Child & Children

Child:

- **Purpose: Used by widgets that can have only a single child.**
- **Usage: When you want to include one widget inside another.**
- **Example: In a Center widget, the child property specifies a single widget to be centered.**

Children:

- **Purpose: Used by widgets that can have multiple children.**
- **Usage: When you want to include a list of widgets inside another widget.**
- **Example: In a Column widget, the children property is a list of widgets arranged vertically.**

Widgets with child:

1. **Center: Centers a single child widget.**
2. **Container: Can contain one child widget with styling options.**
3. **Padding: Adds padding around a single child widget.**
4. **Align: Aligns a single child widget within itself.**
5. **SizedBox: Provides a fixed size for a single child widget.**
6. **Scaffold: Can contain one body widget (but often includes other properties like appBar and floatingActionButton).**

Widgets with children:

1. **Column: Arranges multiple children widgets vertically.**
2. **Row: Arranges multiple children widgets horizontally.**
3. **ListView: Displays a scrollable list of multiple children widgets.**
4. **Stack: Overlays multiple children widgets on top of each other.**
5. **GridView: Displays multiple children widgets in a grid layout.**
6. **Wrap: Arranges multiple children widgets in a wrap-around layout.**

Differences Between Stateless and Stateful widgets

Stateless Widget:
A widget that does not store or manage any state and remains unchanged throughout its lifecycle.

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

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

  @override
  Widget build(BuildContext context) {
    return const Scaffold();
  }
}
```

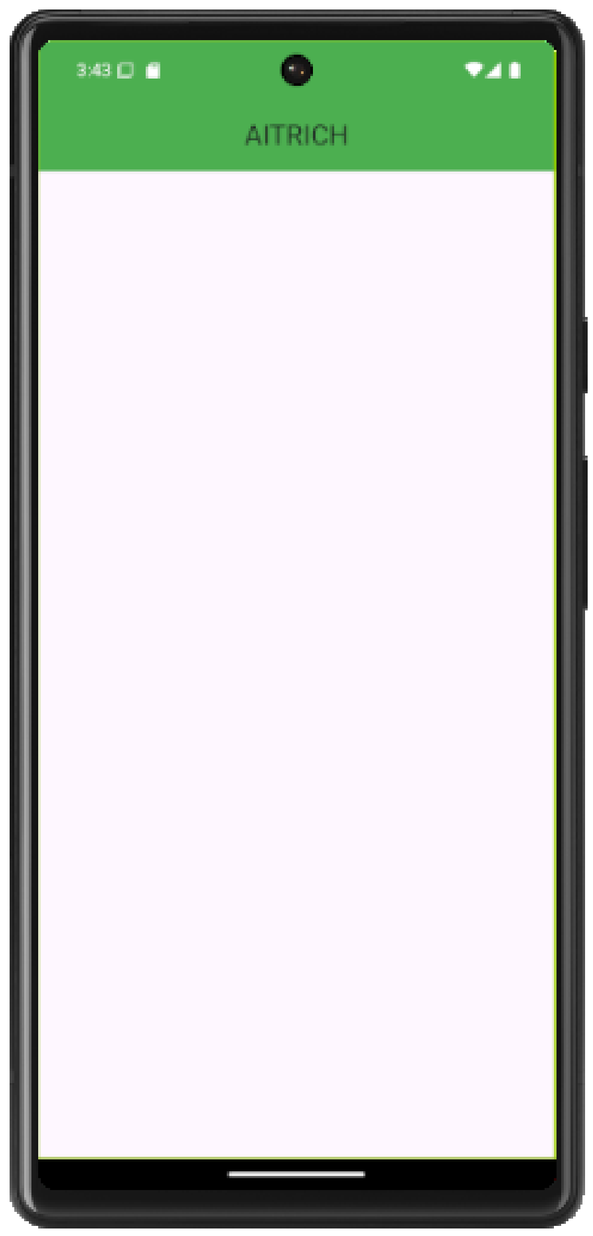
Stateful Widget:
A widget that can store and manage state, allowing it to rebuild and update its UI in response to changes.

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

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

  @override
  State<Myapp> createState() => _MyappState();
}

class _MyappState extends State<Myapp> {
  @override
  Widget build(BuildContext context) {
    return const Scaffold();
  }
}
```

Appbar

In Flutter, an AppBar is a top toolbar typically used in a Scaffold widget. It provides space for a title, navigation icons, and action buttons. You can customize it with properties like title, leading, actions, and backgroundColor.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.green,
      title: const Text("AITRICH"),
      centerTitle: true,
    ), // AppBar
  ); // Scaffold
}
```



This Flutter code creates a screen with a green top bar (called AppBar). The title in the center of the bar is "AITRICH". The Scaffold organizes the screen, and the AppBar is placed at the top with a green background and centered title.

```dart
      appBar: AppBar(
        iconTheme: const IconThemeData(color: Colors.white),
        backgroundColor: Colors.green,
        title: const Text(
          "AITRICH",
          style: TextStyle(color: Colors.white),
        ), // Text
        centerTitle: true,
        leading: const Icon(Icons.settings),
        actions: const [
          Icon(Icons.search),
        ],
      ), // AppBar
```

This code customizes the AppBar by setting the iconTheme to white, using a green backgroundColor, applying white text to the title, adding a settings Icon on the left with leading, and including a search Icon in the actions section.

- **iconTheme: Sets all icons to white.**
- **backgroundColor: Green background.**
- **title: White text "AITRICH" centered.**
- **centerTitle: Centers the title.**
- **leading: Adds a settings icon on the left.**
- **actions: Adds a search icon on the right.**

Container

In Flutter, a Container is a widget used to create a rectangular visual element with customizable properties. It acts as a box that can hold other widgets and can be styled in various ways.

```dart
return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.green,
      title: const Text(
        "AITRICH",
        style: TextStyle(color: Colors.white),
      ), // Text
      centerTitle: true,
    ), // AppBar
    body: Container(
      width: 100,
      height: 100,
      padding: const EdgeInsets.all(8),
      margin: const EdgeInsets.all(16),
      decoration: BoxDecoration(
        color: Colors.blue,
        border: Border.all(color: Colors.black),
        borderRadius: BorderRadius.circular(12),
      ), // BoxDecoration
      child: const Center(child: Text('Hello')),
    ) // Container
    ); // Scaffold
```

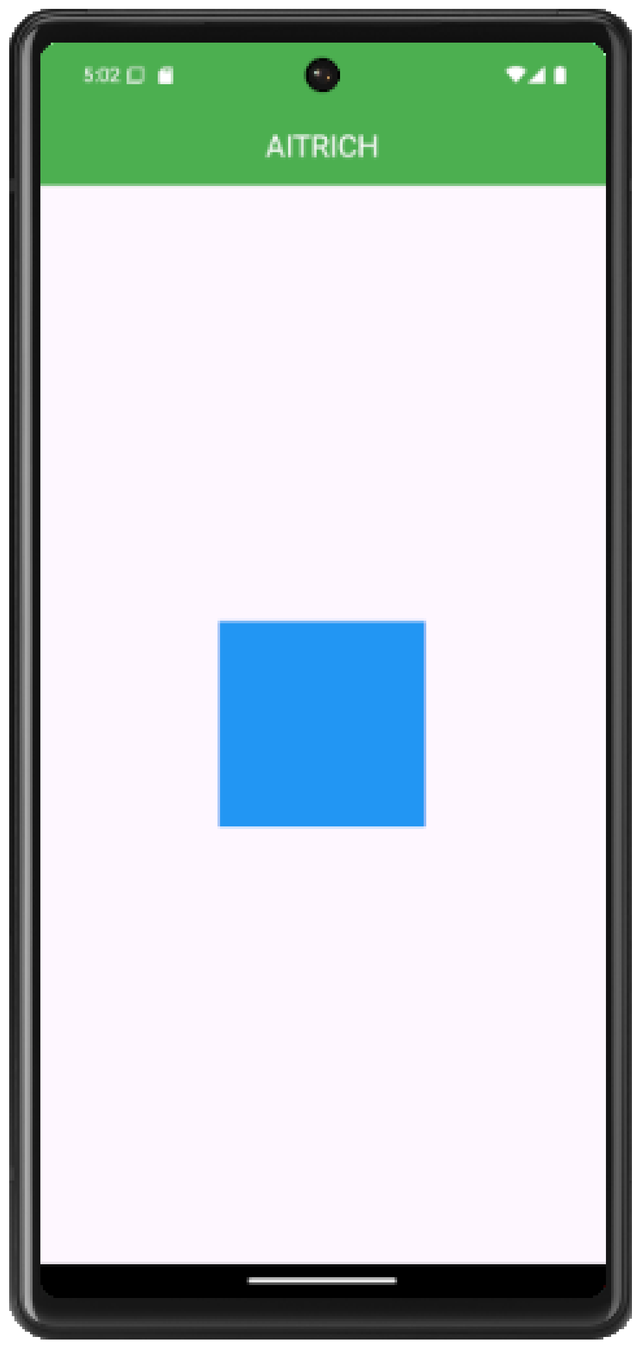
Key Feature

- **Size: Control the width and height.**
- **Padding: Adds space inside the container around its child.**
- **Margin: Adds space outside the container, separating it from other widgets.**
- **Decoration: Customizes appearance with background color, borders, and border radius.**
- **Alignment: Positions child widgets within the container.**

Center

In this code, the Center widget is used to position its child widget in the center of the available space.

