



# Aula 3 – Widgets e Estado

Desenvolvimento de aplicações híbridas com Flutter

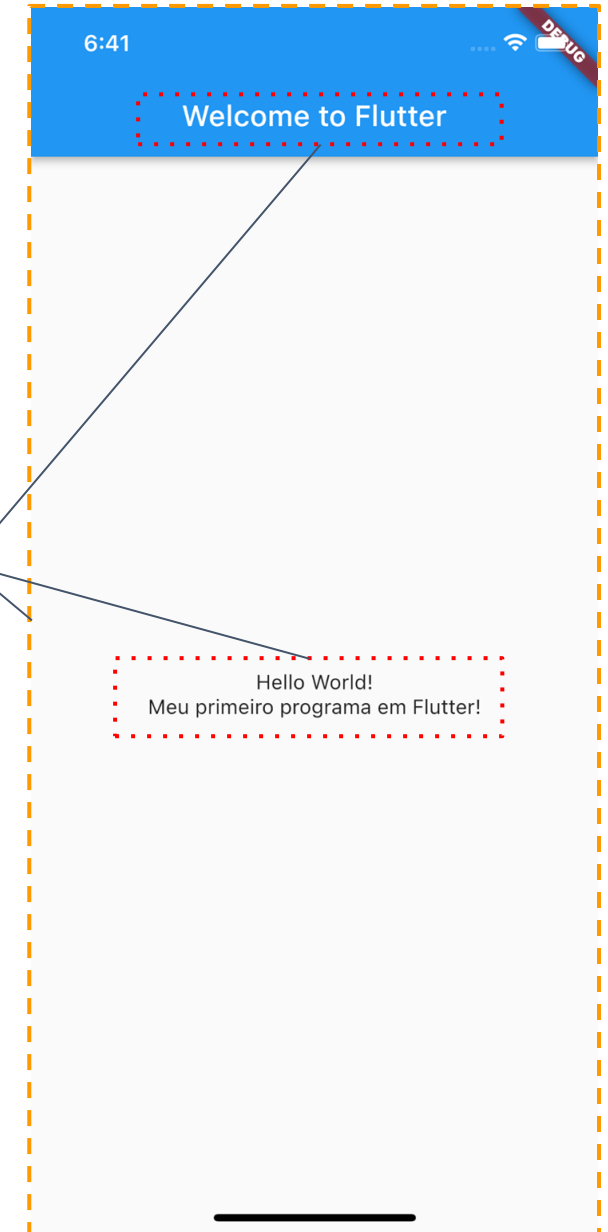
22-23 de Outubro/21

Widget e UI

# Widget e UI

- Widgets são como “blocos de construção” responsáveis por construir a UI.
- Quase tudo na UI é definido por Widgets (Ex. Componentes, Animações, Ajustes de layout)

