



# Componentes Gráficos

**N**este módulo, vamos explorar os fundamentos da construção de interfaces no Flutter usando widgets. Widgets são os blocos de construção básicos de uma aplicação Flutter e tudo no Flutter é um widget. Discutiremos como utilizar widgets básicos como Text, Container, Row, Column, e Scaffold para criar interfaces de usuário interativas e responsivas. Veremos exemplos práticos e comentados de código que ilustram como esses widgets são usados para estruturar a interface do aplicativo.

## Construindo Interfaces com Widgets

### Conceito de Widgets

Widgets são os elementos fundamentais de uma aplicação Flutter. Cada parte da interface do usuário é composta por widgets. Eles podem ser visuais, como botões e textos, ou estruturais, como colunas e linhas.

### Widgets Básicos

#### 1. Text Widget

O Text widget é usado para exibir texto na tela.

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

```
void main() {
```

```
  runApp(MyApp());
```

```

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text('Exemplo de Texto'),

        ),

        body: Center(

          child: Text('Olá, Mundo!'),

        ),

      ),

    );

  }

}

```

Explicação: Neste exemplo, criamos um aplicativo simples que exibe “Olá, Mundo!” no centro da tela. O widget Text é usado para exibir o texto.

## 2. Container Widget

O Container é um widget flexível usado para criar caixas com propriedades específicas, como margens, padding e alinhamento.

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

```
void main() {
```

```
  runApp(MyApp());
```

```
}
```

```
class MyApp extends StatelessWidget {
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
```

```
      home: Scaffold(
```

```
        appBar: AppBar(
```

```
          title: Text('Exemplo de Container'),
```

```
        ),
```

```
        body: Center(
```

```
          child: Container(
```

```
            padding: EdgeInsets.all(16.0),
```

```
            color: Colors.blue,
```

```
            child: Text('Texto dentro de um Container', style: TextStyle(color: Colors.white)),
```

```
    ),  
  
    ),  
  
    ),  
  
    );  
  
  }  
  
}
```

Explicação: Aqui, o Container envolve o Text widget e adiciona padding e cor de fundo. Isso permite personalizar a aparência e o layout do texto.

### 3. Row e Column Widgets

Row e Column são usados para organizar widgets em linhas e colunas, respectivamente.

```
import 'package:flutter/material.dart';  
  
void main() {  
  
  runApp(MyApp());  
  
}  
  
class MyApp extends StatelessWidget {  
  
  @override  
  
  Widget build(BuildContext context) {  
  
    return MaterialApp(  
  
      home: Scaffold(  
  

```

```
appBar: AppBar(
```

```
  title: Text('Exemplo de Row e Column'),
```

```
),
```

```
body: Column(
```

```
  mainAxisAlignment: MainAxisAlignment.center,
```

```
  children: <Widget>[
```

```
    Row(
```

```
      mainAxisAlignment: MainAxisAlignment.center,
```

```
      children: <Widget>[
```

```
        Text('Linha 1, '),
```

```
        Text('Item 1'),
```

```
      ],
```

```
    ),
```

```
    Row(
```

```
      mainAxisAlignment: MainAxisAlignment.center,
```

```
      children: <Widget>[
```

```
        Text('Linha 2, '),
```

```
        Text('Item 2'),
```

```
      ],
```

```
        ),  
      ],  
    ),  
  ),  
);  
  
}  
  
}
```

Explicação: Usamos Column para organizar as linhas verticalmente e Row para organizar os textos horizontalmente. Isso ajuda a estruturar a interface do usuário de forma ordenada.

#### **4. Scaffold Widget**

O Scaffold é um widget que fornece uma estrutura básica de layout para um aplicativo.

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {
```

```
return MaterialApp(  
  
  home: Scaffold(  
  
    appBar: AppBar(  
  
      title: Text('Exemplo de Scaffold'),  
  
    ),  
  
    body: Center(  
  
      child: Text('Corpo do Scaffold'),  
  
    ),  
  
    floatingActionButton: FloatingActionButton(  
  
      onPressed: () {},  
  
      child: Icon(Icons.add),  
  
    ),  
  
  ),  
  
);  
  
}
```

Explicação: O Scaffold fornece uma estrutura visual com AppBar no topo, body no meio e um FloatingActionButton flutuante. Isso facilita a criação de uma interface padronizada.

Nesta aula, exploraremos como adicionar animações aos seus aplicativos Flutter. Animações são essenciais para criar experiências de usuário envolventes e responsivas. Discutiremos o uso de widgets animados básicos, como `AnimatedContainer` e `AnimatedOpacity`, e veremos exemplos práticos de como animar mudanças de estado e criar transições suaves.

## **Animações no Flutter**

### Conceito de Animações

Animações no Flutter tornam a interface do usuário mais dinâmica e interativa. Flutter fornece diversos widgets animados para facilitar a criação de animações.

### Widgets de Animação Básicos

#### **1. `AnimatedContainer`**

- O `AnimatedContainer` permite animar mudanças em suas propriedades, como tamanho, cor e borda.

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

void main() {

  runApp(MyApp());

}

class MyApp extends StatefulWidget {

  @override

  _MyAppState createState() => _MyAppState();

}
```



```
class _MyAppState extends State<MyApp> {

  bool _isExpanded = false;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text('Exemplo de AnimatedContainer'),

        ),

        body: Center(

          child: GestureDetector(

            onTap: () {

              setState(() {

                _isExpanded = !_isExpanded;

              });

            },

            child: AnimatedContainer(

              width: _isExpanded ? 200.0 : 100.0,

              height: _isExpanded ? 200.0 : 100.0,
```

```

        color: _isExpanded ? Colors.blue : Colors.red,

        alignment: _isExpanded ? Alignment.center :
AlignmentDirectional.topCenter,

        duration: Duration(seconds: 1),

        child: Text('Toque Aqui', style: TextStyle(color: Colors.white)),

    ),

),

),

),

);

}

}