```
body: Center(
  child: Container(
    color: Colors.blue,
    width: 150, // Width of the container
    height: 150, // Height of the container
  ), // Container
), // Center
```



- **Center: A widget that aligns its child widget both horizontally and vertically in the center of its parent.**
- **child: The widget that will be centered, in this case, a Container.**

Text

In Flutter, the Text widget is used to display a string of text with various styling options.

Text Widget:

- **Purpose: Displays a string of text.**
- **"Aitrich": The text content that will be shown.**

style Property (of Text):

- **Type: TextStyle**
- **Purpose: Defines the appearance of the text.**

TextStyle Widget:

- **color: Colors.red**
  - **Purpose: Sets the text color to red.**

fontSize: 40

  - **Purpose: Specifies the size of the text as 40 pixels.**

fontWeight: FontWeight.bold

  - **Purpose: Makes the text bold.**


```
body: const Center(
  child: Text(
    "Aitrich",
    style: TextStyle(
        color: Colors.red, fontSize: 40,
        fontWeight: FontWeight.bold), // TextStyle
  ), // Text
), // Center
```

Textfield

In Flutter, a TextField is a widget that allows users to input text. It can be used for various types of input, such as entering a username, password, or any other text-based data. You can customize its appearance, including its border, hint text, and whether the text is obscured or not.

```dart
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Padding(
        padding: const EdgeInsets.symmetric(horizontal: 20.0),
        child: TextField(
          decoration: InputDecoration(
            labelText: 'Username',
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(20.0),
            ), // OutlineInputBorder
          ), // InputDecoration
        ), // TextField
      ), // Padding
      const SizedBox(height: 20.0),
      Padding(
        padding: const EdgeInsets.symmetric(horizontal: 20.0),
        child: TextField(
          obscureText: true,
          decoration: InputDecoration(
            labelText: 'Password',
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(20.0),
            ), // OutlineInputBorder
          ), // InputDecoration
        ), // TextField
      ), // Padding
    ],
  ), // Column
), // Center
); // Scaffold
```

- **Center: Centers its child widget.**
- **Column: Arranges children vertically, centered in this case.**
- **Padding: Adds horizontal padding (20.0 pixels) around each TextField.**
- **TextField: Creates input fields; one for 'Username' and one for 'Password'.**
  - **Username Field: Shows text normally.**
  - **Password Field: Hides text with dots.**
- **SizedBox: Adds 20.0 pixels of vertical space between the two TextField widgets.**

mainAxisAlignment & crossAxisAlignment

In Flutter, mainAxisAlignment and crossAxisAlignment are properties used to control the alignment of child widgets within flex containers, such as Row and Column.

mainAxisAlignment

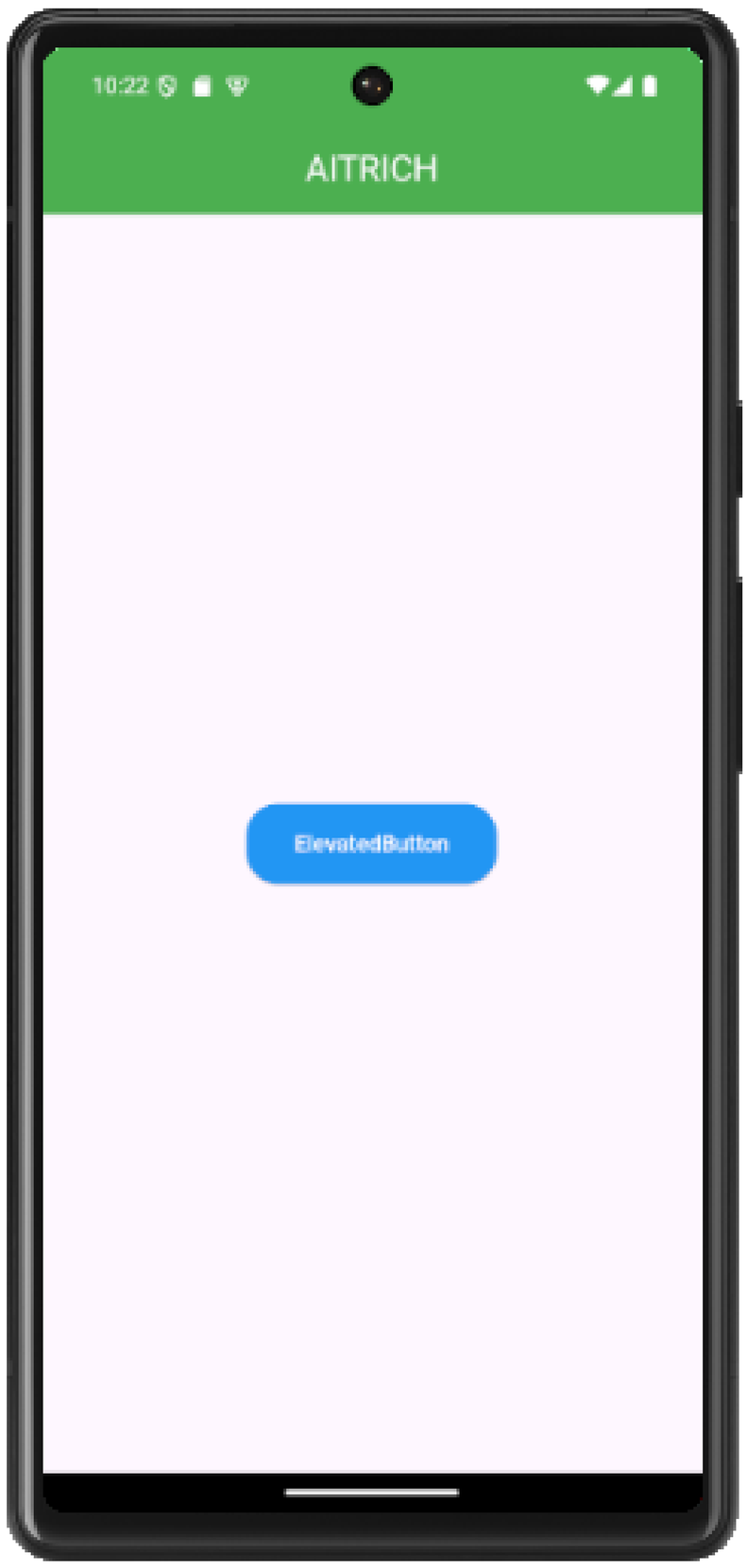
- **Purpose: Aligns children along the main axis of the container.**
- **Main Axis: For Row, the main axis is horizontal. For Column, it is vertical.**
- **Options:**
  - **MainAxisAlignment.start: Aligns children at the beginning of the main axis.**
  - **MainAxisAlignment.end: Aligns children at the end of the main axis.**
  - **MainAxisAlignment.center: Centers children along the main axis.**
  - **MainAxisAlignment.spaceBetween: Distributes children with space between them.**
  - **MainAxisAlignment.spaceAround: Distributes children with space around them.**
  - **MainAxisAlignment.spaceEvenly: Distributes children with equal space around them.**

crossAxisAlignment

- **Purpose: Aligns children along the cross axis of the container.**
- **Cross Axis: For Row, the cross axis is vertical. For Column, it is horizontal.**
- **Options:**
  - **CrossAxisAlignment.start: Aligns children at the beginning of the cross axis.**
  - **CrossAxisAlignment.end: Aligns children at the end of the cross axis.**
  - **CrossAxisAlignment.center: Centers children along the cross axis.**
  - **CrossAxisAlignment.stretch: Stretches children to fill the cross axis.**

Elevated Button

