Flutter Material Widgets em Português

UM GUIA PARA ESTUDO OU CONSULTAS RÁPIDAS

Marco A. Braghim

1ª revisão

INTRODUÇÃO

Flutter é a estrutura de projeto básico (framework) do Google para construção de aplicativos mobile, sites e softwares desktops cross-platform, o que significa que os programas construídos com ele devem funcionar em uma grande gama de plataformas diferentes como Android, IOS, MacOS, Windows, Linux, etc... Talvez até o momento o Flutter Web e o Flutter Desktop ainda não tenham sido oficialmente lançados, mas eles já foram anunciados e devem entrar em produção em breve.

Uma das coisas que estranhamos um pouco logo de cara quando começamos as primeiras brincadeiras e testes com Flutter é o seu modo de construção de visual hierárquico, quero dizer, o modo como cada elemento que compõe a tela visualmente é embutido um dentro do outro, declarativamente. Nada de errado com isso, na verdade toda construção de visual em software é feita assim de alguma forma, se você pensar em HTML, por exemplo.

Dentro do Flutter todos os elementos são widgets, até mesmo um pequeno texto ou um ícone. Nós não vamos entrar aqui no mérito de discutir se isso é uma boa prática ou não, até porque acreditamos que isso depende bastante também da sua própria organização de projeto e familiaridade com o framework, quanto mais tempo você passa desenvolvendo com esta tecnologia mais se acostuma com ela e a produtividade aumenta gradativamente.

O que vamos trazer aqui é um guia que vai te dar um norte para consultar rapidamente o que cada widget faz e qual se aplica melhor à sua necessidade. Vamos dar alguns exemplos e tentar quando possível também dizer onde não usar, o que pode ajudar a economizar seu tempo.

Mas cuidado, existem muitos widgets oficiais do Flutter e nós vamos aqui focar apenas naqueles que são usados com o Material Design, mas não todos. Digo “oficial” porque durante o desenvolvimento você vai criando seus próprios widgets, mas eles são seus e não oficiais. Para consultar a lista completa de widgets do Flutter visite o link abaixo:

DICA: Antes de começar a de fato a desenhar o visual do seu aplicativo consulte a Ferramenta de cores Material Design (link abaixo) para definir a combinação de cores que se encaixa perfeitamente ao propósito do seu aplicativo.

Documentação Oficial: https://flutter.dev/
Widget Catalog: https://flutter.dev/docs/development/ui/widgets
Ferramenta de cores Material Design:
https://material.io/resources/color/#!/?view.left=0&view.right=0&primary.color=01579B&secondary.color=EF6C00&secondary.text.color=ffffff

Ps.: Para consultar rapidamente um widget de seu interesse, se este documento não for impresso, utilize CTRL+F para pesquisar.

TEXTOS, IMAGENS E RECURSOS

No Flutter tudo são widgets, até mesmo um pequeno texto ou um ícone. Para conseguir construir o visual perfeito do seu aplicativo você vai precisar saber usar os principais e mais simples widgets disponíveis.

Text

Este widget foi desenvolvido especialmente para adicionar texto ao seu aplicativo. Ele pode ser totalmente estilizado, mas conta com algumas limitações, por exemplo, para que o texto seja selecionável deve-se usar o SelectableText em seu lugar.

É possível também criar textos com estilos mais “ricos” como negrito, itálico e até mesmo separar pequenos blocos de texto para aplicar um estilo um pouco diferente, vamos supor que você tenha uma frase e apenas uma palavra dessa frase deverá ficar em itálico, então use o método Text.rich() ou simplesmente o widget RichText.

```
// Texto simples
Text('Um texto qualquer');
Text('Um texto amarelo', style: TextStyle(color: Colors.yellow));

// É possível e recomendado herdar os estilos definidos no topo
// do aplicativo, assim sempre que algo mudar passará a valer
// para o aplicativo inteiro
Text(
  'Título herdado',
  style: TextStyle(
    fontSize: Theme.of(context).textTheme.title.fontSize
  )
);
```

Um exemplo com texto um pouco mais estilizado:

```
// Este é um exemplo de texto com alta ocorrência de estilização
Text.rich(
  TextSpan(
    text: 'Um texto com',
    children: <TextSpan>[
      TextSpan(
        text: ' grande',
        style: TextStyle(fontWeight: FontWeight.bold)
      ),
      TextSpan(
        text: ' style!',
        style: TextStyle(fontStyle: FontStyle.italic)
      ),
    ],
  ),
);
```

O efeito produzido será:

Um texto com **grande** *style!*

Icon

Este widget vai criar um ícone da lista de ícones do Material Design na posição definida e você poderá definir aqui também seu tamanho e cor. Quando você definir fontes de texto personalizadas dentro do seu aplicativo então provavelmente será necessário fazer o download da fonte MaterialIcons-Regular.ttf e apontá-la dentro do arquivo assets/FontManifest.json.

```
Icon(Icons.adb);
```

O exemplo acima produzirá este ícone:  .

