

# PM – PROGRAMAÇÃO MOBILE

## INTRODUÇÃO FLUTTER

[edson.sensato@espm.br](mailto:edson.sensato@espm.br)



## Introdução

---

SDK para desenvolvimento de *apps* criado pelo Google

Permite o desenvolvimento multiplataforma: Android, iOS, Web ou até mesmo Desktop

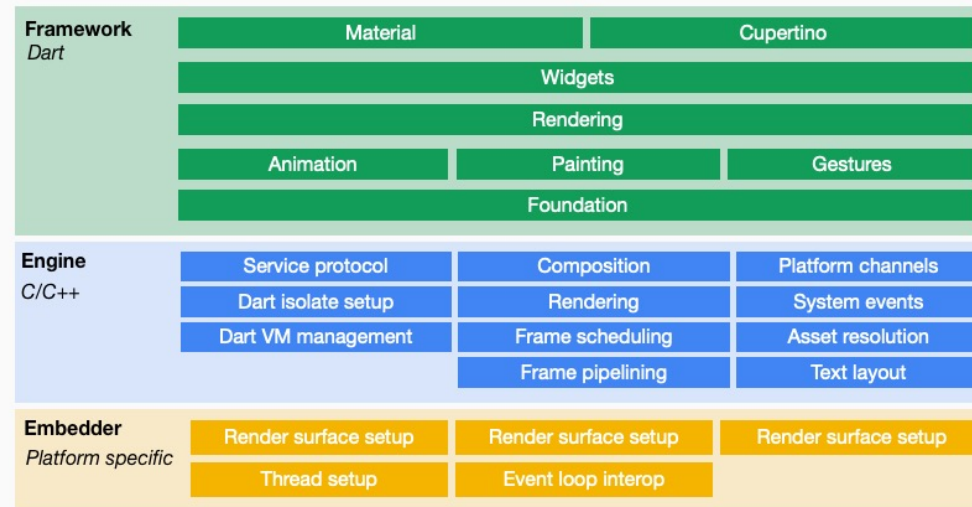
Utiliza uma linguagem de desenvolvimento própria, o **Dart**

Possui um *plugin* para ser utilizado junto com o **Android Studio**



## Camadas

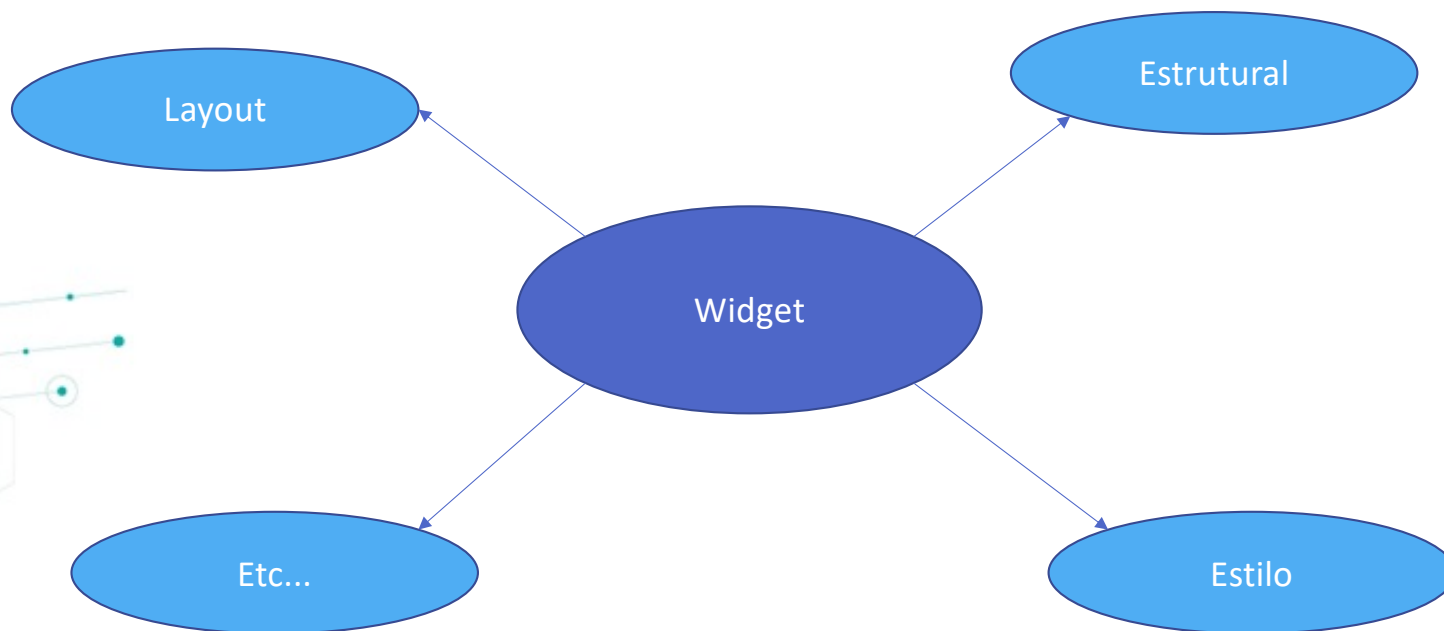
### Flutter system overview



Fonte: <https://flutter.dev/docs/resources/technical-overview>

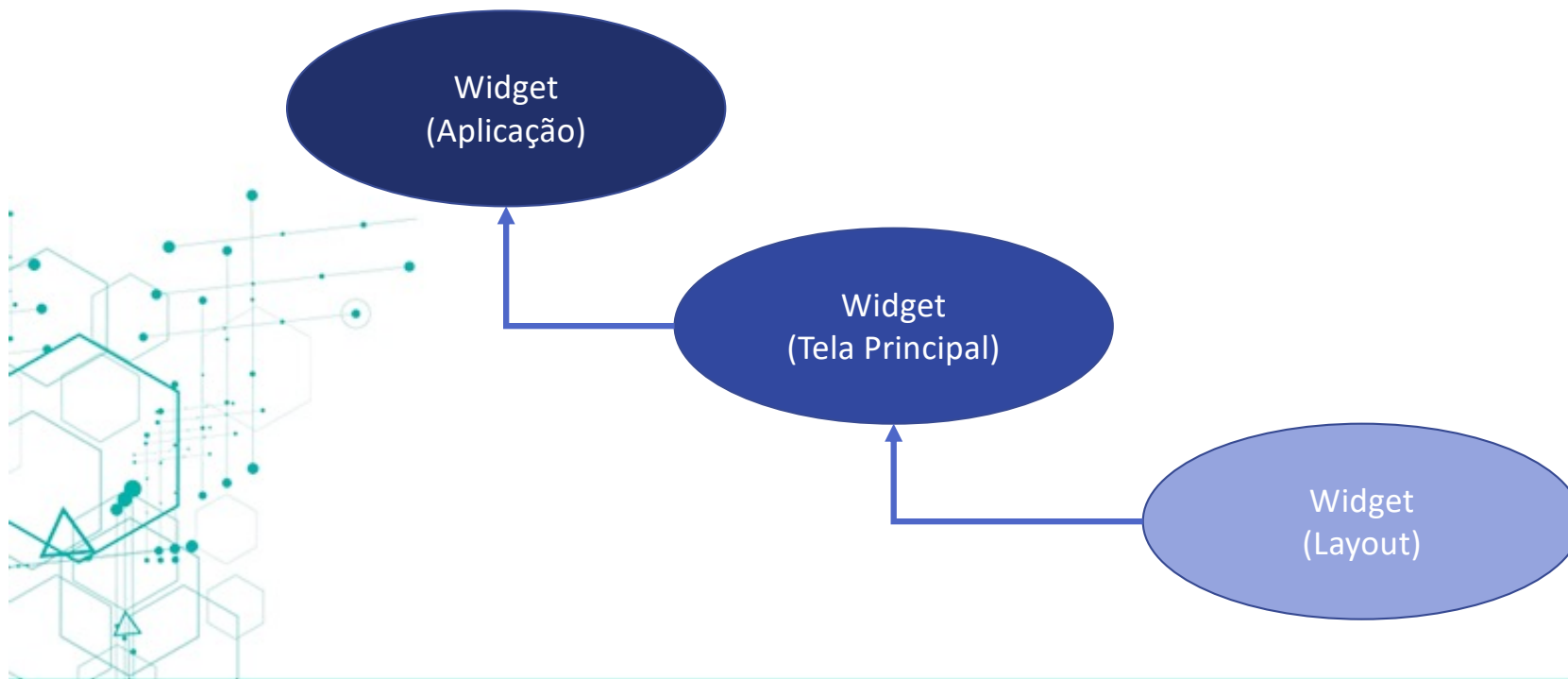
## Widget

---



## Hierarquia Widgets

---



## Linguagem Dart

---

Criada pelo *Google* para ser utilizada no *AdWords*

Orientada a objetos, utiliza uma sintaxe similar ao C (*C-style*)

Compilada também em *Just in Time (JIT)* que permite rapidamente compilar as alterações realizadas no código sem ter que compilar a aplicação inteira (em tempo de desenvolvimento)

Compilada nativamente (em tempo de lançamento – release, *Ahead of Time - AOT*)

*Playground* on-line:

<https://dartpad.dev>



## Dart Alô Mundo

---

// primeiro programa Dart

```
void main() {
```

```
    print('Alô Mundo Dart!!!');
```

```
}
```

// segundo programa Dart

```
void main() => print('Alô Mundo Dart!!!');
```

## Variáveis

---

```
var msg = 'Alo';
```

```
double preco = 12.4;
```

```
int qtde;
```

```
print(2000.toString());
```

Tipos são inferidos automaticamente

**var** deve ser utilizado quando o tipo de dados da variável deva ser inferido

Tudo em flutter é um objeto: null, int, true, etc...