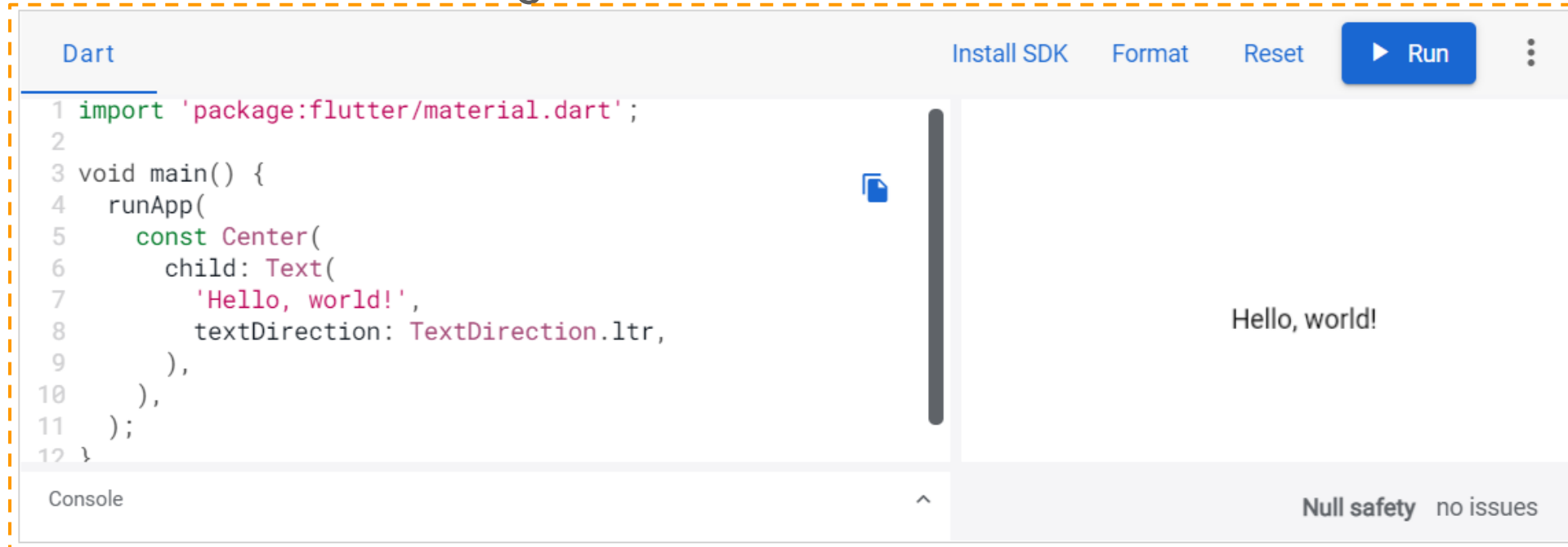
Widgets



# Widget e UI

## Hello Widget

1. Importar as dependências
2. Chamar a função 'runApp' do framework, passando o Widget que será a raiz da Árvore de Widgets



```
Dart
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(
5     const Center(
6       child: Text(
7         'Hello, world!',
8         textDirection: TextDirection.ltr,
9       ),
10    ),
11  );
12 }
```

Install SDK   Format   Reset   ▶ Run   ⋮

Hello, world!

Console   ^   Null safety   no issues

# Widget e UI

Por baixo dos panos...

Árvore de  
Widgets

Center

Outros Widgets...

Text

Árvore de  
Elementos

Stateless  
Element

Outros  
Elementos...

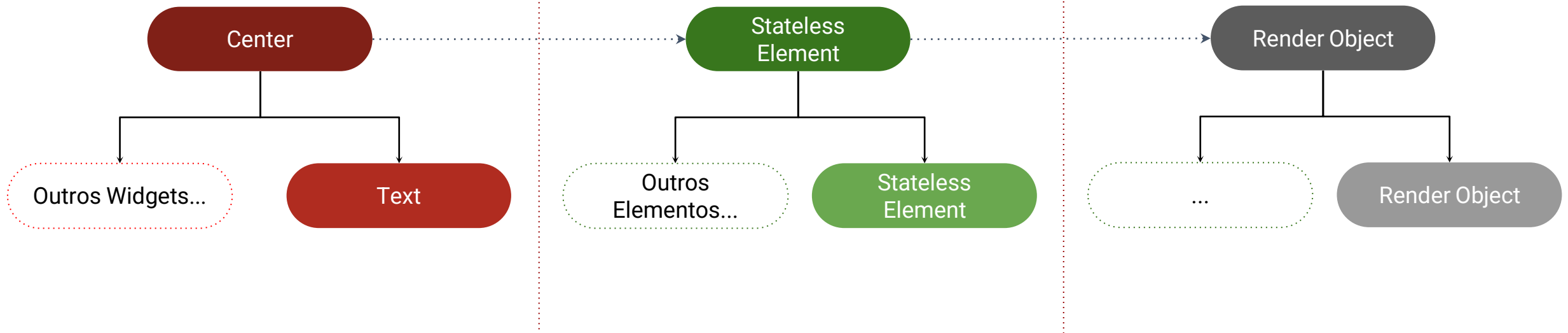
Stateless  
Element

Árvore de  
Renderização

Render Object

...

Render Object



# Widget e UI

Alguns Widgets...

# Widget e UI

## Container

Cria um layout retangular, que serve como um Wrapper para um Widget filho.

*Parâmetros:*

- width -> double
- height -> double
- color -> Color
- padding -> EdgeInsetsGeometry
- margin -> EdgeInsetsGeometry
- child -> Widget
- decoration -> Decoration
- alignment -> Alignment

# Widget e UI

## Text

Cria um texto na tela.

*Alguns Parâmetros:*

- text -> String
- textStyle -> TextStyle
- textAlign -> TextAlign
- textDirection -> TextDirection





