

Desenvolvimento para Dispositivos Móveis

Scaffold, Container, Columns e Rows

Profº: Joseph Donald

Contatos:

☎ (83) 98228-8607

📷 @josephdonald

✉ 030106382@prof.uninassau.edu.br

“Se você tem uma maçã e eu tenho outra; e nós trocamos as maçãs, então cada um terá sua maçã. Mas se você tem uma ideia e eu tenho outra, e nós as trocamos; então cada um terá duas ideias.”

George Bernard Shaw



```
lib > main.dart > _HomeState
1  import 'package:flutter/material.dart';
2
3  Run | Debug | Profile
4  void main() {
5    runApp(MaterialApp(home: Home()));
6  }
7
8  class Home extends StatefulWidget {
9    const Home({super.key});
10
11    @override
12    State<Home> createState() => _HomeState();
13  }
14
15  class _HomeState extends State<Home> {
16    @override
17    Widget build(BuildContext context) {
18      return Placeholder();
19    }
20  }
```

Scaffold

Linha 1: Importa a biblioteca flutter/material.dart, que contém os componentes visuais (widgets) baseados no **Material Design**.

Linha 3 a 5:

- main(): Ponto de entrada do aplicativo Flutter.
- runApp(): Inicia o aplicativo e define o widget raiz.
- MaterialApp: Um widget que configura o Material Design no app.
- home: Home(): Define a tela inicial do aplicativo como sendo a classe Home.

Linha 7 a 12:

- Home é um StatefulWidget, ou seja, um widget que pode mudar de estado ao longo do tempo.
- O createState() cria uma instância do estado _HomeState, que conterá a lógica da interface.

Linha 14 a 19:

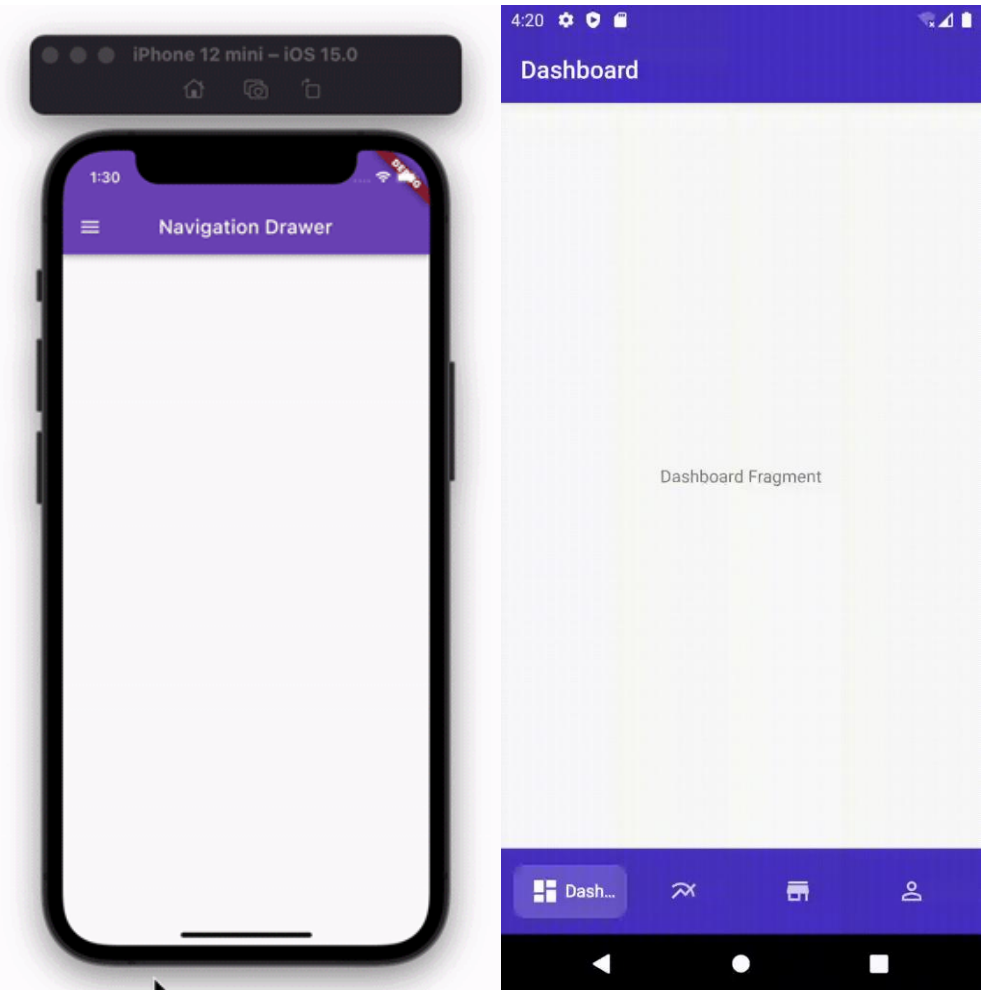
- A classe _HomeState herda de State<Home>, tornando-se responsável por gerenciar o estado do widget Home.

Método build(context):

- É chamado sempre que o widget precisa ser reconstruído.
- Retorna um Placeholder(), que é um widget temporário usado como espaço reservado.