## Null Safety

---

```
void main() {  
  int x = 0;  
  x = null; // ERRO de compilação!
```

```
  int? y;
```

```
  y = null; // OK
```

```
  if (y != null) {  
    print('Alo ${y.isEven}');  
  }  
}
```

```
  print('Alo ${y?.isEven}');
```

Por padrão nenhuma variável pode receber um valor nulo

```
var x = {'maria': {'nota': 10}};
```

```
print(x['joao']?['nota']);
```

## Typedef

---

```
typedef Calculadora(int x, int y);
```

```
int somar(int x, int y) {  
    return x + y;  
}
```

```
int subtrair(int x, int y) {  
    return x - y;  
}
```

```
Calculadora calc = somar;
```

```
print(calc(1, 20));
```

Permite criar uma variável ponteiro para uma função

## Classes

```
class Heroi {  
  
    String nome = '';  
    var anoCriacao = 1900;  
    int? forca;  
  
}  
  
void main() {  
    var heroi = Heroi();  
    heroi.nome = 'Batman';  
    print('- Nome ${heroi.nome}, criado em: ${heroi.anoCriacao}');  
}
```

Classes são declaradas de forma semelhante ao Java

Variáveis devem ser iniciadas por padrão

Não é necessário declarar explicitamente o tipo de dados

Variáveis opcionais (sem valores iniciados) devem ser indicadas por '?'

## Construtores

```
class Heroi {  
  
    String nome = '';  
    var anoCriacao = 1900;  
    int? forca;  
  
    Heroi (this.nome, this.anoCriacao) {  
        forca = 10;  
    }  
}
```

Construtores possuem o mesmo nome da classe

**this** pode ser utilizado como atalho nos argumentos do construtor para definir valores diretamente nos atributos de mesmo nome

## Parâmetros Opcionais

```
Herói ({this.nome = '', this.anoCriacao = 1900}) {  
    forca = 10;  
}
```

```
Herói.forcaDefault(this.nome, this.anoCriacao, [this.forca = 50]);
```

```
var herói1 = Herói(anoCriacao: 1939, nome: 'Batman');  
var herói2 = Herói.forcaDefault('Homem Aranha', 1962);  
print('- Nome ${herói1.nome}, criado em: ${herói1.anoCriacao}, força: ${herói1.forca}');  
print('- Nome ${herói2.nome}, criado em: ${herói2.anoCriacao}, força: ${herói2.forca}');
```

`{}` indicam parâmetros nomeados e opcionais

`[]` indicam parâmetros posicionais e opcionais

Mais de um construtor pode ser declarado desde que possua um nome específico (`forcaDefault`)

## Listas

---

```
var filmes = [];  
  
var heroi1 = Heroi(anoCriacao: 1939, nome: 'Batman');  
  
heroi1.filmes.add('Batman Returns');  
heroi1.filmes.add('Batman Forever');  
  
print('Total filmes: ${heroi1.filmes.length}');  
heroi1.filmes.forEach((filme){  
  print('- ${filme}');  
});
```

Listas são representadas pelo objeto `List`

Métodos comuns: `add`, `length`

`forEach` pode ser utilizado para percorrer os itens da lista

## Tratamento de Exceções

---

on pode ser substituído por catch...

```
try {  
    print(heroi1.filmes[10]);  
  
    } on RangeError {  
        print('Indice invalido');  
    } finally {  
        print('Processado');  
    }
```

## Mapas

---

```
var caracteristicas = {};  
  
var heroi2 = Heroi.forcaDefault('Homem Aranha', 1962);  
heroi2.caracteristicas['ataque'] = 10;  
heroi2.caracteristicas['defesa'] = 20;  
  
print('Ataque: ${heroi2.caracteristicas['ataque']}');
```

Mapas são representadas pelo objeto Map



## Enums

---

```
enum Caracteristicas {  
    ataque, defesa  
}  
  
heroi2.caracteristicas[Caracteristicas.ataque] = 10;  
heroi2.caracteristicas[Caracteristicas.defesa] = 20;  
  
print('Ataque: ${heroi2.caracteristicas[Caracteristicas.ataque]}');
```

## Final e Const

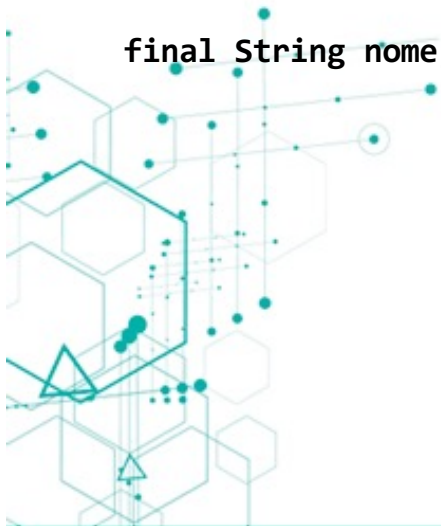
---

Representam valores que não podem ser alterados

**const** deve ser imediatamente iniciado com um valor e **final** não precisa

```
static const superForca = 100;
```

```
final String nome;
```



## Operadores

`as` → *typecast*

```
int x = 10;  
double v = (x as double);
```

`is (is!)` → verifica se um objeto é de um tipo específico  
`print (x is int);` → true

`??=` → atribui valor a uma variável somente se ela tiver valor *null*

```
String v1 = 'Oi';  
v1 ??= 'Alo'; // ignorado pois v1 não é nulo!  
String v2 = null;  
v2 ??= 'Olá'; // executa pois v2 era nulo!  
print(v1); → Oi  
print(v2); → Olá
```

## Pacotes

---

Importa um pacote:

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

Importa uma classe na pasta folder:

```
import 'folder/MinhaClasse.dart'
```

Importa uma classe (**Random**) na de um pacote ('**dart.math**'):

```
import 'dart:math' show Random;
```

Principais pacotes: <https://dart.dev/guides/libraries/useful-libraries>

## Herança

```
class Master {  
    String master;  
  
    Master(String master) {  
        this.master = master;  
    }  
  
    get master {  
        return this.master;  
    }  
}
```

```
class Detail extends Master {  
    String detail;  
  
    Detail(String master, String detail) : super(master) {  
        this.detail = detail;  
    }  
  
    @override String toString() => "$master, $detail";  
}
```



**extends** indica herança

Construtor da classe "pai" pode ser acionado com **super**

## Encapsulamento

---

```
class Carta {
```

```
    String _valor; // _ para privado dentro do arquivo
```

```
    String _naipe;
```

```
    set valor (String valor) {  
        this._valor = valor;  
    }
```

```
    String get valor {  
        return this._valor;  
    }
```

```
}
```

```
Carta carta = Carta();
```

```
    carta.valor = 'As';
```

```
    print(carta.valor);
```

## Programação Assíncrona

```
import 'dart:math' show Random;

class Carta {

  Future<int> sortear() async {
    return await sorteio();
  }

  Future<int> sorteio() async {
    var rng = new Random();
    for (int i = 0; i < 1000000; i++) {
      // loop para "gastar" tempo
    }
    return rng.nextInt(100);
  }
}
```

```
Carta carta = Carta();
carta.sortear().then((numero){
  print (numero);
});
...
```

**async** define uma função como assíncrona

Seu retorno não é imediato, não bloqueia e portanto retorna um **Future**

**await** indica que quando o **sorteio()** retornar alguma coisa então a função sortear também retornará algo (o número sorteado no caso)