```

Explicação: AnimatedContainer muda de tamanho e cor ao ser tocado. A mudança é animada com a duração de 1 segundo, proporcionando uma transição suave.

## 2. AnimatedOpacity

O AnimatedOpacity permite animar mudanças na opacidade de um widget.

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

```
void main() {
```

```
  runApp(MyApp());
```

```
}
```

```
class MyApp extends StatefulWidget {
```

```
  @override
```

```
  _MyAppState createState() => _MyAppState();
```

```
}
```

```
class _MyAppState extends State<MyApp> {
```

```
  bool _visible = true;
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
```

```
      home: Scaffold(
```

```
        appBar: AppBar(
```

```
          title: Text('Exemplo de AnimatedOpacity'),
```

```
        ),
```

```
        body: Center(
```

```
          child: Column(
```

```
            mainAxisAlignment: MainAxisAlignment.center,
```

```
            children: <Widget>[
```

```
              AnimatedOpacity(
```

```
        opacity: _visible ? 1.0 : 0.0,

        duration: Duration(seconds: 1),

        child: Text('Visível/Invisível'),

    ),

    ElevatedButton(

        onPressed: () {

            setState(() {

                _visible = !_visible;

            });

        },

        child: Text('Alternar Visibilidade'),

    ),

  ],

),

),

),

),

);

}

}
```

Explicação: AnimatedOpacity permite que a opacidade de um widget mude suavemente entre visível e invisível. O botão alterna a visibilidade do texto com uma animação de 1 segundo.

Nesta aula, discutiremos como integrar elementos multimídia, como imagens e vídeos, em aplicativos Flutter. Veremos como usar o widget Image para exibir imagens de várias fontes e como utilizar o pacote video\_player para reproduzir vídeos. Também abordaremos como adicionar áudio e criar uma experiência multimídia completa.

## **Integração de Elementos Multimídia**

### Uso de Imagens

#### **1. Image Widget**

O Image widget é usado para exibir imagens na aplicação.

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

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(
```

```

appBar: AppBar(

  title: Text('Exemplo de Imagem'),

),

body: Center(

  child: Image.network('https://flutter.dev/assets/homepage/carousel/slide_1-
bg-opaque-
5c6c77e6560f66dbce5cf0c1e64f0c17279e0e0a923c6db5b69d2ef1c85de14d.png'),

),

),

);

}

}

```

Explicação: Este exemplo carrega uma imagem da internet usando Image.network. O widget Image suporta várias fontes de imagem, incluindo assets locais, rede e memória.

## Reproduzindo Vídeos

### **1. Video Player**

Para reproduzir vídeos, usamos o pacote video\_player.

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

```
import 'package:video_player/video_player.dart';
```

```
void main() {
```

```
  runApp(MyApp());
```

```
}
```

```
class MyApp extends StatefulWidget {
```

```
  @override
```

```
  _MyAppState createState() => _MyAppState();
```

```
}
```

```
class _MyAppState extends State<MyApp> {
```

```
  late VideoPlayerController _controller;
```

```
  @override
```

```
  void initState() {
```

```
    super.initState();
```

```
    controller = VideoPlayerController.network(https://flutter.github.io/assets-for-api-docs/assets/videos/butterfly.mp4)
```

```
    ...initialize().then(() {
```

```
      setState(() {}); // Atualiza o estado quando a inicialização estiver  
concluída.
```

```
    });
```

```
  }
```

```
  @override
```

```
Widget build(BuildContext context) {

  return MaterialApp(

    home: Scaffold(

      appBar: AppBar(

        title: Text('Exemplo de Vídeo'),

      ),

      body: Center(

        child: _controller.value.isInitialized

          ? AspectRatio(

              aspectRatio: _controller.value.aspectRatio,

              child: VideoPlayer(_controller),

            )

          : CircularProgressIndicator(),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: () {

          setState(() {

            _controller.value.isPlaying ? _controller.pause() : _controller.play();

          });

        },

      ),

    ),

  );

}
```



```

    },

    child: Icon(_controller.value.isPlaying ? Icons.pause : Icons.play_arrow),

  ),

),

);

}

```

```
@override
```

```

void dispose() {

  super.dispose();

  _controller.dispose();

