



FLUTTER

FLUTTER #7

Copyright © 2023 Accenture. All rights reserved.

FLUTTER



PROJETO CONTADOR



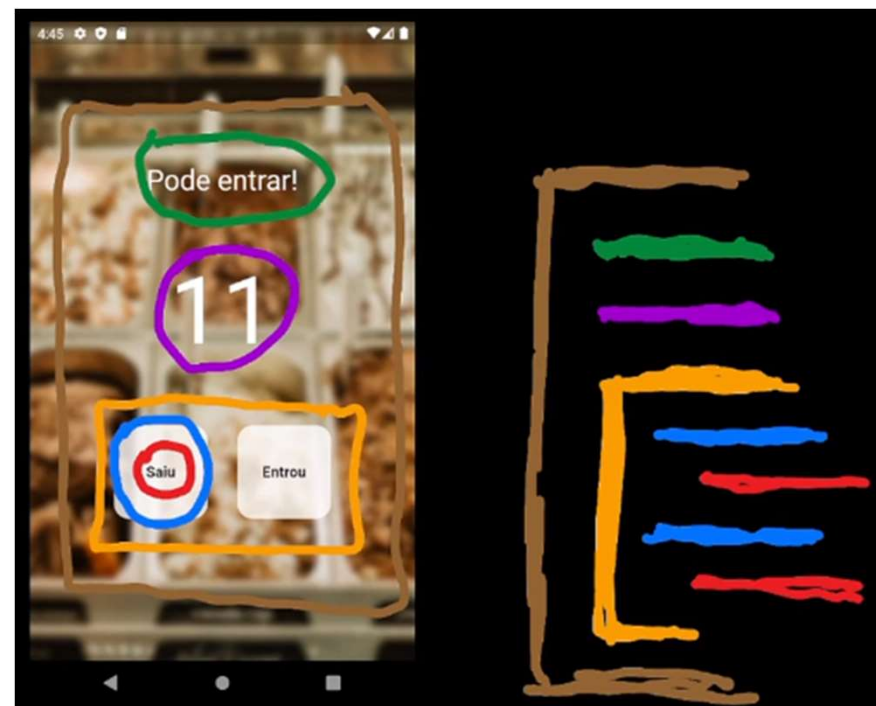


Usaremos os Widgets!

1. Texto.
2. Botão.

Planejar o projeto:

1. Nosso projeto terá 2 botões.
2. Nosso projeto terá 4 textos.



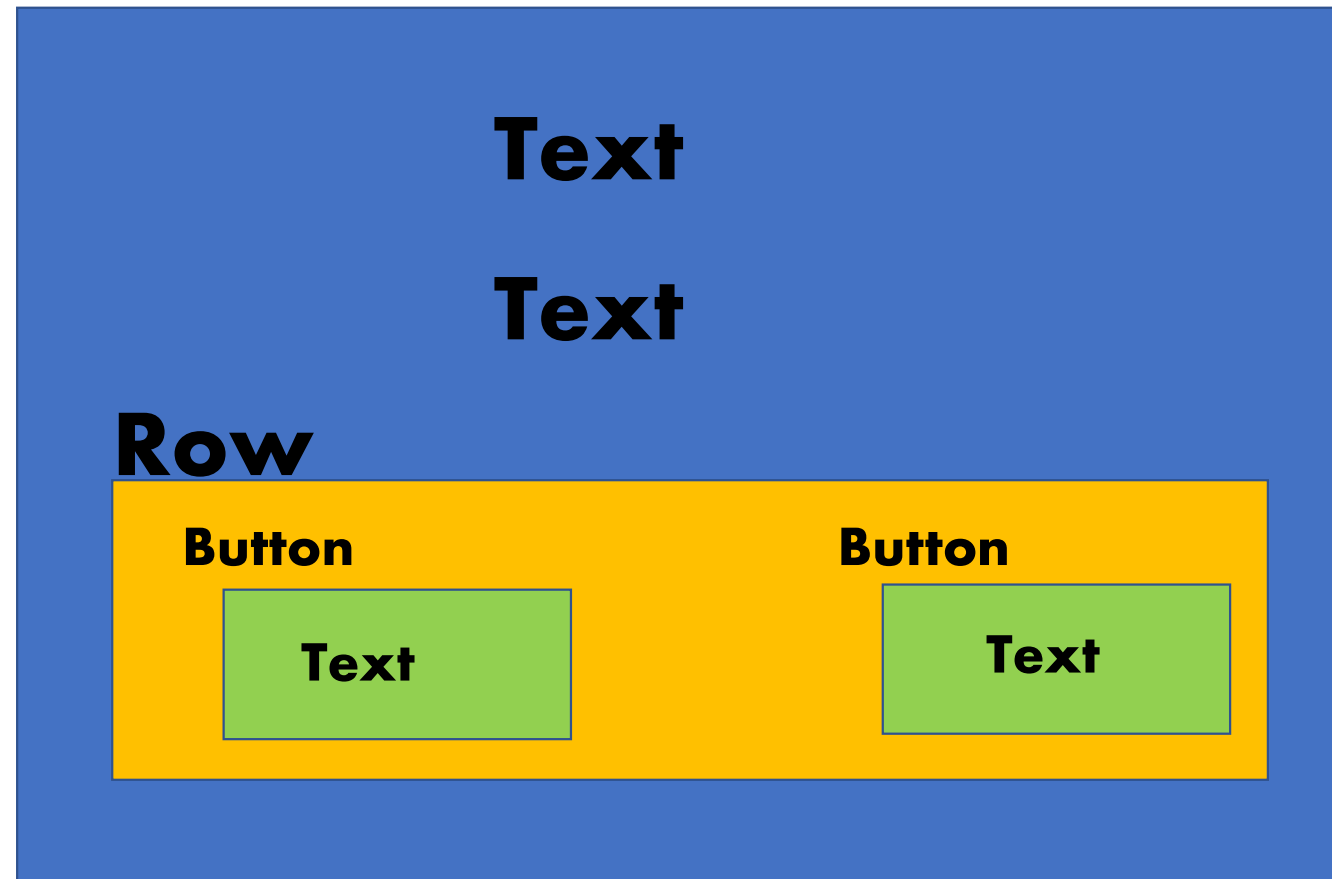
FLUTTER



Organizar a tela em linhas e colunas

Column

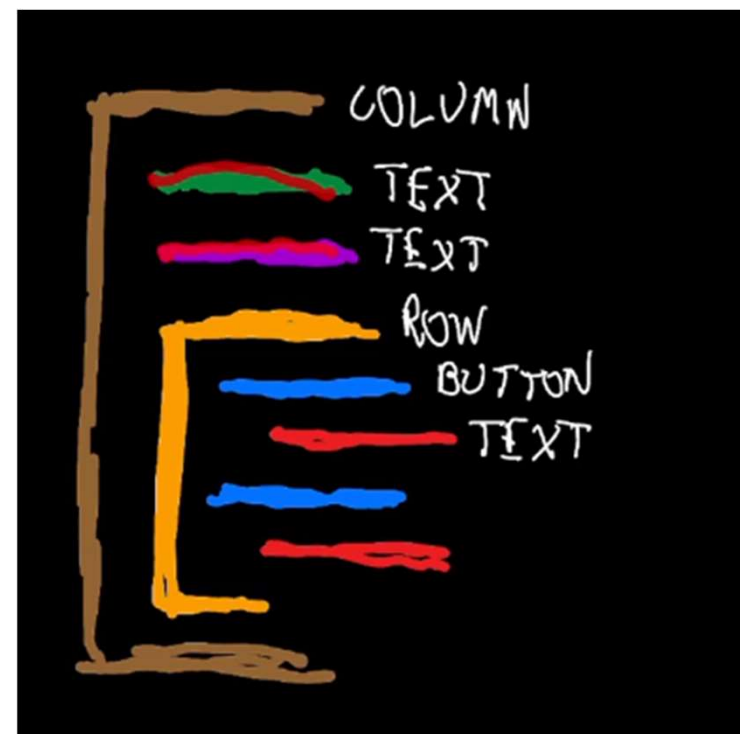
1. Uma coluna. E uma linha
2. Na coluna:
 1. Temos 2 widgets text.
 2. 1 widget row (linha)
3. Na Linha.
 1. 2 widgets button.
 2. 2 widgets text.