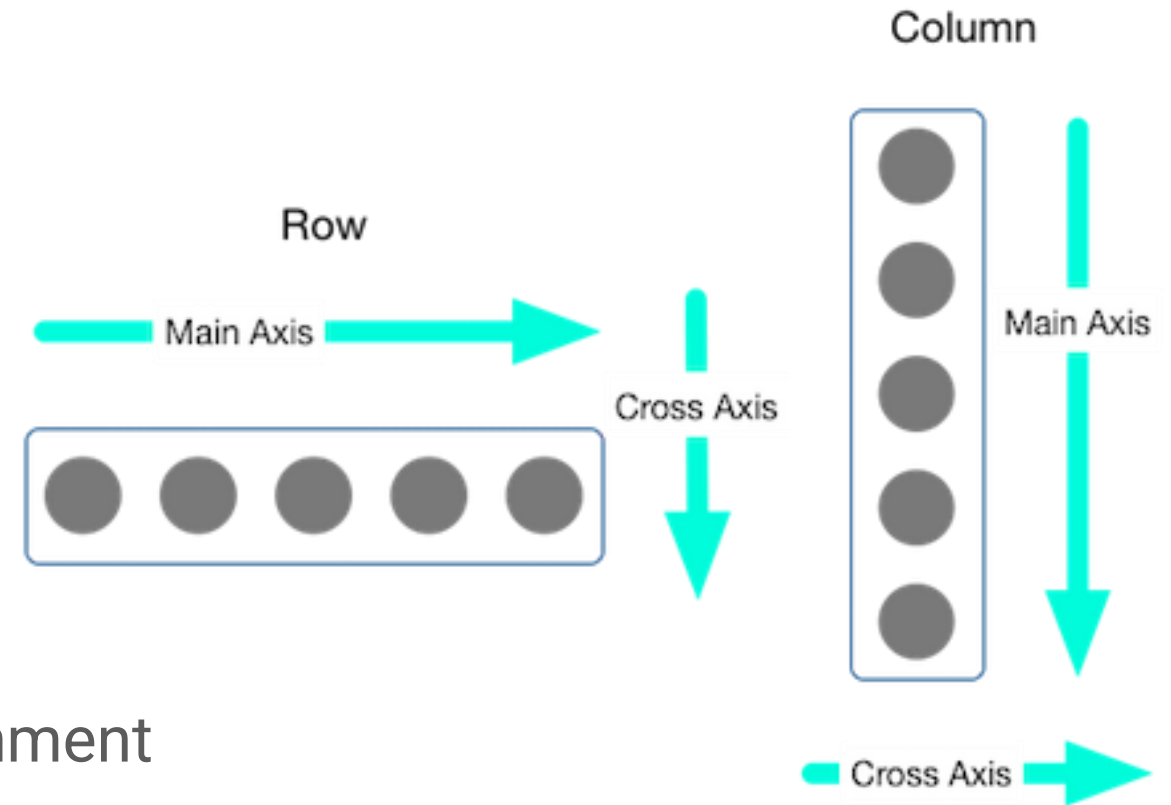
# Widget e UI

## Row, Column

Criam layouts com os itens alinhados na vertical(Column) e horizontal(Row).

*Alguns Parâmetros:*

- children -> List<Widget>
- direction -> Axis
- mainAxisAlignment -> MainAxisAlignment



# Widget e UI

## Stack

Permite empilhar widgets um na frente do outro.

*Alguns Parâmetros:*

- children -> List<Widget>
- alignment -> AlignmentGeometry

# Exercício 1: Widgets e UI

Criar uma tela com quatro textos organizados como um grid 2x2. Os textos devem ter o fundo azul e uma margem entre eles.

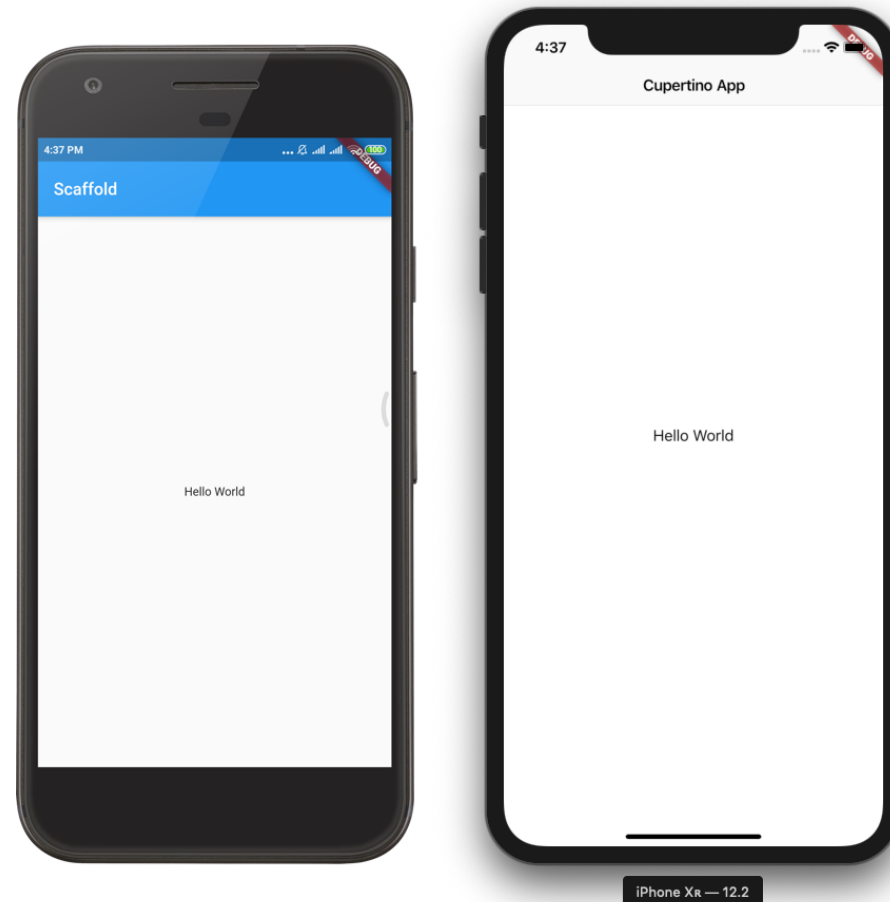
Ex:

