Widgets

Usabilidade, Desenvolvimento Web, Mobile e Jogos

Prof. Gustavo Custodio
gustavo.custodio@anhembi.br

Widgets

Widgets

- Cada widget no Flutter é utilizado para realizar uma única tarefa pequena.

Widgets

- Cada widget no Flutter é utilizado para realizar uma única tarefa pequena.
  - Um widget `Text` mostra texto na tela;
  - Um widget `Padding` cria um espaço entre widgets;
  - Um widget `Scaffold` fornece uma estrutura para uma tela.

Widgets

- Um Widget sem estato (*Stateless*) é o bloco primário para construir interfaces de usuário.

Widgets

- Um Widget sem estato (*Stateless*) é o bloco primário para construir interfaces de usuário.
  - O Flutter consegue facilmente renderizar muitos widgets desse tipo sem dificuldade.
  - Widgets sem estado são imutáveis, quando criados, não podem ser mais modificados.

Widgets

- Vamos criar uma tela utilizando widgets imutáveis.

Widgets

- Vamos criar uma tela utilizando widgets imutáveis.
- Utilizaremos:

Widgets

- Vamos criar uma tela utilizando widgets imutáveis.
- Utilizaremos:
  - `Scaffold`;
  - `Container`;
  - `Padding`.

Widgets

- Vamos criar uma tela utilizando widgets imutáveis.
- Utilizaremos:
  - `Scaffold`;
  - `Container`;
  - `Padding`.
- Criaremos tudo dentro de uma classe chamada `ImmutableWidget`.

Widgets

- Crie um novo projeto em Flutter:

Widgets

- Crie um novo projeto em Flutter:
  - `flutter create exemplo-widgets`.
  - ou então utilize o atalho `Ctrl+Shift+P` do VSCode.

Widgets

- Crie um novo projeto em Flutter:
  - `flutter create exemplo-widgets`.
  - ou então utilize o atalho `Ctrl+Shift+P` do VSCode.
- Crie um arquivo `immutable_widget.dart` na pasta `lib`.

Widgets

- No arquivo `main.dart`, insira o seguinte código:

Widgets

- No arquivo `main.dart`, insira o seguinte código:

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

void main() {
  runApp(const ExemploWidgets());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: ImmutableWidget(),
    ); // MaterialApp
  }
}
```

Widgets

- Observe que estamos importanto o arquivo `immutable_widget.dart` para nossa classe principal, de acordo com a linha:

Widgets

- Observe que estamos importanto o arquivo `immutable_widget.dart` para nossa classe principal, de acordo com a linha:
  - `import './immutable\_widget.dart';`
  - Agora vamos adicionar conteúdo a esse arquivo.

Widgets

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

class ImmutableWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      color: Colors.green,
      child: Padding(
        padding: EdgeInsets.all(40),
        child: Container(
          color: Colors.purple,
          child: Padding(
            padding: const EdgeInsets.all(50),
            child: Container(
              color: Colors.blue,
            ),
          ),
        ),
      ),
    );
  }
}
```

Widgets

- O resultado é um retângulo de três cores.

Widgets

- O resultado é um retângulo de três cores.

Containers, Columns e Rows

- O método `build` retorna um `Container` (verde)

Containers, Columns e Rows

- O método `build` retorna um `Container` (verde)
  - que contém um `Padding`
    - onde o `EdgeInsets.all(40)` define um espaço de 40 pixels nos 4 lados

Containers, Columns e Rows

- O método `build` retorna um `Container` (verde)
  - que contém um `Padding`
    - onde o `EdgeInsets.all(40)` define um espaço de 40 pixels nos 4 lados
  - que contém outro `Container` (roxo)

Containers, Columns e Rows

- O método `build` retorna um `Container` (verde)
  - que contém um `Padding`
    - onde o `EdgeInsets.all(40)` define um espaço de 40 pixels nos 4 lados
  - que contém outro `Container` (roxo)
  - que contém outro `Padding`

Containers, Columns e Rows

- O método `build` retorna um `Container` (verde)
  - que contém um `Padding`
    - onde o `EdgeInsets.all(40)` define um espaço de 40 pixels nos 4 lados
  - que contém outro `Container` (roxo)
  - que contém outro `Padding`
  - que contém um `Container` azul.

Containers, Columns e Rows

- Observe que cada widget contido dentro de outro é um parâmetro opcional chamado `child`.

Containers, Columns e Rows

- Observe que cada widget contido dentro de outro é um parâmetro opcional chamado `child`.
- Alguns widgets possuem um parâmetro opcional chamado `children`
  - ou seja, múltiplos widgets contidos nele.