```
    ElevatedButton(
  onPressed: () {
    print('Button Pressed');
  },
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.blue,

    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(20),
    ), // RoundedRectangleBorder
    padding: const EdgeInsets.symmetric(horizontal: 30, vertical: 15),
  ),
  child: const Text(
    'ElevatedButton',
    style: TextStyle(color: Colors.white),
  ), // Text
), // ElevatedButton
```



- **ElevatedButton: This creates a button with a raised appearance.**
- **onPressed: Defines the action that occurs when the button is pressed. In this case, it prints "Button Pressed" to the console.**
- **style: Customizes the button's appearance using the styleFrom method.**
- **backgroundColor: Sets the button's background color to blue.**
- **shape: Adjusts the button's shape by rounding the corners.**
- **borderRadius: Specifies a corner radius of 20, making the button's corners rounded.**
- **padding: Adds internal spacing inside the button, with 30 pixels on the sides and 15 pixels on the top and bottom.**
- **child: Defines the content inside the button, which is a text label in this case.**
- **Text: Displays the label "ElevatedButton" inside the button.**
- **TextStyle: Sets the text color to white.**

This creates a blue button with rounded corners and padding, displaying white text that reads "ElevatedButton." When pressed, it prints a message to the console.

Text Button

A TextButton in Flutter is a simple, flat button that displays only text. It's used for less prominent actions and has no elevation or background by default. It’s typically used for secondary actions or inline buttons.

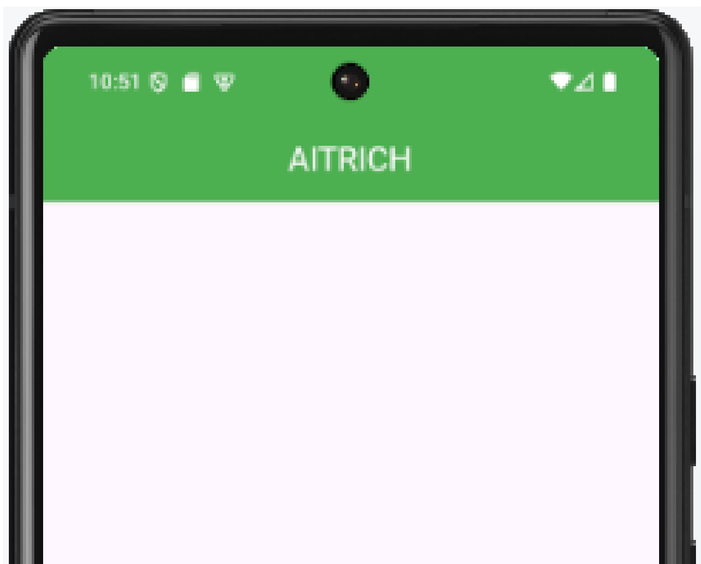
```dart
TextButton(
    onPressed: () {
      // Action to perform when the button is pressed
      print('TextButton pressed!');
    },
    child: Text('Click Me'),
  ), // TextButton
```

- **TextButton:**
  - **Purpose: Creates a button that displays text and does not have an elevated or outlined appearance.**
- **onPressed:**
  - **Type: VoidCallback (a function with no parameters and no return value).**
  - **Purpose: Specifies the action to be performed when the button is tapped.**
  - **Example: The code inside the curly braces {} will be executed when the button is pressed. In this case, it prints 'TextButton pressed!' to the console.**
- **child:**
  - **Type: Widget.**
  - **Purpose: Defines the content inside the button. For TextButton, it is typically a Text widget.**

Example: const Text('Click Me') creates a Text widget with the label "Click Me". The const keyword indicates that this widget is a constant and can be optimized by Flutter for better performance.

Icon Button

In Flutter, an IconButton is a widget that displays a clickable icon. It is used to trigger actions in the user interface when tapped.



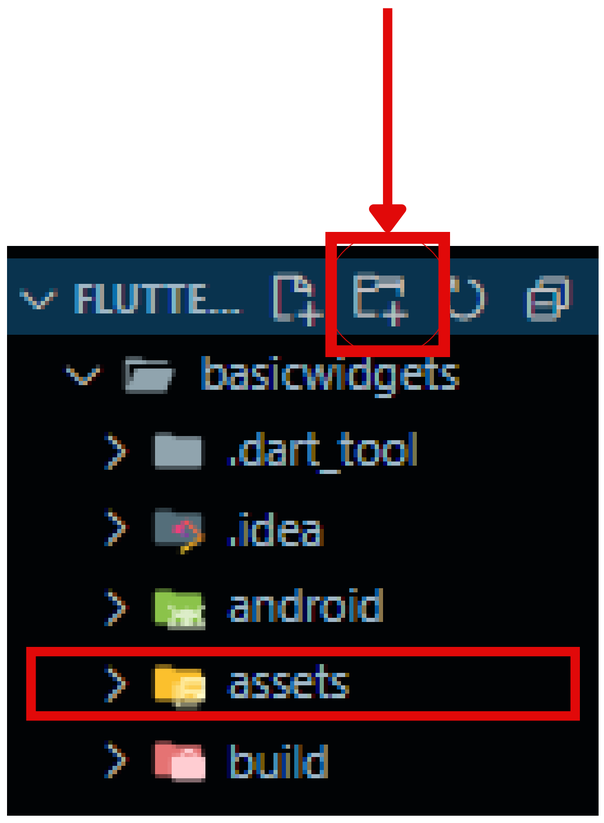

```
    IconButton(
  icon: const Icon(Icons.favorite), // The icon to display
  onPressed: () {
    // Action to perform when the button is pressed
    print('Icon Button Pressed');
  },
  color: Colors.red, // Color of the icon
  iconSize: 50.0, // Size of the icon
) // IconButton
```

1. **IconButton:**
   - **IconButton: This is the widget being created. It displays an icon that can be tapped to trigger an action.**
2. **icon:**
   - **icon: const Icon(Icons.favorite): This specifies the icon to be displayed in the button.**
     - **const Icon(Icons.favorite): Creates an Icon widget with a heart symbol (using Icons.favorite).**
     - **const: Marks the Icon as a compile-time constant, which can help with performance.**
3. **onPressed:**
   - **onPressed: () { ... }: This defines what happens when the button is pressed.**
     - **() { ... }: An anonymous function that executes when the button is tapped.**
     - **print('Icon Button Pressed');: This line of code prints the message "Icon Button Pressed" to the console.**
4. **color:**
   - **color: Colors.red: Sets the color of the icon.**
     - **Colors.red: Specifies the icon color as red.**
5. **iconSize:**
   - **iconSize: 50.0: Defines the size of the icon in pixels.**
     - **50.0: The icon will be 50 pixels wide and 50 pixels tall.**

Asset image

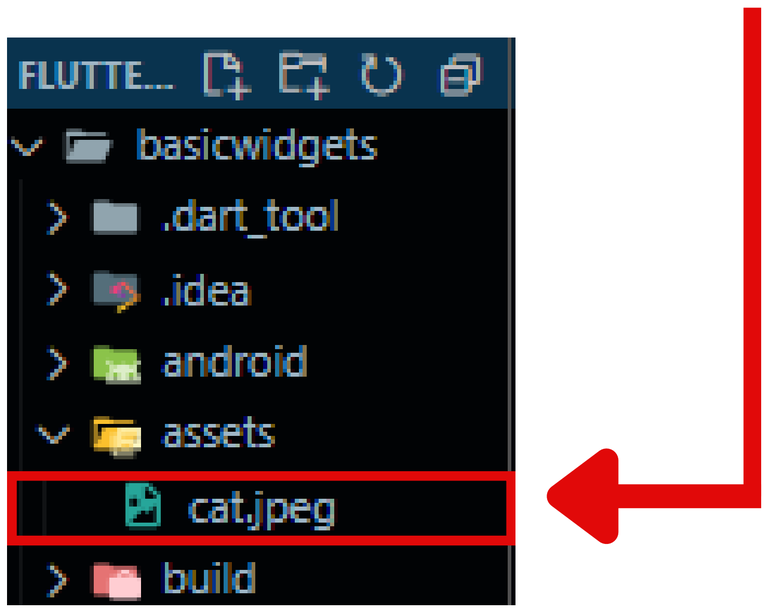
In Flutter, an assets image refers to images that are bundled with your app and stored in a designated directory within your project. These images are packaged into the app's binary, making them readily available at runtime.

Press this to create a new folder.



Create a folder in your project named assets, as shown in the given example screenshot.

While creating a new Flutter folder named 'assets,' add an image to that folder.



After adding the image to the folder, go to pubspec.yaml in the project and make some changes under the assets: section.