Texto 1      Texto 2

Texto 3      Texto 4

# Widget e UI

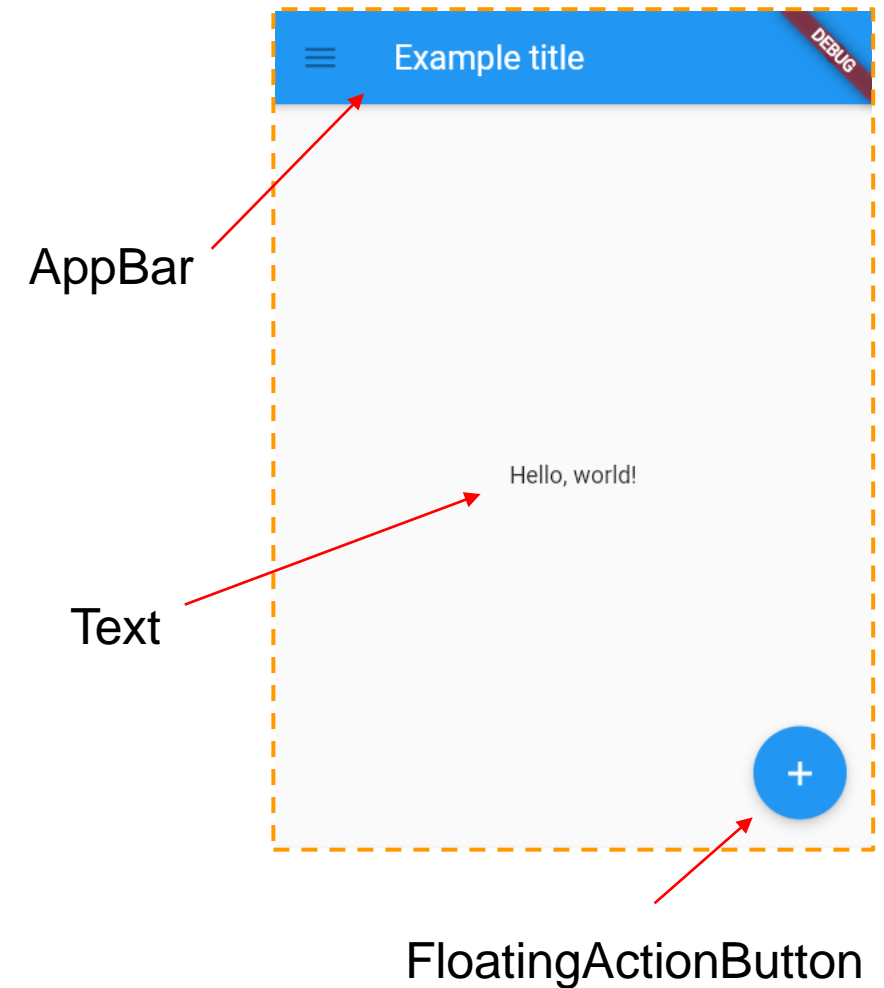
## Material Design vs Cupertino



# Widget e UI

## Usando Material Design

- Conjunto de Widgets que aplicam os Guidelines do Material Design na UI.
- Esses Widgets replicam os componentes típicos do ambiente Android.



# Widget e UI

## MaterialApp

Permite aplicar o estilo do Material Design na aplicação, por isso, fica geralmente na raiz da árvore de Widgets.

### *Alguns Parâmetros:*

- title -> String
- home -> Widget
- theme → ThemeData

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

void main() {
  runApp(
    MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Exemplo Material"),
        ), // AppBar
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: const <Widget> [
              Text(
                'Texto Centralizado',
              ) // Text
            ], // <Widget>[]
          ), // Column
        ), // Center
      ), // Scaffold
    ) // MaterialApp
  );
}
```

# Widget e UI

## ThemeData

Define o tema global no Material App

*Alguns Parâmetros:*

- primaryColor -> Color
- primaryColorDark -> Color
- accentColor → Color

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

void main() {
  runApp(
    MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Exemplo Material"),
        ), // AppBar
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: const <Widget> [
              Text(
                'Texto Centralizado',
              ) // Text
            ], // <Widget>[]
          ), // Column
        ), // Center
      ), // Scaffold
    ) // MaterialApp
  );
}
```

# Widget e UI

## Scaffold

Define uma estrutura básica de layout. Com ele podemos definir uma AppBar, um Body entre outros.