A lista de ícones disponíveis pode ser encontrada em https://material.io/resources/icons. Para estes ícones estarem disponíveis no seu aplicativo é necessário habilitá-los através do pubspec.yaml da seguinte forma:

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

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

Image

Provavelmente seu aplicativo contará com imagens que serão renderizadas de diferentes formas e lugares, para fazer isso este widget poderá ajudá-lo mesmo quando esta imagem estiver armazenada em servidores online. Por isso este widget contém diversas formas de ser utilizado como segue nos exemplos abaixo. **Verifique nos comentários do código abaixo o propósito de cada método detalhado.**

```
Column(
 mainAxisAlignment: MainAxisAlignment.center,
 children: <Widget>[

   // Carrega uma imagem que foi previamente alocada
   // no pacote do aplicativo e definida no arquivo pubspec.yaml
   Image.asset('assets/images/phone.png'),

   // Busca e renderiza uma imagem diretamente de um
   // endereço acessível na internet.
   // PS.: localhost aqui tem 99% de chance de estar errado
   Image.network('https://picsum.photos/id/1025/150/150'),

   // Renderiza uma imagem do sistema de arquivos do dispositivo
```

```
    // como uma imagem nova baixada no aparelho.
    // Ps.: Este exemplo aqui gerará uma Exception.
    Image.file(File('caminho/para/o/arquivo.jpg')),

    // Carrega uma imagem através de seus dados em formato de texto puro (raw).
    Image.memory(bytes)
  ],
);
```

AssetBundle

Segundo o próprio manual, é uma coleção de recursos, ou seja, imagens, estilos, dados, arquivos de texto, ou qualquer tipo de arquivo útil a sua aplicação que se tornam acessíveis. O acesso a estes recursos é assíncrono e isso significa que o aplicativo não irá travar enquanto carrega algum desses dados, mas fará isso em segundo plano.

Você pode acessar o rootBundle que são recursos que são empacotados junto do aplicativo quando este for exportado para produção, isso deve ser apontado em seu arquivo **pubspec.yaml** como segue.

```
# The following section is specific to Flutter.
flutter:
  # To add assets to your application, add an assets section, like this:
  assets:
    - assets/images/phone.png
    - assets/somedata.txt
```

Para fazer a leitura do conteúdo de um arquivo salvo como asset utilize o objeto rootBundle que estará disponível em qualquer parte do código.

```
// O código abaixo vai capturar o conteúdo do arquivo
// asset informado
rootBundle.loadString('assets/somedata.txt').then((content) {
  print(content);
});
```

UTILITÁRIOS E INTERATIVOS

Center

A função deste simples widget é colocar seu conteúdo exatamente no centro do espaço disponível a ele. Mais útil quando se trata de widgets pequenos, mas pode funcionar muito bem até mesmo para centralizar uma lista como ListView em um cenário talvez muito improvável. Imagine uma lista que está buscando dados de uma API e enquanto isso você pode adicionar um Center(child: Text('carregando...')) para adicionar este texto exatamente no centro da tela, ou quem sabe no lugar do Text um LinearProgressIndicator ou CircularProgressIndicator.

Chip

Imagine que você tem um grupo de tags para marcar em uma foto de seu aplicativo, por exemplo, você pode querer adicionar então essas tags em formato de Chips que são pequenos conteúdos que podem facilmente ser anexados a interações, mantendo um visual muito bonito e simples.

```
Chip(
 backgroundColor: Colors.blue,
 label: Text('Maria Rita'),
 labelStyle: TextStyle(color: Colors.white),
 avatar: CircleAvatar(
   backgroundColor: Colors.black45,
   child: Text('M'),
 ),
);
```

FlatButton

Modelo mais simples de botão, a princípio um FlatButton parece apenas um texto comum, mas muda temporariamente quando há nele uma ação de toque. Assim como qualquer outro botão também acompanha o efeito Ripple Effect ou splash, que faz o clique parecer como se fosse um pingo d'água animado.

```
FlatButton(
 child: Text('Teste'),
 onPressed: () {
   print('Ação de clique');
 },
)
```

RaisedButton

Um simples e belo botão que realizará alguma ação dentro de seu aplicativo. Este botão é retangular e tem aparência de estar acima do relevo natural da tela.

```
RaisedButton(
 child: Text('Teste'),
 color: Colors.blue,
 textColor: Colors.white,
```