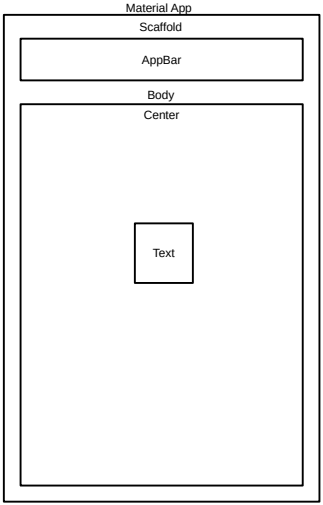
Containers, Columns e Rows

- Observe que cada widget contido dentro de outro é um parâmetro opcional chamado `child`.
- Alguns widgets possuem um parâmetro opcional chamado `children`
  - ou seja, múltiplos widgets contidos nele.
  - O widget `Column` é um exemplo.

Hello World

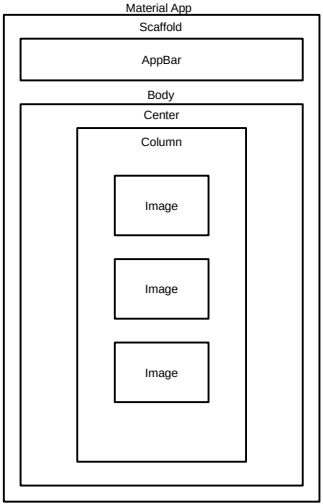
- Observe o código do `Hello World`.

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```



Containers, Columns e Rows

- Representamos o app com um diagrama mostrando como os widgets estão aninhados.
- Agora vamos tentar criar o seguinte app:



Imagens

- Antes de continuar, precisamos aprender como adicionar imagens a um proojeto.

Imagens

- Antes de continuar, precisamos aprender como adicionar imagens a um proojeto.
- Crie uma pasta chamada `images` na raiz do projeto.
- Baixe as imagens que deseja adicionar e coloque-as nessa pasta.

Imagens

- É preciso adicionar as imagens como `assets` no projeto.

Imagens

- É preciso adicionar as imagens como `assets` no projeto.
- Abra o arquivo `pubspec.yaml` e procure pela parte do arquivo:

```
1       # assets:
2       # - images/a_dot_burr.jpeg
3       # - images/a_dot_ham.jpeg
```

Imagens

- É preciso adicionar as imagens como `assets` no projeto.
- Abra o arquivo `pubspec.yaml` e procure pela parte do arquivo:

```
# assets:
# - images/a_dot_burr.jpeg
# - images/a_dot_ham.jpeg
```

- Remova o # das linhas (descomente) e substitua pelo caminho das suas imagens.
- Agora o flutter reconhecerá as imagens.

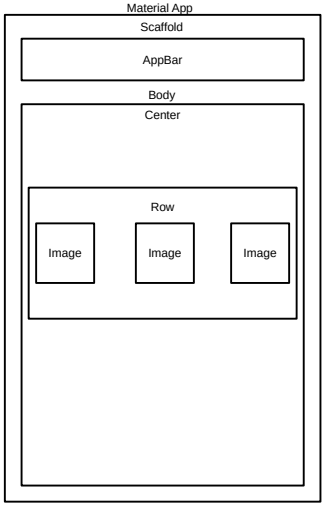
Containers, Columns e Rows

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Colunas e Rows',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Colunas e Rows'),
        ),
        body: Center(
          child: Column(
            children: [
  // Adicionando imagens na coluna
              Image.asset('images/lands_01.jpg'),
              Image.asset('images/lands_02.jpg'),
              Image.asset('images/lands_03.jpg'),
            ],
          ),
        ),
      ),
    );
  }
}
```

- Temos como resultado três imagens alinhadas em uma coluna.

Containers, Columns e Rows

- É também possível dispor conteúdo em uma mesma linha.



Containers, Columns e Rows

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

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

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

 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Colunas e Rows',
     home: Scaffold(
       appBar: AppBar(
         title: const Text('Colunas e Rows'),
       ),
       body: Center(
         child: Row(
           children: [
 // Adicionando imagens na coluna
             Image.asset('images/lands_01.jpg'),
             Image.asset('images/lands_02.jpg'),
             Image.asset('images/lands_03.jpg'),
           ],
         ),
       ),
     ),
   );
 }
}
```

- Temos como resultado três imagens alinhadas em uma linha agora.