### *Alguns Parâmetros:*

- appBar -> PreferredSizeWidget
- body -> Widget
- drawer -> Widget

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

void main() {
  runApp(
    MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Exemplo Material"),
        ), // AppBar
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: const <Widget> [
              Text(
                'Texto Centralizado',
              ) // Text
            ], // <Widget>[]
          ), // Column
        ), // Center
      ), // Scaffold
    ) // MaterialApp
  );
}
```



# Widget e UI

## AppBar

Cria uma barra de ação, típica do Material Design.

*Alguns Parâmetros:*

- actions -> List<Widget>?
- backgroundColor -> Color?

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

void main() {
  runApp(
    MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: Scaffold(
        appBar: AppBar(
          title: const Text("Exemplo Material"),
        ), // AppBar
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: const <Widget> [
              Text(
                'Texto Centralizado',
              ) // Text
            ], // <Widget>[]
          ), // Column
        ), // Center
      ), // Scaffold
    ) // MaterialApp
  );
}
```

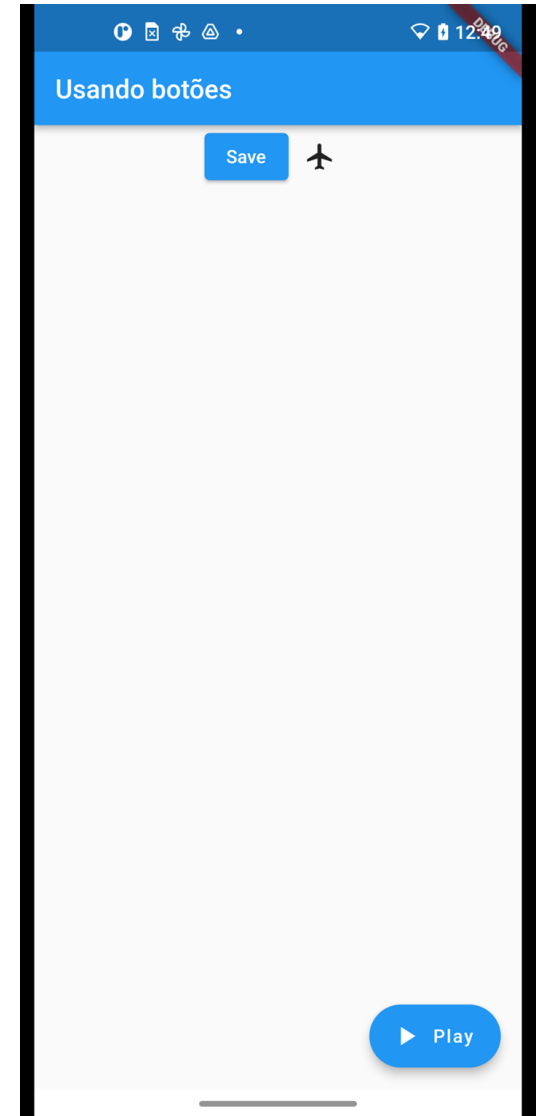
# Widget e UI

## Botão

Existe uma variedade de Widgets de botão como: FloatingActionButton, ElevatedButton, IconButton

