



# Flutter





Flutter

Estrutura do Flutter

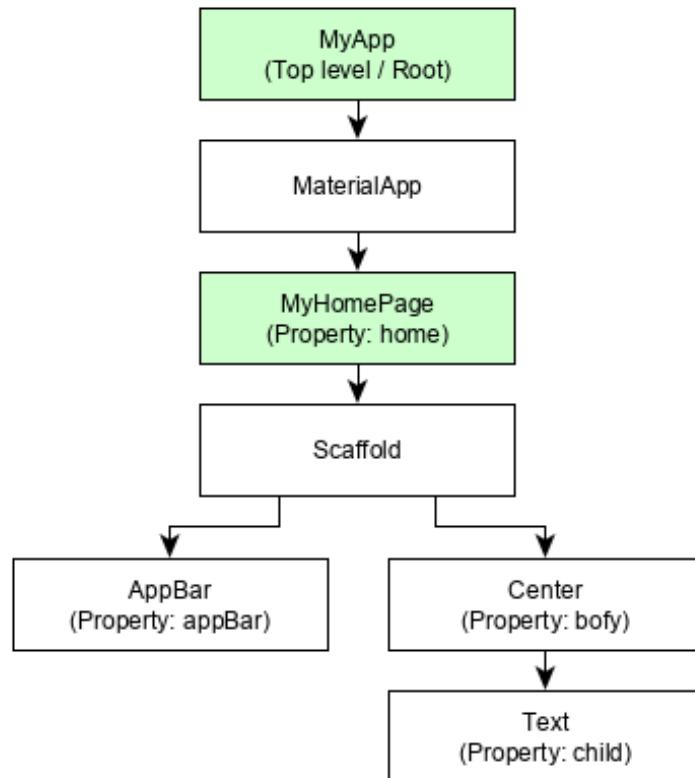
Widgets

# Widgets

---

- O conceito central do framework **Flutter** é:
  - Tudo é um **widget**.
  - **Widgets**:
    - são basicamente componentes de interface do usuário usados para criar a interface do usuário do aplicativo.
- No **Flutter**, o próprio aplicativo é um **widget**.
  - O aplicativo é o widget de nível superior e sua interface do usuário é construída usando um ou mais filhos (widgets), que por sua vez são construídos usando seus filhos widgets.
  - Esse recurso de composição nos ajuda a criar uma interface de usuário de qualquer complexidade.

# Estrutura básica de um APP

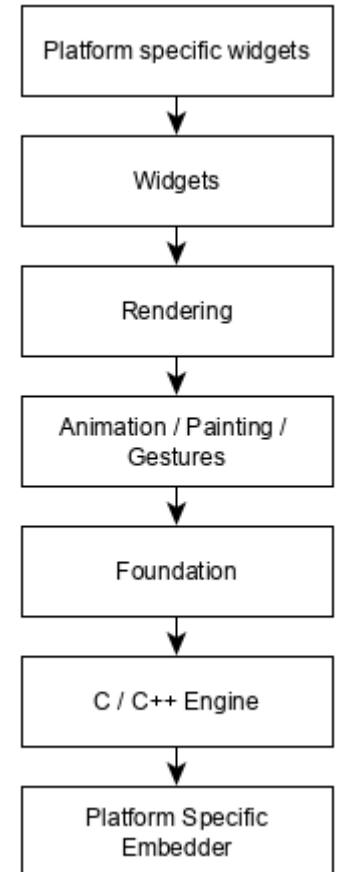


# Camadas

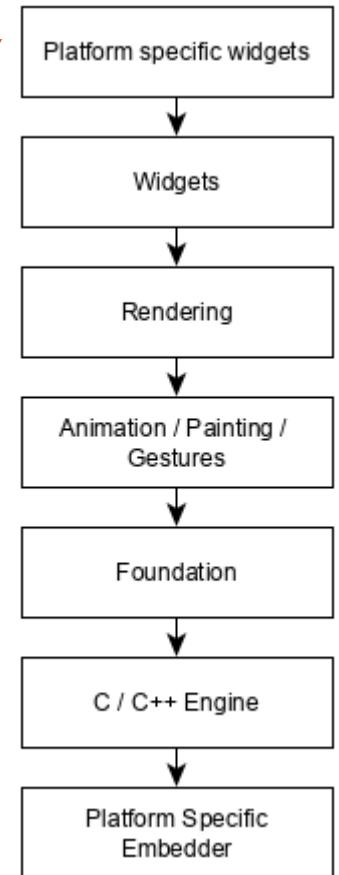
---

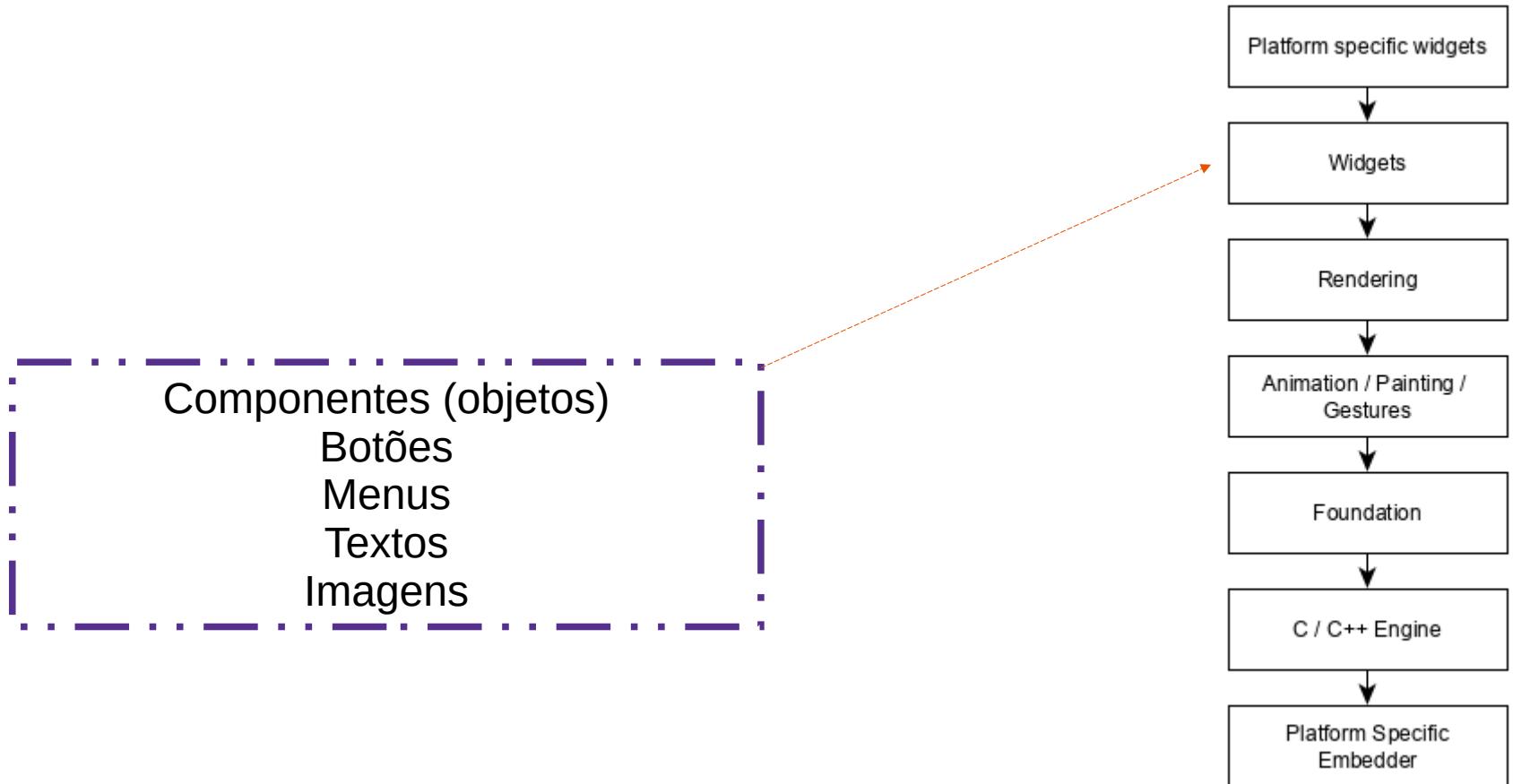
- O conceito mais importante do framework **Flutter** é que ele é agrupado em múltiplas categorias em termos de complexidade e claramente organizado em camadas de complexidade decrescente.
- Uma camada é construída usando a camada imediatamente seguinte.
  - A camada mais alta é específica para **widgets** do **Android** e **iOS**.
  - A próxima camada contém todos os **widgets** nativos do **Flutter**.
  - A camada seguinte é a camada de **renderização**, que é um componente de renderização de baixo nível e renderiza tudo no aplicativo **Flutter**.
- As camadas descem até o código principal específico da plataforma.