Organizar a tela em linhas e colunas

1. Uma coluna. E uma linha
2. Na coluna:
 1. Temos 2 widgets text.
 2. 1 widget row (linha)
3. Na Linha.
 1. 2 widgets button.
 2. 2 widgets text.



FLUTTER



Criando o Projeto



FLUTTER



Criando o projeto

1. Criar o projeto.
2. Editar o Main.dart
3. Adicionando Collumn
4. Adicionando row.
5. Adicionando os botões.
6. Estilizando o app.
7. Colocar a imagem de fundo.
8. Ajustes finais.

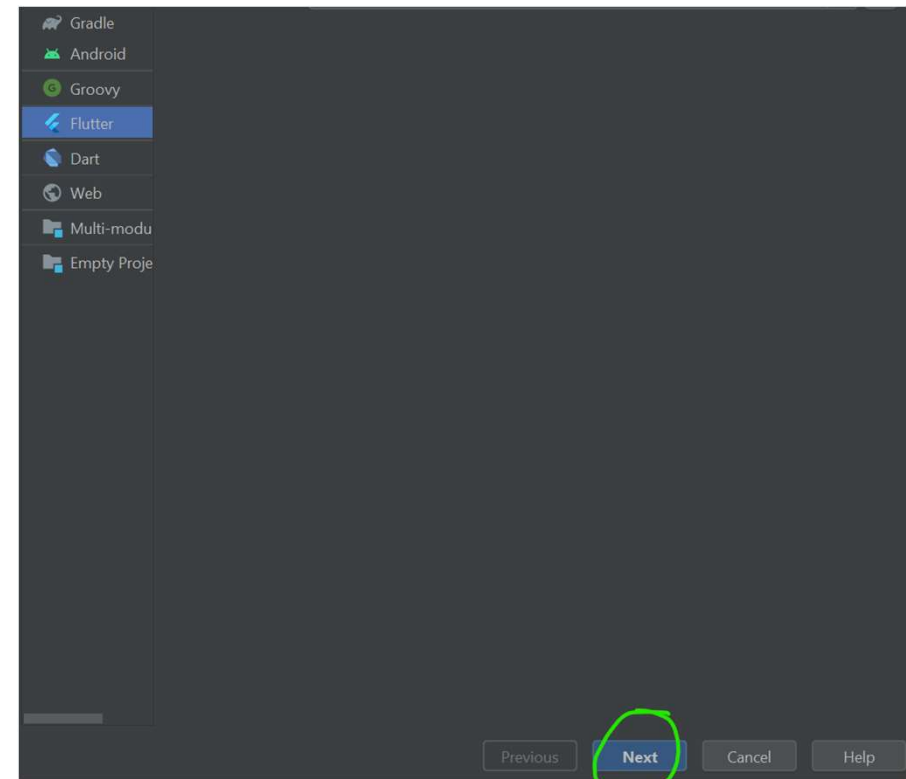
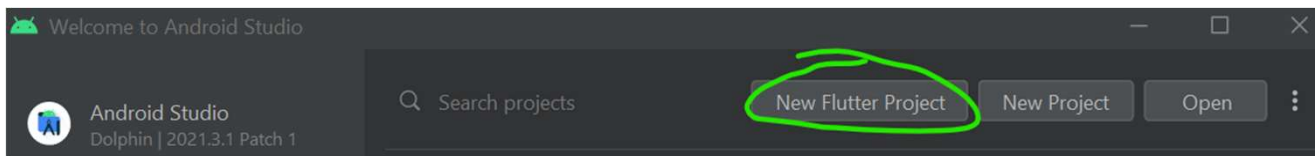


FLUTTER



Executando no browser.

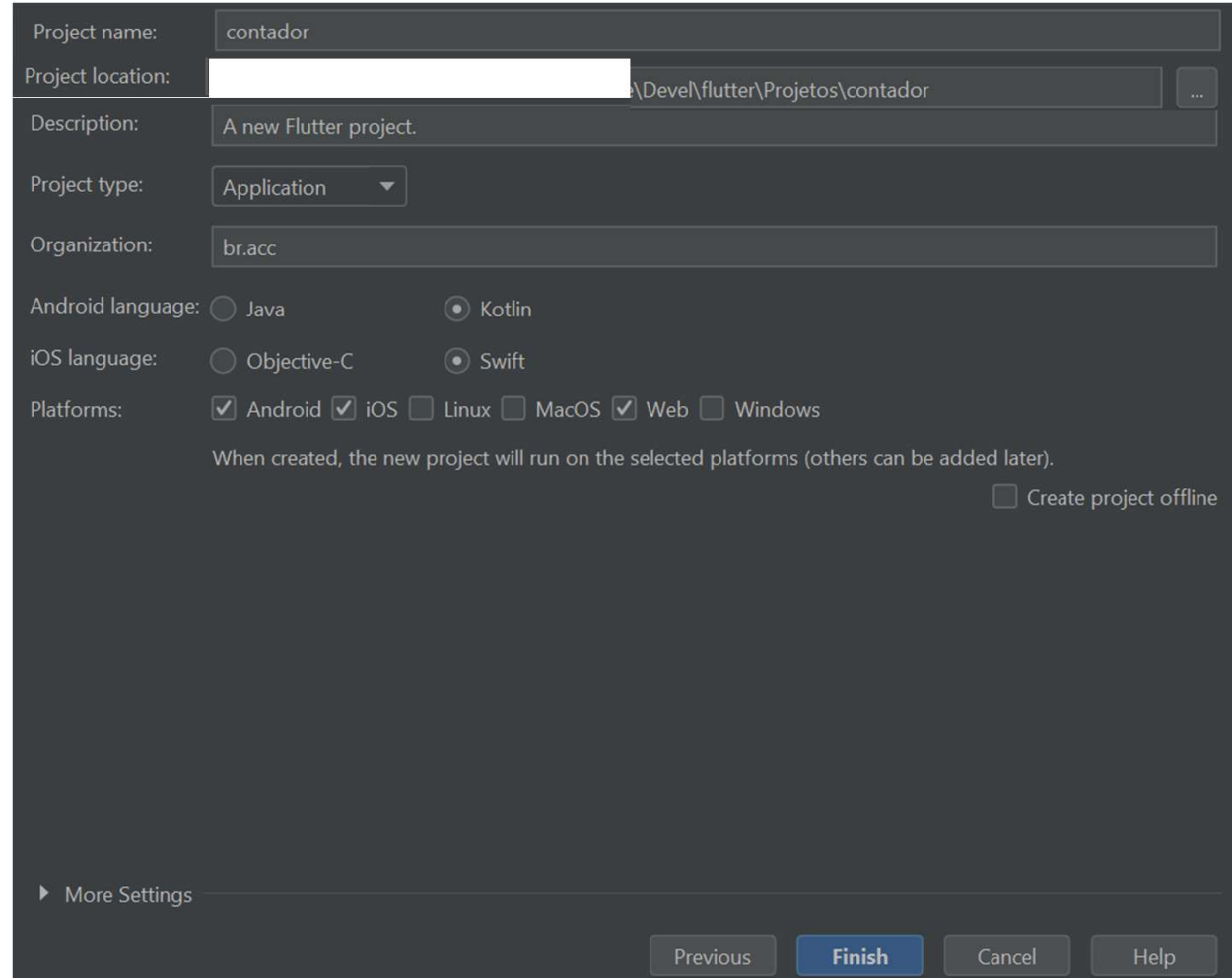
1. Abra o Android Studio.
2. Selecione New Flutter project.
3. Especifique o SDK