*Alguns Parâmetros:*

- onPressed -> VoidCallback
- icon-> Icon
- label -> Text



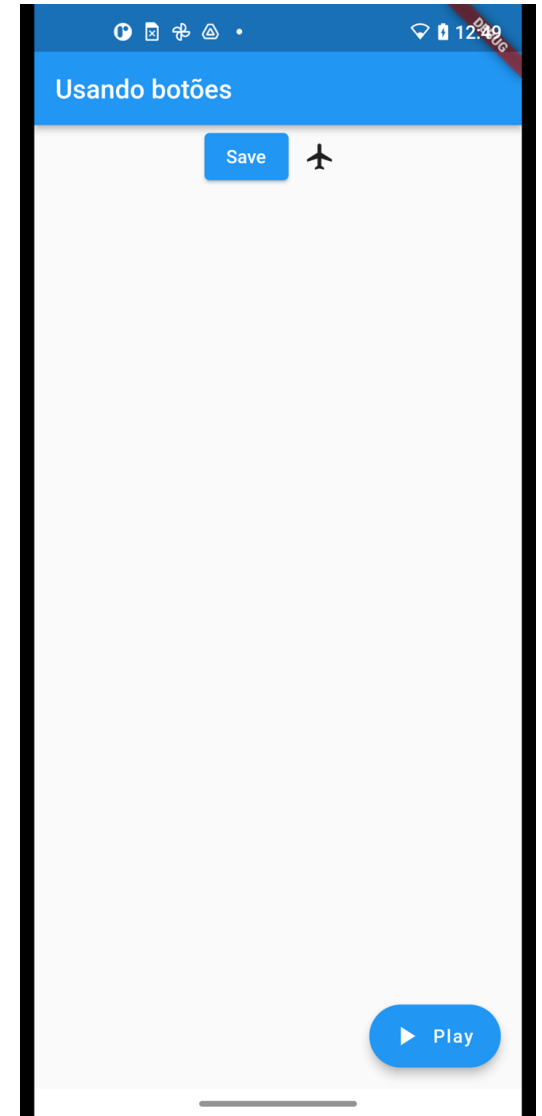
# Widget e UI

## Botão

Existe uma variedade de Widgets de botão como: FloatingActionButton, ElevatedButton, IconButton

*Alguns Parâmetros:*

- onPressed -> VoidCallback
- icon-> Icon
- label -> Text



# Widget e UI

## Botão

Existe uma variedade de Widgets de botão como: FloatingActionButton, ElevatedButton, IconButton

*Alguns Parâmetros:*

- onPressed -> VoidCallback
- icon-> Icon
- label -> Text

```
MaterialApp(  
  title: "Minha aplicação",  
  home: Scaffold(  
    appBar: AppBar(  
      title: const Text("Usando botões"),  
    ), // AppBar  
    body: Row(  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: [  
        ElevatedButton(  
          onPressed: () {},  
          child: const Text('Save'),  
        ), // ElevatedButton  
        IconButton(  
          onPressed: () {},  
          icon: const Icon(Icons.flight),  
        ), // IconButton  
      ],  
    ), // Row  
    floatingActionButton: FloatingActionButton.extended(  
      onPressed: () {},  
      icon: const Icon(Icons.play_arrow),  
      label: const Text("Play")  
    ), // FloatingActionButton.extended  
  ), // Scaffold  
) // MaterialApp
```

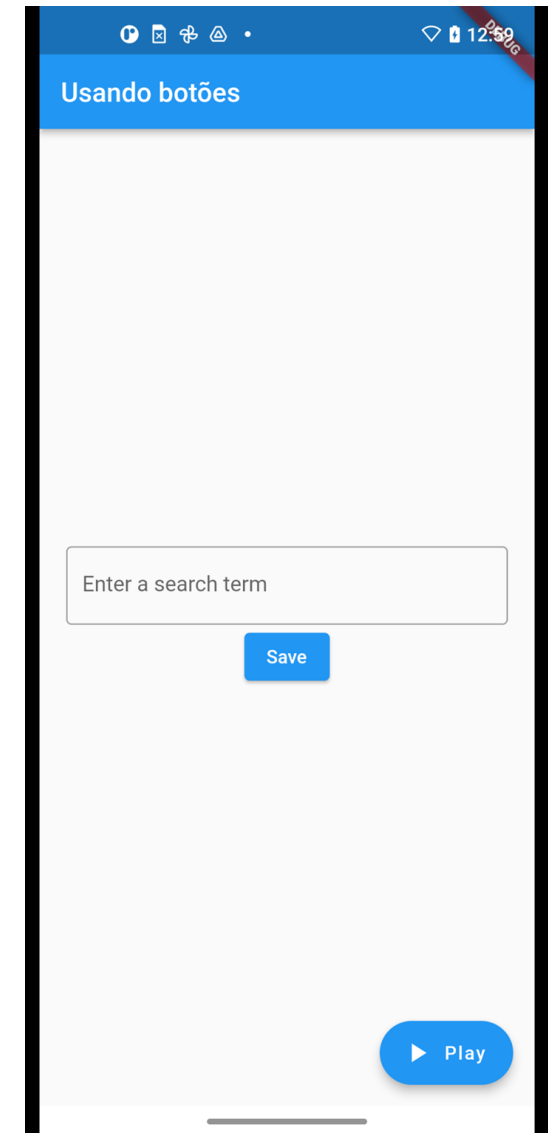
# Widget e UI

## Caixa de texto

*Alguns Parâmetros:*

- decoration-> InputDecoration?
- controller-> TextEditingController?

```
Container(  
  margin: const EdgeInsets.symmetric(vertical: 0, horizontal: 20),  
  child: const TextField(  
    decoration: InputDecoration(  
      border: OutlineInputBorder(),  
      hintText: 'Enter a search term'), // InputDecoration  
    ), // TextField  
  ), // Container
```



# Exercício 2: Widgets e UI

Criar uma tela utilizando Widgets do Material.