- Aqui temos uma visão geral de uma camada no **Flutter**

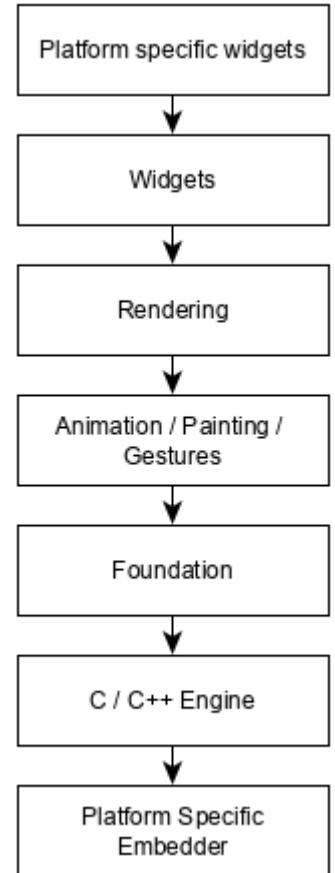


Material Design ou Cupertino

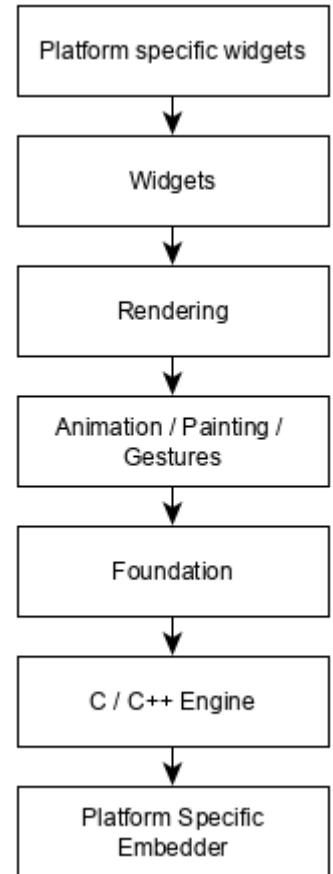




- Na arquitetura do Flutter, a camada de renderização é uma parte essencial do framework, responsável pelo layout e pela renderização da interface do usuário.
- Ela converte a hierarquia abstrata de widgets em objetos concretos e renderizáveis que o mecanismo subjacente pode exibir como pixels na tela.

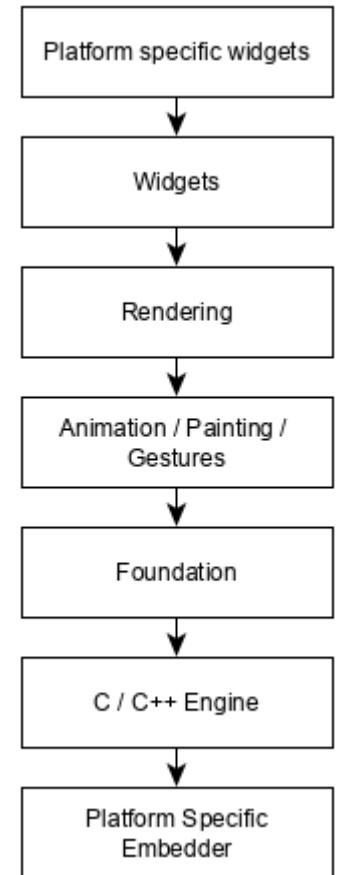


- A "camada fundamental" no Flutter refere-se às classes utilitárias de nível mais baixo dentro da estrutura geral do Flutter.
- Essa camada fornece **blocos de construção essenciais**, como animações, gestos e utilitários básicos escritos na linguagem Dart, sobre os quais todas as camadas superiores, incluindo widgets e renderização, são construídas.

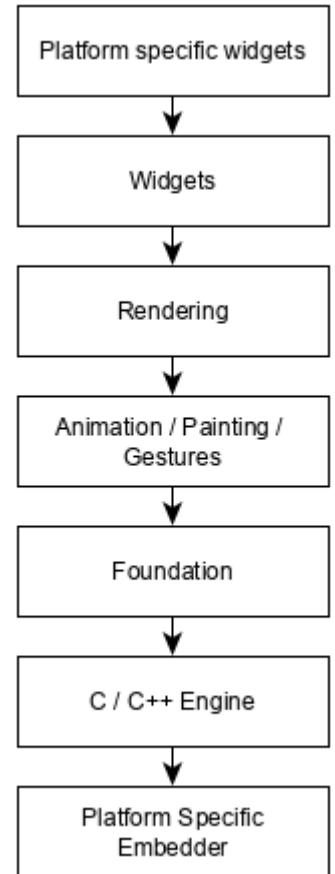


- O núcleo do ambiente de execução
- que lida com gráficos (através dos mecanismos Skia ou **Impeller** – gráficos 2D), layout de texto, entrada/saída de arquivos/rede e o ambiente de execução Dart.

O **Impeller** é o mecanismo de renderização moderno e de alto desempenho que elimina as lentidões na compilação de shaders, com foco específico no Metal no iOS e no Vulkan no Android.



- A "camada incorporada específica da plataforma" no Flutter é chamada de Platform Embedder. Essa camada de baixo nível, em código nativo, integra o mecanismo do Flutter ao sistema operacional hospedeiro, gerenciando tarefas cruciais do sistema, como renderização de superfícies, entrada e acessibilidade.



# Conceitos de Estados

---

- Em um APP alguns objetos precisam de controle de estado.
  - Ou seja, em alguns **widgets** será necessário controlar o seu estado (quando houver mudança) haverá uma renderização automática sempre que seu estado interno for alterado.
  - Os **widgets** do Flutter suportam a manutenção de estado através de um widget especial, o **StatefulWidget**.
  - Para suportar a manutenção de estado, o **widget** precisa ser derivado de  **StatefulWidget**
- A renderização é otimizada encontrando a diferença entre a interface do usuário do widget antigo e a nova, e renderizando apenas as alterações necessárias.

# Gestos

---

- Os **widgets** do **Flutter** suportam interação por meio de um **widget** especial, o **GestureDetector**.
- O **GestureDetector** é um widget invisível com a capacidade de capturar interações do usuário, como:
  - Toques, arrastos, etc., em seus widgets filhos.
  - Muitos widgets nativos do **Flutter** suportam interação por meio do uso do **GestureDetector**.
  - Também podemos incorporar recursos interativos em widgets existentes, compondo-os com o widget **GestureDetector**.

# Resumo da Arquitetura do Flutter

---

- No **Flutter**, tudo é um **widget** e um **widget** complexo é composto de **widgets** já existentes.
- Recursos interativos podem ser incorporados sempre que necessário usando o **widget GestureDetector**.
- O estado de um **widget** pode ser mantido sempre que necessário usando o **widget StatefulWidget**.
- O **Flutter** oferece um design em camadas, de forma que qualquer camada pode ser programada dependendo da complexidade da tarefa.



Flutter

# Usando Widgets Comuns



Flutter

#0

MaterialApp

# Classe MaterialApp

---

- **MaterialApp** é uma **classe** ou **widget** predefinido no **Flutter**.
  - É um dos principais componentes ou central de um aplicativo **Flutter**.
  - O widget **MaterialApp** fornece uma camada protetora para outros widgets do **Material Design**.

# Classe MaterialApp

---

- O widget **MaterialApp** é um widget de alto nível no Flutter que define a estrutura básica de um aplicativo **Material Design**.
- Ele funciona como um invólucro para a interface do usuário do seu aplicativo, fornecendo uma base para navegação, temas e outras configurações gerais do aplicativo.

# Classe MaterialApp

---

- Ao usar o **MaterialApp**, você tem acesso a componentes do Material Design como:
  - Scaffold, AppBar, FloatingActionButton e muito mais,
  - Garantindo que seu aplicativo siga uma linguagem de design consistente.
- Pense no **MaterialApp** como o ponto de entrada do seu aplicativo, que encapsula tudo, desde o título até as rotas de navegação e o estilo visual.
- Normalmente, ele é o widget raiz do seu aplicativo Flutter, e a maioria dos outros widgets são aninhados dentro dele.

# MaterialApp: propriedades

---

- O widget **MaterialApp** oferece uma ampla gama de propriedades para personalizar o comportamento e a aparência do seu aplicativo.
  - Aqui estão as mais importantes:
    - **title**: Define o nome do aplicativo, exibido no gerenciador de tarefas ou no seletor de aplicativos do dispositivo.
    - **theme**: Especifica o tema visual do aplicativo, incluindo cores, tipografia e estilos de widgets.
    - **home**: O widget padrão exibido quando o aplicativo é iniciado, geralmente um Scaffold.
    - **routes**: Um mapa de rotas nomeadas para navegação entre telas.
    - **initialRoute**: A primeira rota a ser exibida se você estiver usando rotas nomeadas em vez de um widget de página inicial.
    - **navigatorKey**: Uma chave para controlar o navegador do aplicativo programaticamente.
    - **locale** e **supportedLocales**: Habilitam a localização para aplicativos multilíngues.
  - Essas propriedades tornam o MaterialApp uma ferramenta poderosa para configurar a estrutura e a aparência do seu aplicativo.

# MaterialApp: por que usá-lo?

---

- O widget **MaterialApp** simplifica o desenvolvimento de aplicativos ao:
  - **Fornecer estrutura de Material Design:**  
Garante que seu aplicativo siga os princípios do Material Design, tornando-o intuitivo para os usuários.
  - **Habilitar navegação:**  
Oferece suporte integrado para rotas e pilhas de navegação.
  - **Suporte a temas:**  
Permite um estilo consistente em todos os widgets.
  - **Facilitar a localização:**  
Simplifica o suporte a vários idiomas e regiões.
  - **Simplificar a configuração:**  
Centraliza as configurações de todo o aplicativo em um só lugar.