FLUTTER



1. Name: contador.

A screenshot of the Flutter project creation dialog in an IDE. The dialog has a dark gray background and contains several input fields and checkboxes. The 'Project name' field is filled with 'contador'. The 'Project location' field is empty, with a file explorer icon to its right. The 'Description' field is filled with 'A new Flutter project.'. The 'Project type' dropdown is set to 'Application'. The 'Organization' field is filled with 'br.acc'. Under 'Android language', 'Kotlin' is selected. Under 'iOS language', 'Swift' is selected. Under 'Platforms', 'Android', 'iOS', and 'Web' are checked. A note at the bottom states: 'When created, the new project will run on the selected platforms (others can be added later).'. There is an unchecked checkbox for 'Create project offline'. At the bottom right, there are four buttons: 'Previous', 'Finish' (highlighted in blue), 'Cancel', and 'Help'. A 'More Settings' link is visible at the bottom left of the dialog area.

Project name: contador

Project location: \Devel\flutter\Projetos\contador

Description: A new Flutter project.

Project type: Application

Organization: br.acc

Android language: ☐ Java ☒ Kotlin

iOS language: ☐ Objective-C ☒ Swift

Platforms: ☒ Android ☒ iOS ☐ Linux ☐ MacOS ☒ Web ☐ Windows

When created, the new project will run on the selected platforms (others can be added later).

☐ Create project offline

More Settings

Previous Finish Cancel Help



FLUTTER



1. Abra o **main.dart**.
2. Apague seu conteúdo.

```
1 import 'package:flutter/material.dart';  
2  
3 void main() {  
4   runApp(const MyApp());  
5 }  
6  
7 class MyApp extends StatelessWidget {  
8   const MyApp({Key? key}) : super(key: key);  
9   @override  
10  Widget build(BuildContext context) {  
11    return MaterialApp(  
12      home: Container(  
13        color: Colors.black,  
14        alignment: Alignment.center,  
15        child: Text('ola Mundo'),  
16      ), // Container  
17    ); // MaterialApp  
18  }  
19 }
```



FLUTTER



Adicionando o scaffold



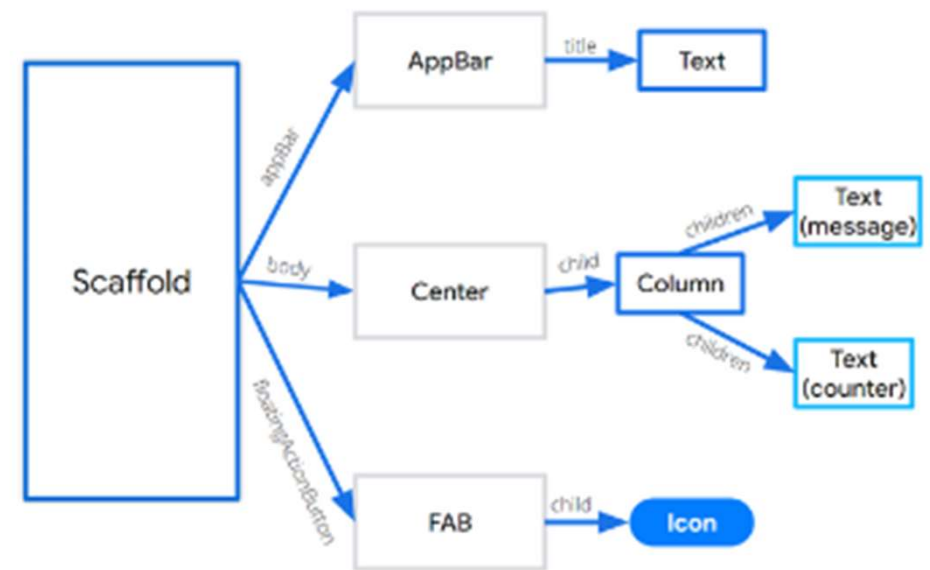
FLUTTER



Adicionando o Scaffold

1. Scaffold

- Implementa a estrutura visual do Material Design básico e permite definir outros widgets no seu interior.