A tela deve possuir:

- Uma AppBar com o título “Exercício 4” na cor verde.
- Uma lista com 5 itens.
- Cada Item possui um texto (“Item 1”, “Item 2” ... “Item 5”).
- Os Itens devem ser separados por divisores.

# State: Stateless e Stateful

# State: Stateless e Stateful

- Flutter é um Framework Reativo
  - A UI reflete o estado atual da aplicação.
  - Quando o estado muda, os Widgets necessários são reconstruídos.
  - Os Widgets podem ser divididos em dois tipos: Stateless e Stateful Widgets



# State: Stateless e Stateful

## Stateless Widget

- Usado quando os dados dentro do Widget não mudam.
- Pode ser reconstruído, caso o Widget Pai seja.
- Alguns exemplos são:
  - Text
  - Button
  - Icon
  - Image

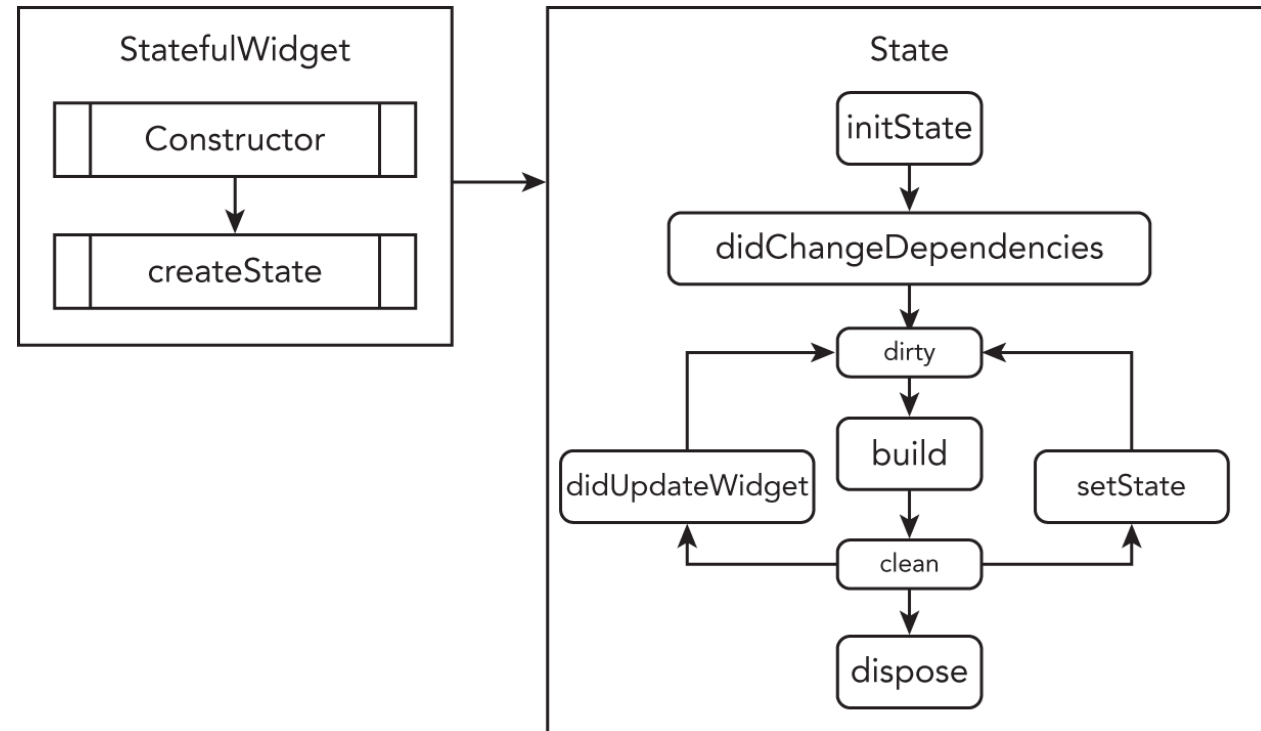
```
class AppDescription extends StatelessWidget {  
  const AppDescription({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return const Text("Essa é a descrição do app");  
  }  
}
```

```
class AppDescription extends StatelessWidget {  
  final String appName;  
  
  const AppDescription(this.appName, {Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Text("Essa é a descrição do app $appName");  
  }  
}
```

# State: Stateless e Stateful

## Stateful Widget

- Possui um estado que muda, sendo necessário reconstruir o Widget.
- É responsável por criar e retornar um objeto de State para o framework no método *createState()*



# State: Stateless e Stateful

## Exemplo 1

```
class Counter extends StatefulWidget {  
  const Counter({Key? key}) : super(key: key);  
  
  @override  
  _CounterState createState() => _CounterState();  
}
```

```
class _CounterState extends State<Counter> {  
  int _counter = 0;  
  
  void _increment() {  
    setState(() {  
      _counter++;  
    });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Row(  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: <Widget>[  
        ElevatedButton(  
          onPressed: _increment,  
          child: const Text('Increment'),  
        ), // ElevatedButton  
        const SizedBox(width: 16),  
        Text('Count: $_counter'),  
      ], // <Widget>[]  
    ); // Row  
  }  
}
```