```yaml
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  assets:
    - assets/cat.jpeg
  #   - images/a_dot_ham.jpeg
```

After making the changes in pubspec.yaml, the next step is to display the image in the app. Here is your code and output :

```dart
Center(
  child: Image.asset('assets/cat.jpeg'),
), // Center
```

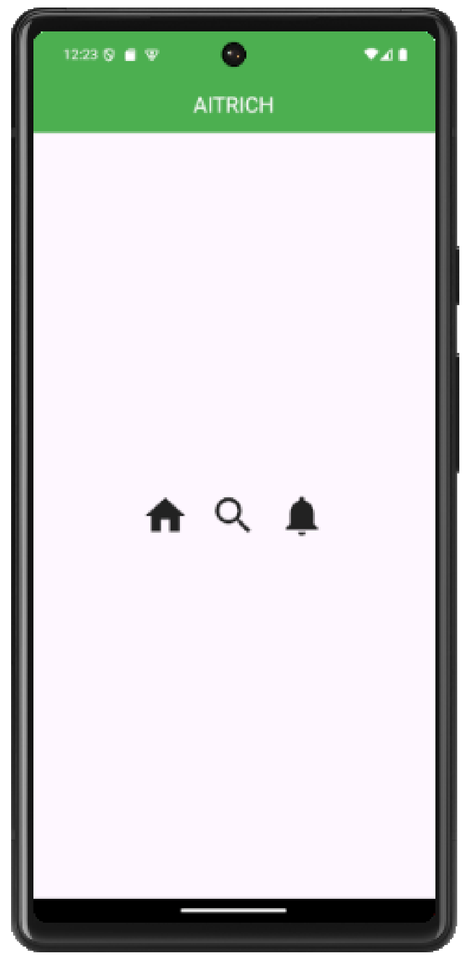
Row

In Flutter, a Row is a widget that arranges its children horizontally in a line.

Key Points:

- **children: List of widgets displayed side by side.**
- **mainAxisAlignment: Controls horizontal alignment (e.g., start, center, spaceBetween).**
- **crossAxisAlignment: Controls vertical alignment (e.g., center, start).**
- **mainAxisSize: Defines the space the row takes (e.g., max, min).**

```dart
Row(
  mainAxisAlignment:
      MainAxisAlignment.center, // Centers the icons horizontally
  children: [
    Icon(Icons.home, size: 50.0), // First icon
    SizedBox(width: 20.0), // Adds space between icons
    Icon(Icons.search, size: 50.0), // Second icon
    SizedBox(width: 20.0), // Adds space between icons
    Icon(Icons.notifications, size: 50.0), // Third icon
  ],
), // Row
```



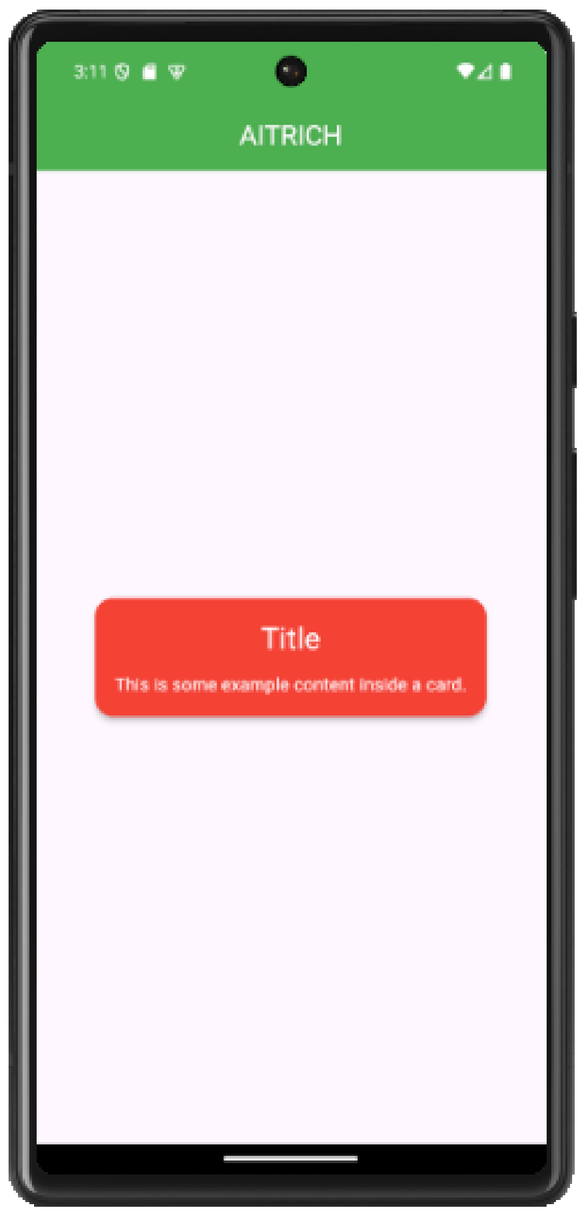
Card

In Flutter, a Card widget is a container that provides a rectangular area with rounded corners, elevation (shadow), and optional padding. It's often used to visually group related content and make it stand out from the rest of the UI.

Key Features of the Card Widget:

- **Elevation: Adds a shadow beneath the card, giving it a raised effect.**
- **Rounded Corners: By default, the card has slightly rounded corners.**
- **Padding: You can add padding inside the card to control the spacing between the card's content and its edges.**
- **Content: You can place any widget inside the Card, like Text, Image, Column, or ListTile.**