Exercício

- Vamos tentar colocar uma imagem seguida de um texto.
  - Use o `CircleAvatar` para imagens arredondadas.

Widgets

AppBar

AppBar

- O AppBar é a barra que fica no topo dos aplicativos.

AppBar

- O `AppBar` é a barra que fica no topo dos aplicativos.
- É comum que os apps possuam ícones de menu, de edição ou de busca nessas barras.

AppBar

- O `AppBar` é a barra que fica no topo dos aplicativos.
- É comum que os apps possuam ícones de menu, de edição ou de busca nessas barras.

AppBar

- O menu à esquerda é definido pelo atributo `leading` do `AppBar`.
- O símbolo de busca é parte do atributos `actions`.
  - Um atributo que define quais ações o usuário pode realizar em um app.
  - Recebe uma lista de ações possíveis.

AppBar

- Exemplo de código.

AppBar

- Exemplo de código.

```
1
2     class MyApp extends StatelessWidget {
3       const MyApp({Key? key}) : super(key: key);
4
5       // This widget is the root of your application.
6       @override
7       Widget build(BuildContext context) {
8         return MaterialApp(
9           home: BarraApp(),
10        );
11      }
12    }
```

AppBar

- Exemplo de código.

AppBar

- Exemplo de código.

```
class BarraApp extends StatelessWidget {
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.indigo,
        title: Text('Aplicativo com Barra'),
        leading: Padding(
          padding: const EdgeInsets.all(10.0),
          child: Icon(Icons.menu),
        ),
        actions: <Widget>[
          Padding(
            padding: const EdgeInsets.all(15.0),
            child: Icon(Icons.search),
          ),
        ],
      ),
    );
  }
}
```

AppBar

- `Icons.menu` representa o ícone com 3 linhas horizontais;

AppBar

- `Icons.menu` representa o ícone com 3 linhas horizontais;
- `Icons.search` representa o ícone de busca.

AppBar

- `Icons.menu` representa o ícone com 3 linhas horizontais;
- `Icons.search` representa o ícone de busca.
- Vamos adicionar um ícone de editar ao lado da busca:
  - `Icons.edit`.

AppBar

- O menu no canto superior esquerdo é um lugar comum para adicionar a navegação dentro do aplicativo.

AppBar

- O menu no canto superior esquerdo é um lugar comum para adicionar a navegação dentro do aplicativo.
  - Na próxima aula veremos como adicionar um comportamento nesse menu.
  - E, consequentemente, como navegar por telas diferentes.

Widgets

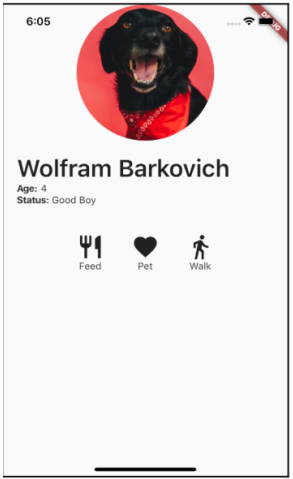
Prática

Prática

- Vamos criar uma tela de perfil para um cachorro.

Prática

- Vamos criar uma tela de perfil para um cachorro.



Prática

- Primeiro vamos começar com um código de uma tela vazia.

Prática

- Primeiro vamos começar com um código de uma tela vazia.
- Teremos três funções para construir a tela:

Prática

- Primeiro vamos começar com um código de uma tela vazia.
- Teremos três funções para construir a tela:
  - `_buildProfileImage(BuildContext context);`
  - `_buildProfileDetails(BuildContext context);`
  - `_buildActions(BuildContext context).`

Prática

- O `BuildContext` é utilizado para localizar cada widget na árvore de widgets.

Prática

- O `BuildContext` é utilizado para localizar cada widget na árvore de widgets.
- Dessa forma é necessário passá-lo como parâmetro sempre que retornamos um widget que ajudará a construir essa árvore.

Prática

- Crie um arquivo chamado `profile_screen.dart`.

Prática

- Crie um arquivo chamado `profile_screen.dart`.
- No `main.dart` adicione o seguinte código:

Prática

- Crie um arquivo chamado `profile_screen.dart`.
- No `main.dart` adicione o seguinte código:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ProfileScreen(),
    );
  }
}
```

Prática

- Agora adicione o código no arquivo `profile_screen.dart`:

Prática

- Agora adicione o código no arquivo `profile_screen.dart`:

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

class ProfileScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: <Widget>[
          _buildProfileImage(context),
          _buildProfileDetails(context),
          _buildActions(context),
        ],
      ),
    );
  }

  Widget _buildProfileImage(BuildContext context) {
    return Container();
  }
  Widget _buildProfileDetails(BuildContext context) {
    return Container();
  }
  Widget _buildActions(BuildContext context) {
    return Container();
  }
}
```

Prática

- Por enquanto o app não terá nenhuma funcionalidade.