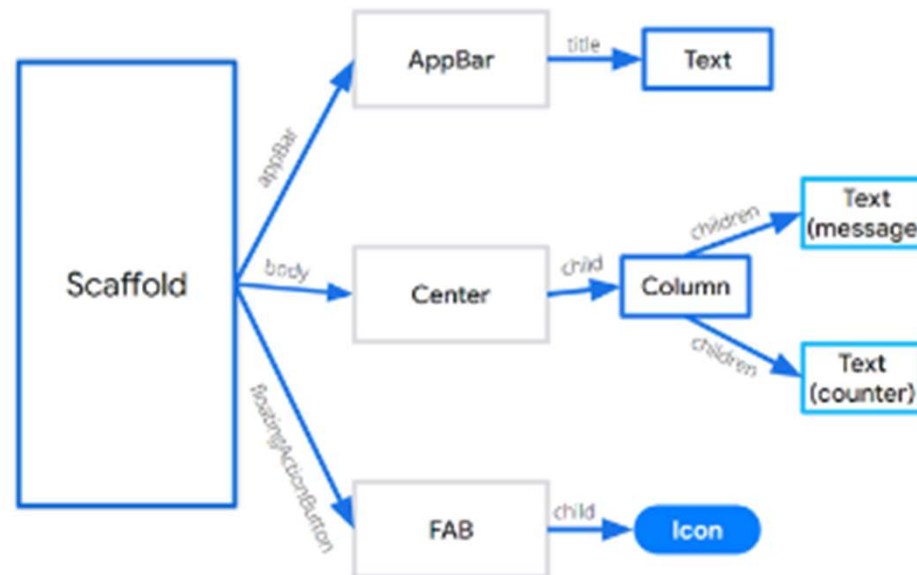




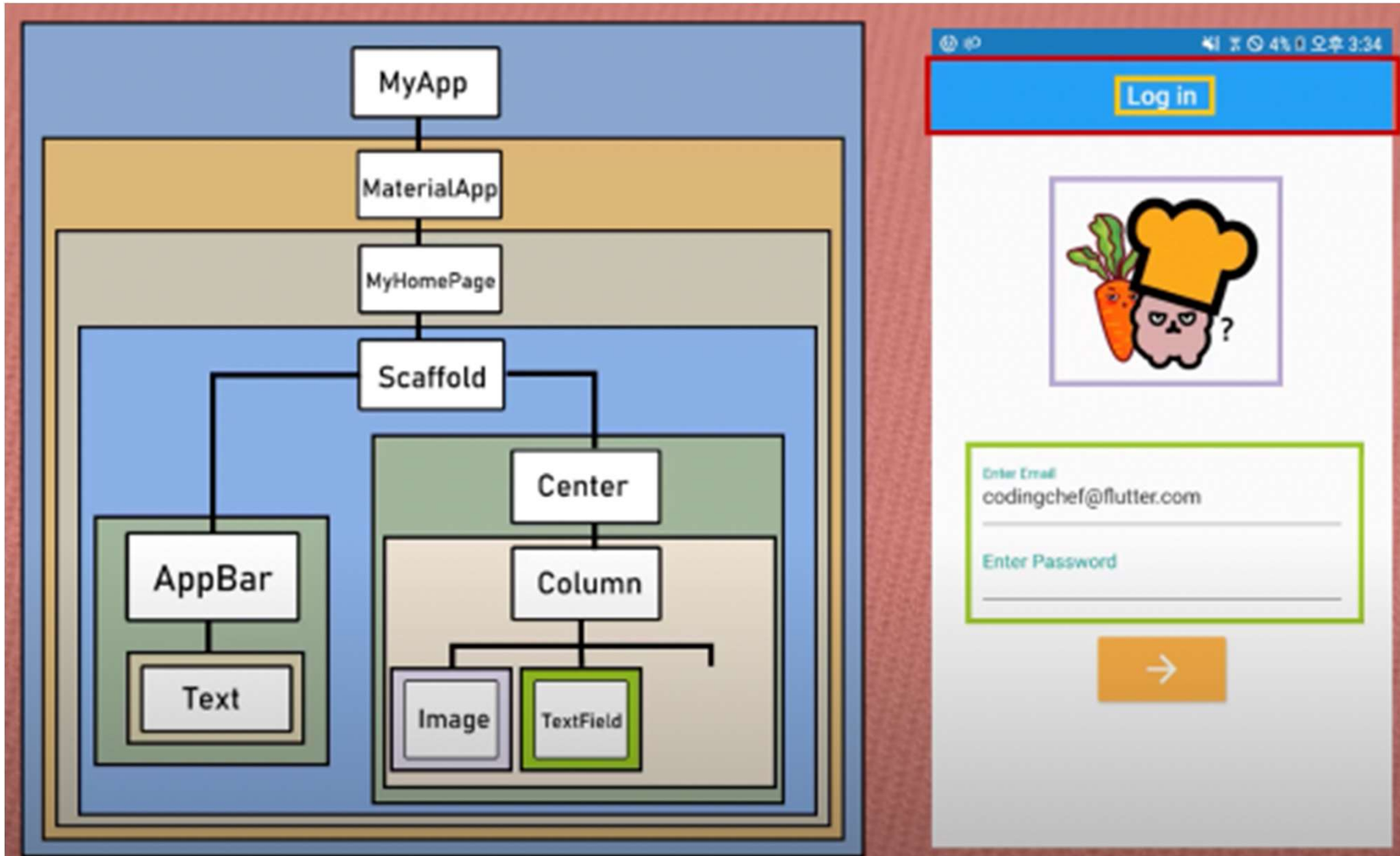
Adicionando o Scaffold

1. Widget Tree

- É a estrutura que representa como nossos widgets estão organizados.



FLUTTER



FLUTTER



Adicionando a Coluna





Adicionando a Coluna

2. Column

- Com esse Widget alinhamos os Widgets na tela do app no sentido vertical, como se fossem parte de uma coluna.



- Sobreponha o conteúdo deste slide no main.dart.

```
16 class HomePage extends StatelessWidget {  
17   const HomePage({Key? key}) : super(key: key);  
18  
19   @override  
20   Widget build(BuildContext context) {  
21     return Scaffold(  
22       backgroundColor: Colors.black,  
23       body: Column(  
24         ) // Column  
25     ); // Scaffold  
26   }  
27 }  
28  
29
```





Adicionando a Coluna

3. Children

- Widgets que são filhos de outros widgets.

4. Text

- Widgets que contém um texto.
- **Acrescente dois widgets Text.**



```
25 | children: const [  
26 |   Text(  
27 |     'Pode Entrar',  
28 |     style: TextStyle(  
29 |       fontSize: 30,  
30 |       color: Colors.white,  
31 |       fontWeight: FontWeight.w700,  
32 |     ), // TextStyle  
33 |   ), // Text  
34 |   Text(  
35 |     '0',  
36 |     style: TextStyle(  
37 |       fontSize: 100,  
38 |       color: Colors.white,  
39 |     ), // TextStyle  
40 |   ), // Text  
41 | ],
```

FLUTTER



Adicionando a Linha





Adicionando a row(linha)

5. children

- A row deve ser adicionada aos filhos da coluna.

6. row

- Com esse Widget alinhamos os Widgets na tela do app no sentido horizontal, como se fizessem parte de uma linha.

```
Row(  
  
)
```



FLUTTER



Adicionando Botões





Adicionando botões

7. children

- A row também tem filhos.

7.1. TextButton

- Também tem filho.

7.1.1. Text

- Texto “Saiu”.



```
Row(  
  children:[  
    TextButton(  
      onPressed: null,  
      child: Text('Saiu'),  
    ),  
  ],  
)
```





Adicionando uma função ao botão

8. Adicione o evento onPressed

- Toda vez que o botão for pressionado, o evento **onPressed** irá chamar a função *decrement* (sem os parêntesis para não dar retorno)

9. Adicione a função decrement

```
TextButton(  
  onPressed: decrement,  
  child: Text('Saiu'),  
)
```



```
void decrement(){  
  print('decrement'); // imprime no console  
}
```



Adicionando uma função ao botão

10. Duplique o Botão

11. Adicione a função increment

```
TextButton(  
  onPressed: increment,  
  child: Text('Entrou'),  
)
```



```
void increment(){  
  print('increment'); // imprime no console  
}
```



Adicionando uma função ao botão

10. Duplicue o Botão

11. Adicione a função increment

```
TextButton(  
  onPressed: increment,  
  child: Text('Entrou'),  
)
```



```
void increment(){  
  print('increment'); // imprime no console  
}
```



FLUTTER



Estilizando o app





Centralizando os botões

12. CrossAxisAlignment

- Alinha os filhos no eixo transversal.