```dart
Card(
 color: Colors.red,
 elevation: 4, // Elevation for shadow effect
 shape: RoundedRectangleBorder(
   borderRadius: BorderRadius.circular(15),
 ), // RoundedRectangleBorder
 child: const Padding(
   padding: EdgeInsets.all(16.0),
   child: Column(
     mainAxisSize: MainAxisSize.min,
     children: [
       Text(
         'Title',
         style: TextStyle(fontSize: 24, color: Colors.white),
       ), // Text
       SizedBox(height: 10),
       Text(
         'This is some example content inside a card.',
         style: TextStyle(color: Colors.white),
       ), // Text
     ],
   ), // Column
 ), // Padding
), // Card
```



This code creates a Flutter Card widget with the following key features:

1. **Background Color: The card has a red background (color: Colors.red).**
2. **Elevation: It has a shadow effect with elevation set to 4 (elevation: 4).**
3. **Rounded Corners: The corners are rounded with a 15-pixel radius (shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(15))).**
4. **Padding: The content inside the card has padding of 16 pixels on all sides (Padding(padding: EdgeInsets.all(16.0))).**
5. **Content Layout: Inside the card, a Column arranges a title and content vertically.**
6. **Styling: The title is styled with a larger font (24 pixels) and both texts are white (TextStyle(fontSize: 24, color: Colors.white)).**

Circle Avatar

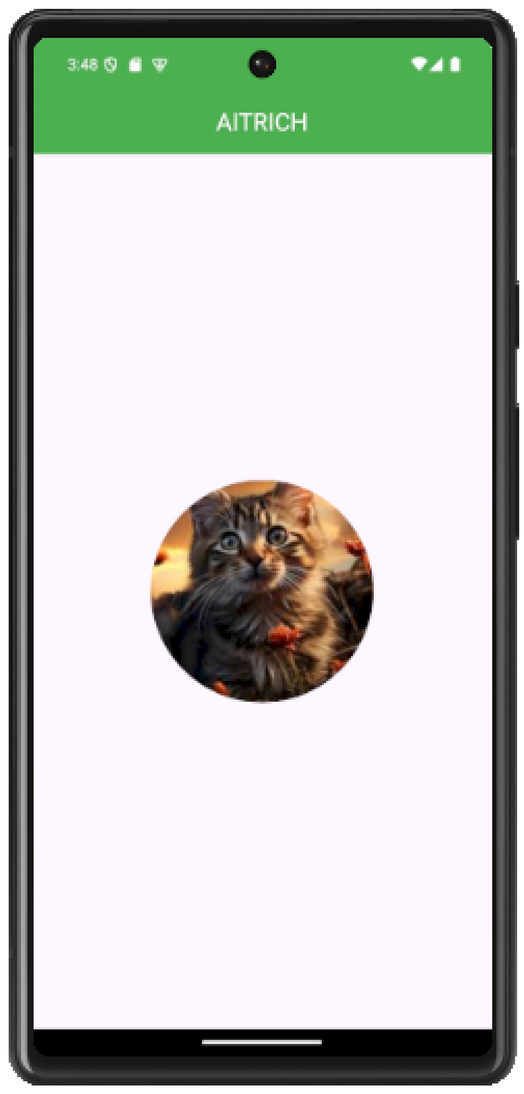
In Flutter, a CircleAvatar is a widget that displays a circular image or icon. It's commonly used for user profile pictures or other small, circular images.

Key Features of CircleAvatar:

- **Circular Shape: Automatically crops the child (usually an image or icon) into a circular shape.**
- **Size Control: The size of the circle is controlled by the radius property.**
- **Background Image: You can set an image from your assets or network as the background using backgroundImage.**
- **Child Widget: It can display text or an icon if no image is provided.**

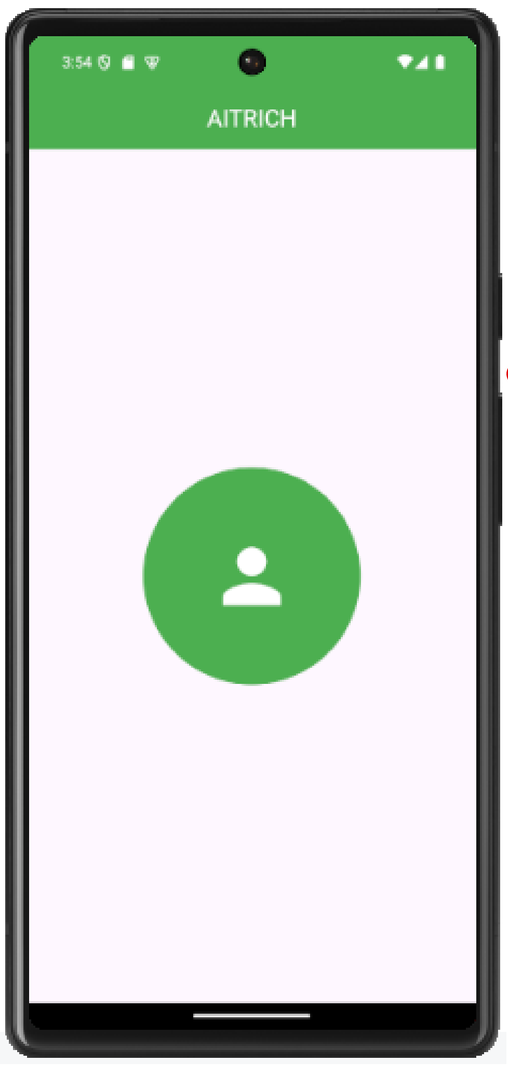
CircleAvatar with Image

```
  CircleAvatar(
  radius: 100,
  backgroundImage: AssetImage('assets/cat.jpeg'),
) // CircleAvatar
```



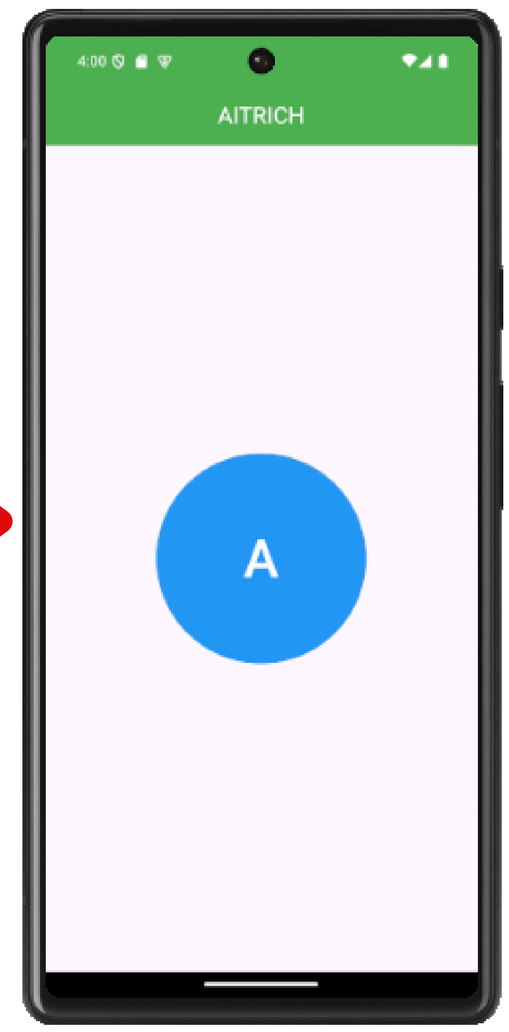
CircleAvatar with Icon

```
  CircleAvatar(
  radius: 100,
  backgroundColor: Colors.green,
  child: Icon(Icons.person,
    color: Colors.white,
    size: 80,
  ), // Icon
) // CircleAvatar
```



CircleAvatar with Text

```
  CircleAvatar(
  radius: 100,
  backgroundColor: Colors.blue,
  child: Text(
    'A',
    style: TextStyle(color: Colors.white, fontSize: 50),
  ), // Text
) // CircleAvatar
```



- **radius: Defines the radius of the circle.**
- **backgroundImage: Sets an image as the background.**
- **backgroundColor: Specifies the color behind the image if no backgroundImage is set.**
- **child: Allows for a widget (like text or an icon) to be displayed inside the circle.**

FloatingActionButton

A FloatingActionButton in Flutter is a circular button that floats above the content of the screen. It is typically used for primary actions, such as adding new items or initiating important tasks. It’s positioned in the bottom right corner by default but can be customized.