Prática

- Por enquanto o app não terá nenhuma funcionalidade.
- Cada método corresponde a uma parte diferente do app.

Prática

- Por enquanto o app não terá nenhuma funcionalidade.
- Cada método corresponde a uma parte diferente do app.
- Vamos começar com o `_buildProfileImage(BuildContext context);`

Prática

```
Widget _buildProfileImage(BuildContext context) {
  return Container(
    width: 200, // largura
    height: 200, //altura
    child: CircleAvatar(
      backgroundImage: NetworkImage(
          'https://www.collinsdictionary.com/images/full/dog_230497594.jpg'),
    ),
  );
}
```

Prática

```
Widget _buildProfileImage(BuildContext context) {
  return Container(
    width: 200, // largura
    height: 200, //altura
    child: CircleAvatar(
      backgroundImage: NetworkImage(
          'https://www.collinsdictionary.com/images/full/dog_230497594.jpg'),
    ),
  );
}
```

- Isso resulta na foto do cachorro aparecendo na tela.

Prática

- Depois disso, adicionamos informações sobre o cachorro.

Prática

- Depois disso, adicionamos informações sobre o cachorro.

```
Widget _buildProfileDetails(BuildContext context) {
  return Padding(
    padding: const EdgeInsets.all(20.0),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
        Text(
          'Wolfram Barkovich',
          style: TextStyle(fontSize: 35, fontWeight: FontWeight.w600),
        ),
        _buildDetailsRow('Age', '4'),
        _buildDetailsRow('Status', 'Good Boy'),
      ],
    ),
  );
}
```

Prática

- O texto é adicionado em um widget separado.

Prática

- O texto é adicionado em um widget separado.

```
Widget _buildDetailsRow(String heading, String value) {
  return Row(
    children: <Widget>[
      Text(
        '$heading: ',
        style: TextStyle(fontWeight: FontWeight.bold),
      ),
      Text(value),
    ],
  );
}
```

Prática

- Depois disso, adicionamos alguns botões falsos que simulam a interação com o cachorro.

Prática

- Depois disso, adicionamos alguns botões falsos que simulam a interação com o cachorro.

```
Widget _buildActions(BuildContext context) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      _buildIcon(Icons.restaurant, 'Feed'),
      _buildIcon(Icons.favorite, 'Pet'),
      _buildIcon(Icons.directions_walk, 'Walk'),
    ],
  );
}
```

Prática

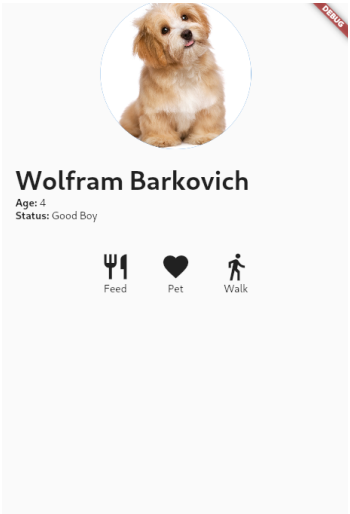
```
1
2  Widget _buildIcon(IconData icon, String text) {
3    return Padding(
4      padding: const EdgeInsets.all(20.0),
5      child: Column(
6        children: <Widget>[Icon(icon, size: 40), Text(text)],
7      ),
8    );
9  }
```

Prática

- Com isso, obtemos o seguinte resultado:

Prática

- Com isso, obtemos o seguinte resultado:



Exercício

- Coloque uma imagem de fundo atrás da foto do cachorro.
- Use o widget `Stack` para isso.

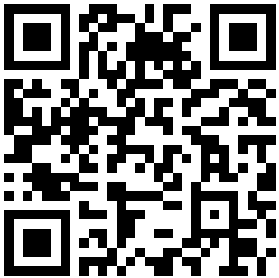
Exercício

- Crie o `layout` de um aplicativo de perguntas e respostas utilizando os widgets do flutter.

Referências

Simone Alessandria and Brian Kayfitz.
Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart.
Packt Publishing Ltd, 2021.

Conteúdo



https://gustavotcustodio.github.io/usabilidade.html

Obrigado

gustavo.custodio@anhembi.br