Scaffold

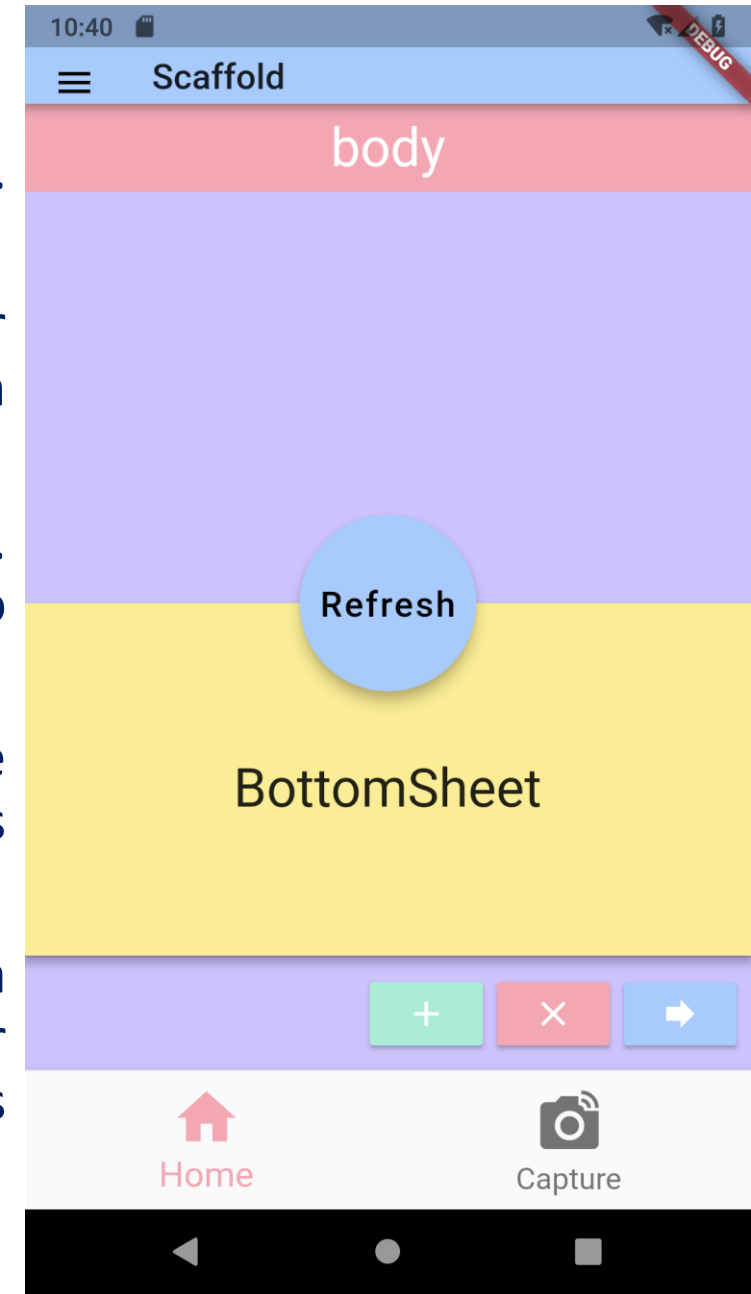
- O Scaffold no Flutter é um widget de layout que ajuda a estruturar a interface do usuário (UI) de um aplicativo.
- Ele fornece um esqueleto (ou “scaffolding”, em inglês) básico que pode ser personalizado para se adequar às necessidades do aplicativo.
- O Scaffold é composto por vários widgets filhos, como AppBar, Drawer, BottomNavigationBar, FloatingActionButton e Body. Cada um desses widgets filhos é responsável por uma área específica da tela e pode ser personalizado conforme necessário.



Scaffold

A seguir, estão alguns dos principais atributos do Scaffold no Flutter:

- **AppBar**: define a barra de aplicativos na parte superior da tela. Pode conter um título, ações e outros widgets personalizados.
- **body**: define o conteúdo principal da tela. Pode ser qualquer widget, como um ListView, um GridView, um Container ou um widget personalizado.
- **floatingActionButton**: define um botão de ação flutuante na tela. Pode ser usado para acionar ações comuns, como adicionar um novo item à lista ou abrir uma tela de configurações.
- **bottomNavigationBar**: define uma barra de navegação na parte inferior da tela. Pode ser usada para navegar entre as diferentes telas do aplicativo ou para acessar outras funcionalidades.
- **drawer**: define um menu lateral que pode ser aberto deslizando a tela da esquerda para a direita. Pode ser usado para acessar configurações, opções de ajuda, perfis de usuário ou outras funcionalidades.



Scaffold (Exemplo de código)

Sample Code

```
1  import 'package:flutter/material.dart';
2
3  // ignore: unused_element
4  class _MyStatefulWidgetState extends State<StatefulWidget> {
5    int _count = 0;
6
7    @override
8    Widget build(BuildContext context) {
9      return Scaffold(
10       appBar: AppBar(
11         title: const Text('Sample Code'),
12       ), // AppBar
13       body: Center(child: Text('You have pressed the button $_count times.')),
14       floatingActionButton: FloatingActionButton(
15         onPressed: () => setState(() => _count++),
16         tooltip: 'Increment Counter',
17         child: const Icon(Icons.add),
18       ), // FloatingActionButton
19     ); // Scaffold
20   }
21 }
```

You have pressed the button 0 times.



child: Aceita apenas um único widget.

- ✓ É usado quando um widget pode ter apenas um filho. Assim, o valor esperado para child é um único widget.

Exemplos de Widgets:

- ✓ Container, Center, Expanded e etc...

```
Container(  
  width: 200,  
  height: 200,  
  color: Colors.blue,  
  child: Text(  
    'Olá, Flutter!',  
    style: TextStyle(color: Colors.white),  
  ),  
);
```

children: Aceita vários widgets.

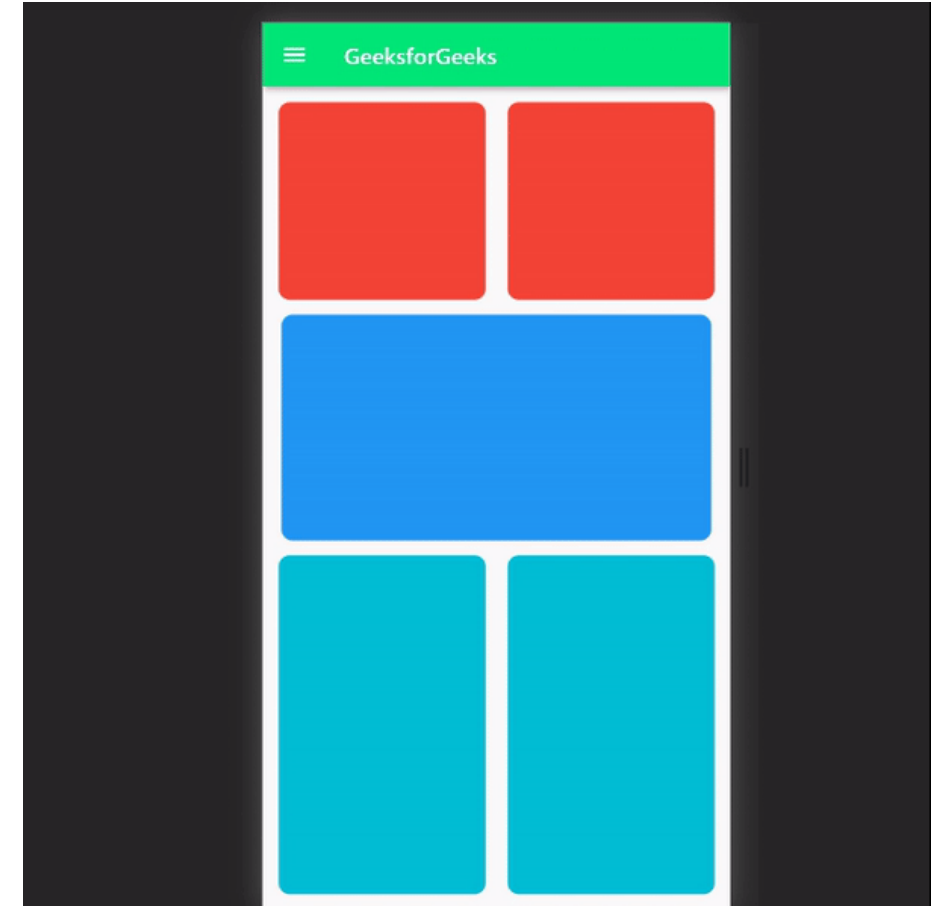
- ✓ É usado em widgets que podem ter múltiplos filhos. Assim, o valor esperado para children é uma lista de widgets (List<Widget>).