```dart
floatingActionButton: FloatingActionButton(
  onPressed: () {
    // Action to be performed when the button is pressed
  },
  backgroundColor: Colors.red,
  child: const Icon(
    Icons.add,
    color: Colors.white,
  ), // Icon
), // FloatingActionButton
```

This code configures a FloatingActionButton in Flutter:

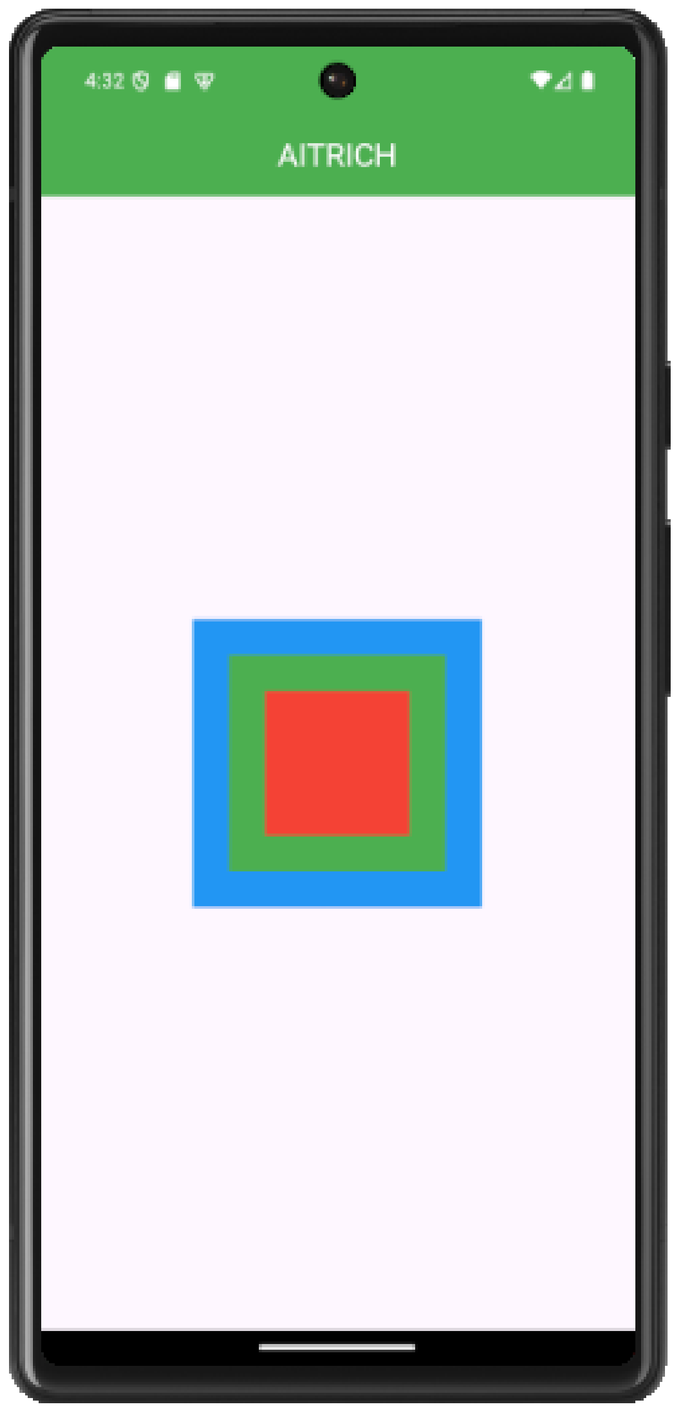
- **onPressed: Defines the action to perform when the button is pressed (currently empty).**
- **backgroundColor: Colors.red Sets the button’s background color to red.**
- **child: Icon(Icons.add, color: Colors.white) Displays a white plus sign icon on the button.**

In Flutter, you can position the FloatingActionButton using the floatingActionButtonLocation property in a Scaffold. Here are the key locations:

- **endFloat: Bottom-right corner (default).**
- **centerFloat: Centered at the bottom.**
- **endTop: Top-right corner.**
- **startTop: Top-left corner.**
- **startFloat: Bottom-left corner.**

Stack

In Flutter, the Stack widget allows you to overlay widgets on top of each other. Here's a simple example that demonstrates how to use Stack to position widgets:



Explanation:

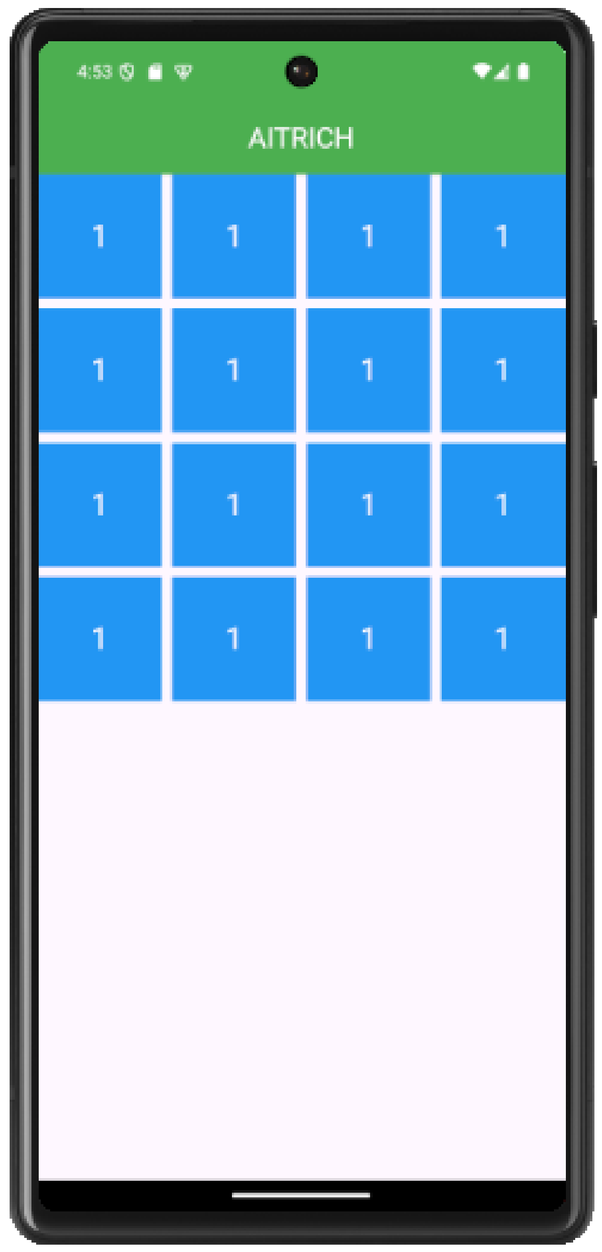
1. Stack: The Stack widget allows for layering of child widgets.
2. alignment: Alignment.center: Aligns all children widgets to the center of the stack.
3. children: List of widgets to be stacked:
   - Bottom Layer: A blue container of size 200x200.
   - Middle Layer: A green container of size 150x150, which will be placed on top of the blue container.
   - Top Layer: A red container of size 100x100, which will be placed on top of the green container.

```dart
Stack(
 alignment: Alignment.center,
 children: [
   // First widget (bottom layer)
   Container(
     width: 200,
     height: 200,
     color: Colors.blue,
   ), // Container
   // Second widget (middle layer)
   Container(
     width: 150,
     height: 150,
     color: Colors.green,
   ), // Container
   // Third widget (top layer)
   Container(
     width: 100,
     height: 100,
     color: Colors.red,
   ), // Container
 ],
), // Stack
```

Grid

In Flutter, a Grid refers to a layout structure where items are arranged in a two-dimensional matrix of rows and columns. It is useful for displaying a collection of items in a grid-like pattern.

```dart
GridView.builder(
  gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 4, // Number of columns in the grid
    crossAxisSpacing: 8.0, // Horizontal spacing between items
    mainAxisSpacing: 8.0, // Vertical spacing between items
  ), // SliverGridDelegateWithFixedCrossAxisCount
  itemCount: 16, // Total number of items (4x4 grid)
  itemBuilder: (context, index) {
    return Container(
      alignment: Alignment.center,
      color: Colors.blue,
      child: const Text(
        '1', // Number to display
        style: TextStyle(
          fontSize: 24,
          color: Colors.white,
        ), // TextStyle
      ), // Text
    ); // Container
  },
), // GridView.builder
```

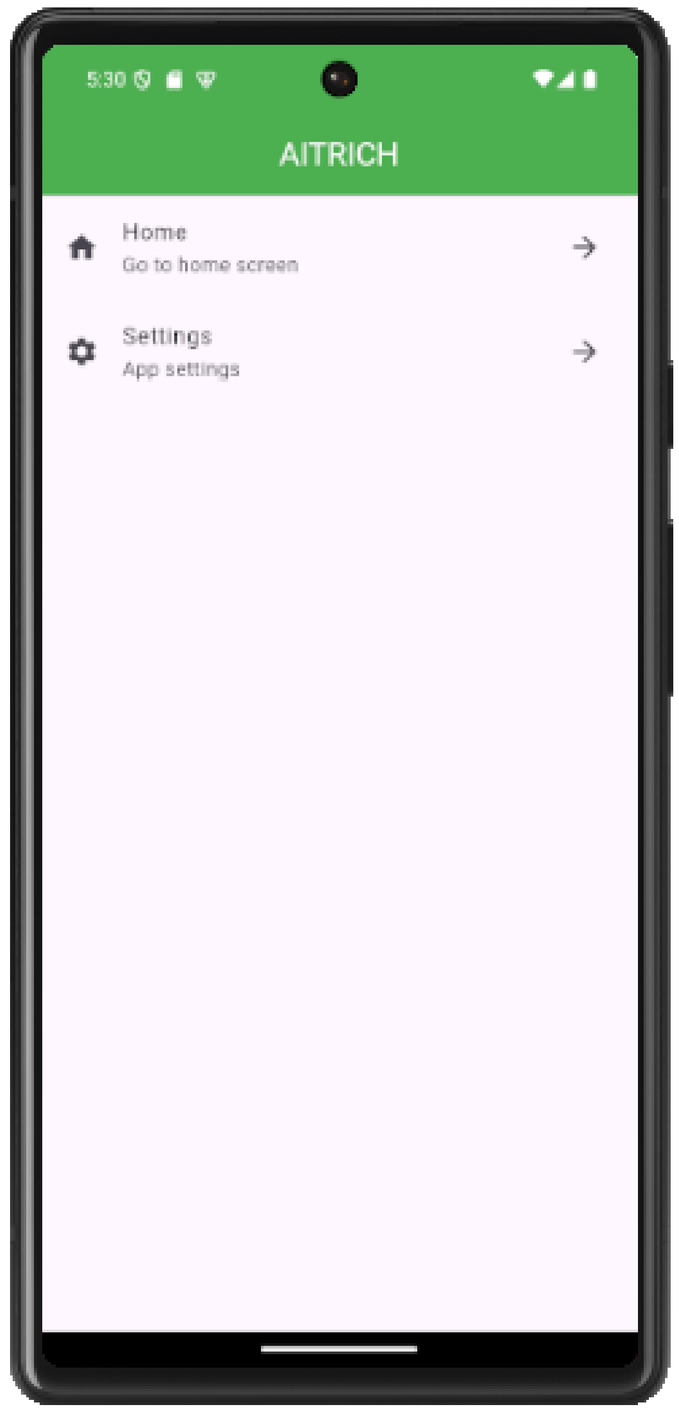


- GridView.builder: This creates a grid of items where you can efficiently generate each item as needed.
- gridDelegate: Defines how the grid is arranged:
- 4 columns: The grid will have 4 columns.
- 8 pixels spacing: There will be 8 pixels of space between items both horizontally and vertically.
- itemCount: 16: There will be a total of 16 items in the grid, making it a 4x4 grid.
- itemBuilder: This function creates each grid item:
- Each item is a blue box with a white number '1' in the center.

ListTile

In Flutter, the ListTile widget is used to create a single row in a list, often containing a leading icon, a title, a subtitle, and an optional trailing widget. It's commonly used in lists, settings, and menus.