Sem o **MaterialApp**, você precisaria implementar manualmente muitos desses recursos, aumentando a complexidade e o tempo de desenvolvimento.

# MaterialApp: exemplificando

```
@override  
Widget build(BuildContext context) {  
  return MaterialApp(  
    title: 'Themed App',  
    theme: ThemeData(  
      primarySwatch: Colors.teal,  
      scaffoldBackgroundColor: Colors.white,  
      appBarTheme: const AppBarTheme(  
        backgroundColor: Colors.teal,  
        foregroundColor: Colors.white,  
      ),  
    ),  
    darkTheme: ThemeData(  
      primarySwatch: Colors.teal,  
      scaffoldBackgroundColor: Colors.grey[900],  
      appBarTheme: const AppBarTheme(  
        backgroundColor: Colors.tealAccent,  
        foregroundColor: Colors.black,  
      ),  
    ),  
    themeMode: ThemeMode.system,  
    home: const MyHomePage(title: 'Flutter Demo Home Page'),  
  );  
}
```

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    title: 'Themed App',  
    theme: ThemeData(  
      primarySwatch: Colors.teal,  
      scaffoldBackgroundColor: Colors.white,  
      appBarTheme: const AppBarTheme(  
        backgroundColor: Colors.teal,  
        foregroundColor: Colors.white,  
      ),  
    ),  
    darkTheme: ThemeData(  
      primarySwatch: Colors.teal,  
      scaffoldBackgroundColor: Colors.grey[900],  
      appBarTheme: const AppBarTheme(  
        backgroundColor: Colors.tealAccent,  
        foregroundColor: Colors.black,  
      ),  
    ),  
    themeMode: ThemeMode.system,  
    initialRoute: '/home',  
    routes: {  
      '/home': (context) => MyHomePage(title: 'Flutter Demo Home Page'),  
      '/settings': (context) => const SettingsScreen(),  
    },  
  );  
}
```

# Material Design

---

- O Material Design

- É um sistema adaptável de diretrizes, componentes e ferramentas para a criação de interfaces de usuário consistentes, acessíveis e visualmente atraentes em diversas plataformas (**dispositivos móveis, web e desktop**).



Flutter

#1

Scaffold

Pesquisar no Bolicho.shop

# Scaffold

O widget **Scaffold** implementa a estrutura para um layout visual básico do **Material Design**, permitindo que se adicione facilmente vários **widgets**, como:

**AppBar, Body, BottomAppBar, FloatingActionButton, Drawer, SnackBar, BottomSheet.**

Dentre outros **widgets**.



Castanhas



Chocolates



Guloseimas



Zerc

Nossos Produtos

Ver Todos



15%

Balas logurte  
Vai Curinthia!!!  
Boavistense

R\$ 24.24



15%

Pirulito Cherry  
Pop  
Boavistense

R\$ 19.76



Home



Carrinho



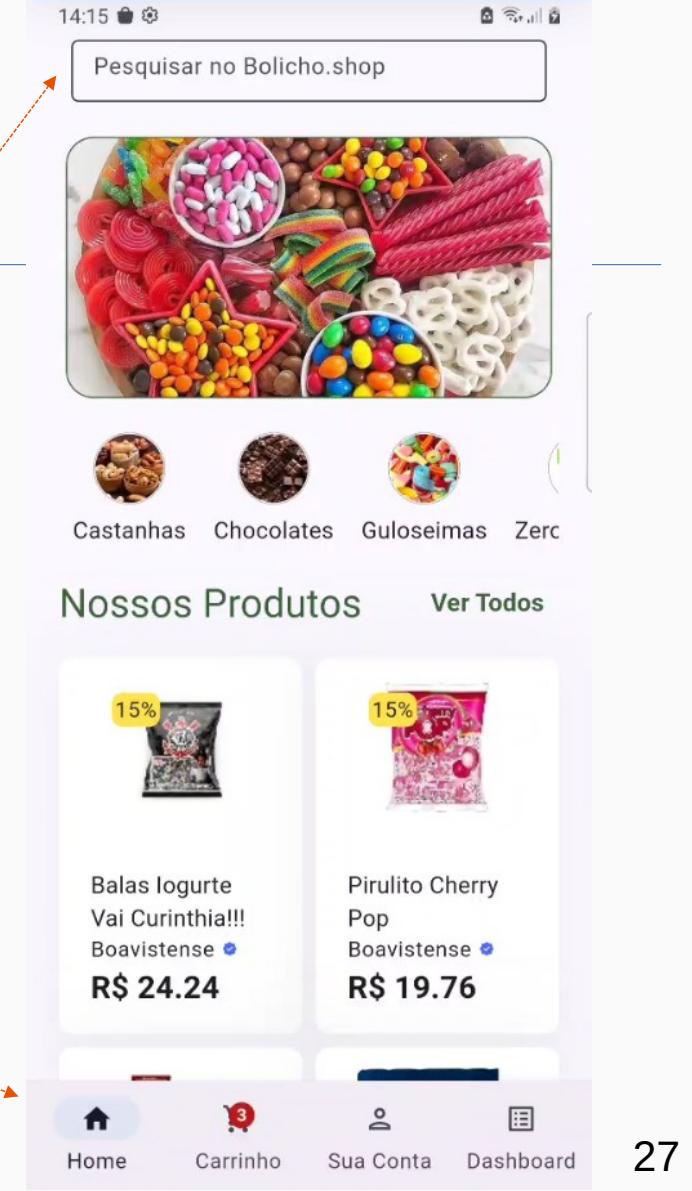
Sua Conta



Dashboard

# A estrutura do widget **Scaffold**

- **AppBar**
- **Body**
- **NavigationBar**



```

4  class HomePage extends StatelessWidget {
5    HomePage({super.key});
6    List<String> images = ["image1.jpg", "image2.jpg", "image3.png"];
7    @override
8    Widget build(BuildContext context) {
9      return Scaffold(
10        appBar: AppBar( // AppBar ...
11        body: SafeArea( // SafeArea ...
12        ); // Scaffold
13    }
14 }
```

14:15 ☀️

Pesquisar no Bolicho.shop



Castanhas



Chocolates



Guloseimas



Zerc

## Nossos Produtos

[Ver Todos](#)



Balas logurte

Vai Curinthia!!!

Boavistense 🇧🇷

**R\$ 24.24**



15%

Pirulito Cherry

Pop

Boavistense 🇧🇷

**R\$ 19.76**



Home



Carrinho



Sua Conta



Dashboard



# Flutter

#2

Utilizando APPBAR

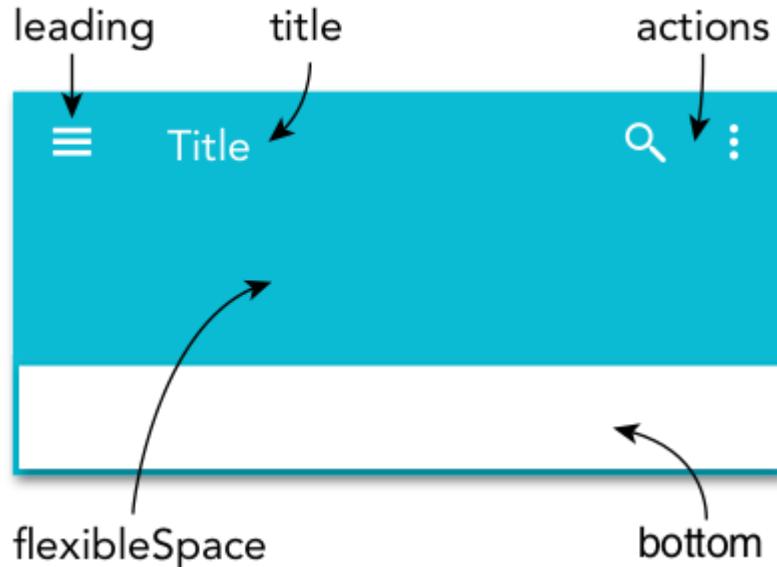
# AppBar: Finalidade e Recursos

---

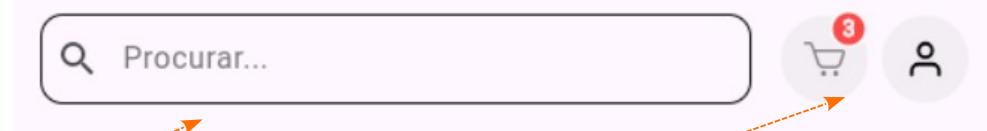
- A **AppBar** desempenha várias funções essenciais em um aplicativo **Flutter**:
  - **Marca e Título**: Exibe com destaque o título da tela atual ou o logotipo do aplicativo.
  - **Navegação**: Geralmente inclui controles de navegação, como um botão "voltar" para retornar à tela anterior ou um ícone de menu para abrir um menu lateral.
  - **Ações**: Oferece um local para ações comuns por meio de ícones (por exemplo, pesquisa, configurações, notificações) ou um **PopupMenuButton** para opções menos frequentes.

# AppBar: Finalidade e Recursos

- Características do AppBar no Scaffold

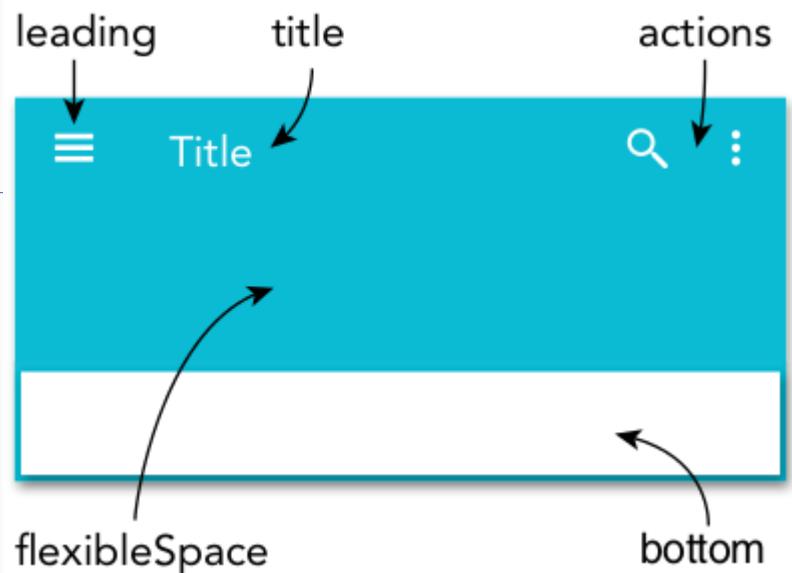


# AppBar: Exemplo de Uso