13. MainAxisAlignment

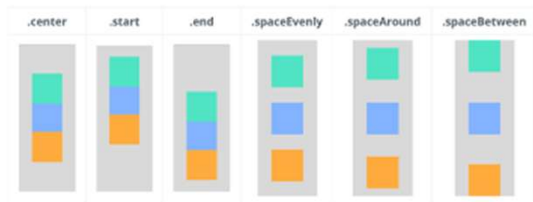
- Alinha os filhos no eixo principal

Column

CrossAxis
Alignment



MainAxis
Alignment



Row

CrossAxis
Alignment



MainAxis
Alignment



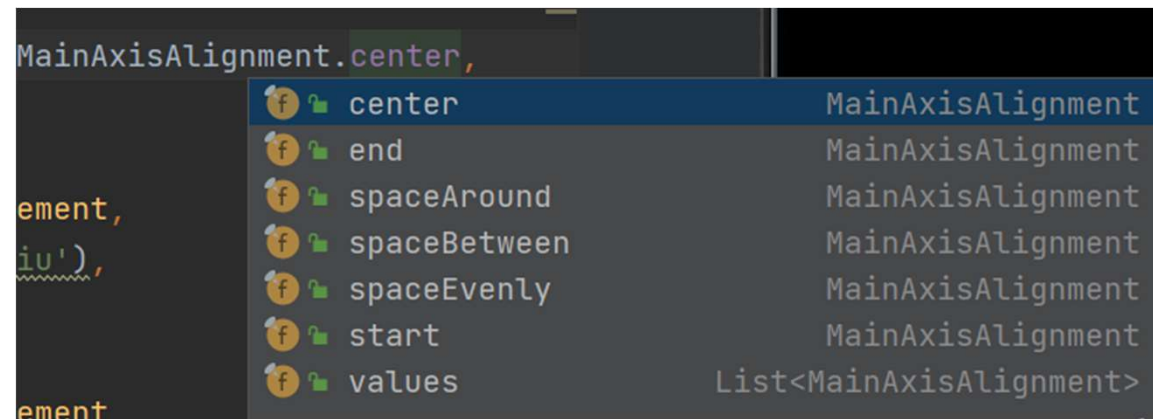
FLUTTER



Centralizando os botões



```
Row(  
  mainAxisAlignment: MainAxisAlignment.center,  
)
```





Estilizando os botões

12. style: T

- Alinha os filhos no eixo transversal.

13. MainAxisAlignment

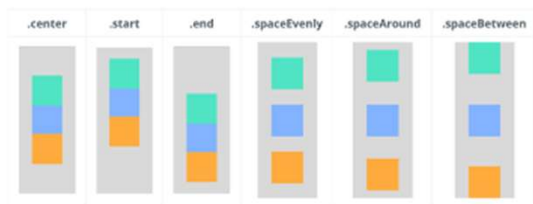
- Alinha os filhos no eixo principal

Column

CrossAxis
Alignment



MainAxis
Alignment



Row

CrossAxis
Alignment



MainAxis
Alignment



FLUTTER



Estilizando os botões

```
style: TextButton.styleFrom(  
  backgroundColor: Colors.white,  
  fixedSize: const Size(100,100)  
  shape: RoundedRectangleBorder(  
    borderRadius: BorderRadius.circular(24),  
  )  
),
```



FLUTTER



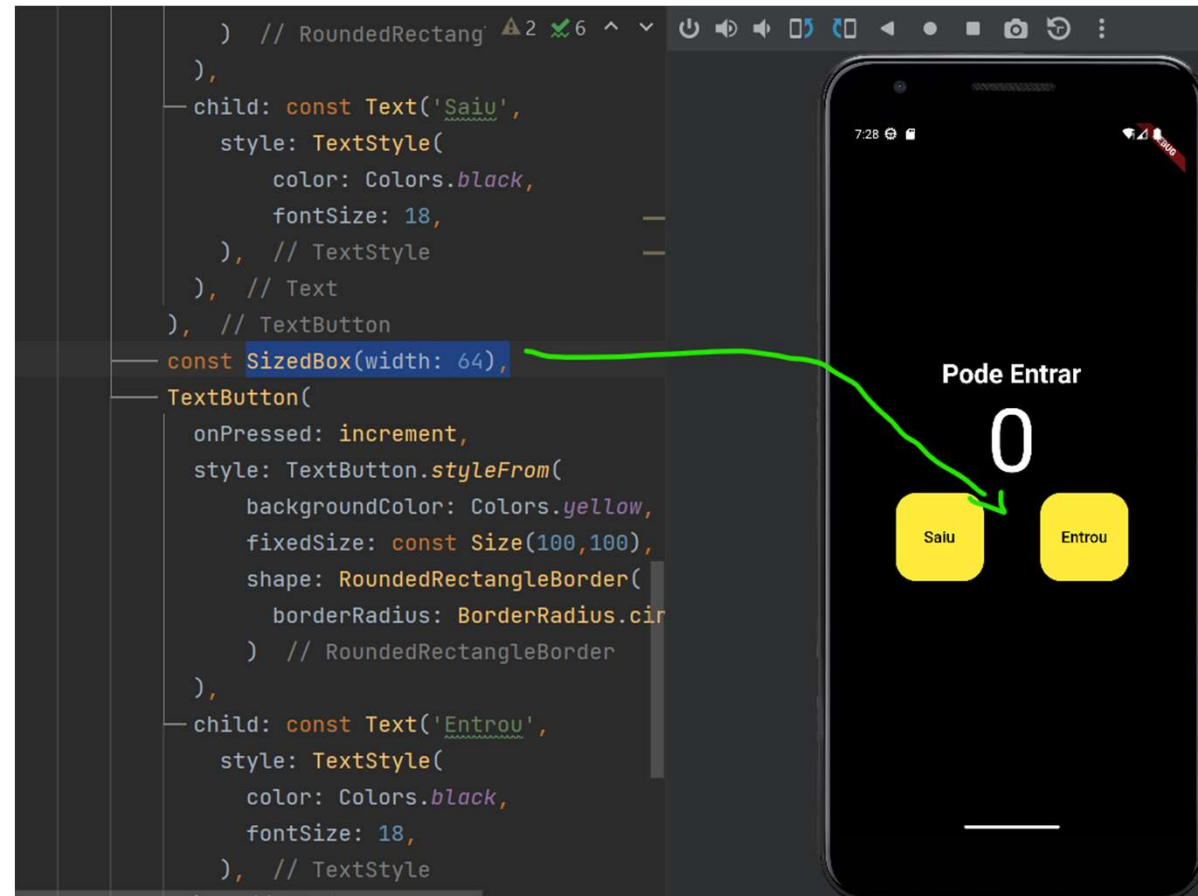
Separando os botões

`SizedBox(width: 64),`

Widget invisível, aplicado
entre os botões.

ou

Widget padding

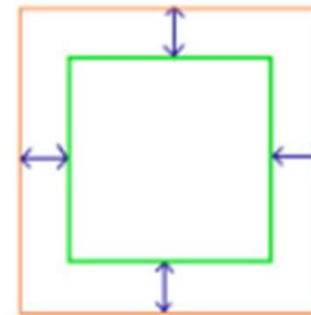




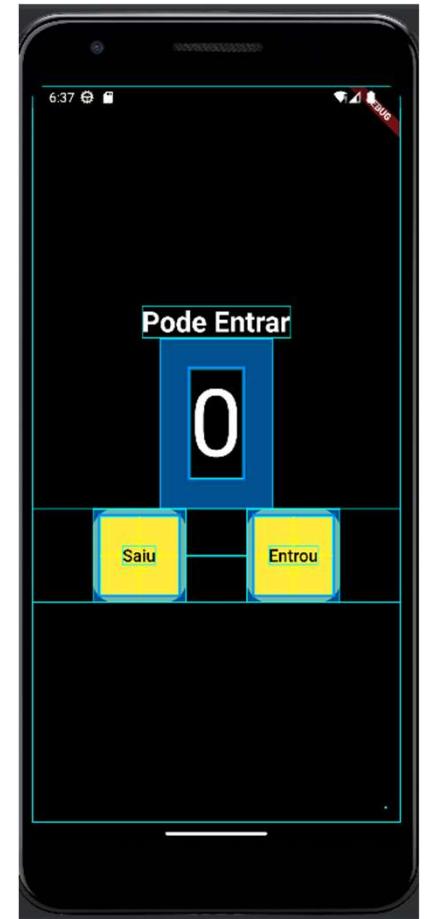
Separando os textos

Se você quiser adicionar algum espaço extra ao redor de um widget.

```
Padding(  
  padding : const EdgeInsets.all(32),  
  child :  
)
```