**then** aguarda por um retorno **Future** para executar

## Instalação Flutter

---

<https://docs.flutter.dev/get-started/install>

```
flutter config --android-sdk Documents/espm/software/android-sdk
```

```
flutter doctor
```

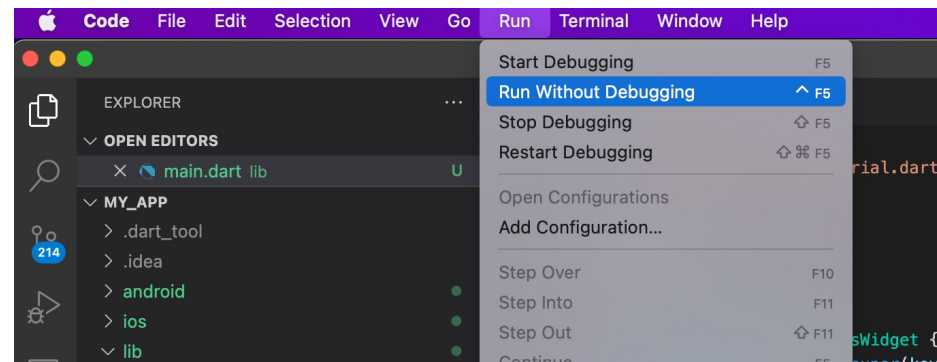
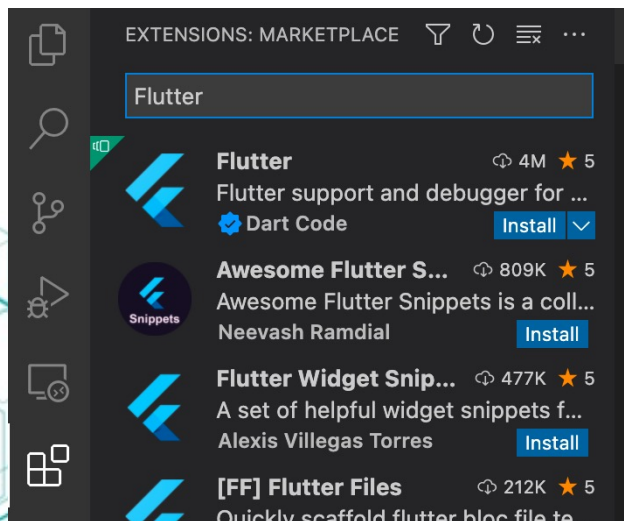
Criando um projeto:

```
flutter create my_app  
cd my_app  
flutter analyze  
flutter test  
flutter run
```



## Integração VS Code

<https://docs.flutter.dev/development/tools/vs-code>



## StatefulWidget / StatelessWidget

**StatefulWidget** possuem (preservam) estado

Com isso são dinâmicos, podem ter seu aspecto (estado) alterado pelo usuário

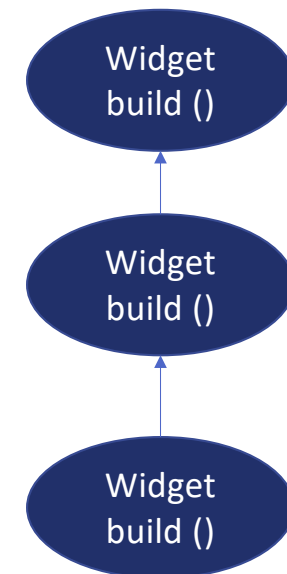
Por exemplo, um **Widget** que exibe um contador que é incrementado quando o usuário clica em um botão deve ser um **StatefulWidget**

Digamos que representam o estado (**model**) e as funções de negócio do app

O estado é representado por um objeto **State**

Já os **StatelessWidgets** são justamente o **contrário** do **StatefulWidget**, portanto são estáticos, não permitindo a interação (alteração do estado) do usuário

Ambos possuem uma função **build(BuildContext context)** que retorna um **Widget** formando uma hierarquia de **Widgets**



## Criando um StatelessWidget

---

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
  }  
}
```

Um **StatelessWidget** pode ser criado simplesmente especializando a classe **StatelessWidget**

Obrigatoriamente a função **build** deve ser implementada para retornar o **Widget** (por exemplo a interface gráfica) que irá compor o **StatelessWidget**

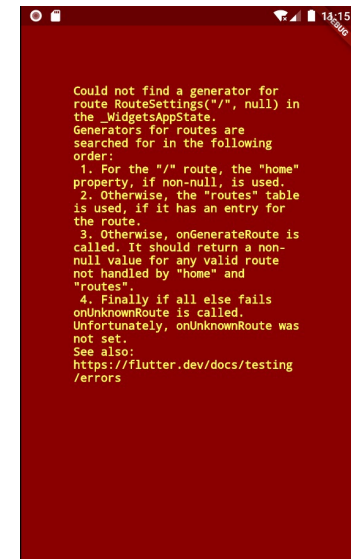
## MaterialApp Widget

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

**MaterialApp** é um **Widget** que implementa vários **Widgets** que são comumente encontrados em aplicações que utilizam os princípios do *Material Design*. Por exemplo: **title**, **theme**, página principal (**home**), etc...

Deve-se definir ao menos a propriedade **home** que será o **Widget** inicial a ser exibido

Sobre *Material Design*: <https://material.io/design/>



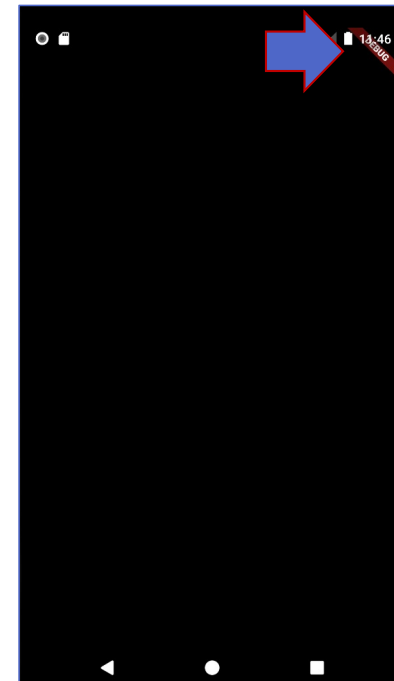
## Definindo a Home

```
class AloFullter extends StatelessWidget{

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Container()
    );
  }
}
```

**Container** é um gerenciador de layout simples

**debugShowCheckedModeBanner** indica se o banner de modo debug deve ou não ser exibido



## Scaffold

```
Widget build(BuildContext context) {  
  return MaterialApp(  
    debugShowCheckedModeBanner: false,  
    home: Scaffold(  
      appBar: AppBar(title: Text('Alô Flutter Bar')),  
      body: Text('Alô Flutter'),  
      bottomNavigationBar: BottomAppBar(child: Text('Aqui embaixo!')),  
      floatingActionButton: FloatingActionButton()  
    ),  
  );  
}
```

**Scaffold** é um outro tipo de gerenciador de layout mais avançado que o **Container**

Nele podemos definir uma barra de aplicação (**AppBar**), um corpo, botões de navegação (**Bottom Navigation**), botões flutuantes (**Floating Action**) e rodapés (**Persistence Footer**)

Opções para **AppBar**:

**title:**  
**Icon(Icons.home)** →  
define um ícone ao  
invés de texto

**backgroundColor:**  
**Colors.amber** → *cor  
de fundo*

## Container

---



Utilizar **EdgeInsets** para **margin**

Exemplo de decoration: **BoxDecoration(border: Border.all(width: 5.0))**

## Column

Representa uma coluna onde **widgets** podem ser adicionado na propriedade **children**

Cada **widget** será apresentado na vertical, um abaixo do outro

```
Column(  
  mainAxisAlignment: MainAxisAlignment.center,  
  crossAxisAlignment: CrossAxisAlignment.stretch,  
  children: <Widget>[  
    Text('Linha 1'),  
    Divider(),  
    Text('Linha 2'),  
    Divider()  
  ],  
)
```



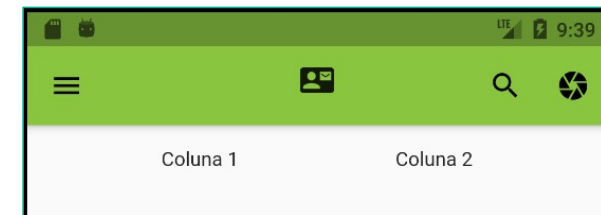


## Row

Representa uma linha onde **widgets** podem ser adicionado na propriedade **children**

Cada **widget** será apresentado na horizontal, um ao lado do outro