```
class HomePage extends StatelessWidget {  
  const HomePage({super.key});  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        toolbarHeight: 80,  
        title: Form(  
          child: TextFormField(  
            onChanged: (value) {},  
            decoration: InputDecoration( // InputDecoration ...  
            ), // TextFormField  
          ), // Form  
        actions: [  
          IconButtonWithCounter(numOfitem: 3, svgSrc: cartIcon, press: () {}),  
          const SizedBox(width: 8),  
          IconButtonWithCounter(svgSrc: personIcon, press: () {}),  
          const SizedBox(width: 20),  
        ],  
      ), // AppBar  
      body: SafeArea(
```

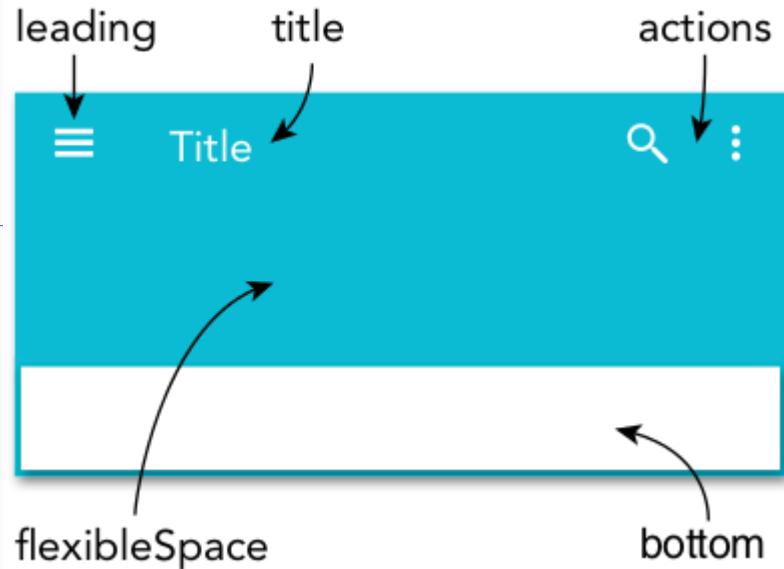
Adicione ao AppBar um IconButton principal. Se você substituir a propriedade **leading** (principal), geralmente é um IconButton ou BackButton.



IconButton or BackButton.

```
leading: IconButton(  
    icon: Icon(Icons.menu),  
    onPressed: () { },  
),
```

A propriedade actions recebe uma lista de widgets; adicionaremos dois widgets **IconButton**.



```
actions: <Widget>[  
    IconButton(  
        icon: Icon(Icons.search),  
        onPressed: () {},  
    ),  
    IconButton(  
        icon: Icon(Icons.more_vert),  
        onPressed: () {},  
    ),  
,  
,
```



Flutter

#3

# SafeArea

## Sem SafeArea



Os celulares têm um recorte parcial que obscurece a tela geralmente localizado na parte superior do dispositivo.

O widget **SafeArea** adiciona automaticamente preenchimento suficiente ao widget filho para evitar invasões do sistema operacional.

## Com SafeArea



```
body: Padding(  
    padding: EdgeInsets.all(16.0),  
    child: SafeArea(  
        child: SingleChildScrollView(  
            child: Column(  
                children: <Widget>[  
  
                ],  
            ),  
        ),  
    ),  
),  
,
```



Flutter

#4

# Container

# Container

---

- Um **Container** no **Flutter** é um widget de conveniência que permite personalizar o layout, o estilo, o tamanho, o preenchimento, a margem e o alinhamento em um só lugar.
- Ele envolve um único widget filho, mas pode conter vários filhos quando usado dentro de widgets como **Row**, **Column** ou **Stack**.

O widget  
**Container**  
ajudará-lo a  
compor, decorar  
e posicionar.

Se enveloparmos  
um widget em um  
**Container**,  
podermos  
personalizá-lo com  
cores, tamanho,  
margens, formas,  
dentre outros...

# Exemplo

```
Container(  
    child: Text('IFMT'),  
    decoration: BoxDecoration(  
        shape: BoxShape.circle,  
        color: Colors.orange,  
    ),  
    margin: EdgeInsets.all(25.0),  
    padding: EdgeInsets.all(40.0),  
    alignment: Alignment.center,  
    width: 200,  
    height: 100,  
);
```

# Container: Exemplo

```
258 Widget build(BuildContext context) {  
259     return Container(  
260         width: double.infinity,  
261         margin: const EdgeInsets.all(20),  
262         padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 16),  
263         decoration: BoxDecoration(  
264             color: const Color(0xFF4A3298),  
265             borderRadius: BorderRadius.circular(20),  
266         ), // BoxDecoration  
267         child: const Text.rich(  
268             TextSpan(  
269                 style: TextStyle(color: Colors.white),  
270                 children: [  
271                     TextSpan(text: "Descontão\n"),  
272                     TextSpan(  
273                         text: "Cashback de 20%",  
274                         style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),  
275                     ), // TextSpan  
276                 ],  
277             ), // TextSpan  
278         ), // Text.rich  
279     ); // Container  
280 }
```





Flutter

#5

Column

## Column



Um widget **Column** exibe seus filhos verticalmente. Contém uma propriedade **children** contendo um array de `List<Widget>`. **Children** se alinham verticalmente sem ocupar toda a altura da tela.

Cada widget filho pode ser incorporado em um widget **Expanded** para preencher o espaço disponível. Você pode usar **CrossAxisAlignment**, **MainAxisAlignment** e **MainAxisSize** para alinhar e dimensionar quanto espaço é ocupado no eixo principal.

```
Column(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  mainAxisSize: MainAxisSize.max,  
  children: <Widget>[  
    Text('Column 1'),  
    Divider(),  
    Text('Column 2'),  
    Divider(),  
    Text('Column 3'),  
  ],  
)
```

# Coluna: exemplo

```
1 import 'package:flutter/material.dart';
2 import 'package:flutter_svg/flutter_svg.dart';
3
4 class HomePage extends StatelessWidget {
5   HomePage({super.key});
6   List<String> images = ["image1.jpg", "image2.jpg", "image3.png"];
7   @override
8   Widget build(BuildContext context) {
9     return Scaffold(
10       appBar: AppBar( // AppBar ...
11         body: SafeArea(
12           child: SingleChildScrollView(
13             // padding: EdgeInsets.symmetric(vertical: 16),
14             child: Column(
15               children: [
16                 // HomeHeader(),
17                 ConstrainedBox( // ConstrainedBox ...
18                   child: BannerDesc(),
19                   ProdutosPop(),
20                   SizedBox(height: 20),
21                   ProdutosRecentes(),
22                 ],
23               ],
24             ), // Column
25           ), // SingleChildScrollView
26         ), // SafeArea
27       ); // Scaffold
28     }
29 }
```



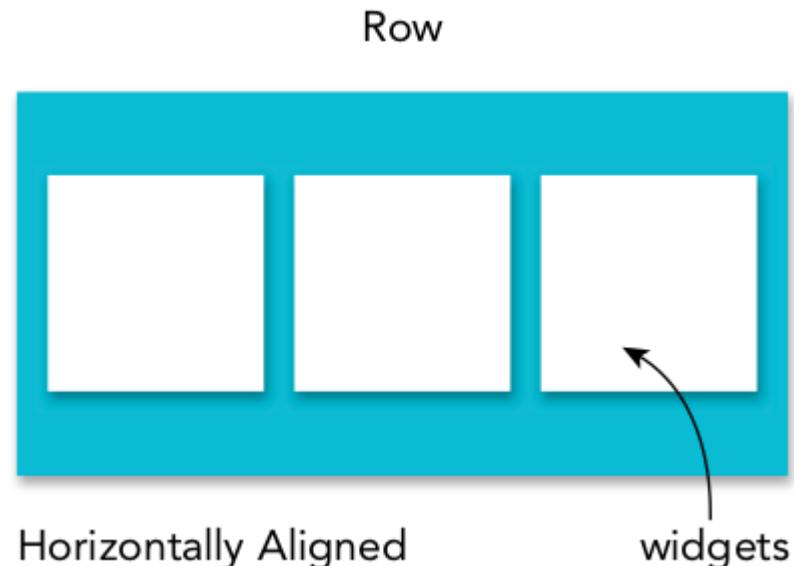


Flutter

#6

Row

Um widget **Row** exibe seus filhos horizontalmente. Contém, também, uma propriedade `children` contendo um array de **List<Widget>**. As mesmas propriedades que a Coluna contém são aplicadas ao widget Linha, exceto que o alinhamento é horizontal, não vertical.