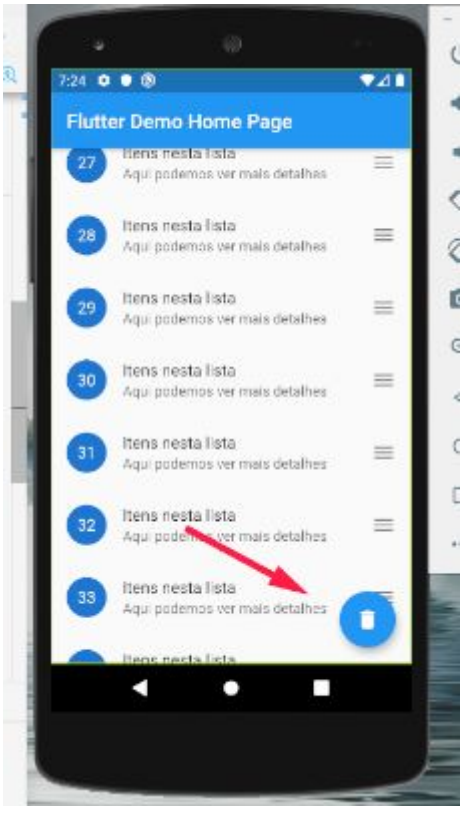
```
  onPressed: () {
    print('Ação de clique');
  },
)
```

FloatingActionButton

Botão redondo flutuante, perfeito para ser usado em listas como um botão de ação rápida. Este modelo de botão é usado no WhatsApp do Android para iniciar uma nova conversa, por exemplo.

```
33  class _MyHomePageState extends State<MyHomePage> {
34    @override
35    Widget build(BuildContext context) {
36      return Scaffold(
37        appBar: AppBar(
38          title: Text(widget.title),
39        ), // AppBar
40        body: Center(
41          child:Column(
42            mainAxisAlignment: MainAxisAlignment.center,
43            children: <Widget>[],
44          ), // Column
45        ), // Center
46        floatingActionButton: FloatingActionButton(
47          onPressed: () => print('Ação de clique'),
48          tooltip: 'Increment',
49          child: Icon(Icons.delete),
50        ), // FloatingActionButton
51      ); // Scaffold
52    }
53  }
```

Scaffold necessário



IconButton

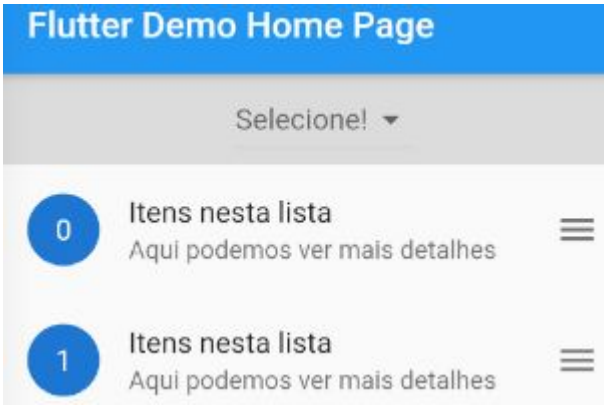
Botão em formato de ícone, este tipo de botão é também tanto simples quanto funcional e sem o evento de click não há nele um fundo ou borda. Este botão também dispara o efeito de splash ou Ripple Effect.

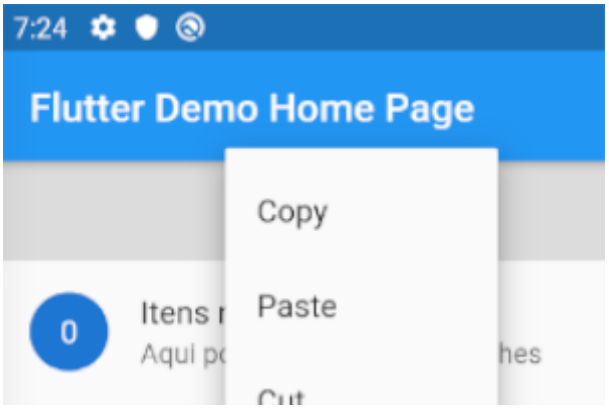
```
IconButton(
 color: Colors.black54,
 icon: Icon(Icons.menu),
 onPressed: () => print('Click'),
);
```

DropdownButton

Um botão que pode conter diversas opções a serem selecionadas, perfeito para ser usado em formulários para campos com variadas opções. Este é o equivalente no Flutter a tag <select> do HTML.

```
// NULL por padrao
String dropdownValue;

// Retorna a lista de opções dropdown
Widget _dropdownOptions() {
 return DropdownButton<String>(
   items: [
     DropdownMenuItem<String>(child: Text('Copy'), value: 'copy'),
     DropdownMenuItem<String>(child: Text('Paste'), value: 'paste'),
     DropdownMenuItem<String>(child: Text('Cut'), value: 'cut')
   ],
   onChanged: (String val) => setState(() {
     dropdownValue = val;
   }),
   hint: Text('Selecione!'),
   value: dropdownValue,
 );
}
```