Exemplos de Widgets:

- ✓ Column, Row, Stack e etc.

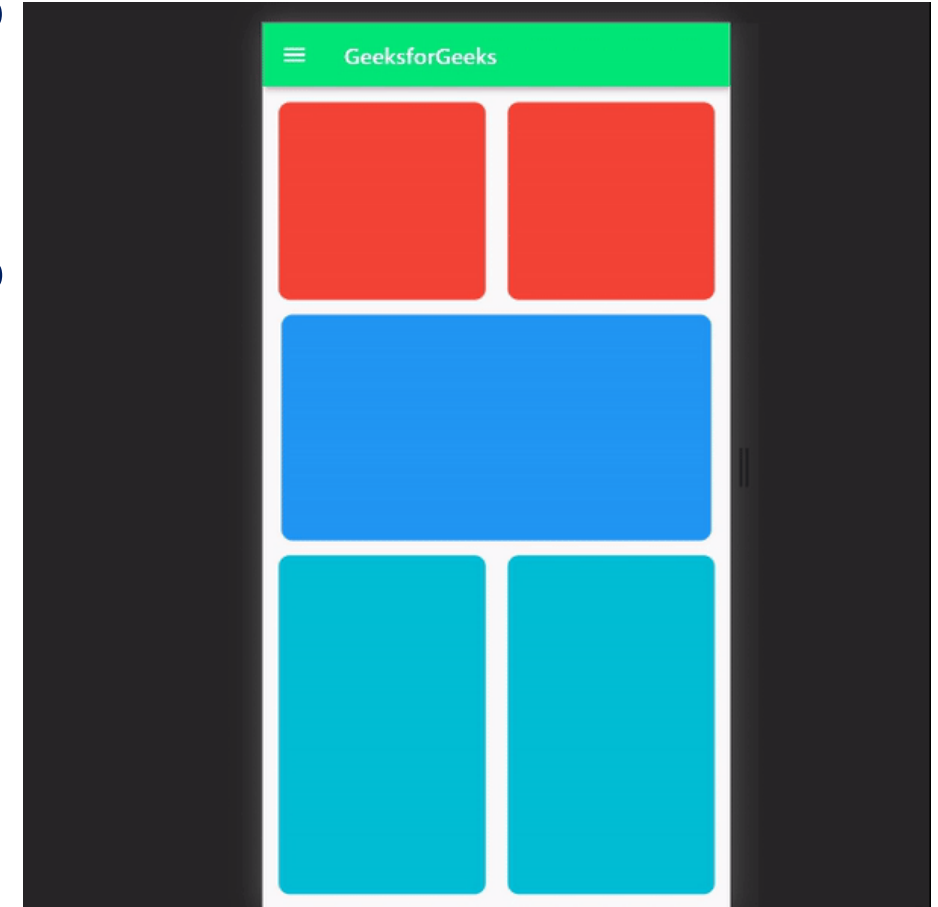
```
Column(  
  children: [  
    Text('Item 1'),  
    Text('Item 2'),  
    Text('Item 3'),  
  ],  
);
```

- O Container é um widget no Flutter que permite criar um retângulo visual com propriedades personalizáveis.
- Ele é usado para decorar outros widgets e dar-lhes uma aparência personalizada.
- O Container é bastante versátil e pode ser usado para definir a aparência de um widget de várias maneiras, como definir a cor de fundo, o tamanho, o espaçamento, a borda, entre outros.



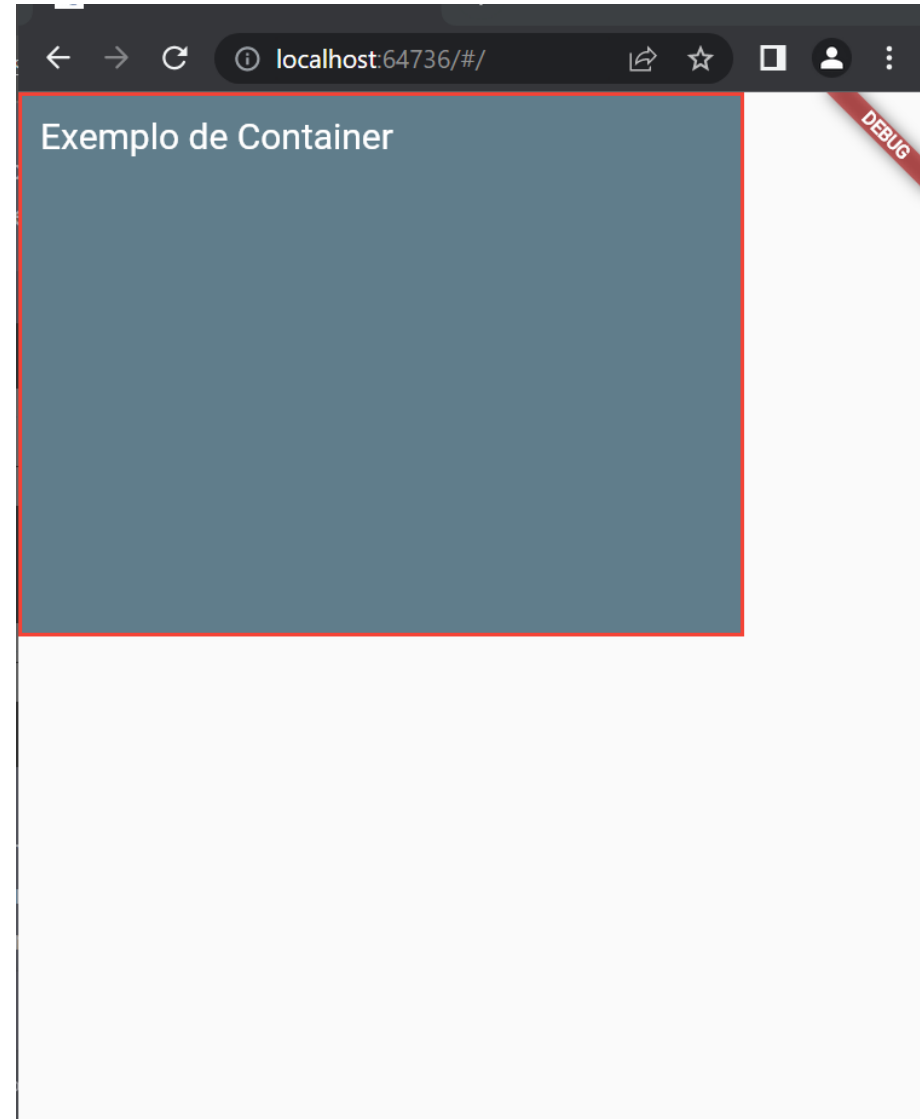
A seguir, estão alguns dos principais atributos do Container no Flutter:

- **alignment**: define a posição do widget filho dentro do Container.
- **color**: define a cor de fundo do Container.
- **width**: define a largura do Container.
- **height**: define a altura do Container.
- **margin**: define o espaço ao redor do Container.
- **padding**: define o espaço interno do Container.
- **decoration**: define a aparência visual do Container. Pode incluir bordas, gradientes e sombras.