```
Row( mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  children: <Widget>[  
    Padding(padding: EdgeInsets.all(16.0), child: Text('Coluna 1')),  
    Padding(padding: EdgeInsets.all(16.0), child: Text('Coluna 2')),  
  ])
```



## Estilos para Texto

---

```
Text(  
  'Alo',  
  style: TextStyle(color: Colors.amber),  
)
```

Referência: <https://api.flutter.dev/flutter/painting/TextStyle-class.html>

## Floating Action Button

---

```
floatingActionButton: FloatingActionButton(  
  child: Icon(Icons.home),  
  onPressed: () {  
    print('Botão pressionado!');  
  }  
)
```

A propriedade **child** define o conteúdo do botão (que pode ser um ícone)

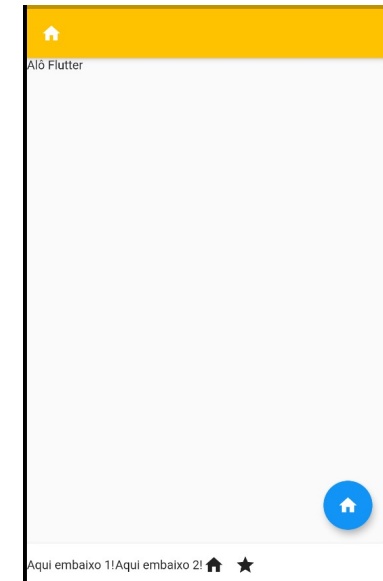
Já o **onPressed** permite implementar uma função que será acionada quando o botão for pressionado

## Bottom Navigation Bar

```
bottomNavigationBar: BottomAppBar(  
  child: Row(children: <Widget>[  
    Text('Aqui embaixo 1!'),  
    Text('Aqui embaixo 2!'),  
    Icon(Icons.home),  
    IconButton(icon: Icon(Icons.star),  
      onPressed: (){  
        print('Botão estrela pressionado!');  
      })  
  ]) )  
)
```

**Row** cria uma linha para que os **widgets** sejam exibidos lado a lado quando definidos em um *array* (<Widget>[ ])

**IconButton** é um ícone que pode receber eventos de *click*

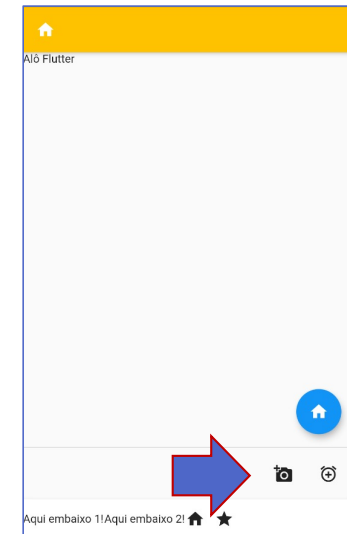


## Persistent Footer Buttons

```
persistentFooterButtons: <Widget>[IconButton(icon: Icon(Icons.add_a_photo),
onPressed: (){
  print('Botão foto pressionado!');
}),IconButton(icon: Icon(Icons.add_alarm),
onPressed: (){
  print('Botão alarme pressionado!');
})]
```

Exibe uma barra de botões permanentes no rodapé

Os botões são definidos em um *array* (<Widget>[])



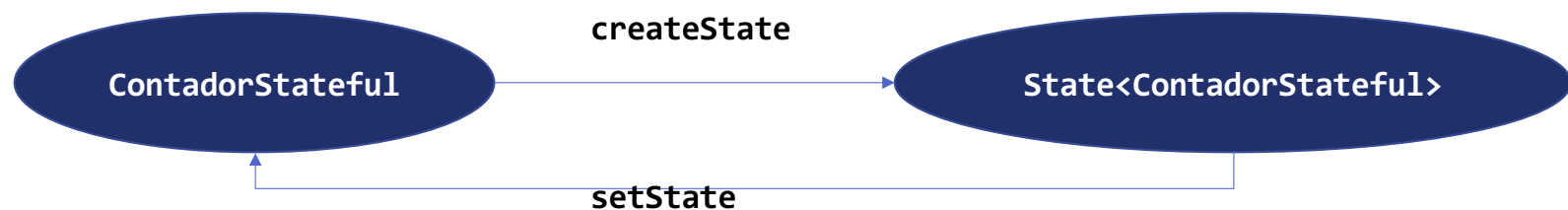
## StatefulWidget

Recordando, os **StatefulWidget** são dinâmicos, isto é, a cada momento podem exibir valores (estado) diferentes

Por exemplo, ao navegar por uma galeria de fotos, a cada clique no botão próxima foto irá atualizar a foto exibida atualmente

Os dados (estado) de um **StatefulWidget** é encapsulado em um objeto **State** que é instanciado por meio de **createState**

Qualquer alteração nos valores do **State** será então refletida de volta na camada visual por meio de **setState()**



## Criando um Contador

```
class Contador extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: Scaffold(  
        body: Center(  
          child: Text('Widget Stateful', style: new TextStyle(  
            fontSize: 30.0,  
            color: Colors.blueAccent  
          )),  
        ),  
      ),  
    );  
  }  
}
```

Criar uma nova classe conforme ao lado do tipo **StatelessWidget** apenas para criar um *layout* inicial

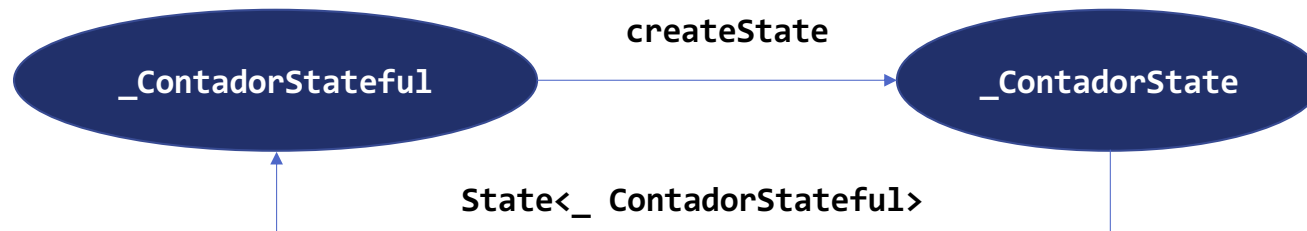
Futuramente o texto **Widget Stateful** será substituído por uma caixa de textos dinâmica e um botão que irá aumentar o valor do contador...

## Estrutura Básica

```
class _ContadorStateful extends StatefulWidget {  
  @override  
  State<_ContadorStateful> createState() => _ContadorState();  
}  
  
class _ContadorState extends State<_ContadorStateful> {  
  @override  
  Widget build(BuildContext context) {  
      
  }  
}
```

Criar uma nova classe conforme ao lado do tipo **StatelessWidget** apenas para criar um *layout* inicial

Futuramente o texto **Widget** **Stateful** será substituído por uma caixa de textos dinâmica e um botão que irá aumentar o valor do contador...





## Definindo Variáveis de Estado

---

```
class _ContadorState extends State<_ContadorStateful> {  
  
  int _contador = 0; // variável que representa o contador atual (estado)  
  
  @override  
  Widget build(BuildContext context) {  
    }  
}
```

Neste exemplo o estado será representado apenas pela variável **\_contador** que armazenará o valor atual do contador e poderá ser incrementado de uma em uma unidade pelo usuário

## Alterando o Estado

```
@override
Widget build(BuildContext context) {
  return Row(mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      Text('$_contador', style: new TextStyle(
        fontSize: 30.0,
        color: Colors.blueAccent
      )),
      IconButton(icon: Icon(Icons.add), onPressed: (){
        setState(() {
          _contador++;
        });
      })
    ],
  );
}
```

A chamada à função **setState()** obriga que toda a camada visual seja atualizada para refletir as últimas alterações nas variáveis de estado

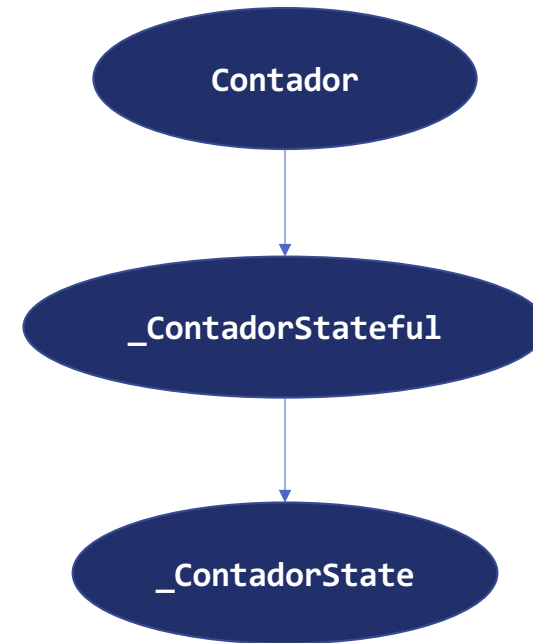
No exemplo, será efetuado um incremento na variável privada **\_contador**

Observar que o **Widget Text** em sua propriedade que define o texto a ser exibido referencia a variável de estado **\_contador**

## Atualizando o StatelessWidget