# State: Stateless e Stateful

## Exemplo 2

```
class _SearchToolState extends State<SearchTool> {  
  final inputController = TextEditingController();  
  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: [  
        Container(  
          margin: const EdgeInsets.symmetric(vertical: 0, horizontal: 20),  
          child: TextField(  
            controller: inputController,  
            decoration: const InputDecoration(  
              border: OutlineInputBorder(),  
              hintText: 'Enter a search term'), // InputDecoration  
            ), // TextField  
          ), // Container  
        ElevatedButton(  
          onPressed: () {},  
          child: const Text('Save'),  
        ), // ElevatedButton  
      ],  
    ); // Column  
  }  
}
```

# State: Stateless e Stateful

## Shallow Widget Tree

- Tentar separar logicamente os Widgets para manter uma árvore mais organizada e o código mais legível
- Live Code!

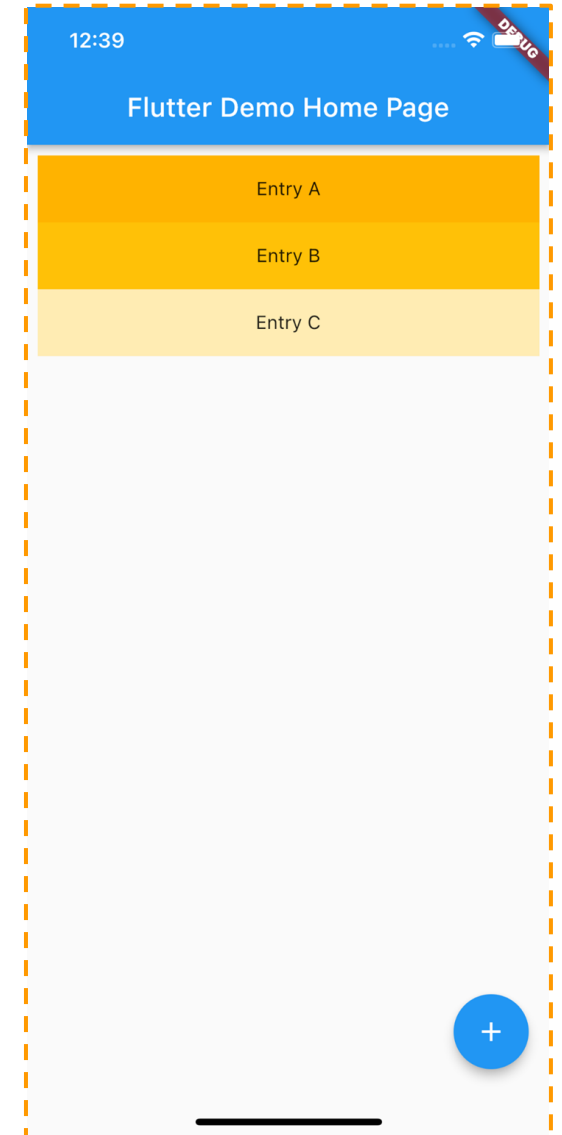
# Exercício 9

- Criar uma interface com um texto, uma caixa de Input de texto, e um botão.
- Quando o botão for pressionado, o texto mudará para o que estiver escrito na caixa de input.

**List View**

# Lista Estatica

- O Widget ListView é uma lista com scroll de widgets organizados linearmente;
- O construtor padrão do ListView recebe uma lista e funciona bem para listas pequenas;





# Lista Estatica

```
ListView(  
  padding: const EdgeInsets.all(8),  
  children: <Widget>[  
    Container(  
      height: 50,  
      color: Colors.amber[600],  
      child: const Center(child: Text('Entry A')),  
    ),  
    Container(  
      height: 50,  
      color: Colors.amber[500],  
      child: const Center(child: Text('Entry B')),  
    ),  
    Container(  
      height: 50,  
      color: Colors.amber[100],  
      child: const Center(child: Text('Entry C')),  
    ),  
  ],  
)
```

# Lista View Dinâmica

- ***ListView.builder:***
  - Renderiza os itens quando estiverem visíveis ao invés de todos de uma vez;
  - Recomendado para lista dinâmica/infinita/longa;

# Lista View Dinâmica

```
ListView.builder(  
  reverse: false,  
  itemBuilder: (_,int index) => ItemList(this.items[index]),  
  itemCount: this.items.length,  
)  
  
class ItemList extends StatelessWidget{
```

# Desafio

Desafio Desenvolvimento Híbrido com Flutter -  
Básico