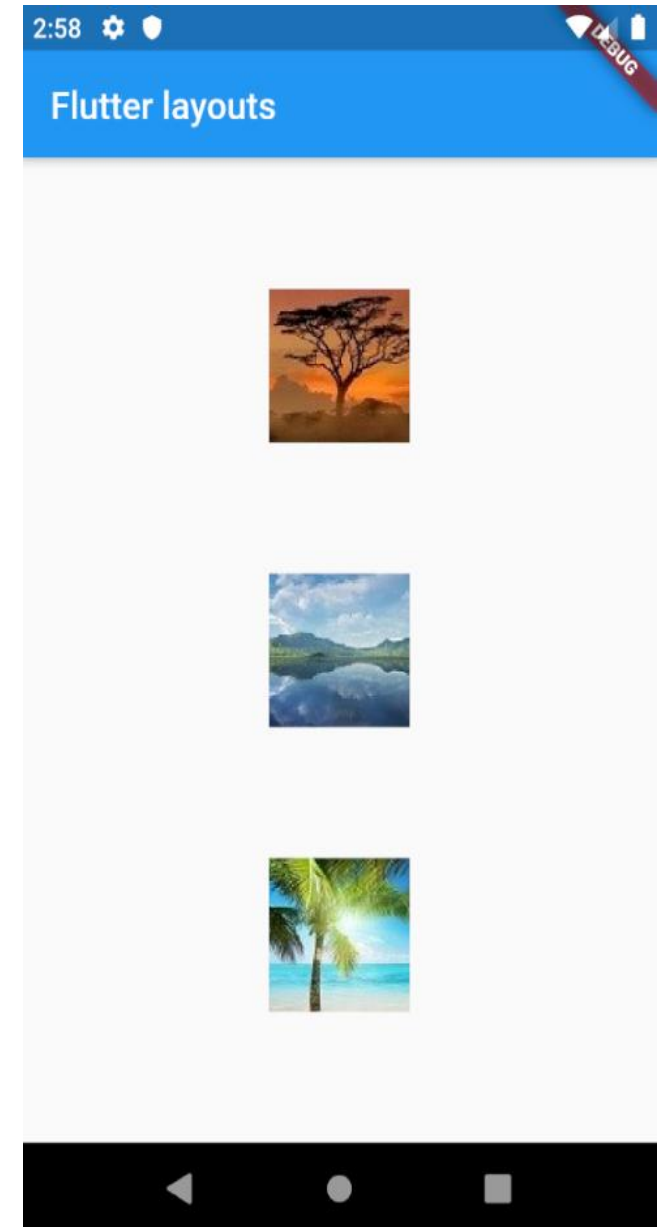
Container (Exemplo de código)

```
12 class _HomeState extends State<Home> {  
13   @override  
14   Widget build(BuildContext context) {  
15     return Scaffold(  
16       body: Container(  
17         decoration: BoxDecoration(  
18           color: Colors.blueGrey,  
19           border: Border.all(  
20             color: Colors.red,  
21             width: 2,  
22           ), // Border.all  
23         ), // BoxDecoration  
24         padding: const EdgeInsets.all(10),  
25         height: 300,  
26         width: 400,  
27         child: const Text(  
28           'Exemplo de Container',  
29           style: TextStyle(color: Colors.white, fontSize: 20),  
30         ), // Text  
31       ), // Container  
32     ); // Scaffold  
33   }  
34 }
```



Column

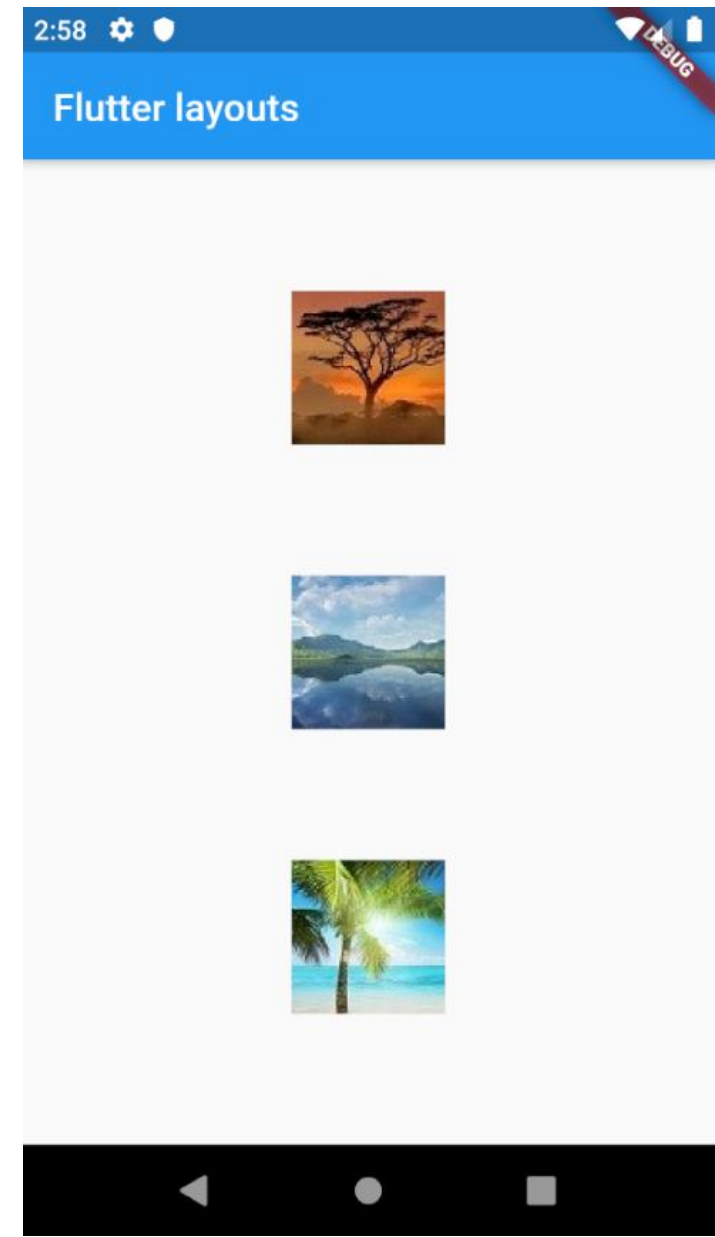
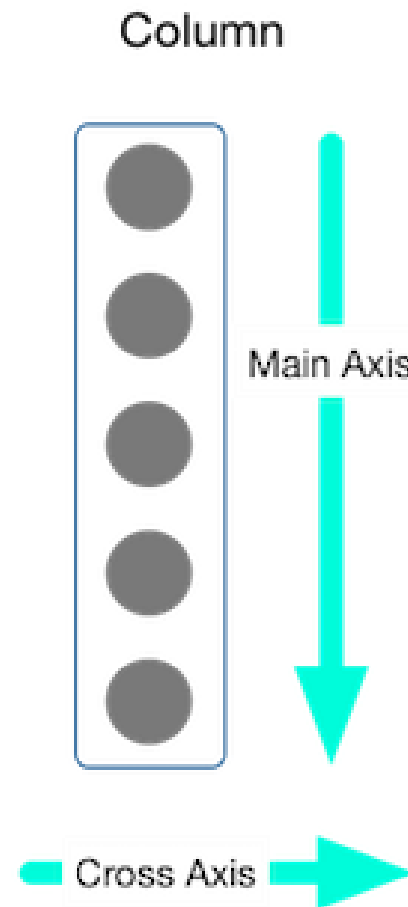
- O Column é um widget de layout no Flutter que permite empilhar outros widgets verticalmente.
- Ele é usado para criar layouts de colunas em um aplicativo, onde os widgets são dispostos em uma coluna vertical. O Column pode conter qualquer widget filho, incluindo outros widgets de layout.
- O funcionamento básico do Column é bastante simples. Ele organiza seus filhos em uma coluna vertical, alinhando-os ao topo por padrão.