```
class Contador extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
  
    return MaterialApp(  
      home: Scaffold(  
        body: Center(  
          child: _ContadorStateful()  
        )  
      )  
    );  
  }  
}
```

Finalmente, o **\_ContadorStateful** deve ser referenciado na classe Contador para que seja exibido na UI



## Text Field

---

Para obter o valor do texto informado em um **TextField** basta acessar a propriedade **text** de **TextEditingController**

```
var txtController = TextEditingController();
```

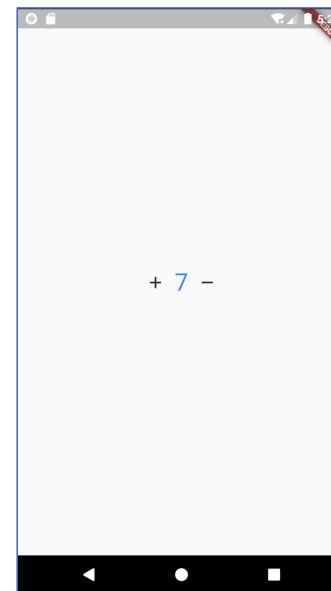
```
TextField(  
  controller: txtController,  
)
```

## Micro Exercício

---

Implementar o botão "menos" para diminuir em uma unidade o contador deixando o layout conforme definido ao lado...

Criar um **TextField** para que o usuário atualize o valor atual do contador



## Exercício

Implementar um simples jogo de dados

Quando o usuário clicar no **Floating Button** um número entre 1 e 6 deve ser sorteado e exibido em um **Text**

Obs: utilizar a classe **Random** vista anteriormente...



## Legibilidade do Código

---

A árvore de componentes (**widgets**) pode crescer rapidamente mesmo em uma simples app Flutter

Com isso a legibilidade e manutenção do código pode ficar bastante comprometida

Para evitar que a árvore se torne grande demais pode-se aplicar algumas técnicas de implementação:

- Constante → cada **widget** é representado por uma constante (**final**)
- Método → definia um método para instanciar e retornar o **widget**
- Classe → (preferível) uma sub-classe de **StatelessWidget** que retorna o **widget** sobrescrevendo o método **build**

## Classe para Criar Widget

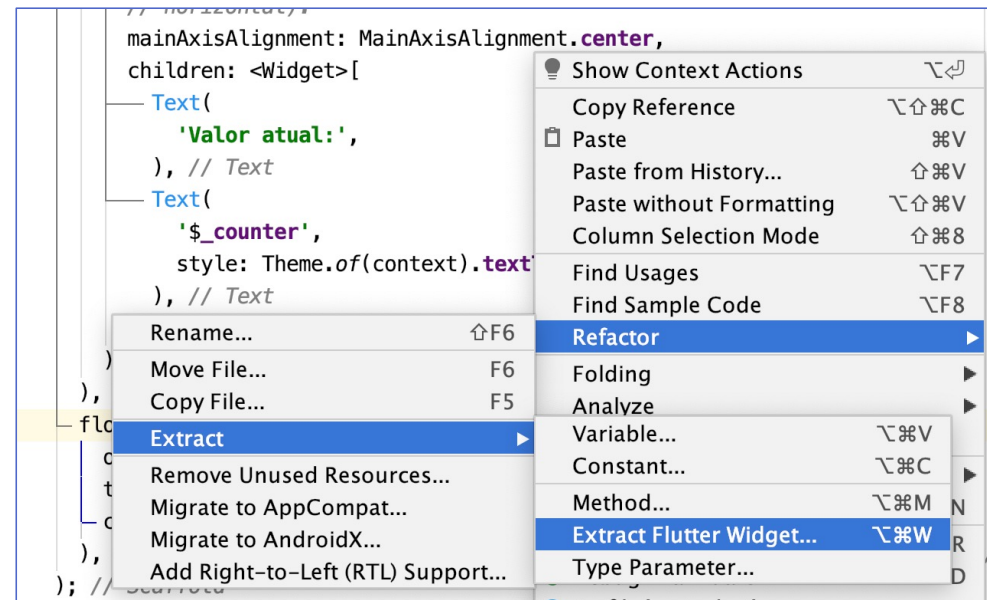
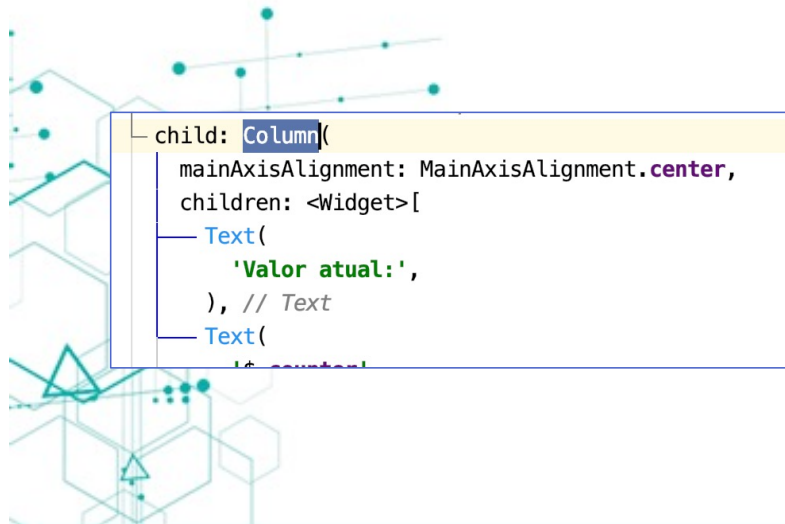
---

Seguir os passos:

- Criar uma subclasse de **StatelessWidget**
- Criar um construtor como **const**
- Implementar o método **build** (sobrescrito)
- Instanciar a classe também utilizando um **const**



## Atalho...



## Código Gerado

```
class ColunaWidget extends StatelessWidget {  
  const ColunaWidget({  
    Key key  
  }) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      mainAxisAlignment: MainAxisAlignment.center,  
      children: <Widget>[  
        Text( 'Boa' ),  
        Text( 'Noite!' )  
      ],  
    );  
  }  
}
```



```
return Scaffold(  
  body: Center(  
    child: ColunaWidget(),  
  )  
);
```

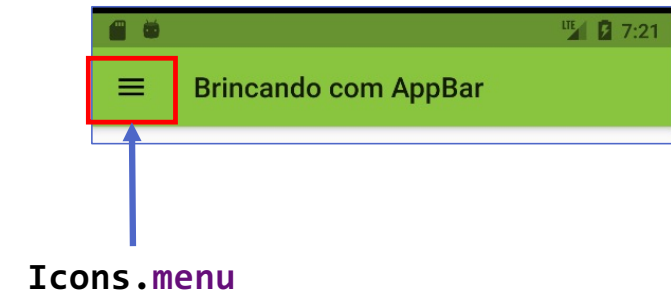
## AppBar Widget Leading