}

}

```

Explicação: Este exemplo utiliza o pacote video\_player para carregar e reproduzir um vídeo da internet. O VideoPlayerController gerencia o estado do vídeo, enquanto o AspectRatio e VideoPlayer exibem o vídeo. Um FloatingActionButton permite pausar e reproduzir o vídeo.

Nesta aula, aprenderemos como personalizar a aparência de aplicativos Flutter usando temas. Discutiremos como definir temas globais que afetam toda a aplicação e temas específicos para widgets. Exploraremos como usar o widget Theme e a classe ThemeData para ajustar cores, fontes e outros aspectos visuais.

## Customização e Temas

### Definindo Temas Globais

#### 1. Tema Global

O tema global é definido usando o widget `MaterialApp` e a classe `ThemeData`.

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

```
void main() {
```

```
  runApp(MyApp());
```

```
}
```

```
class MyApp extends StatelessWidget {
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
```

```
      theme: ThemeData(
```

```
        primarySwatch: Colors.blue,
```

```
        textTheme: TextTheme(
```

```
          bodyText1: TextStyle(color: Colors.red, fontSize: 20),
```

```
        ),
```

```
      ),
```

```
home: Scaffold(  
  
  appBar: AppBar(  
  
    title: Text('Exemplo de Tema Global'),  
  
  ),  
  
  body: Center(  
  
    child: Text('Texto com tema personalizado'),  
  
  ),  
  
),  
  
);  
  
}
```

Explicação: Definimos um tema global para o aplicativo usando ThemeData, onde configuramos uma cor primária (primarySwatch) e personalizamos a aparência do texto (textTheme).

### Tema Específico para Widgets

#### **1. Tema Específico**

Podemos definir temas específicos para widgets usando o widget Theme.

```
import 'package:flutter/material.dart';  
  
void main() {  
  
  runApp(MyApp());  
  
}
```

```
}
```

```
class MyApp extends StatelessWidget {
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return MaterialApp(
```

```
      home: Scaffold(
```

```
        appBar: AppBar(
```

```
          title: Text('Exemplo de Tema Específico'),
```

```
        ),
```

```
        body: Center(
```

```
          child: Theme(
```

```
            data: ThemeData(
```

```
              buttonTheme: ButtonThemeData(
```

```
                buttonColor: Colors.green,
```

```
            ),
```

```
          ),
```

```
          child: ElevatedButton(
```

```
            onPressed: () {},
```

```
            child: Text('Botão com Tema Específico'),
```

```
    ),  
  
    ),  
  
    ),  
  
    ),  
  
    );  
  
}  
  
}
```

Explicação: Usamos o widget Theme para definir um tema específico para um botão. O ThemeData ajusta a cor do botão, mostrando como podemos personalizar a aparência de widgets individuais dentro do aplicativo.

Esses exemplos e explicações detalhadas devem fornecer uma base sólida para qualquer iniciante em Flutter começar a criar interfaces interativas, implementar animações, integrar elementos multimídia e personalizar a aparência de seus aplicativos. Para mais informações, consulte a documentação oficial do Flutter: [Flutter Documentation](https://flutter.dev/docs).

## Materiais Extras

Você pode realizar o download do arquivo contendo os materiais extras utilizados ao longo das aulas por meio do seguinte link:

<https://drive.google.com/file/d/1mg7lqMI8Pt2zl0rHlsFS0Qew00YN-sEX/view?usp=sharing>.

## Conteúdo Bônus

Como conteúdo bônus, sugiro o vídeo “Stateless e Stateful Widgets e Ciclo de Vida” do curso de Flutter ministrado pelo Prof. Diego Antunes, disponível no YouTube. Nesse vídeo, o professor explica detalhadamente as diferenças entre Stateless e Stateful Widgets, essenciais para o desenvolvimento com Flutter. Além disso, ele explora o ciclo de vida de um objeto State, utilizado em Stateful Widgets, abordando como gerenciar e otimizar o comportamento de interfaces dinâmicas.

### **Referências Bibliográficas**

BOYLESTAD, R. L.; NASHELSKY, L. Dispositivos Eletrônicos e Teoria de Circuitos. 11. ed. Pearson, 2013.

DEITEL, P. J.; DEITEL, H. M. Ajax, Rich Internet Applications e Desenvolvimento Web para Programadores. Pearson, 2008.

DUARTE, W. Delphi para Android e iOS: Desenvolvendo Aplicativos Móveis. Brasport, 2015.

FELIX, R.; SILVA, E. L. da. Arquitetura para Computação Móvel. 2. ed. Pearson, 2019.

LEE, V.; SCHNEIDER, H.; SCHELL, R. Aplicações Móveis: Arquitetura, Projeto e Desenvolvimento. Pearson, 2005.

MARINHO, A. L.; CRUZ, J. L. da. Desenvolvimento de Aplicações para Internet. 2. ed. Pearson, 2019.

MOLETTA, A. Você na Tela: Criação Audiovisual para a Internet. Summus, 2019.

SILVA, D. (Org.) Desenvolvimento para dispositivos móveis. Pearson, 2017.

**Ir para exercício**