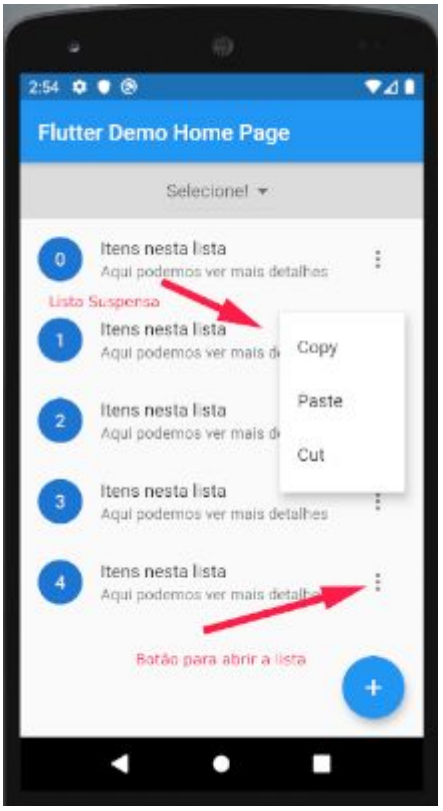


PopupMenuButton

Existe pouca diferença deste widget para o DropdownButton, este deve ser usado para promover um menu de **ações** que pode ser acessado em diferentes partes do seu aplicativo. Por exemplo, você pode adicionar do lado

direito superior de seu aplicativo um menu para o usuário acessar seu perfil ou fazer logout, neste caso este é o widget recomendado. O exemplo abaixo mostra um menu suspenso com ações pertinentes a itens de uma lista.

```
PopupMenuButton<String>(
 itemBuilder: (BuildContext context) {
   return [
     PopupMenuItem<String>(value: 'copy', child: Text('Copy')),
     PopupMenuItem<String>(value: 'paste', child: Text('Paste')),
     PopupMenuItem<String>(value: 'cut', child: Text('Cut'))
   ];
 },
 icon: Icon(Icons.more_vert),
 onSelected: (String val) => print(val),
 tooltip: 'Selecione uma ação',
);
```



Quando houver esse tipo de situação, procure trocar o método utilizado nestes exemplos como String para **enum**. Enum é um tipo de dado que não necessariamente tem um valor, mas possibilita uma comparação muito mais exata e confiável do que Strings que podem sofrer com erros de digitação, quebra de caracteres por motivos de encoding de texto, entre outros possíveis problemas. Utilizar enums é especialmente simples, basta declarar fora do escopo de classe e pronto.

```
1  import 'package:flutter/material.dart';
2
3  void main() => runApp(MyApp());
4
5  enum acao { copy, paste, cut }
6
7  class MyApp extends StatelessWidget {
8
9    @override
10   Widget build(BuildContext cor
11     return MaterialApp(
12       title: 'Flutter Demo',
13       theme: ThemeData(
14         primarySwatch: Colors.b
15       ), // ThemeData
```

Um exemplo de enum declarado fora do escopo de classe.

Veja como implementar o exemplo acima, mas dessa vez utilizando enum.

```
PopupMenuButton<acao>(
 itemBuilder: (BuildContext context) {
   return [
     PopupMenuItem<acao>(value: acao.copy, child: Text('Copy')),
     PopupMenuItem<acao>(value: acao.paste, child: Text('Paste')),
     PopupMenuItem<acao>(value: acao.cut, child: Text('Cut'))
   ];
```

```
 },
 icon: Icon(Icons.more_vert),
 onSelected: (acao val) => print(val),
 tooltip: 'Selecione uma ação',
)
```

Ps.: O efeito produzido na tela é exatamente o mesmo.

BottomSheet

Com este widget é possível abrir um bloco para adicionar conteúdo à vontade, como uma lista de botões por exemplo, ou detalhes de um usuário. É possível utilizar este widget também como modal.

Para que ele funcione é preciso adicioná-lo ao Scaffold. No exemplo abaixo nós criamos uma chave GlobalKey<ScaffoldState> de nome scaffoldKey e adicionamos ao atributo key do próprio Scaffold. Depois disso no evento onPressed do FloatingActionButton abrimos o BottomSheet. Lembrando que o FloatingActionButton é também um atributo de Scaffold.

```
// ...
floatingActionButton: FloatingActionButton(
 onPressed: () {
   scaffoldKey.currentState.showBottomSheet((BuildContext context) {
     return Container(
       color: Colors.grey[800],
       height: 200,
       width: MediaQuery.of(context).size.width,
       child: ButtonBar(
         children: <Widget>[
           RaisedButton(
             child: Text('Fechar', style: TextStyle(color: Colors.white)),
             color: Colors.blue,
             onPressed: () => Navigator.of(context).pop()
           )
         ],
       )
     );
   });
 },
 tooltip: 'Increment',
 child: Icon(Icons.add),
),
// ...
```

Hero

Você pode escolher uma imagem que terá uma animação ao entrar em uma tela específica ficando maior e mais imponente, ou seja, uma imagem que será animada e compartilhada entre duas telas. Imagine que você tenha uma lista de usuários e nesta lista hajam as fotos deles. Ao pressionar sobre um usuário desta lista o aplicativo abrirá o perfil deste usuário que contém esta foto em um tamanho maior, para animar esta interação este é o widget perfeito.