```
Row(  
    mainAxisAlignment: MainAxisAlignment.start,  
    mainAxisSize: MainAxisSize.max,  
    spaceEvenly,  
    children: <Widget>[  
        Row(  
            children: <Widget>[  
                Text('Row 1'),  
                Padding(padding: EdgeInsets.all(16.0),),  
                Text('Row 2'),  
                Padding(padding: EdgeInsets.all(16.0),),  
                Text('Row 3'),  
            ],  
        ),  
    ],  
,
```



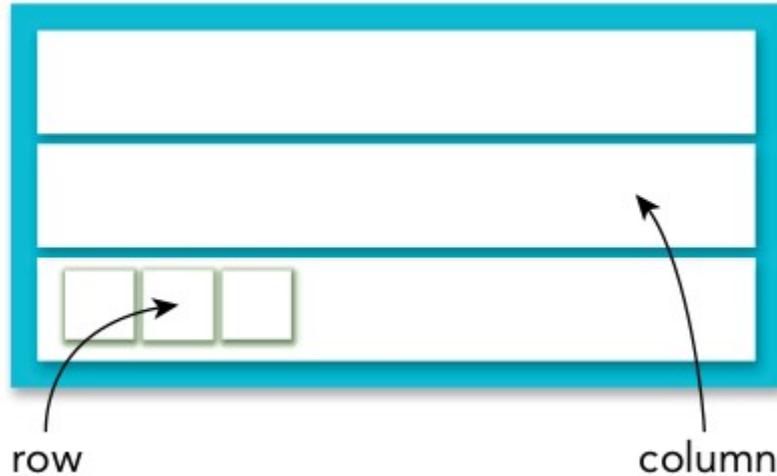
# Flutter

## #6.1

Aninhamento de linhas e  
colunas

Uma ótima maneira de criar layouts exclusivos é combinar widgets de Column e Row para necessidades particulares.

Imagine ter uma página de diário com texto em uma coluna com uma linha aninhada contendo uma lista de imagens



```
Column(  
  mainAxisAlignment: MainAxisAlignment.start,  
  mainAxisSize: MainAxisSize.max,  
  spaceEvenly:  
  children: <Widget>[  
    Text('Columns and Row Nesting 1'),  
    Text('Columns and Row Nesting 2'),  
    Text('Columns and Row Nesting 3'),  
    Padding(padding: EdgeInsets.all(16.0)),  
    Row(  
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
      children: <Widget>[  
        Text('Row Nesting 1'),  
        Text('Row Nesting 2'),  
        Text('Row Nesting 3'),  
      ],  
    ),  
  ],  
)
```

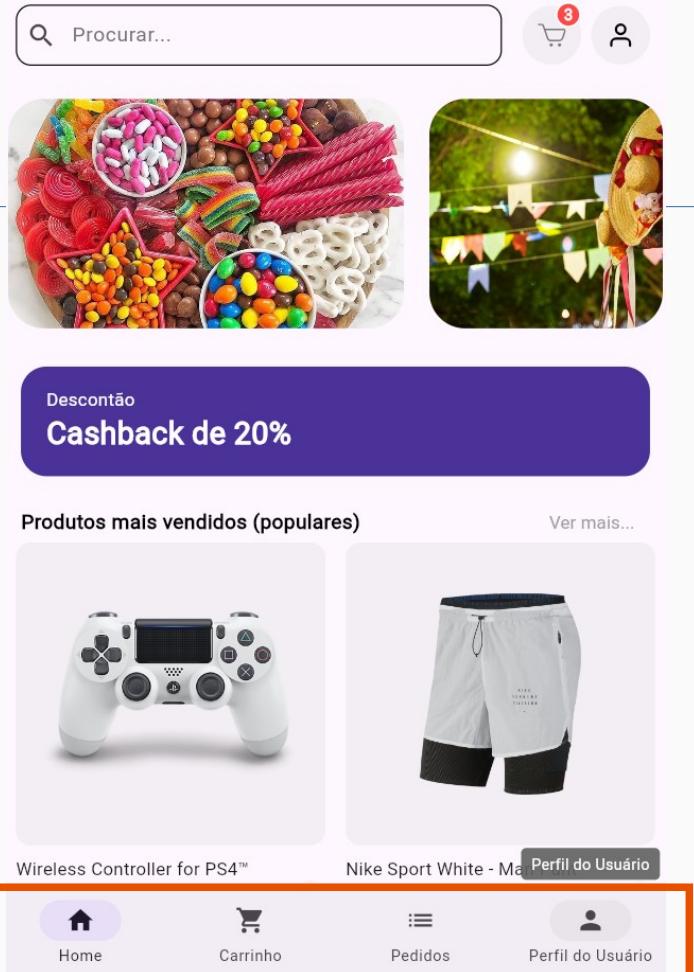
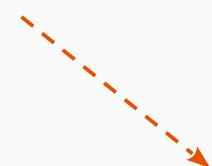


Flutter

#7

BottomNavigationBar

**BottomNavigationB  
ar** é um widget do  
Material Design que  
permite a navegação  
entre telas com o  
auxílio de uma barra  
na parte inferior da  
tela



Quando o  
**BottomNavigationItem** é  
selecionado, a  
página  
apropriada é  
criada.

The screenshot shows a mobile application interface. At the top, there is a search bar with a magnifying glass icon and the placeholder text "Procurar...". To its right are three circular icons: a shopping cart with a red notification badge containing the number "3", and a user profile icon. A horizontal blue line separates the header from the main content area.

In the main content area, there are two images: one showing a variety of colorful candies and another showing a string of colorful flags hanging outdoors at night.

A purple button with white text is displayed, reading "Descontão" and "Cashback de 20%".

The section below is titled "Produtos mais vendidos (populares)" and includes a "Ver mais..." link. It features two product cards: one for a "Wireless Controller for PS4™" (a white DualShock 4 controller) and another for "Nike Sport White - Ma" (a pair of white and black athletic shorts).

At the bottom, a navigation bar is visible with four items: "Home" (selected, indicated by a purple background), "Carrinho" (Cart), "Pedidos" (Orders), and "Perfil do Usuário" (User Profile). The number "51" is located in the bottom right corner.

# BottomNavigation: Demo



Descontão  
**Cashback de 20%**

Produtos mais vendidos (populares)

Ver mais...

Two product cards. The left card shows a white PS4 DualShock 4 controller with the text 'Wireless Controller for PS4™'. The right card shows a pair of white and black Nike Sportswear shorts with the text 'Nike Sport White - Ma... Perfil do Usuário'. At the bottom, there is a navigation bar with four items: 'Home' (selected), 'Carrinho', 'Pedidos', and 'Perfil do Usuário'. Arrows point from the text labels above to their corresponding icons in the navigation bar.

```
8  Widget build(BuildContext context) {  
9    return Scaffold(  
10      bottomNavigationBar: NavigationBar(  
11        selectedIndex: 0,  
12        onDestinationSelected: (int index) {},  
13        destinations: [  
14          NavigationDestination(icon: Icon(Icons.home), label: 'Home'),  
15          NavigationDestination(  
16            icon: Icon(Icons.shopping_cart),  
17            label: 'Carrinho',  
18          ), // NavigationDestination  
19          NavigationDestination(icon: Icon(Icons.list), label: 'Pedidos'),  
20          NavigationDestination(  
21            icon: Icon(Icons.person),  
22            label: 'Perfil do Usuário',  
23          ), // NavigationDestination  
24        ],  
25      ), // NavigationBar  
26      appBar: AppBar( // AppBar ...  
27    )  
28  )
```

# BottomNavigation: Criando

---

- Quando um opção é selecionada no **BottomNav** temos um sincronismo que permite a alteração de páginas no **Scaffold** que o contém.
  - É necessário que a página extenda a classe StatelessWidget para que o **Scaffold** “escute” as mudanças ocorridas no **BottomNav**.
  - Esse procedimento acontece com o controle da modificação de uma variável que será utilizada por ambos: **Scaffold** e **BottomNav**.

# BottomNavigation: Criando

- É necessário instanciar Scaffold para começar a composição com o BottomNav ou NavigationBar

```
1 import 'package:flutter/material.dart';
2
3 class MainPage2 extends StatelessWidget {
4   const MainPage2({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return Scaffold(
9       );
10    );
11  }
12 }
```

# BottomNavigation: Criando

- É necessário instanciar **Scaffold** para começar a composição com o **BottomNav** ou **NavigationBar**

- Observe a propriedade
  - “**destinations**”
    - Ela se refere às opções disponibilizada ao usuário