Padding





Separando os textos

EdgeInsets

EdgeInsets ajuda a criar o preenchimento externo ou interno para um Widget com base nos parâmetros visuais, esquerda, superior, direita e inferior. Não depende da direção do texto





Separando os textos

- EdgeInsets.only** define os valores para cada lado: Esquerda, superior, direita e inferior
- .all** define o mesmo valor para todos os quatro lados esquerda, superior, direita e inferior
- .fromLTRB** define os valores esquerdo, superior, direito e inferior.
- .symetric** cria um espaçamento simétrico na horizontal e vertical

EdgeInsets.only (ex1)

```
Container (  
  color: Colors.greenAccent,  
  padding: EdgeInsets.only(left: 120, top: 50, right: 80),  
  child: ElevatedButton (  
    child: Text("Button"),  
    onPressed: (){}  
  )  
)
```

EdgeInsets.all (ex1)

```
Container (  
  margin: EdgeInsets.all(80),  
  color: Colors.greenAccent,  
  child: Text(  
    "Hi There!",  
    style: TextStyle(fontSize: 28)  
  )  
)
```

EdgeInsets.fromLTRB (ex1)

```
Padding (  
  padding: EdgeInsets.fromLTRB(80, 100, 70, 50),  
  child: ElevatedButton (  
    child: Text("Button"),  
    onPressed: (){}  
  )  
)
```

EdgeInsets.symmetric (ex1)

```
Container (  
  color: Colors.greenAccent,  
  padding: EdgeInsets.symmetric(horizontal: 120, vertical: 50),  
  child: ElevatedButton (  
    child: Text("Button"),  
    onPressed: (){}  
  )  
)
```



FLUTTER



Estilizando o app



FLUTTER



Imagem de Fundo

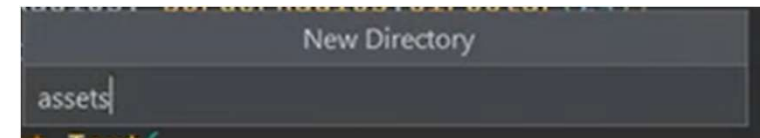
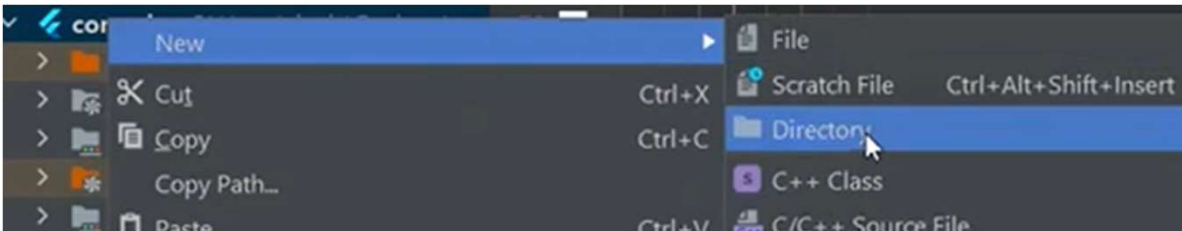
1. Criar uma pasta para guardar as imagens.
2. Configurar o pubspecs.yaml.
3. Criar um container que vai receber a imagem.
4. Colocar a imagem no container via decoration.



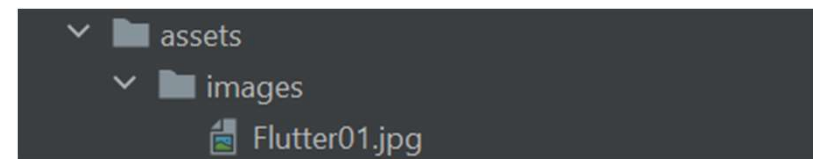


Imagem de Fundo

1. Criar uma pasta para guardar as imagens.



- Crie a pasta Assets.
- Dentro de Assets crie a pasta images.
- Cole o arquivo com a imagem.



FLUTTER



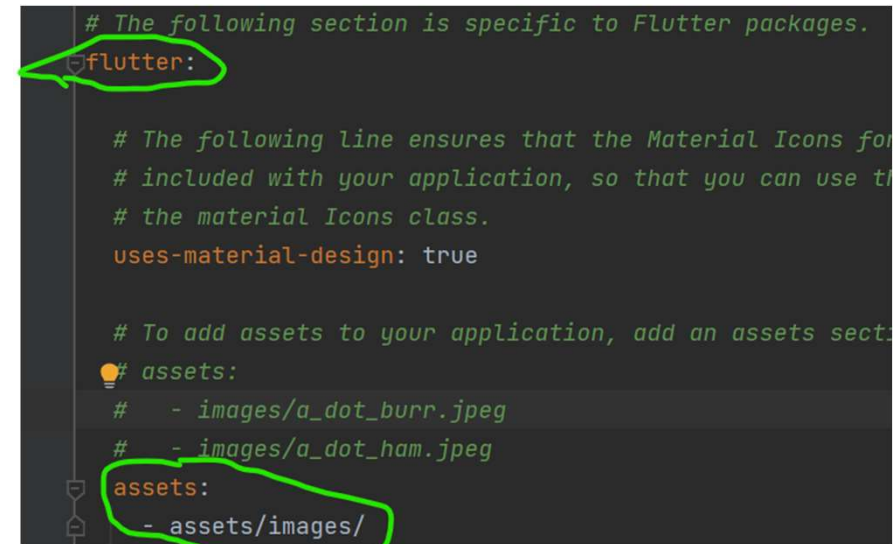
Imagem de Fundo

2. Configurar o pubspecs.yaml.

Neste arquivo, na sessão **flutter:** adicione as linhas conforme abaixo:

assets:

- assets/images/



The screenshot shows a code editor with the following content:

```
# The following section is specific to Flutter packages.
flutter:

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

  # To add assets to your application, add an assets section to
  # the file.
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

  assets:
    - assets/images/
```

Annotations in the image include a green circle around the `flutter:` key and a green rectangle around the `assets:` key and its list item `- assets/images/`.





Imagem de Fundo

3. Criar o container que vai receber a imagem.

- Voltando pra o arquivo **main.dart**.
- Pressione **ALT+ENTER** na Column.
- Selecione **Wrap with Container**.