Para fazer isso, basta usar a mesma tag nos widgets Hero que quer animar. Verifique abaixo o código para ambos os widgets que serão animados entre si em duas telas diferentes configuradas através das rotas.

```
// TELA 1
// Note que a escala da imagem fará com que
// ela seja redimensionada, neste caso 5x menor
Hero(
 tag: 'imageDog',
 child: ClipRRect(
   child: Image.asset('assets/images/dog-look.jpg', scale: 5),
   borderRadius: BorderRadius.circular(15),
 ),
),

// TELA 2
// Perceba que neste caso a imagem não será
// redimensionada, pois a escala é 1 (original)
Hero(
 tag: 'imageDog',
 child: ClipRRect(
   child: Image.asset(
     'assets/images/dog-look.jpg',
     scale: 1,
     fit: BoxFit.cover
   ),
   borderRadius: BorderRadius.circular(15),
 ),
)
```

Para ver o resultado desta animação acesse: https://www.thizer.com/assets/dog-hero.mp4

RefreshIndicator

Um widget que deve ser combinado com um ListView, por exemplo, mas que também pode ser combinado com outros widgets do tipo Scrollable. Quando recebe um overscroll que vamos traduzir aqui para facilitar a comunicação de “deslize exagerado”, então este widget é apresentado na tela para indicar que a lista está sendo recarregada. Este indicador deve desaparecer após completar o carregamento.

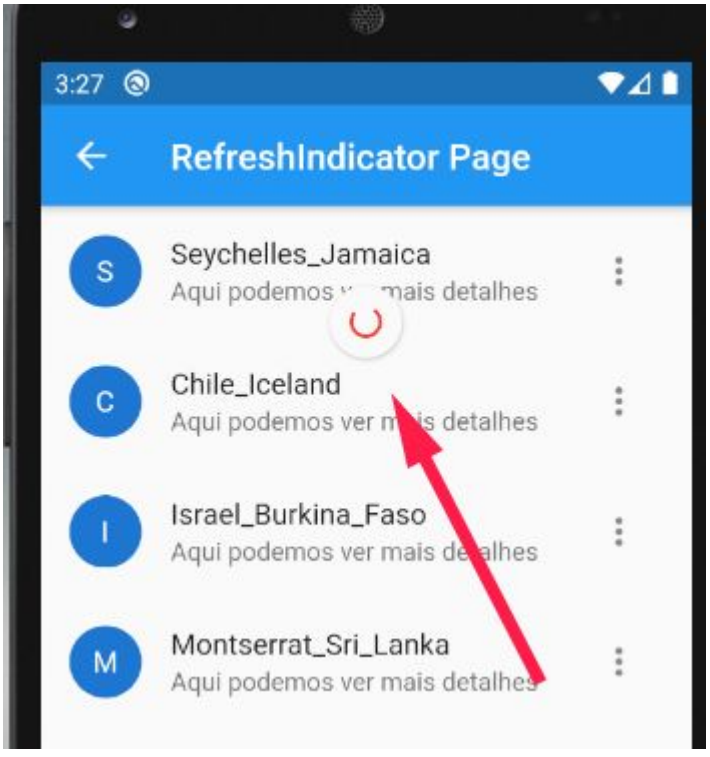
Se o componente Scrollable não tiver espaço suficiente para ter um overscroll considere adicionar a ele no atributo physics o AlwaysScrollableScrollPhysics.

O exemplo abaixo apesar de bastante extenso mostra uma página inteira que implementa esse widget, não acredito que sem um exemplo assim completo possa de fato sanar as dúvidas de quem está tentando utilizá-lo pela primeira vez.

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

class RefreshPage extends StatefulWidget {
```

```dart
 RefreshPage({Key key, this.title}) : super(key: key);

 final String title;

 @override
 _RefreshPageState createState() => _RefreshPageState();
}

class _RefreshPageState extends State<RefreshPage> {

 List<dynamic> listData = List<dynamic>();

 @override
 initState() {
   super.initState();

   // Busca a primeira listagem quando carrega a tela
   this._getList();
 }

 Future<Null> _getList() async {

   // Busca uma lista aleatoria atravez de API
   // Requer adicionar requisição ao pacote dio
   // no arquivo pubspec.yaml
   //
   // neste exemplo => dio: ^3.0.4
   Dio dio = Dio();
   Response resp = await dio.get('http://names.drycodes.com/10?nameOptions=countries');

   // Força o recarregamento da lista
   setState(() {
     this.listData = resp.data;
   });
 }

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text(widget.title)),
     body: Column(
       children: <Widget>[
         Expanded(
           child: RefreshIndicator(
             color: Colors.red,
             onRefresh: () => _getList(),
```