```
1 import 'package:flutter/material.dart';
2
3 class MainPage2 extends StatelessWidget {
4   const MainPage2({super.key});
5   @override
6   Widget build(BuildContext context) {
7     return Scaffold(
8       bottomNavigationBar: NavigationBar(
9         destinations: destinations
10       ), // NavigationBar
11     ); // Scaffold
12   }
13 }
```

# BottomNavigation: Criando

---

- **BottomNav** ou **NavigationBar** nada mais são do que uma **lista** (vetor).
  - O que nos permite adicionar várias opções que são indexadas.
  - Este índice será utilizado para sincronizar a apresentação das páginas no **Scaffold**.
  - No caso do **NavigationBar**, os **destinations** que também recebe uma lista de “destinos” a serem apresentados

# BottomNavigation: Criando

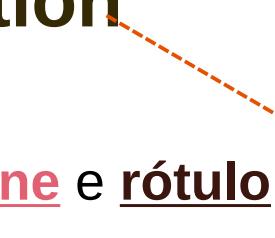
- Utilizaremos uma list (“[ ]”)a que conterá as opções a serem disponibilizadas ao usuário

```
1 import 'package:flutter/material.dart';
2
3 class MainPage2 extends StatelessWidget {
4   const MainPage2({super.key});
5   @override
6   Widget build(BuildContext context) {
7     return Scaffold(
8       bottomNavigationBar: NavigationBar(
9         destinations: []
10      ), // NavigationBar
11    ); // Scaffold
12  }
13 }
```

# BottomNavigation: Criando

- Perceba o uso de:
  - **NavigationDestination**
    - Este widget incluirá:
      - Uma opção com ícone e rótulo

```
1 import 'package:flutter/material.dart';
2
3 class MainPage2 extends StatelessWidget {
4   const MainPage2({super.key});
5   @override
6   Widget build(BuildContext context) {
7     return Scaffold(
8       bottomNavigationBar: NavigationBar(
9         destinations: [
10           NavigationDestination(
11             icon: icon,
12             label: label
13           ) // NavigationDestination
14         ],
15       ), // NavigationBar
16     ); // Scaffold
17   }
18 }
```



# BottomNavigation: Criando

- Perceba o uso de:
  - **NavigationDestination**
    - Este widget incluirá:
      - Uma opção com ícone e rótulo

```
1 import 'package:flutter/material.dart';
2
3 class MainPage2 extends StatelessWidget {
4   const MainPage2({super.key});
5   @override
6   Widget build(BuildContext context) {
7     return Scaffold(
8       bottomNavigationBar: NavigationBar(
9         destinations: [
10           NavigationDestination(
11             icon: Icon(Icons.home),
12             label: 'Início'
13           ) // NavigationDestination
14         ],
15       ), // NavigationBar
16     ); // Scaffold
17   }
18 }
```

# BottomNavigation: Criando

- **Destinations** com:

- Quatro opções



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

class MainPage2 extends StatelessWidget {
  const MainPage2({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      bottomNavigationBar: NavigationBar(
        destinations: [
          NavigationDestination(icon: Icon(Icons.home), label: 'Home'),
          NavigationDestination(
            icon: Icon(Icons.shopping_cart),
            label: 'Carrinho',
          ), // NavigationDestination
          NavigationDestination(icon: Icon(Icons.list), label: 'Pedidos'),
          NavigationDestination(icon: Icon(Icons.person), label: 'Você'),
        ],
      ), // NavigationBar
    ); // Scaffold
  }
}
```

# BottomNavigation: Criando

## ● SelectedIndex

- Define qual é a opção atual
  - Através de um índice vinculado
    - O valor atribuído a ele define a posição/opção da navegação

```
class MainPage2 extends StatelessWidget {  
  const MainPage2({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      bottomNavigationBar: NavigationBar(  
        selectedIndex: 0,  
        onDestinationSelected: (value) {},  
        destinations: [  
          NavigationDestination(icon: Icon(  
            Icons.shopping_cart,  
            color: Colors.purple,  
          ),  
          label: 'Carrinho',  
        ),  
        NavigationDestination(icon: Icon(  
          Icons.home,  
          color: Colors.purple,  
        ),  
        label: 'Home',  
      ],  
    );  
  }  
}
```

# BottomNavigation: Criando

- **onDestinationSelected**

- Controlará a variável que:
  - Atualizará o parâmetro **selectedIndex**
    - Com isso, teremos uma mudança na página apresentada pelo **Scaffold**.

```
class MainPage2 extends StatelessWidget {  
  const MainPage2({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      bottomNavigationBar: NavigationBar(  
        selectedIndex: 0,  
        onDestinationSelected: (value) {},  
        destinations: [  
          NavigationDestination(icon: Icon(Icons.shopping_cart),  
            label: 'Carrinho',), // NavigationDestination  
          NavigationDestination(icon: Icon(Icons.home),  
            label: 'Home',),  
          NavigationDestination(icon: Icon(Icons.person),  
            label: 'Profile',),  
        ],  
      ), // NavigationBar  
    ); // Scaffold  
  }  
}
```

# BottomNavigation: Criando

- **onDestinationSelected**

- Observe que **onDestinationSelected** é uma função que:
  - Atualizará uma variável que será utilizada por **selectedIndex**



```
class MainPage2 extends StatelessWidget {  
  const MainPage2({super.key});  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      bottomNavigationBar: NavigationBar(  
        selectedIndex: 0,  
        onDestinationSelected: (value) {},  
        destinations: [  
          NavigationDestination(icon: Icon(Icons.shopping_cart),  
            label: 'Carrinho',), // NavigationDestination  
          NavigationDestination(icon: Icon(Icons.home),  
            label: 'Home',), // NavigationDestination  
        ],  
      ), // NavigationBar  
    ); // Scaffold  
  }  
}
```

# BottomNavigation: Criando

- **onDestinationSelected**

- Agora temos uma função implementada que altera de fato o valor de uma variável: **\_index**
  - Que será utilizada em **selectedIndex**
    - Assim o **NavigationBar** vincula o **Scaffold** à uma nova página no parâmetro **body**

```
class MainPage2 extends StatelessWidget {  
  const MainPage2({super.key});  
  @override  
  Widget build(BuildContext context) {  
    int _index = 0;  
    return Scaffold(  
      bottomNavigationBar: NavigationBar(  
        selectedIndex: _index,  
        onDestinationSelected: (value) {  
          _index = value;  
        },  
        destinations: [  
          NavigationDestination(icon: Icon(  
            Icons.shopping_cart  
          ), label: 'Carrinho',), // NavigationDestination  
          NavigationDestination(icon: Icon(  
            Icons.home  
          ), label: 'Home',), // NavigationDestination  
          NavigationDestination(icon: Icon(  
            Icons.logout  
          ), label: 'Logout',), // NavigationDestination  
        ],  
      ), // NavigationBar
```

# BottomNavigation: Criando

---

- Mesmo **onDestinationSelected** sendo acionado nada acontecerá pois:
  - Deveremos ter uma classe do tipo **StateFull**
    - Pois assim, poderemos utilizar o controle de estado de um objeto, no caso a variável **\_index**
    - Portanto, deveremos converter a classe do tipo **StateLess** para **StateFull** para podermos utilizar uma função (**setState**) que controla o estado de algum objeto

# Sincronizando a Navegação

- Para sincronizar o **Scaffold**, temos:
  - De converter a classe para **Stateful** e:
    - Adicionar o método **setState** no corpo da função que controla os valores dos índices.
    - Observe o reposicionamento da criação da variável **\_index**
  - Depois, temos de adicionar as páginas no parâmetro **body** do **Scaffold**

```
class MainPage2 extends StatefulWidget {  
  const MainPage2({super.key});  
  @override  
  State<MainPage2> createState() => _MainPage2State();  
}  
  
class _MainPage2State extends State<MainPage2> {  
  int _index = 0;  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      bottomNavigationBar: NavigationBar(  
        selectedIndex: _index,  
        onDestinationSelected: (value) {  
          setState(() {  
            _index = value;  
          });  
        },  
        destinations: [  
          NavigationDestination(icon: Icon(Icons.home))  
        ]  
      ),  
    );  
}
```

# Sincronizando a Navegação

---

- O parâmetro/argumento **body** do **Scaffold** aceita uma lista de **widgets** não indexada ou indexada:
  - Quando a lista for indexada isto permitirá a mudança de acordo com o valor atual o índice utilizado
  - Lembre que o parâmetro **body** é onde adicionamos os **widgets** que deverão ser apresentados no corpo do **Scaffold**, podendo ser:
    - Páginas ou qualquer outro widget

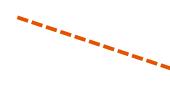
# Body: vinculando páginas/widgets

- Exemplo de **body** com a inserção de **widgets** por meio do uso de **coluna**

```
class _ MainPage2State extends State<MainPage2> {  
    int _index = 0;  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            body: Column(children: [  
                //lista de widgets  
            ], // Column  
            bottomNavigationBar: NavigationBar(  
                selectedIndex: _index,  
                onDestinationSelected: (value) {  
                    setState(() {  
                        _index = value;  
                    });  
                },  
                destinations: [  
                    NavigationDestination(icon: Icon(Icons.add),  
                    NavigationDestination(  
                        icon: Icon(Icons.shopping_cart),  
                        label: 'Carrinho',  
                    ),  
                ],  
            ),  
        );  
    }  
}
```