```dart
ListView(
 children: [
  ListTile(
   leading: const Icon(Icons.home), // Leading icon
   title: const Text('Home'), // Main text
   subtitle: const Text('Go to home screen'), // Subtitle
   trailing: const Icon(Icons.arrow_forward), // Trailing icon
   onTap: () {
    // Action when tapped
    print('Home tapped');
   },
  ), // ListTile
  ListTile(
   leading: const Icon(Icons.settings),
   title: const Text('Settings'),
   subtitle: const Text('App settings'),
   trailing: const Icon(Icons.arrow_forward),
   onTap: () {
    print('Settings tapped');
   },
  ), // ListTile
  // Add more ListTile items as needed
 ],
), // ListView
```



Explanation

- ListTile: The primary widget used to create a row in the list.
  - leading: A widget displayed at the start of the tile, often an icon or an image.
  - title: The main text displayed in the tile.
  - subtitle: An optional text displayed below the title.
  - trailing: A widget displayed at the end of the tile, often an icon or button.
  - onTap: A callback function that is called when the tile is tapped.

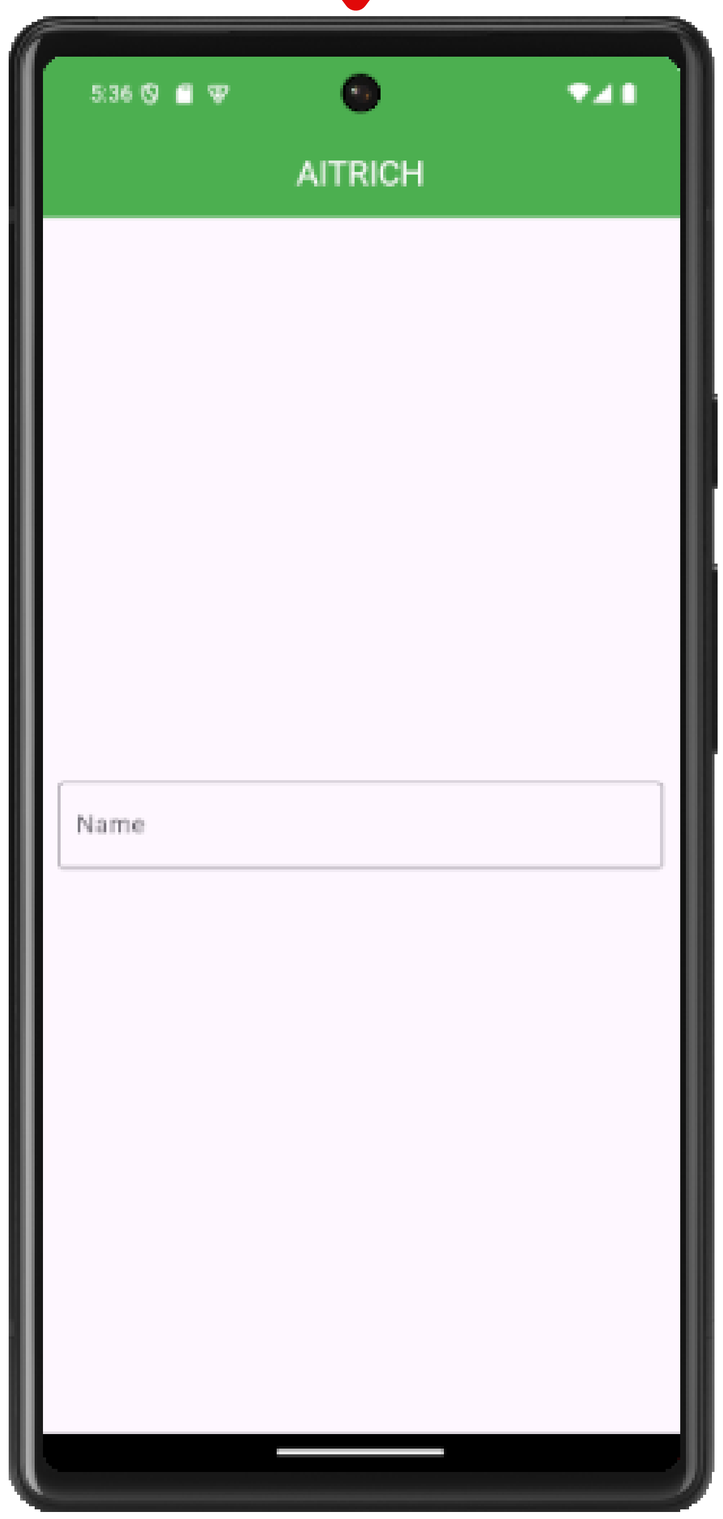
Properties

- leading: A widget displayed before the title. Commonly an Icon or Image.
- title: The main text in the ListTile.
- subtitle: Additional text displayed below the title.
- trailing: A widget displayed after the title. Typically an Icon, Switch, or Button.
- onTap: A callback function for handling tap events.

TextField

In Flutter, the TextField widget is used to create an editable text input field. It allows users to enter and edit text in a form or other UI components.