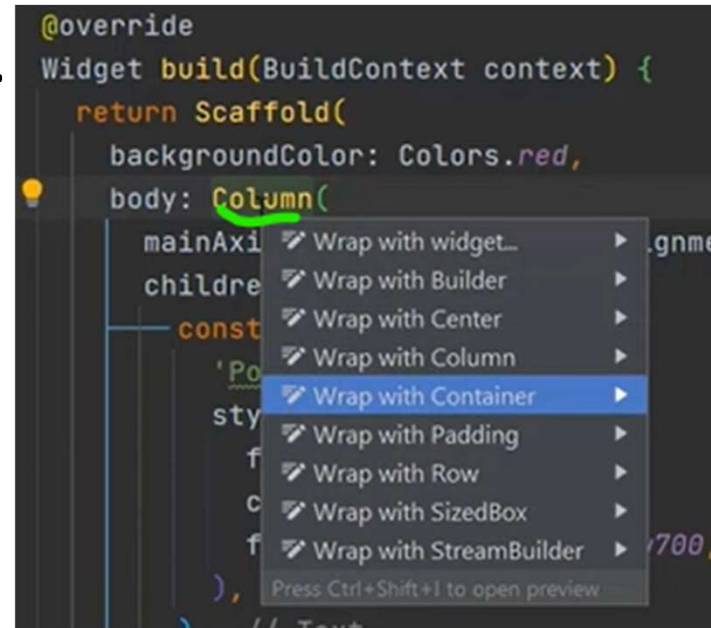




Imagem de Fundo

4. Colocar a imagem no container via decoration

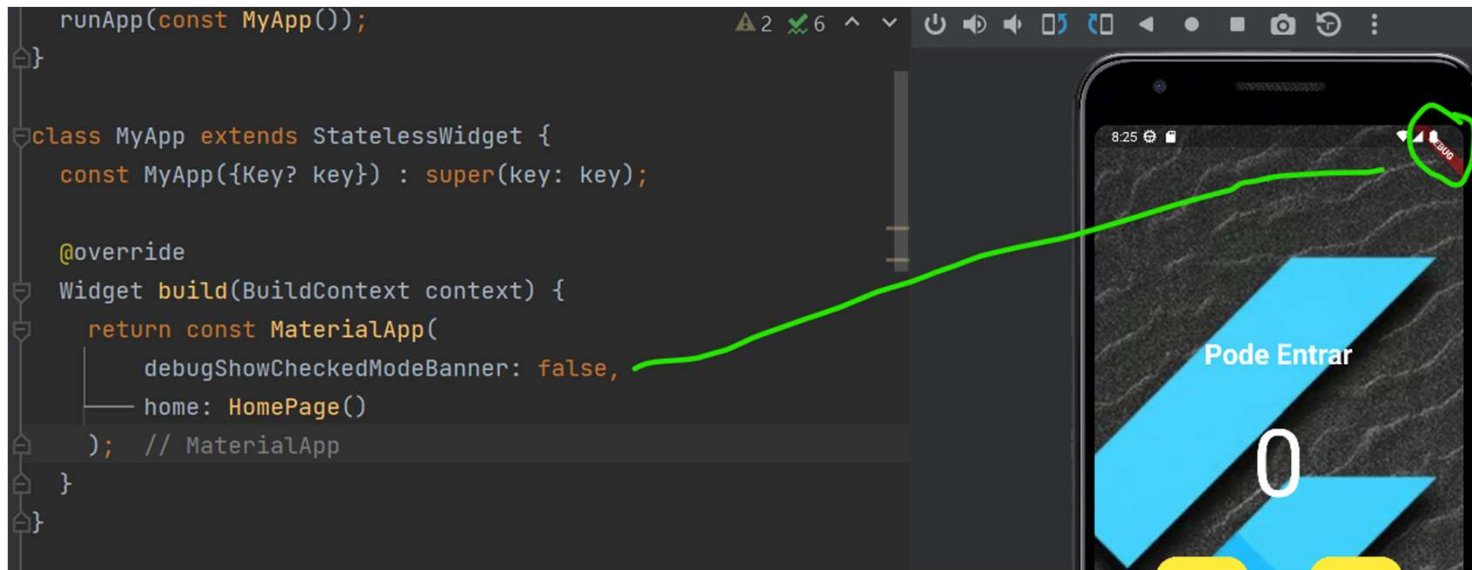
```
decoration: const BoxDecoration(  
  image: DecorationImage(  
    image: AssetImage('assets/images/Flutter01.jpg'),  
    fit: BoxFit.cover,  
  ),  
  fill,  
  contain,  
  fitWidth,  
  fitHeight,  
  none,  
  scaleDown,
```





Remover o Banner de Debug

debugShowCheckedModeBanner: **false**



FLUTTER



Ajustes Finais





Exibir a contagem

1. Criar uma variável inteira.
2. Transformar de inteiro para string.
3. Alterar o state da variável, para que ela seja exibida.

```
class HomePage extends StatelessWidget {  
  const HomePage({Key? key}) : super(key: key);  
  
  int count = 0;  
  
  void decrement() {  
    count--;  
  }  
  
  void increment() {  
    count++;  
  }  
}
```



FLUTTER



Exibir a contagem

Exibir a variável.

```
const Padding(  
  padding : EdgeInsets.all(40),  
  child : Text(  
    '0',  
    style: TextStyle(  
      fontSize: 100,  
      color: Colors.white,  
    ), // TextStyle  
  ), // Text  
) // Padding
```



```
Padding(  
  padding : const EdgeInsets.all(40),  
  child : Text(  
    '$count',  
    style: const TextStyle(  
      fontSize: 100,  
      color: Colors.white,  
    ), // TextStyle  
  ), // Text  
) // Padding
```

O padding não é mais constante pois temos uma variável dentro dele.

O **const** desceu para os elementos filhos.





Stateless vs Stateful

1. A variável count pertence a um widget stateless.
2. Por isso ela não será “renderizada” quando o seu valor alterar.
3. Solução é transformar o widget para stateful

```
class HomePage extends StatelessWidget {  
  HomePage({Key? key}) : super(  
    package:flutter/src/widgets/framework.dart  
    abstract class StatelessWidget extends Widget  
  
    A widget that does not require mutable state.  
    A stateless widget is a widget that describes part of the user interface by building a  
    widgets that describe the user interface more concretely. The building process con  
    description of the user interface is fully concrete (e.g., consists entirely of RenderO  
    describe concrete RenderObjects).  
    www.youtube.com/watch?v=wE7khGHVYY >  
  
    int count = 0;  
  
    void decrement() {  
      count--;  
      print(count);  
    }  
}
```



Stateless vs Stateful

- Se for necessário gerenciar estado, use stateful!

Stateless Widget

Um *Stateless Widget* é um widget sem controle de estado. Mas o que isso significa? Significa que esse tipo de widget não tem alterações dinâmicas, ou seja, as informações ali são estáticas, não mudam ao longo do processo.

```
class WidgetExample extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

Stateful Widget

Agora falando sobre o *Stateful Widget*, é um widget com controle de estado. O que significa que podemos ter informações que se alteram ao longo do processo, ou seja, é com ele que conseguimos construir várias coisas interativas.

```
class WidgetExample extends StatefulWidget {  
  const WidgetExample({super.key});  
  
  @override  
  State<WidgetExample> createState() => _WidgetExample();  
}
```





Transformando para Stateful

1. Acima do widget stateless digite: stful <enter>

```
class HomePage extends StatefulWidget {  
  const HomePage({Key? key}) : super(key: key);  
  
  @override  
  _HomePageState createState() => _HomePageState();  
}  
  
class _HomePageState extends State<HomePage> {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```





Transformando para Stateful

1. Recorte o conteúdo do widget stateless.
2. Cole no widget stateful, sobre a área azul, conforme na figura abaixo.
3. Apague o stateless.

```
class HomePage extends StatefulWidget {  
  const HomePage({Key? key}) : super(key: key);  
  
  @override  
  _HomePageState createState() => _HomePageState();  
}  
  
class _HomePageState extends State<HomePage> {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

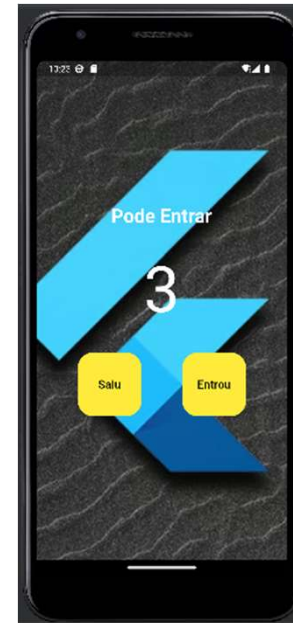




Transformando para Stateful

1. Adicione a função `setState()`.
2. Ela informa ao widget que é necessário chamar o método `build`.
3. Sempre que o **count** for atualizado a tela será renderizada.