```
AppBar(
  leading: IconButton(
    icon: Icon(Icons.menu),
    onPressed: () {
      print('Menu pressionado!');
    },
  ),
  title: Text('Brincando com AppBar'),
)
```

**AppBar** é um **Widget** para representar a barra de ferramentas apresentada na parte superior do app

A propriedade **leading** define um ícone que pode ser utilizado para a abertura de um menu de opções, por exemplo

Dentro de **leading** o **onPressed** é o evento acionado quando o usuário pressiona o botão

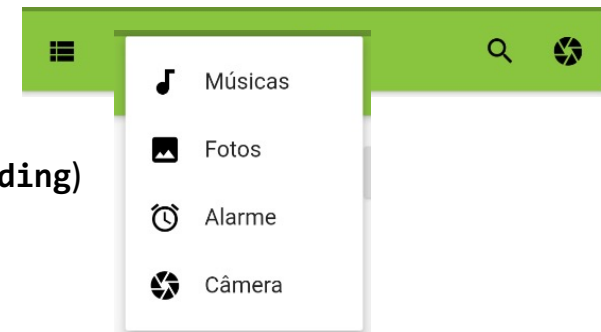


## Popup Menus

Apresentam opções para o usuário final na forma de um menu suspenso

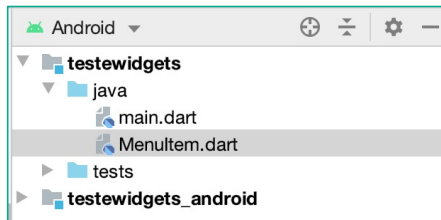
Para implementar, seguir os passos abaixo:

1. Criar uma classe (*model*) para conter a descrição e o ícone associado (pode conter mais informações)
2. Gerar a lista de itens que serão exibidos
3. Encapsular a lista em um objeto **PopupMenuButton**
4. Associar o **PopupMenuButton** a um item para a abertura (por exemplo no **leading**)



## Popup Menus – Classe Model

---



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

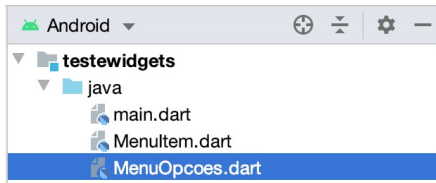
class MenuItem {

  final String title;
  final Icon icon;

  MenuItem({this.title, this.icon});

}
```

## Popup Menus – Gerar Lista Opções



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

```
class MenuOpcoes extends StatelessWidget {  
  const MenuOpcoes({Key key}) : super(key: key);
```

```
  @override
```

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

```
    List<MenuItem> itensMenu = [  
      MenuItem(title: 'Músicas', icon: Icon(Icons.audiotrack)),  
      MenuItem(title: 'Fotos', icon: Icon(Icons.photo)),  
      MenuItem(title: 'Alarme', icon: Icon(Icons.alarm)),  
      MenuItem(title: 'Câmera', icon: Icon(Icons.camera)),  
    ];
```

```
  }  
}
```

## Popup Menus – Encapsular PopupMenu

```
return PopupMenuButton<MenuItem>(
  icon: Icon(Icons.view_list),
  onPressed: ((item) { print('selecionado: ${item.title}'); }),
  itemBuilder: (BuildContext context) {
    return itensMenu.map((MenuItem menuItem) {
      return PopupMenuItem<MenuItem>(
        value: menuItem,
        child: Row(//define o 'desenho' para cada um dos itens da lista
          children: <Widget>[
            Icon(todoMenuItem.icon.icon),
            Padding(padding: EdgeInsets.all(8.0)),
            Text(todoMenuItem.title),
          ],
        ),
      );
    }).toList();
  }
);
```

## Popup Menus – Associar Exibição

---

```
appBar: AppBar(  
  actions: <Widget> [  
    ...  
  ],  
  leading: MenuOpcoes()  
)
```

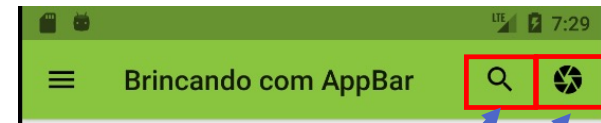
Neste caso o menu (**MenuOpcoes**) foi associado ao **leading** do **appBar**



## AppBar Widget Actions

```
actions: <Widget> [  
  IconButton(  
    icon: Icon(Icons.search),  
    onPressed: () {  
      print('Busca pressionado!');  
    },  
  ),  
  IconButton(  
    icon: Icon(Icons.camera),  
    onPressed: () {  
      print('Camera pressionado!');  
    },  
  ),  
],
```

De modo similar, as **Actions** permitem definir botões no lado oposto ao **leading** para serem utilizados como atalho para funções mais comuns dentro do app



Icons.search

Icons.camera

## AppBar Widget Flexible Space / Safe Area

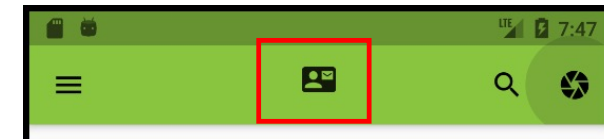
```
appBar: AppBar(  
  flexibleSpace: IconButton(  
    icon: Icon(Icons.contact_mail),  
    onPressed: () {  
      print('Contato pressionado!');  
    },  
  ),  
  
  flexibleSpace: SafeArea(child: IconButton(  
    icon: Icon(Icons.contact_mail),  
    onPressed: () {  
      print('Busca pressionado!');  
    },  
  )),  
)
```

Finalmente, o **flexibleSpace** permite incluir um ícone ao centro

Envolvido pelo **SafeArea** possibilita incluir uma margem envolvendo o **widget** (como no segundo exemplo acima)



**Icons.contact\_mail**



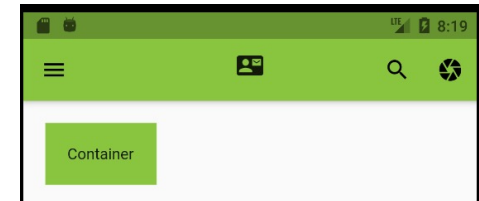
**SafeArea**

## Containers Propriedades

```
Container(  
  height: 100.0,  
  width: 100.0,  
  color: Colors.lightGreen,  
  child: Text('Container')  
)
```

```
Container(  
  padding: EdgeInsets.all(20.0),  
  margin: EdgeInsets.all(20.0),  
  color: Colors.lightGreen,  
  child: Text('Container')  
)
```

Container define uma área delimitada que pode conter outros **widgets**



Algumas propriedades:

- **height** e **width** → altura e largura
- **color** → cor de fundo
- **padding** → espaço interno (**all** ou **only**)
- **margin** → espaço externo (margem)
- **alignment** → alinhamento interno (vide **Alignment**)
- **constraints** → permite definir valores mínimos e máximos do **Container** (vide **BoxConstraints.expand()**)

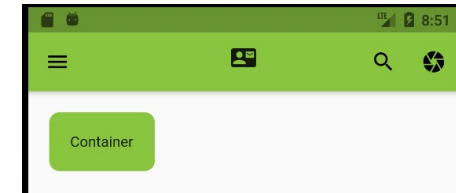
## Decorators - ShapeDecoration

**Decorators** podem ser utilizados para aplicar mais configurações visuais a um **widget**

Existem basicamente dois tipos: **ShapeDecorator** (mais genérico) e **BoxDecorator** (específico para retângulos)

Um **Decorator** pode ser aplicado a um **Container** por meio das propriedades **decoration** (fundo) e **foregroundDecoration** (frente)

```
Container(  
  padding: EdgeInsets.all(20.0),  
  margin: EdgeInsets.all(20.0),  
  child: Text('Container'),  
  decoration: ShapeDecoration(  
    color: Colors.lightGreen,  
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.all(Radius.circular(10.0))))  
)
```



## Decorators - BoxDecoration

---

```
BoxDecoration(  
  color: const Color(0xff7c94b6),  
  image: const DecorationImage(  
    image: NetworkImage('https://flutter.github.io/assets-for-api-docs/assets/widgets/owl-2.jpg'),  
    fit: BoxFit.cover)  
  border: Border.all(  
    color: Colors.black,  
    width: 8),  
  borderRadius: BorderRadius.circular(12)  
)
```

No exemplo acima temos como exibir uma imagem dentro de um Container por meio da propriedade **image** do **BoxDecoration**

**NetworkImage** permite obter uma imagem diretamente da Internet por meio de uma URL

Outro ponto importante é a definição de bordas utilizando a propriedade **border**

## Text / RichText

---

Permitem definir textos somente para leitura

Além disso estilos podem ser definidos por meio da propriedade **style** e classe **TextStyle** (**fontSize**, **color**, **fontStyle**, **textAlign**, etc...)