Column

- No entanto, é possível alterar o alinhamento usando o atributo **mainAxisAlignment**.
- O Column também pode ser esticado verticalmente para preencher todo o espaço disponível usando o atributo **crossAxisAlignment**.



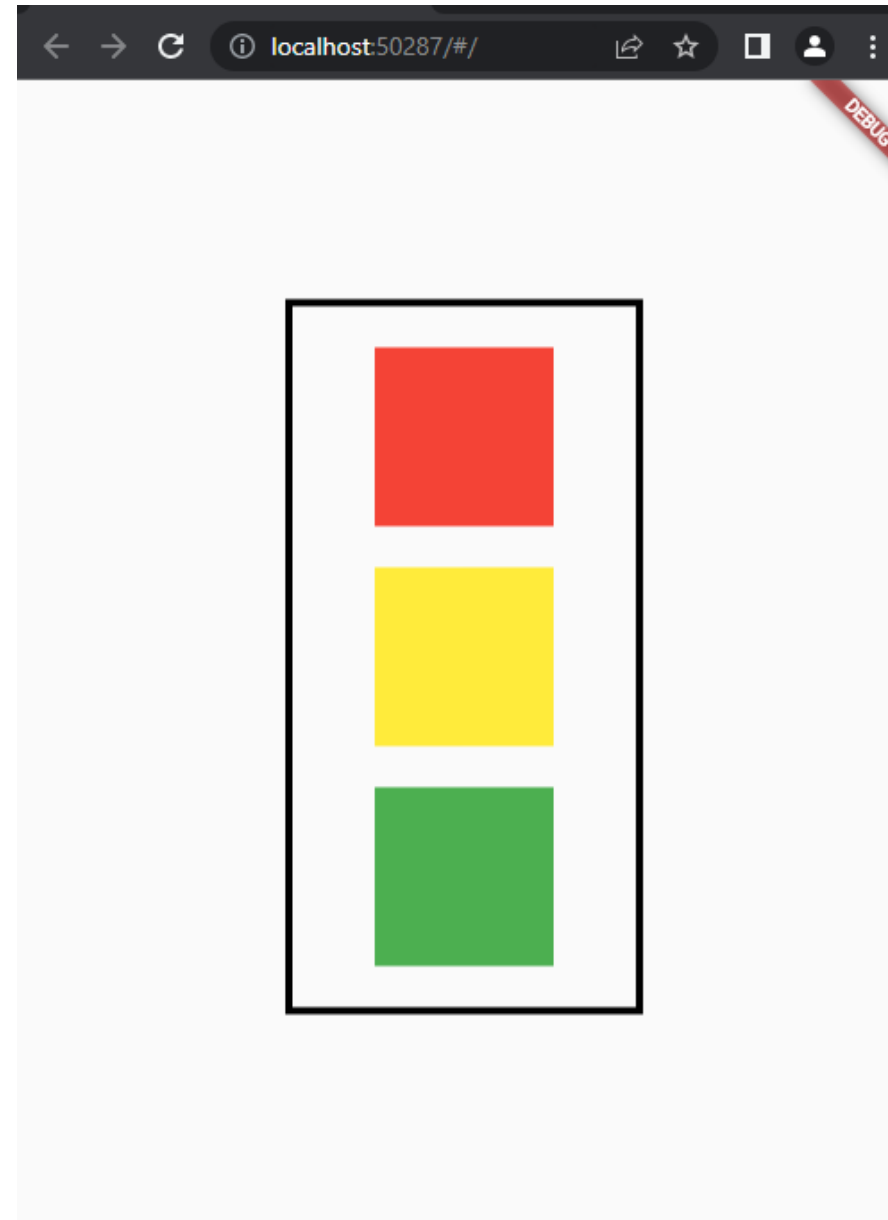
A seguir, estão alguns dos principais atributos do Column no Flutter:

- **children:** define os widgets filhos do Column. Cada widget filho é empilhado na coluna na ordem em que são definidos.
- **mainAxisAlignment:** define como os widgets filhos são alinhados verticalmente dentro do Column. Os valores possíveis incluem start (alinhado ao topo), end (alinhado na parte inferior), center (alinhado no centro), spaceBetween (espaço igual entre os widgets), spaceAround (espaço igual ao redor dos widgets) e spaceEvenly (espaço igual entre os widgets e ao redor deles).
- **crossAxisAlignment:** define como os widgets filhos são alinhados horizontalmente dentro do Column. Os valores possíveis incluem start (alinhado à esquerda), end (alinhado à direita), center (alinhado no centro), baseline (alinhado com a linha de base do texto) e stretch (esticado para preencher todo o espaço disponível).
- **mainAxisSize:** define o tamanho principal do Column, que pode ser máximo (para preencher todo o espaço disponível) ou mínimo (para se ajustar ao tamanho dos filhos).