```
            // Note que o ListView é filho do RefreshIndicator
            child: ListView.builder(
              physics: const AlwaysScrollableScrollPhysics(),
              itemCount: this.listData.length,
              itemBuilder: (BuildContext context, int i) {
                return ListTile(
                  title: Text(this.listData[i]),
                  subtitle: Text('Aqui podemos ver mais detalhes'),
                  leading: CircleAvatar(
                    child: Text(this.listData[i][0]),
                  ),
                  trailing: Icon(Icons.more_vert),
                );
              },
            ),
          ),
        ),
      ],
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => Navigator.pushNamed(context, '/'),
      tooltip: 'Increment',
      child: Icon(Icons.home),
    ),
  );
 }
}
```

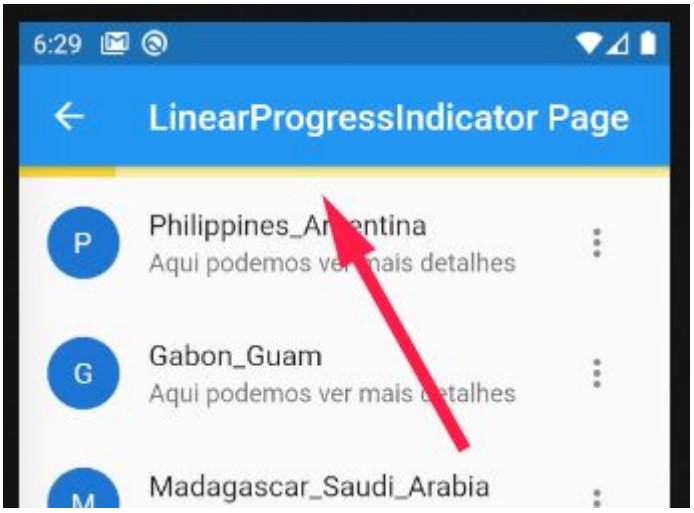


LinearProgressIndicator

Este widget vai indicar que seu aplicativo está executando alguma ação em segundo plano de modo super discreto e bonito, mostrando uma pequena barra de cor personalizável que desliza enquanto seu aplicativo realiza as ações necessárias para chegar ao próximo passo. Para usar este widget é super simples, pois basta posicioná-lo onde você quer que ele apareça, normalmente ele é colocado como o primeiro elemento da tela, logo abaixo do AppBar como no exemplo abaixo.

É possível também determinar em nível de porcentagem o tamanho da barra através do atributo value, de 0.0 até 1, ou seja, para uma barra do tamanho de 75% value é igual a 0.75.

```
LinearProgressIndicator(
 backgroundColor: Colors.yellow[200],
 valueColor: AlwaysStoppedAnimation<Color>(Colors.yellow[600]),
)
```

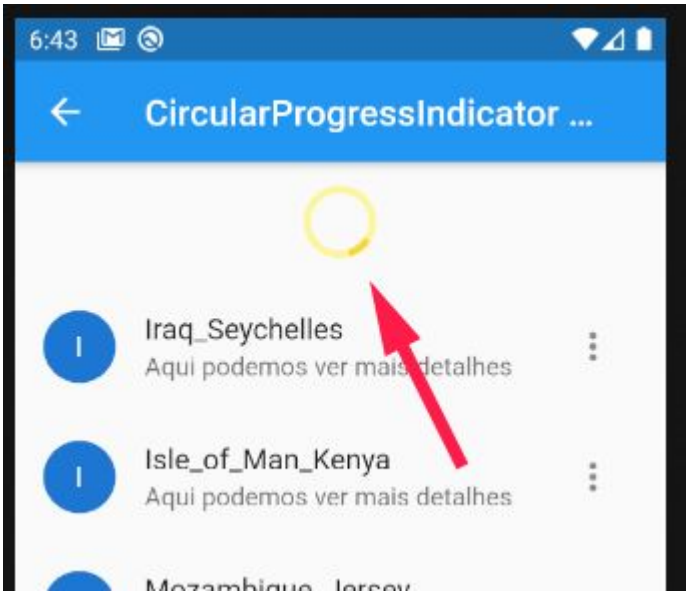


CircularProgressIndicator

Este widget vai indicar que seu aplicativo está executando alguma ação em segundo plano de modo super discreto e bonito, mostrando um círculo animado de cor personalizável enquanto seu aplicativo realiza as ações necessárias para chegar ao próximo passo. Para usar este widget, basta posicioná-lo onde você quer que ele apareça, exatamente igual ao seu parceiro LinearProgressIndicator.

É possível também determinar em nível de porcentagem o quanto do círculo está completo através do atributo value, de 0.0 até 1, ou seja, para um círculo 26% completo o value é igual a 0.26.

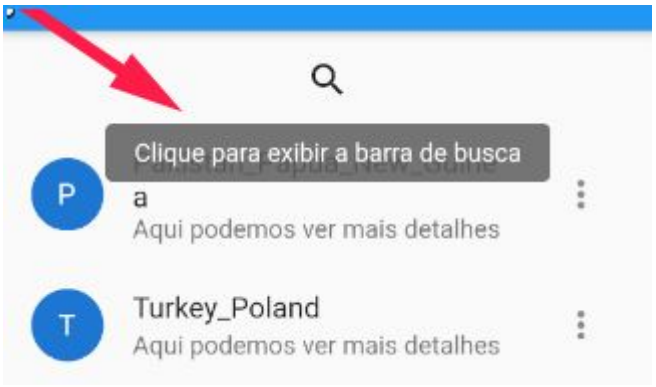
```
CircularProgressIndicator(
 backgroundColor: Colors.yellow[200],
 valueColor: AlwaysStoppedAnimation<Color>(Colors.yellow[600]),
),
```



Tooltip

Utilize este simples widget para mostrar aos seus usuários um texto informativo sobre determinado ponto do layout de seu aplicativo. Ele irá ajudar a explicar melhor a usabilidade que foi planejada para a tela. Para ele aparecer o usuário precisará manter pressionado o toque sobre seu widget filho.