**RichText** possibilita aplicar estilos a diversas partes do texto incluindo vários **TextSpan** na propriedade **children**

Cada **TextSpan** possui por sua vez a propriedade **text** onde o texto a ser exibido é definido e **style** cujo funcionamento é igual ao discutido anteriormente

```
RichText(text: TextSpan(text: 'Boa', style: TextStyle(color: Colors.red),  
children: <TextSpan>[TextSpan(text: ' Noite', style: TextStyle(color: Colors.blue))]  
)),
```

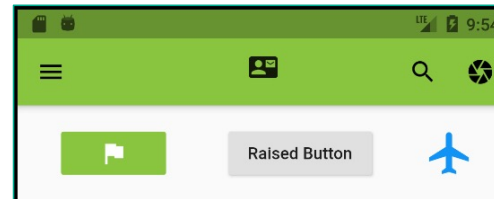
## Botões

Existem alguns tipos de *widget* para botões por exemplo **FlatButton**, **RaisedButton** e **IconButton**

```
FlatButton(  
  onPressed: () { print ('ok'); },  
  child: Icon(Icons.flag),  
  color: Colors.lightGreen,  
  textColor: Colors.white  
))
```

```
IconButton(  
  onPressed: () {},  
  icon: Icon(Icons.flight),  
  iconSize: 42.0,  
  color: Colors.blue  
)
```

```
RaisedButton(  
  onPressed: () {},  
  child: Text('Raised Button')  
)
```



## Radio

```
class _RadioButtonState extends State<RadioButton> {  
  int selecionado = -1;  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Row(children: <Widget>[  
        Radio( value: 0, groupValue: selecionado,  
          onChanged: (int val) {  
            setState(() {  
              selecionado = val;  
            });  
          }  
        ),  
        Text('Banana')  
      ])  
    );  
  }  
}
```

Radio podem ser utilizado para apresentar uma lista de opções na qual apenas uma pode ser selecionada por vez

**value** → valor individual do Radio

**groupValue** → valor selecionado entre as possibilidades

**onChanged** → quando um valor é selecionado



## Checkbox

---

bool **chk** = **true**;

Permitem a seleção individual de valores (**true** / **false**)

```
Checkbox(  
  value: chk,  
  onChanged: (bool val) {  
    print ('seleccionado: $val');  
    setState(() {  
      chk = val;  
    });  
  }  
)
```

## Alertas

---

```
showDialog(context: context,  
child: AlertDialog(  
  title: Text('Alerta'),  
  actions: <Widget>[  
    new FlatButton(onPressed: () => Navigator.pop(context), child: new Text('Ok'))  
  ]  
));
```

Alerta

Ok

## Formulários

Representam uma forma conveniente para permitir a entrada de dados textuais pelo usuário com a possibilidade de validação dos dados

Estrutura básica:

```
class LoginForm extends StatefulWidget {  
  @override  
  State<StatefulWidget> createState() {  
    return _LoginFormState();  
  }  
}
```

```
class _LoginFormState extends State<LoginForm> {  
  final _formKey = GlobalKey<FormState>();  
  
  @override  
  Widget build(BuildContext context) {  
  
    return Form(  
      key: _formKey,  
      child: Column(  
        children: <Widget>[  
          // Aqui vão os itens do form  
        ],  
      ),  
    );  
  }  
}
```

## Incluindo os Itens

Incluir dois **TextFormFields** para username / senha

Observar que a propriedade **validator** permite incluir uma regra para validar o dado que foi informado no campo específico

No caso, os campos não podem ser vazios (**isEmpty**)

**obscureText** pode ser utilizado em campos do tipo senha para ocultar os dados inseridos

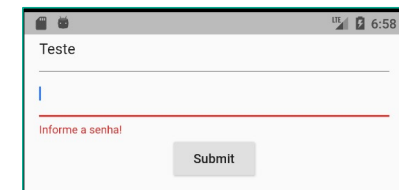
```
TextFormField(  
  validator: (value) {  
    if (value.isEmpty) {  
      return 'Informe o username!';  
    }  
    return null;  
  },  
  TextFormField(  
    obscureText: true,  
    validator: (value) {  
      if (value.isEmpty) {  
        return 'Informe a senha!';  
      }  
      return null;  
    }  
  )  
)
```

## Criando o Botão para Enviar / Validar Dados

Finalmente, um botão **RaisedButton** é adicionado ao formulário para enviar os dados

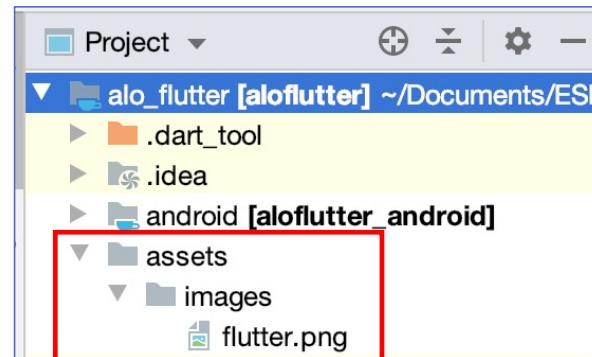
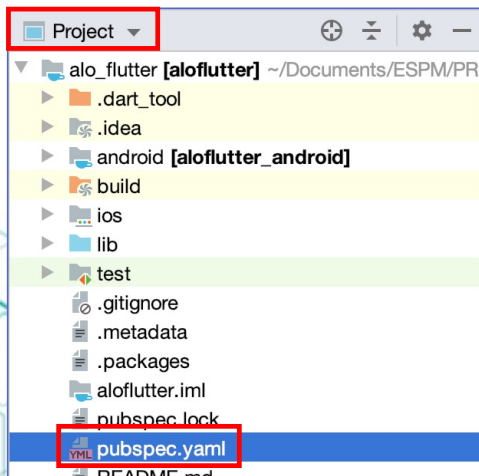
Os dados serão validados (**validate()**) de acordo com as regras definidas no slide anterior (campos vazios)

```
RaisedButton(  
  onPressed: () {  
    if (_formKey.currentState.validate()) {  
  
      // Exibe mensagem  
      Scaffold.of(context)  
        .showSnackBar(SnackBar(content: Text('Enviado...')));  
    }  
  },  
  child: Text('Submit')  
)
```

A screenshot of a mobile application interface. At the top, the title "Teste" is displayed. Below the title is a text input field. Underneath the input field, there is a red error message that reads "Informe a senha!". At the bottom of the form, there is a grey button labeled "Submit". The status bar at the very top shows the time as 6:58.

## Trabalhando com Imagens

Recursos utilizados pela aplicação como imagens e ícones devem ser declarados dentro do arquivo **pubspec.yaml**, na visão **Project** conforme abaixo:



**assets:**

- assets/images/flutter.png

## Referenciando Imagens

---

```
return MaterialApp (  
  home: Scaffold(  
    body: Row(  
      children: <Widget>[  
        Image(  
          image: AssetImage('assets/images/flutter.png')  
        )  
      ]  
    )  
  )  
);
```

**Image.network** também pode ser utilizado para obter uma imagem diretamente da Internet, passando como parâmetro a *URL* da imagem

## Navegação

---

Navegação permite que uma tela acione outra para exibir funcionalidades específicas, por exemplo

Por exemplo, uma tela Abertura (com uma mensagem de boas vindas) aciona uma segunda tela Fechamento, que pode ser fechada, retornando para Abertura

```
return Scaffold (  
  floatingActionButton: FloatingActionButton(  
    child: Icon(Icons.skip_next),  
    onPressed: () => {  
      Navigator.push(context, MaterialPageRoute(  
        builder: (context) => Fechamento()  
      ))  
    }  
  )  
);
```



## Fechando e Retornando

---

```
return Scaffold (  
  floatingActionButton: FloatingActionButton(  
    child: Icon(Icons.skip_previous),  
    onPressed: () => {  
      Navigator.pop(context)  
    }  
  )  
);
```

## Named Routes

---

É possível também nomear as rotas e referenciá-las pelo nome ao invés de utilizar a classe como referência

Uma rota inicial (**initialRoute**) é definida e também as demais (**routes**)

Observar que o atributo **home** deixa de ter sentido pois já definimos a rota principal

```
return MaterialApp(  
  initialRoute: '/',  
  routes: {  
    '/' : (context) => Abertura(),  
    '/fechamento' : (context) => Fechamento()  
  }  
);
```

## Usando Nome da Rota

---

Para utilizar uma rota por meio do nome basta utilizar o método **pushNamed** passando como parâmetro o nome da rota

```
floatingActionButton: FloatingActionButton(  
  child: Icon(Icons.skip_next),  
  onPressed: () => {  
    Navigator.pushNamed(context, '/fechamento')  
  }  
)
```

## Passando Parâmetros

---

É possível efetuar a passagem de parâmetros (valores) entre as rotas

Para isso, seguir os passos:

1. Criar uma classe para conter os parâmetros desejados
2. Passar um objeto da classe com os valores definidos
3. Ler o objeto e extrair os parâmetros no destino



## Criar Classe Parâmetros

---

```
class Mensagem {  
    String conteudo;  
    Mensagem (this.conteudo);  
}
```