# Body: vinculando páginas/widgets

- No caso do uso de navegação, deveremos:
  - Utilizar lista aninhada que é uma estrutura de linhas e colunas que será indexada
  - E com o uso do “ambiente” setState temos uma alteração automática do conteúdo da variável \_index



```
class _MainPage2State extends State<MainPage2> {  
    int _index = 0;  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            body: [][_index],  
            bottomNavigationBar: NavigationBar(  
                selectedIndex: _index,  
                onDestinationSelected: (value) {  
                    setState(() {  
                        _index = value;  
                    });  
                },  
            destinations: [  
                NavigationDestination(icon: Icon(Icons.add),  
                NavigationDestination(  
                    icon: Icon(Icons.shopping_cart),  
                    label: 'Carrinho',  
                ), // NavigationDestination  
                NavigationDestination(icon: Icon(Icons.add),  
                label: 'Carrinho',  
            ],  
        );  
    }  
}
```

# Body: vinculando páginas/widgets

- Agora, poderemos associar as páginas que serão apresentadas:
  - Indicando-as na lista e
  - Observe que a função setState provoca as mudanças de páginas em função da atualização da variável \_index

```
class _ MainPage2State extends State<MainPage2> {  
    int _index = 0;  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            body: [  
                MyHomePage(title: 'Mercearias.shop'),  
                CartScreen(),  
                PedidosPage(),  
                RestaurantePage(),  
            ][_index],  
            bottomNavigationBar: NavigationBar(  
                selectedIndex: _index,  
                onDestinationSelected: (value) {  
                    setState(() {  
                        _index = value;  
                    });  
                },  
                destinations: [  
                    NavigationDestination(icon: Icon(Icons.  
                ]  
            );  
    }  
}
```



Flutter

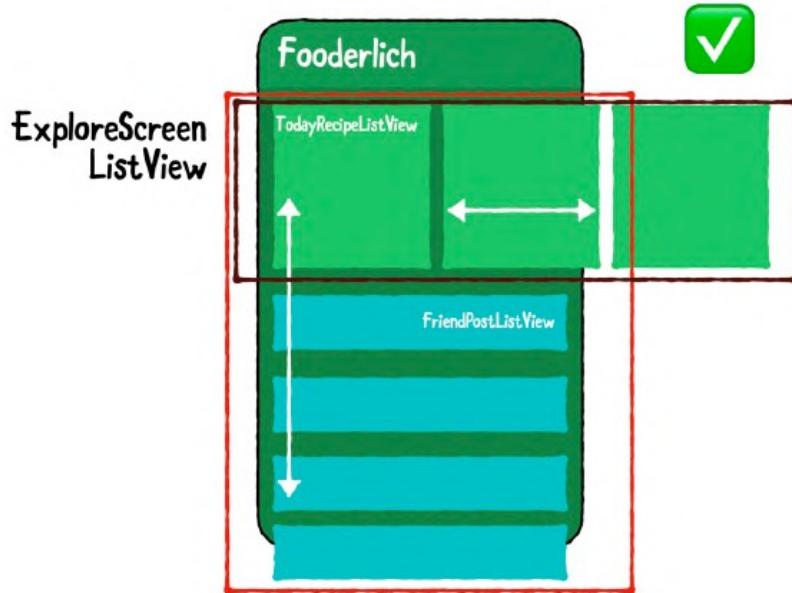
#8

ListView

O widget **ListView** exibe as informações em forma de listagem  
- de cima para baixo ou de um lado para o outro.

Além disso, o widget ListView tem seu próprio recurso de rolagem.

Você pode colocar 100 itens em um **ListView** mesmo que apenas 20 itens caibam na tela. Quando o usuário rola a tela, os itens saem da tela enquanto outros itens se movem.



```
33 Expanded(  
34   child: ListView.builder(  
35     physics: const BouncingScrollPhysics(),  
36     itemCount: favors.length,  
37     itemBuilder: (BuildContext context, int index) {  
38       final favor = favors[index];  
39       return Card(  
40         key: ValueKey(favor.uuid),  
41         margin: const EdgeInsets.symmetric(vertical: 10.0, horizontal: 25.0),  
42         child: Padding(  
43           child: Column(  
44             children: <Widget>[  
45               _itemHeader(favor),  
46               Text(favor.description),  
47               _itemFooter(favor)  
48             ],  
49           ),  
50           padding: EdgeInsets.all(8.0),  
51         ),  
52       );  
53     },  
54   ),
```



Flutter

#9

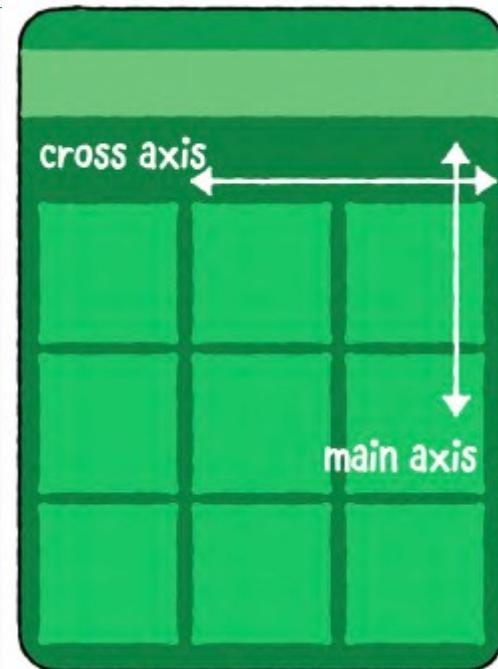
GridView

---

O widget **GridView** exibe as informações em forma de grade.

**GridView** é uma matriz 2D de widgets roláveis. Ele arranja os filhos em uma grade e suporta rolagem horizontal e vertical.

É similar ao **Listview**.





# Flutter

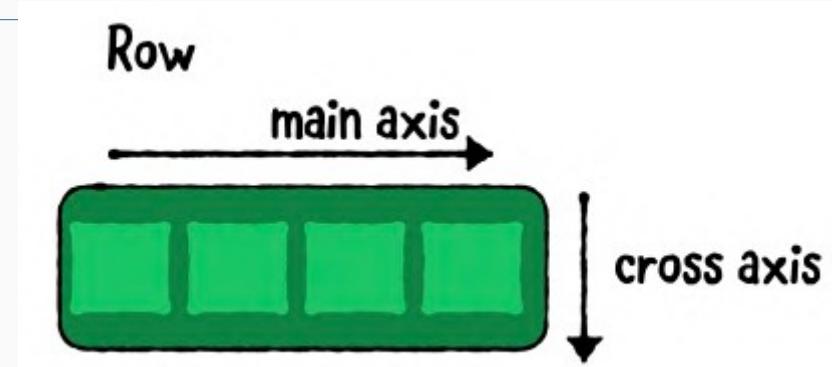
#9.1

Compreendendo o eixo  
transversal e o principal

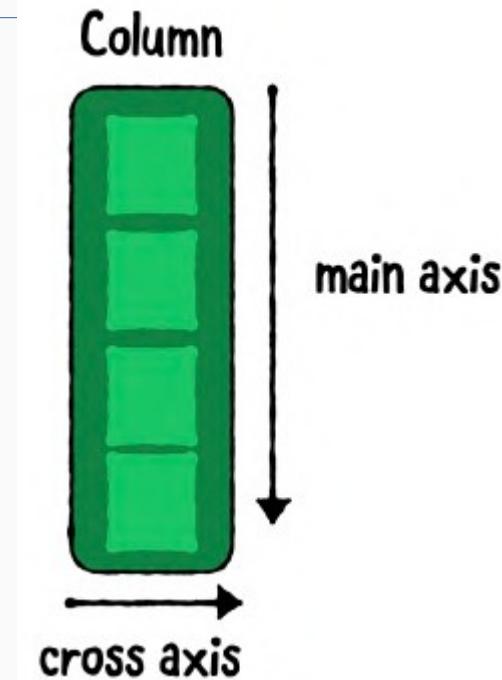
Qual é a diferença entre o eixo principal e o eixo transversal (cruzado)? Lembre-se de que Colunas e Linhas são como **ListView**s, mas sem uma visualização de rolagem.

O eixo principal sempre corresponde à direção de rolagem!