Column (Exemplo de código)

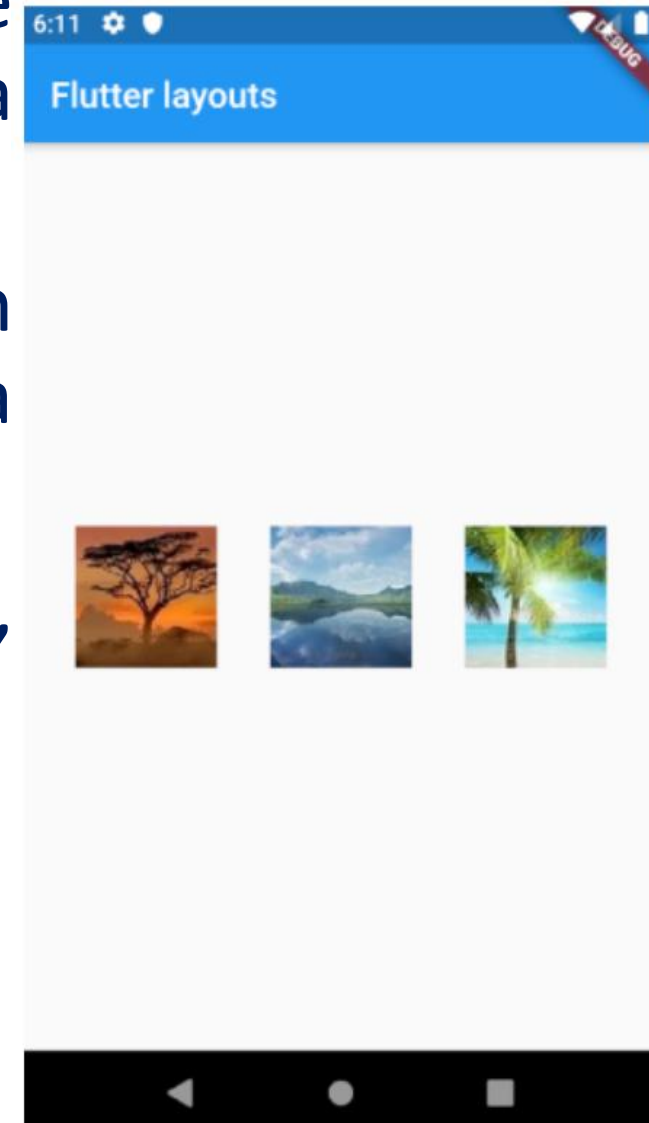
```
12 class _HomeState extends State<Home> {  
13   @override  
14   Widget build(BuildContext context) {  
15     return Scaffold(  
16       body: Center(  
17         child: Container(  
18           decoration: BoxDecoration(border: Border.all(width: 4)),  
19           height: 400,  
20           width: 200,  
21           child: Column(  
22             mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
23             children: [  
24               Container(  
25                 width: 100,  
26                 height: 100,  
27                 color: Colors.red,  
28               ), // Container  
29               Container(  
30                 width: 100,  
31                 height: 100,  
32                 color: Colors.yellow,  
33               ), // Container  
34               Container(  
35                 width: 100,  
36                 height: 100,  
37                 color: Colors.green,  
38               ), // Container  
39             ],  
40           ), // Column  
41         ), // Container // Center  
42       ); // Scaffold  
43     }  
44   }
```





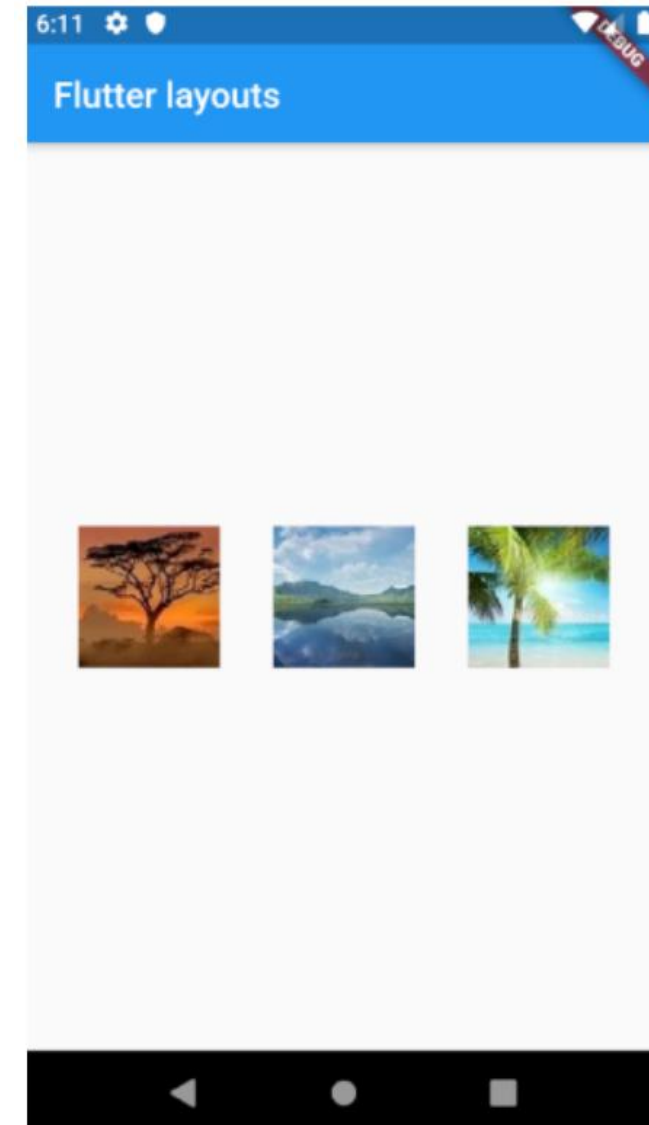
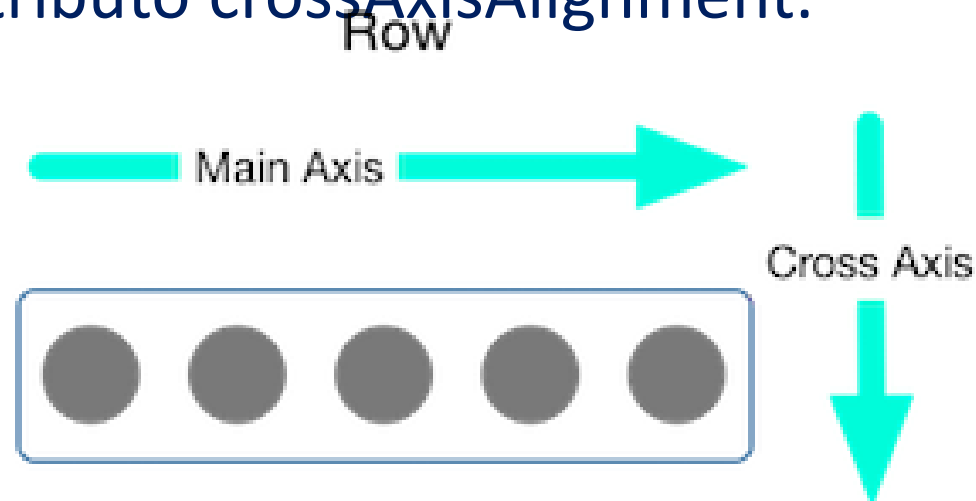
Row

- O Row é um widget de layout no Flutter que permite colocar outros widgets em uma única linha horizontal.
- Ele é usado para criar layouts de linha em um aplicativo, onde os widgets são dispostos em uma única linha horizontal.
- O Row pode conter qualquer widget filho, incluindo outros widgets de layout.