## Passando Parâmetros

---

Parâmetros são passados instanciando a classe criada no passo 1 e atribuindo ao parâmetro **arguments**

```
floatingActionButton: FloatingActionButton(  
  child: Icon(Icons.skip_next),  
  onPressed: () => {  
    Navigator.pushNamed(context, '/fechamento', arguments: Mensagem("Ola"))  
  }  
)
```

## Recebendo Parâmetros

---

```
Widget build(BuildContext context) {  
  
  final Mensagem msg = ModalRoute.of(context).settings.arguments;  
  
  return Scaffold (  
    body: Center(child: Text('${msg.conteudo}')),  
    floatingActionButton: FloatingActionButton(  
      child: Icon(Icons.skip_previous),  
      onPressed: () => {  
        Navigator.pop(context)  
      }  
    )  
  );  
}
```

Os parâmetros são recebidos no **context** e podem ser extraídos conforme exemplo acima

## Card

---

Permite o agrupamento de elementos para criar algo maior

```
Card(  
  shape: OutlineInputBorder(borderSide: BorderSide(color: Colors.deepOrange.withOpacity(0.5))),  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.min,  
    children: <Widget>[  
      const ListTile(  
        leading: Icon(Icons.album),  
        title: Text('Desenvolvimento Mobile'),  
        subtitle: Text('Futter')  
      )  
    ]  
  )  
)
```



## ListView

---

Representa uma lista de valores com barra de rolagem

```
Scaffold(  
  body: SafeArea(child: ListView(children: <Widget>[  
    ListTile(leading: Icon(Icons.autorenew), title: Text('Item 1')),  
    ListTile(leading: Icon(Icons.map), title: Text('Item 2')),  
    ListTile(leading: Icon(Icons.search), title: Text('Item 3'))  
  ]))  
)
```

## ListView Builder

---

Uma segunda forma de instanciar o **ListView** permite maior controle sobre os *widgets* exibidos, utilizando para isso o método **builder**:

```
final List<String> itens = <String>['Item 1', 'Item 2', 'Item 3'];  
final List<int> cores = <int>[600, 500, 100];
```

```
ListView.builder(  
  padding: const EdgeInsets.all(8),  
  itemCount: itens.length,  
  itemBuilder: (BuildContext context, int index) {  
    return Container(  
      height: 50,  
      color: Colors.amber[cores[index]],  
      child: Center(child: Text('${itens[index]}')),  
    );  
  }  
)
```

## ListView Evento onTap

---

onTap permite capturar o evento relacionado à escolha de um item da lista

```
ListView.builder(  
  padding: const EdgeInsets.all(8),  
  itemCount: itens.length,  
  itemBuilder: (BuildContext context, int index) {  
    return ListTile(leading: Icon(Icons.autorenew), title: Text('${itens[index]}'),  
      onTap: () {  
        print(index);  
      });  
  }  
)
```

## GridView

---

Organiza a visualização em células (linhas x colunas)

```
GridView.count(crossAxisCount: 2,  
children: List.generate(10, (index) {  
  return Center(  
    child: Text('Item $index')  
  );  
} )  
)
```

## GridView x Card

---

O **GridView** pode ser combinado com um **Card** para compor as células com ícones, textos, botões, etc...

```
Card(  
  shape: OutlineInputBorder(borderSide: BorderSide(color: Colors.redAccent)),  
  child: Column(crossAxisAlignment: CrossAxisAlignment.stretch,  
    mainAxisAlignment: MainAxisAlignment.center,  
    children: <Widget>[  
      Icon(Icons.directions_car),  
      Text(this.texto, textAlign: TextAlign.center)  
    ]  
  )  
)
```

## Stack

---

**Stack** permite sobrepor *widgets* para criar um *widget* novo

```
Stack(  
  children: <Widget>[  
    Container(  
      color: Colors.green,  
    ), Container(  
      color: Colors.blue,  
      height: 300.0,  
      width: 300.0,  
    ), Container(  
      color: Colors.pink,  
      height: 150.0,  
      width: 150.0,  
    )  
  ],  
);
```

## Alinhamento x Posicionamento

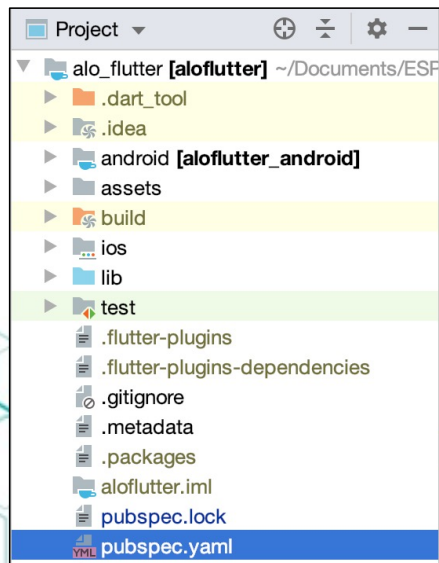
**Align** possibilita posicionar os elementos sobrepostos em diferentes pontos na composição do *layout*

Já **Positioned** permite um controle maior sobre o posicionamento por meio da distância (**top**, **right**, etc...)

```
Stack(  
  children: <Widget>[  
    Container(  
      color: Colors.green  
    ),  
    Align(child: Container(  
      color: Colors.red,  
      width: 200,  
      height: 200  
    ),  
    alignment: Alignment.topRight)  
  ],  
);
```

```
Positioned(child: Container(  
  color: Colors.red,  
  width: 200,  
  height: 200  
),  
top: 100,  
right: 100)
```

## Persistência Adicionando Dependências



```
dependencies:  
  flutter:  
    sdk: flutter  
  
  sqflite:  
    path:
```



## Criar Classe de Dados (Model)

---

```
class Disciplina {  
  
    int id;  
    String nome;  
    String curso;  
  
    Disciplina({this.id, this.nome, this.curso});  
  
    Map<String, dynamic> toMap() {  
        return {'id': id, 'nome': nome, 'curso': curso};  
    }  
}
```

## Criar Banco de Dados

---

```
Future<Database> abrir() async {  
  return await openDatabase("DisciplinasDB",  
    onCreate: (db, version){  
      db.execute("CREATE TABLE TAB_DISCIPLINA (ID INTEGER PRIMARY KEY, NOME TEXT, CURSO TEXT)");  
    },  
    version: 1);  
}
```

## Inserir Registros

---

```
Future<int> inserir(Database db, Disciplina disciplina) async {  
    var id = await db.insert('TAB_DISCIPLINA', disciplina.toMap());  
    print('Inserida ${disciplina} - ${id}');  
    return id;  
}
```



## Listar Registros

---

```
Future<List<Map<String, dynamic>>> listar(Database db) async {  
    return db.query('TAB_DISCIPLINA');  
}
```

```
_MeuDb().listar(db).then((value) {  
    value.forEach((element) {  
        print(element);  
    });  
});
```

## JSON Flutter

Um outro pacote, o `convert ('dart:convert')` permite interpretar a sintaxe *JSON* com a função `json.decode`

```
var jsonMap = '{"id":"123"}'; //Objeto JSON simples
Map<String, dynamic> mapa = json.decode(jsonMap);
print(json.decode(mapa['id']));
```

```
var jsonList = ' [{"id":"123"}, {"id":"456"} ]'; //Objeto JSON Array
List<dynamic> lista = json.decode(jsonList);
lista.forEach((element) {
  print(element['id']);
});
```

```
// Array com Array
var jsonComplexo = ' [{"id":"123", "itens":["A","B"]}, {"id":"456", "itens":["C","D"]} ]';
List<dynamic> listaComplexo = json.decode(jsonComplexo);
listaComplexo.forEach((element) {
  print(element['itens'][0]);
});
```

## Requisições HTTP

---

O pacote `http ('package:http/http.dart')` oferece uma grande facilidade na execução de requisições *HTTP* (*GET*, *POST*, etc...)

Tais requisições podem ser utilizadas para interagir com serviços publicados na Internet (nuvem) utilizando o padrão RESTFul:

```
final URL_1 = 'http://fipeapi.appspot.com/api/1/carros/marcas.json';  
  
Future<Response> executarGet() async {  
  var resp = await get(URL_1); //Objeto Response  
  print('${resp.body}'); //resp.body corpo da resposta da requisição  
}
```

## Requisições POST

---

```
var response = await http.post('https://iam.cloud.ibm.com/identity/token',  
headers: <String, String> {'Content-Type' : 'application/x-www-form-urlencoded',  
'Accept' : 'application/json'},  
body: 'grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey=API_KEY');  
  
if (response.statusCode == 200) {  
    print (response.body);  
} else {  
    throw Exception('Erro ao obter token!');  
}
```