually add an image inside this folder.
  3. Update the `pubspec.yaml` file to include the image. For example, if the image name is `earth.png`, add the following lines:

```
                assets:
                - assets/earth.png
```

```
Scaffold(
  appBar: AppBar(title: Text('Basic Widgets App')),
  body: Image.asset('assets/earth.png'),
);
```

Basic: Icon Widget

- **Icon Widget:**
  - Allows you to `create icon widgets` using a prebuilt list of material icons, accessible through the `Icons` class.
  - You can customize the icon's size and color.
  - Icons in Flutter can be either built-in or custom.
  - Flutter provides a comprehensive list of all available icons in the `Icons` class.
- **Icon Properties:**
  - **icon:** Specifies the name of the icon to display in the application.
  - **color:** Defines the color of the icon.
  - **size:** Specifies the size of the icon in pixels. Typically, icons have equal height and width.

```
Scaffold(
  appBar: AppBar(title: Text('Basic Widgets App')),
  body: Column(
    children: [
      Image.asset('assets/earth.png'),
      Icon(Icons.home, size: 40, color: Colors.blue),
    ],
  ),
);
```