Row

- O funcionamento básico do Row é bastante simples. Ele organiza seus filhos em uma linha horizontal, alinhando-os à esquerda por padrão.
- No entanto, é possível alterar o alinhamento usando o atributo `mainAxisAlignment`.
- O Row também pode ser esticado horizontalmente para preencher todo o espaço disponível usando o atributo `crossAxisAlignment`.

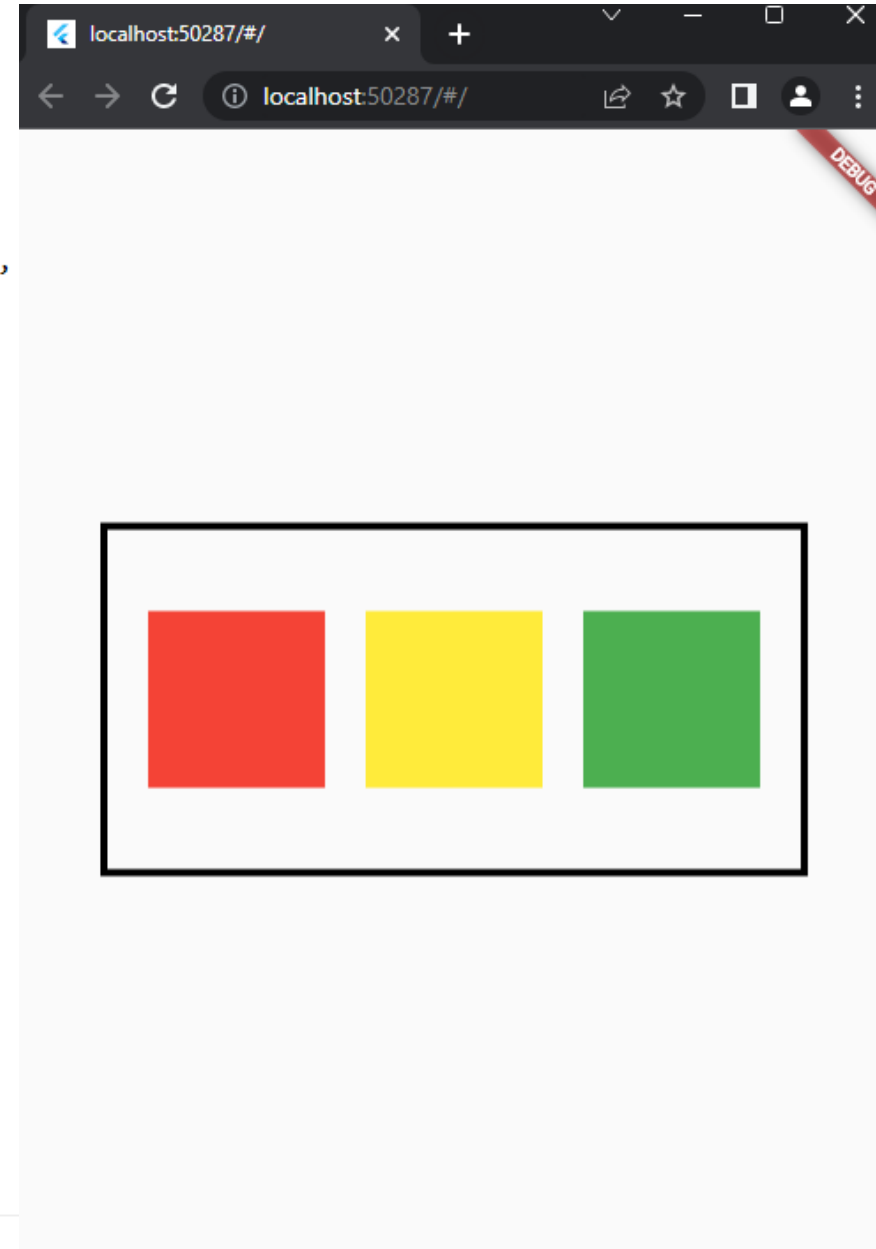


A seguir, estão alguns dos principais atributos do Row no Flutter:

- **children:** define os widgets filhos do Row. Cada widget filho é disposto em uma linha horizontal na ordem em que são definidos.
- **mainAxisAlignment:** define como os widgets filhos são alinhados horizontalmente dentro do Row. Os valores possíveis incluem start (alinhado à esquerda), end (alinhado à direita), center (alinhado no centro), spaceBetween (espaço igual entre os widgets), spaceAround (espaço igual ao redor dos widgets) e spaceEvenly (espaço igual entre os widgets e ao redor deles).
- **crossAxisAlignment:** define como os widgets filhos são alinhados verticalmente dentro do Row. Os valores possíveis incluem start (alinhado ao topo), end (alinhado na parte inferior), center (alinhado no centro), baseline (alinhado com a linha de base do texto) e stretch (esticado para preencher todo o espaço disponível).
- **mainAxisSize:** define o tamanho principal do Row, que pode ser máximo (para preencher todo o espaço disponível) ou mínimo (para se ajustar ao tamanho dos filhos).

Row (Exemplo de código)

```
12 class _HomeState extends State<Home> {  
13   @override  
14   Widget build(BuildContext context) {  
15     return Scaffold(  
16       body: Center(  
17         child: Container(  
18           decoration: BoxDecoration(border: Border.all(width: 4)),  
19           height: 200,  
20           width: 400,  
21           child: Row(  
22             mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
23             children: [  
24               Container(  
25                 width: 100,  
26                 height: 100,  
27                 color: Colors.red,  
28               ), // Container  
29               Container(  
30                 width: 100,  
31                 height: 100,  
32                 color: Colors.yellow,  
33               ), // Container  
34               Container(  
35                 width: 100,  
36                 height: 100,  
37                 color: Colors.green,  
38               ), // Container  
39             ],  
40           ), // Row  
41         ), // Container // Center  
42       ); // Scaffold  
43     }  
44   }
```