Se sua direção de rolagem for horizontal, você pode pensar nisso como uma linha. O eixo principal representa a direção horizontal, conforme mostrado na imagem acima:



Se sua direção de rolagem for vertical, você pode pensar nela como uma Coluna. O eixo principal representa a direção vertical, conforme mostrado ao lado:





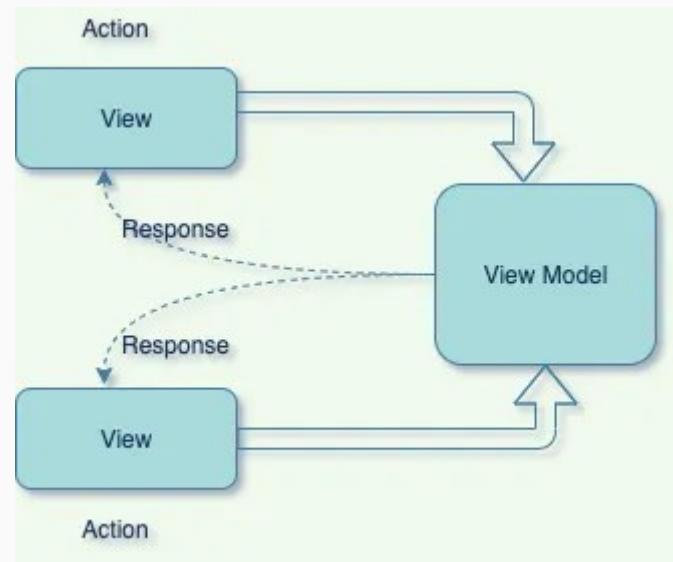
Flutter

#10

MVP

O MVVM é útil para mover a lógica de negócios de exibição para ViewModel e Model.

- ViewModel é o mediador entre View e Model que carrega todos os eventos do usuário e retorna o resultado.





# Flutter

#10.1

Compreendendo a  
estrutura do MVVM



# Flutter

#11

## Responsividade

# Responsividade

---

- Adicionando Responsividade aos Widgets
  - A interface do usuário responsiva altera a interface do usuário da tela/widget do aplicativo de acordo com o tamanho das diferentes telas de celulares.
  - A interface do usuário responsiva é muito útil quando o mesmo aplicativo é feito para celular, web, desktop, relógios (wear apps).
  - A interface do usuário responsiva reorganiza a interface do usuário de acordo com a orientação e o tamanho do dispositivo.

# Responsividade

---





# Flutter

#11.1

Utilizando Responsividade

# Responsividade: Fatores que contribuem para UI's responsivas

---

- Tipos de gerenciadores de responsividade
  - LayoutBuilder
  - MediaQuery
  - Expanded e Flexible
  - AspectRatio e FractionallySizedBox
  - Wrap

# Responsividade: Fatores que contribuem para UI's responsivas

---

- LayoutBuilder

- O LayoutBuilder altera a interface do usuário de acordo com as restrições de tamanho da tela do dispositivo.
- Este widget usa um construtor que retorna um widget e altera a exibição de acordo com a condição especificada pelo usuário.

# Responsividade: Fatores que contribuem para UI's responsivas

---

- MediaQuery

- é a classe/widget que nos permite consultar o tamanho atual da mídia.
- Podemos usar MediaQueryData.size para obter o tamanho da tela.
- Isso será reconstruído automaticamente sempre que o MediaQueryData for alterado.



Flutter

#13

---

# Processos Assíncronos

# Processos Síncronos

---

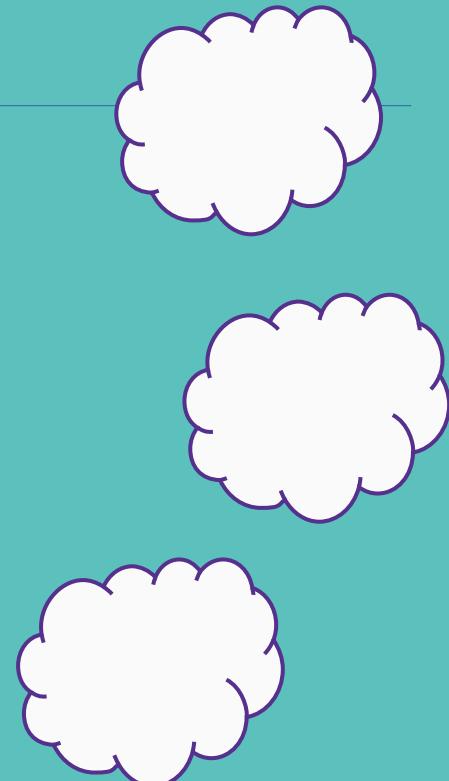
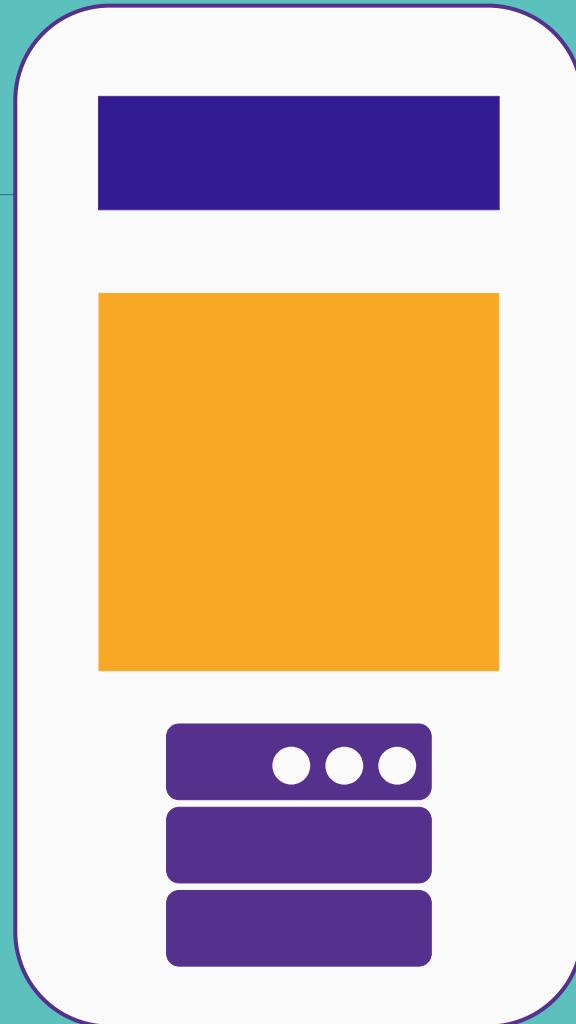
- No Flutter processos são executados de forma síncrona
  - Na programação síncrona um processo só é executado quando outro é finalizado
  - Isto pode causar uma inconsistência ou bloqueio de processos quando se trata de uso de processos que dependam da execução de outro(s) processo(s)
    - Por exemplo: processos que requerem acesso a recursos de banco de dados, arquivos, imagens de câmeras, dentre outros.

# Processos Assíncronos

---

- No Flutter processos assíncronos são aqueles que são executados de forma paralela, ou concomitantemente.
  - Isto quer dizer, que a aplicação poderá executar diversas tarefas
    - Por exemplo:
      - acessar dados, enquanto renderiza uma tela.
      - Autenticar usuário e aguardar a resposta e ainda realizar outra tarefa.
  - Para definir que um processo é assíncrono utilizamos a palavra-chave **async**
    - Desta forma teremos um processo sendo realizado e aguardando resposta, enquanto outro pode ser executado simultaneamente.

# Async faz IO simples





# Flutter

#14

## Navigation

# Navegação com Destinations

---

- O **BottomNavigationBar** é um widget de navegação entre páginas no Flutter.
  - É uma componente que fica localizado parte inferior da tela (página) que permite a **navegabilidade** entre telas disponíveis no app.

- Exemplo de código: return **Scaffold**(  
    **bottomNavigationBar: NavigationBar**(  
        **destinations: const [**  
            **NavigationDestination**(  
                icon: Icon(Icons.home\_outlined),  
                label: 'Home',  
            ),  
            **NavigationDestination**(  
                icon: Icon(Icons.shopping\_cart\_outlined),  
                label: 'Carrinho',  
            ),

# Navegação com Destinations

```
return Scaffold(  
  appBar: AppBar( // AppBar ...  
  body: [ ...  
    bottomNavigationBar: NavigationBar(  
      selectedIndex: _index,  
      onDestinationSelected: (int position) {  
        setState(() {  
          _index = position;  
        });  
      },  
      destinations: const [  
        NavigationDestination(  
          icon: Icon(Icons.home_outlined),  
          label: 'Home',  
        ), // NavigationDestination  
        NavigationDestination(  
          icon: Icon(Icons.shopping_cart_outlined),  
          label: 'Carrinho',  
        ), // NavigationDestination  
        NavigationDestination(  
          icon: Icon(Icons.line_style_outlined),  
          label: 'Pedidos',  
        ), // NavigationDestination
```



Flutter

#15

Enviando e  
Recebendo dados  
entre telas

# Enviando objetos entre páginas

- O envio de objetos entre páginas deve ser feito utilizando um widget de roteamento, o MaterialPageRoute.

```
18     child: ElevatedButton(
19       onPressed: () async {
20         final result = await Navigator.push(
21           context,
22           MaterialPageRoute<String>(
23             builder: (context) => const PaginaDois(),
24           ), // MaterialPageRoute
25         );
26       }
27     );
```

# Enviando objetos entre páginas

- Depois, processa-se o retorno do objeto na segunda página com o seguinte comando

```
61         child: ElevatedButton(  
62             onPressed: () {  
63                 // Feche a tela e retorne "1a mensagem enviada!" como resultado.  
64                 Navigator.pop(context, '1a mensagem enviada!');  
65             },  
66             child: const Text('1a msg!'),  
67         ), // ElevatedButton  
68     ), // Padding  
69     ); // Container
```