```
Tooltip(
 message: 'Clique para exibir a barra de busca',
 child: Icon(Icons.search),
),
```



WIDGETS DE ESTRUTURA

Apresentamos aqui os widgets dos quais sua aplicação não pode lançar mão quando se trata de Material Design, pois eles preparam o aplicativo para receber os widgets que de fato irão montar o tema visual.

São eles:

MaterialApp

Define que seu aplicativo vai usar as diretrizes gerais de montagem de visual baseado no Material Design. Este widget, além de outras coisas, vai configurar o principal **Navigator** para trabalhar com as rotas, quero dizer que é aqui que todas as rotas (páginas) do seu aplicativo devem ser definidas e informadas através do parâmetro routes. Para funcionar pelo menos a rota "/" deve ser informada e isso pode ser feito diretamente através do parâmetro home.

Uma grande parte das configurações do seu aplicativo virão a partir deste widget, pois ele configura diversos padrões e estes são utilizados por todos os widgets filhos, ou seja, seu aplicativo inteiro. Mantenha em mente que

se você quer por exemplo aumentar o tamanho da fonte para todos os títulos ao longo do aplicativo, então provavelmente aqui seja o melhor lugar para fazer isso, pois assim vai ter que fazê-lo apenas uma vez.

A maneira mais comum de usar este widget

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: const Text('MaterialApp Theme'),
   );
 }
}
```

O código acima demonstra o modo mais reduzido possível de construir um aplicativo utilizando o widget MaterialApp.

Parâmetros mais utilizados

✗ **home:** Widget
A rota (página) padrão do aplicativo que também é conhecida como '/'.

✗ **routes:** Map<String, WidgetBuilder>
Uma lista de rotas (páginas) que ficarão disponíveis para serem chamadas através do objeto Navigation.pushNamed.
Quando esta propriedade é fornecida, se torna uma redundância informar também a propriedade **home**. Apesar de ser um erro, ou melhor um engano, isso não irá disparar avisos ou exceptions.
Quando a rota informada não é encontrada a função de callback informada em **onRouteGenerate** é chamada para construir a página.

✗ **initialRoute:** String
Substitui a rota inicial do aplicativo, por padrão a rota inicial é '/' e este parâmetro modifica esta propriedade. Evite usar aqui rotas compostas como por exemplo '/a/b/c'. É permitido isso, mas vai gerar 3 rotas '/a', '/a/b' e '/a/b/c' e caso haja erro em qualquer parte do processo de criar essas 3 rotas este parâmetro será ignorado e o aplicativo voltará a usar a rota padrão '/'.

✗ **locale:** Locale
A localidade inicial do aplicativo. Isso modificará, se configurado corretamente, o idioma em que o aplicativo será apresentado. Ex.: const Locale('pt', 'BR')

✗ **title:** String
Um título que o dispositivo utiliza para identificar o aplicativo.

✗ **color:** Color
A cor primária que será utilizada no aplicativo.

✗ **theme:** ThemeData
Propriedades visuais padrão como cores, fontes de texto e formas que serão utilizados dentro do aplicativo.
Pode aqui aproveitar para combinar também o **darkTheme** que irá fornecer um tema escuro quando for solicitado pelo dispositivo.
O valor padrão para esta propriedade é ThemeData.light().

✗ **darkTheme:** ThemeData
Os atributos de cores de tema quando o 'dark mode' for requisitado pelo sistema. Algumas plataformas fornecem este recurso. A propriedade **themeMode** é que irá controlar isso.

✗ **themeMode:** ThemeMode
Determina qual tema visual será utilizado pelo sistema *ligth mode* ou *dark mode*, caso ambos, *theme* e *darkTheme*, forem informados.
O valor padrão para esta propriedade é *ThemeMode.system* o que significa que quem vai determinar o tema visual entre *light* e *dark* é o próprio dispositivo.