```dart
Padding(
  padding: const EdgeInsets.all(10.0),
  child: TextField(
    decoration: const InputDecoration(
      labelText: 'Name', // Label for the text field
      border:
          OutlineInputBorder(), // Adds a border around the text field
    ), // InputDecoration
    onChanged: (text) {
      // Handle text input changes here
      print('Name entered: $text');
    },
  ), // TextField
), // Padding
```



Padding:

- Purpose: Adds padding around the child widget to provide spacing from its surrounding elements.
- padding: const EdgeInsets.all(10.0), Specifies padding of 10 logical pixels on all sides of the TextField.

TextField:

- Purpose: Provides a field where users can enter and edit text.

decoration: const InputDecoration(

- Purpose: Customizes the appearance of the TextField.
  - labelText: 'Name', Adds a label above the text field that indicates the purpose of the field (in this case, for entering a name). The label appears inside the TextField when it is empty and moves above the field when it has content.
  - border: OutlineInputBorder(), Applies a border around the TextField. The OutlineInputBorder() creates a rectangular border with rounded corners.

onChanged: (text) {

- Purpose: Defines a callback function that is triggered whenever the text in the TextField changes.
  - (text) Parameter that represents the current value of the TextField.
  - print('Name entered: $text'); Prints the updated text to the console whenever the user types or modifies the input. This is useful for debugging or handling real-time text changes.

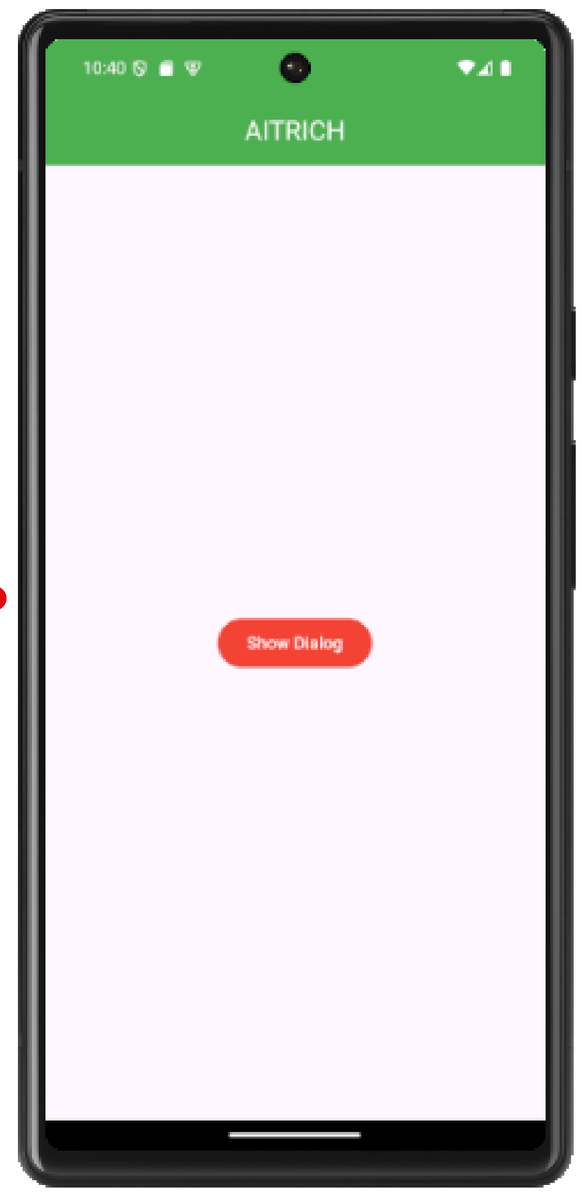
- **Padding: Adds spacing around the TextField.**
- **TextField: Allows user input with a label and a border.**
- **Decoration: Customizes the appearance of the text field.**
- **OnChanged: Handles text input changes and prints the updated text.**

Dialogs

In Flutter, a Dialog is a pop-up UI component that appears on top of the current screen to provide information, ask for confirmation, or interact with the user. Dialogs are typically modal, which means they block user interaction with the rest of the app until they are dismissed or interacted with.

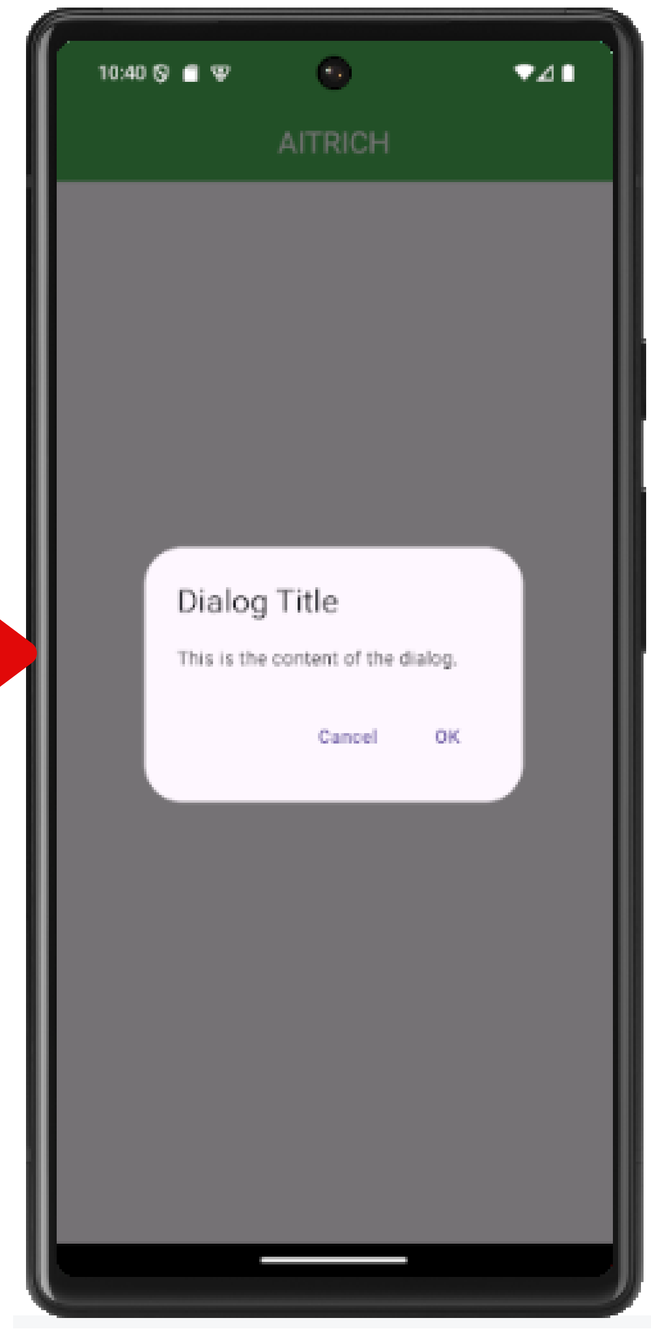
Elevated Button with out Dialogs

```dart
child: ElevatedButton(
  onPressed: () {},
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.red,
  ),
  child: const Text(
    'Show Dialog',
    style: TextStyle(color: Colors.white),
  ), // Text
), // ElevatedButton
```



Elevated Button with Dialogs

```dart
    ElevatedButton(
  onPressed: () {
    // Show the dialog when the button is pressed
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: const Text('Dialog Title'),
          content: const Text('This is the content of the dialog.'),
          actions: <Widget>[
            TextButton(
              child: const Text('Cancel'),
              onPressed: () {
                // Close the dialog
                Navigator.of(context).pop();
              },
            ), // TextButton
            TextButton(
              child: const Text('OK'),
              onPressed: () {
                // Perform any action, then close the dialog
                Navigator.of(context).pop();
              },
            ), // TextButton
          ], // <Widget>[]
        ); // AlertDialog
      },
    );
  },
  child: const Text('Show Dialog'),
), // ElevatedButton
```



This Flutter code creates a red ElevatedButton that, when pressed, shows an alert dialog with a title, some content, and two action buttons: "Cancel" and "OK." Below is a step-by-step explanation of the code:

ElevatedButton:

- **Purpose: This widget creates a button with elevation (a shadow effect) that responds to user interactions.**

onPressed:

- **Purpose: Defines the action to be performed when the button is pressed.**
- **Inside onPressed:**
  - **showDialog: This function is called when the button is pressed. It displays a modal dialog on the screen.**
    - **context: Provides the context in which the dialog is shown.**
    - **builder: A function that returns the dialog widget.**

AlertDialog:

- **Purpose: This widget represents the dialog that pops up when the button is pressed. It typically contains a title, content, and action buttons.**

Dialog Components:

1. **title:**
   - **Text('Dialog Title'): Displays the title of the dialog. The text is constant (const), meaning it won't change during runtime.**
2. **content:**
   - **Text('This is the content of the dialog'): Displays the main content of the dialog, also constant text.**
3. **actions:**
   - **Purpose: These are the buttons or actions available to the user within the dialog.**
   - **TextButton:**
     - **Purpose: A flat button typically used for simple actions.**
     - **child: Text('Cancel'): The label of the button is "Cancel."**
     - **onPressed: When this button is pressed, the dialog is closed using Navigator.of(context).pop().**
   - **Second TextButton:**
     - **Purpose: Similar to the first button, but labeled "OK."**
     - **onPressed: When pressed, it performs any desired action and then closes the dialog with Navigator.of(context).pop().**

style:

- **Purpose: Customizes the appearance of the ElevatedButton.**
- **ElevatedButton.styleFrom: This method allows for style customization.**
  - **backgroundColor: Colors.red: Sets the background color of the button to red.**

child:

- **Purpose: Defines the content inside the button.**
- **Text('Show Dialog'): This is the text displayed on the button.**
  - **style: TextStyle(color: Colors.white): This applies a white color to the text inside the button.**