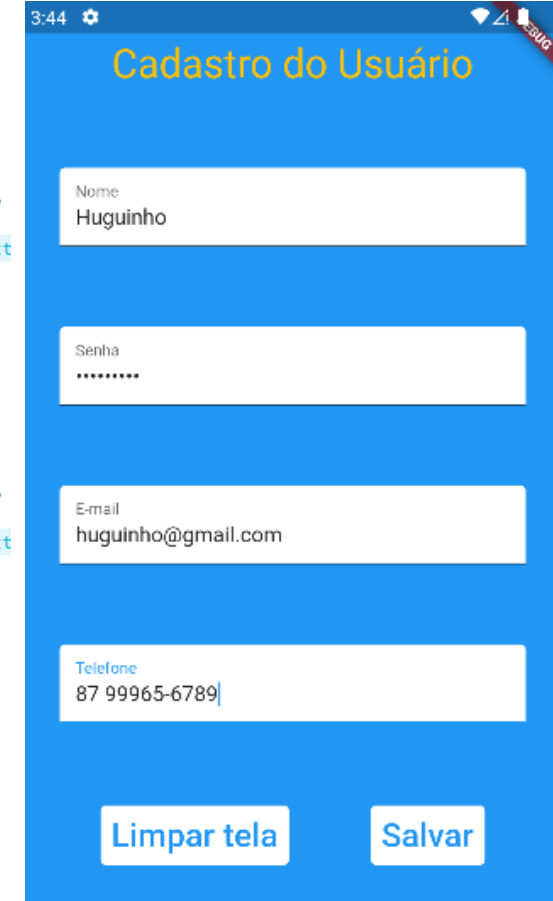


Exemplo – Tela de Cadastro

```

13 class _HomeState extends State<Home> {
14   @override
15   Widget build(BuildContext context) {
16     return Scaffold(
17       body: Container(
18         padding: const EdgeInsets.all(30),
19         color: Colors.blue,
20         width: MediaQuery.of(context).size.width,
21         height: MediaQuery.of(context).size.height,
22         child: Column(
23           mainAxisAlignment: MainAxisAlignment.spaceBetween,
24           children: [
25             const Text("Cadastro do Usuário",
26               style: TextStyle(fontSize: 30, color: Colors.amber)), // Text
27             const TextField(
28               decoration: InputDecoration(
29                 labelText: "Nome", filled: true, fillColor: Colors.white)),
30             const TextField(
31               obscureText: true,
32               decoration: InputDecoration(
33                 labelText: "Senha", filled: true, fillColor: Colors.white)),
34             const TextField(
35               decoration: InputDecoration(
36                 labelText: "E-mail",
37                 filled: true,
38                 fillColor: Colors.white)), // InputDecoration // TextField
39             const TextField(
40               decoration: InputDecoration(
41                 labelText: "Telefone",
42                 filled: true,
43                 fillColor: Colors.white)), // InputDecoration // TextField
44             Row(
45               mainAxisAlignment: MainAxisAlignment.spaceAround,
46               children: [
47                 TextButton(
48                   style: ButtonStyle(
49                     backgroundColor: MaterialStateProperty.all(Colors.white)),
50                   onPressed: () {
51                     print("Botão pressionado!"); // Don't invoke 'print' in product
52                   },
53                   child: const Text(
54                     "Limpar tela",
55                     style: TextStyle(color: Colors.blue, fontSize: 25),
56                   ), // Text
57                 ), // TextButton
58                 TextButton(
59                   style: ButtonStyle(
60                     backgroundColor: MaterialStateProperty.all(Colors.white)),
61                   onPressed: () {
62                     print("Botão pressionado!"); // Don't invoke 'print' in product
63                   },
64                   child: const Text(
65                     "Salvar",
66                     style: TextStyle(color: Colors.blue, fontSize: 25),
67                   ), // Text
68                 ), // TextButton
69               ], // Row
70             ), // Column
71           ], // Container
72         ); // Scaffold
73       }
74     }
75   }
76 }
77

```



3:44

Cadastro do Usuário

Nome
Huguinho

Senha

E-mail
huguinho@gmail.com

Telefone
87 99965-6789

Limpar tela Salvar

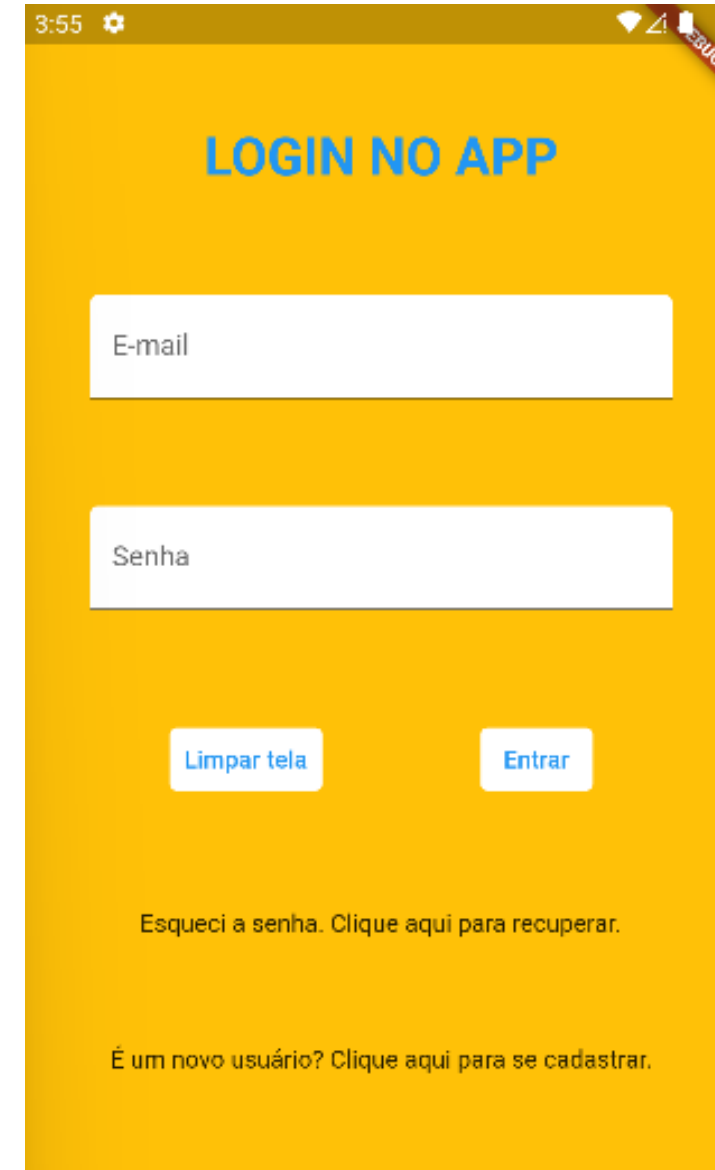
DESAFIO

Crie uma tela de **login** que contenha os campos:

- Usuário (TextField)
- Senha (TextField)
- Limpar Tela (Button)
- Entrar (Button)
- Esqueci a senha (Text)
- Novo usuário (Text)

DESAFIO 1: Criar a tela ao lado com os campos utilizando os Widgets indicados acima.

DESAFIO 2: Fazer que os campos “Esqueci a Senha” e “Novo Usuário” sejam **textos clicáveis**.



The image shows a mobile app login screen with a yellow background. At the top, the status bar shows the time 3:55, a gear icon, and a battery icon. The title "LOGIN NO APP" is displayed in blue text. Below the title are two white text input fields: "E-mail" and "Senha". At the bottom, there are two white buttons: "Limpar tela" and "Entrar". Below the buttons, there are two lines of text: "Esqueci a senha. Clique aqui para recuperar." and "É um novo usuário? Clique aqui para se cadastrar." A red "BUG" label is visible in the top right corner of the app screen.