✗ **debugShowCheckedModeBanner:** bool
Quando habilitado mostrará uma pequena faixa vermelha do lado superior direito com a palavra 'DEBUG' indicando que o aplicativo ainda está em modo de desenvolvimento. Por padrão vem habilitado.

✗ **checkerboardOffscreenLayers:** bool
Esta opção se refere à performance do seu aplicativo, quando há algo como opacidade em algum elemento pode haver uma queda consideravel de performance no aplicativo, se este parâmetro estiver habilitado, será mostrado no objeto com opacidade uma marcação visual com borda e conteúdo semelhante a um tabuleiro de xadrez, isso vai te ajudar a identificar pontos onde possa haver elemento invisível por acidente, por exemplo.

fonte: https://flutter.dev/docs/testing/ui-performance#checking-for-offscreen-layers

✗ **checkerboardRasterCacheImages:** bool
Também se refere à performance, quando habilitado rastreia o aplicativo a fim de encontrar imagens que foram colocadas em cache. Colocar imagens em cache custa bastante ao sistema e por isso essa prática deve ser utilizada com sabedoria e quando for absolutamente necessário.

fonte: https://flutter.dev/docs/testing/ui-performance#checking-for-non-cached-images

✗ **showPerformanceOverlay:** bool
Um gráfico de representação visual da performance do seu aplicativo em tempo real.

✗ **showSemanticsDebugger:** bool
Habilita um painel que mostra as informações de acessibilidade reportadas pelo Flutter.

Demais definições e parâmetros:
https://api.flutter.dev/flutter/material/MaterialApp-class.html#instance-properties

Scaffold

Adiciona a estrutura básica de página com o visual do Material Design. Basicamente para fazer um aplicativo totalmente dentro destes padrões, basta iniciar todas as páginas do seu aplicativo implementando um Scaffold e

suas opções. Isso tornará o aplicativo super simples sem deixar de aproveitar as vantagens visuais e dinâmicas do lindo Material Design.

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: Scaffold( // ← comeca aqui
       appBar: AppBar(
         title: const Text('Sample Code'),
       ),
       body: Center(child: Text('Body Content')),
       floatingActionButton: FloatingActionButton(
         onPressed: () => print(‘hello’),
         tooltip: 'Increment Counter',
         child: const Icon(Icons.add),
       ),
     )
   );
 }
}
```

O exemplo acima irá criar de modo super reduzido e não funcional a tela de exemplo inicial do Flutter quando você cria um novo aplicativo Flutter pelo VS Code por exemplo.

Parâmetros mais utilizados

✗ **appBar:** PreferredSizeWidget
Normalmente alimentado com o widget AppBar que irá criar uma barra no topo do aplicativo.

✗ **body:** Widget
Conteúdo que será renderizado efetivamente na página.

✗ **floatingActionButton:** Widget
Um botão flutuante redondo que irá ficar visivel o tempo todo na tela, como método de ação rápida, normalmente utilizado em listas.

✗ **drawer:** Widget
Um painel lateral com menu de opções que seu aplicativo oferece ao usuário.

✗ **bottomNavigationBar:** Widget
Uma barra de menu na parte inferior do aplicativo com opções de navegação disponíveis ao usuário.

✗ **backgroundColor:** Color
Uma cor da paleta do Material para ser usada como plano de fundo permanente no aplicativo.

✗ **resizeToAvoidBottomInset:** bool
Quando true irá redimensionar a altura do Scaffold para não esconder o conteúdo caso seja aberto o teclado do dispositivo.

Demais definições e parâmetros:
https://api.flutter.dev/flutter/material/Scaffold-class.html#instance-properties

MARCO ANTONIO BRAGHIM

Nascido em uma pequena cidade do interior do Paraná chamada Campo Mourão em 1987, Marco A. Braghim trabalha com tecnologia e desenvolvimento de software desde 2007. Atuou na maior parte do tempo até agora com tecnologias Web e API's de dados.

Sempre apaixonado por tecnologia seu envolvimento se torna inevitável, mesmo ás vezes essa relação sendo entre tapas e beijos como é tão comum na profissão.

No início de 2019 começou a se descobrir como instrutor de programação mobile através da internet e essa relação se tornou apaixonante. Hoje dedica grande parte de seu tempo em benefício da comunidade Flutter para pessoas que falam português e os conteúdos gerados tem impacto em vários países com apoio da comunidade.

NOS ACOMPANHE NAS REDES SOCIAIS

https://www.reddit.com/user/marcobraghim
https://www.facebook.com/marco.braghim

Thizer Aplicativos

https://www.thizer.com/
https://www.facebook.com/thizer.apps/
https://www.facebook.com/groups/flutterpt/
https://www.youtube.com/c/ThizerAplicativos
https://twitter.com/ThizerOficial
https://www.instagram.com/thizerdev/
https://www.reddit.com/r/FlutterBrasil/
https://www.reddit.com/r/FlutterEmPortugues/