```
class _HomePageState extends State<HomePage> {  
  
  int count = 0;  
  
  void decrement(){  
    setState() {  
      count--;  
    };  
    print(count); // imprime no console  
  }  
  
  void increment(){  
    setState() {  
      count++;  
    };  
    print(count); // imprime no console  
  }  
}
```



Código no slide

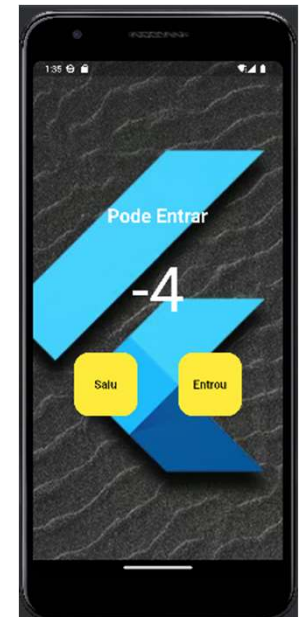


Transformando para Stateful

1. Nosso contador esta ficando negativo.
2. Como resolvemos?

```
bool get isEmpty => count == 0;  
bool get isFull => count == 20;
```

`isEmpty` será true quando o count for igual a zero.
`isFull` será true quando o count for igual a vinte.

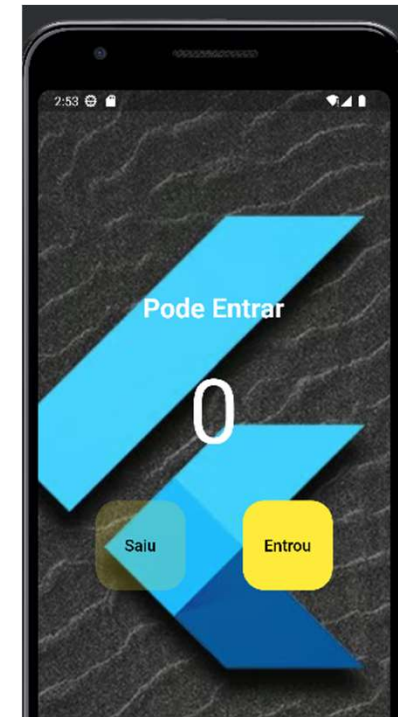


FLUTTER

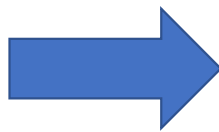
Transformando para Stateful

1. Nosso contador esta ficando negativo.
2. Como resolvemos?

Ao chamar **null** no botão ele será desabilitado.



```
TextButton(  
  onPressed: decrement,  
  style: TextButton.styleFrom(  
    backgroundColor: Colors.yellow,  
    fixedSize: const Size(100,100),  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(24),  
    ) // RoundedRectangleBorder  
  ),  
)
```



```
TextButton(  
  onPressed: isEmpty ? null : decrement,  
  style: TextButton.styleFrom(  
    backgroundColor: isEmpty ? Colors.yellow.withOpacity(0.2) : Colors.yellow,  
    fixedSize: const Size(100,100),  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(24),  
    ) // RoundedRectangleBorder  
  ),  
)
```



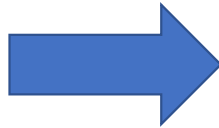


Transformando para Stateful

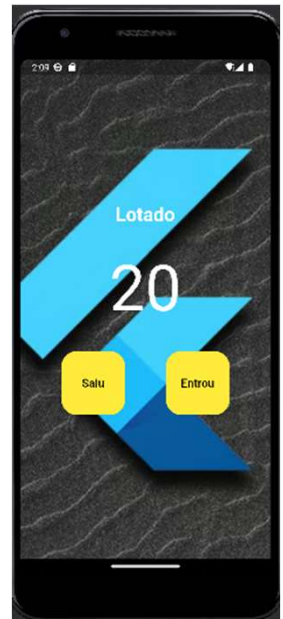
1. Nosso contador esta ficando negativo.
2. Como resolvemos?

Dependendo do valor de `isFull` será escrito o texto "Lotado" ou "Pode Entrar"

```
child: Column(  
  mainAxisAlignment: MainAxisAlignment.center,  
  children: [  
    const Text(  
      'Pode Entrar',  
      style: TextStyle(  
        fontSize: 30,  
        color: Colors.white,  
        fontWeight: FontWeight.w700,  
      ), // TextStyle  
    ), // Text
```



```
child: Column(  
  mainAxisAlignment: MainAxisAlignment.center,  
  children: [  
    Text(  
      isFull ? 'Lotado' : 'Pode Entrar',  
      style: const TextStyle(  
        fontSize: 30,  
        color: Colors.white,  
        fontWeight: FontWeight.w700,  
      ), // TextStyle  
    ), // Text
```



FLUTTER



Operador Ternário



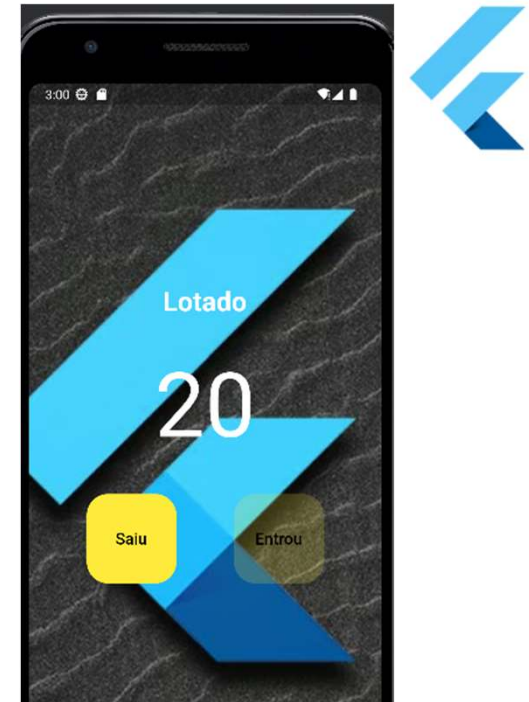
isEmpty ? 'Lotado' : 'Pode entrar'



FLUTTER

Transformando para Stateful

1. Altere a função increment



```
TextButton(  
  onPressed: increment,  
  style: TextButton.styleFrom(  
    backgroundColor: Colors.yellow,  
    fixedSize: const Size(100,100),  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(24),  
    ) // RoundedRectangleBorder  
  ),  
)
```



```
TextButton(  
  onPressed: isFull ? null : increment,  
  style: TextButton.styleFrom(  
    backgroundColor: isFull ? Colors.yellow.withOpacity(0.2) : Colors.yellow,  
    fixedSize: const Size(100,100),  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(24),  
    ) // RoundedRectangleBorder  
  ),  
)
```



FLUTTER



Desafio:

- Colocar a aplicação para funcionar.
- Mudar a cor do texto para **amarelo**, quando contador for maior que 15.
- Mudar a cor do texto para **vermelho**, quando estiver lotado.

