Componentes de Interface

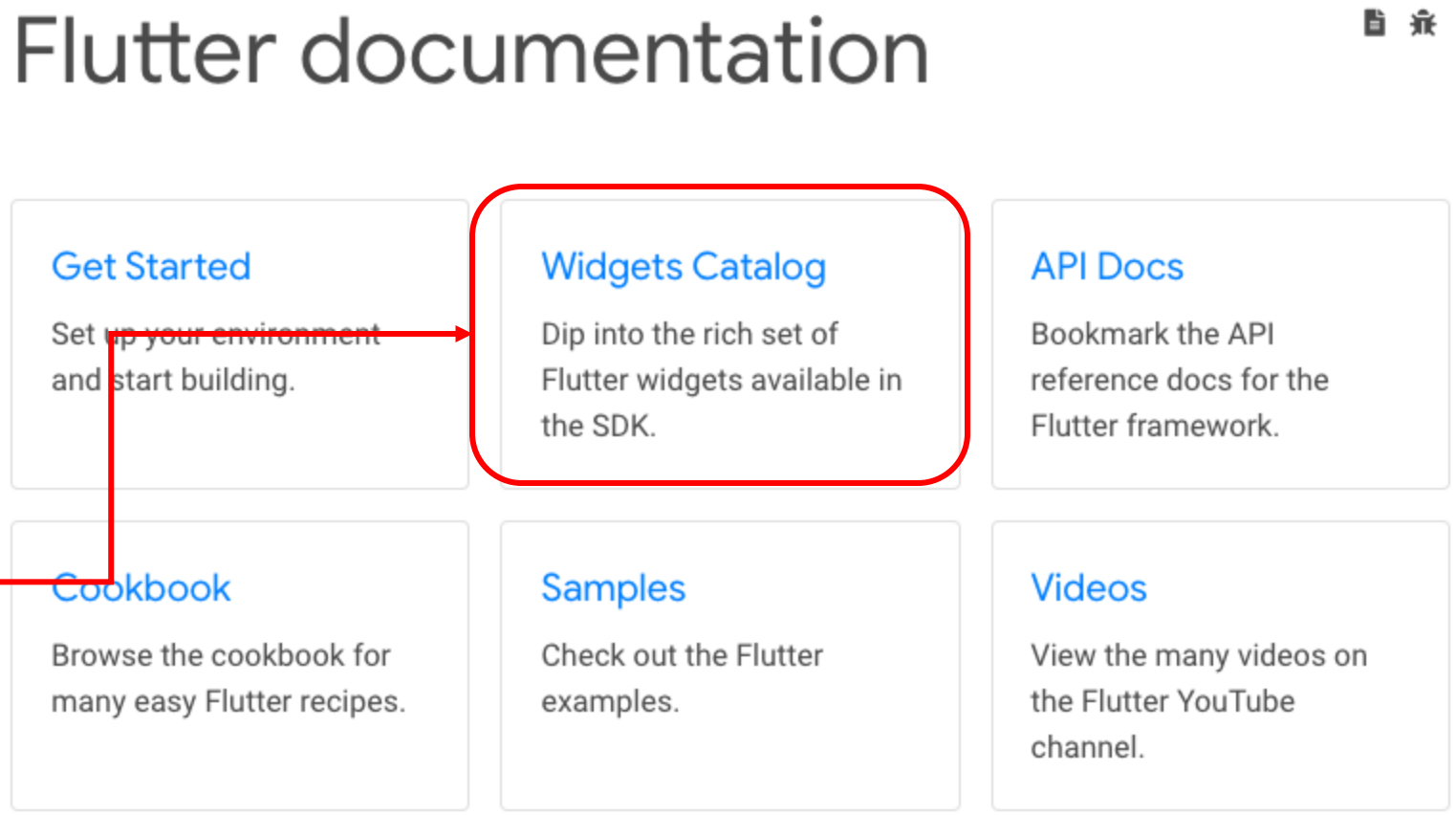
Parte 1

Profa. Ivre Marjorie
(ivre@pucminas.br)

Introdução

- Widgets: Componentes usados para montar o APP
- No Flutter tudo é um widget.
- Nessa aula vamos aprender alguns componentes, além de organizar melhor nosso código.

Catálogo de Widgets

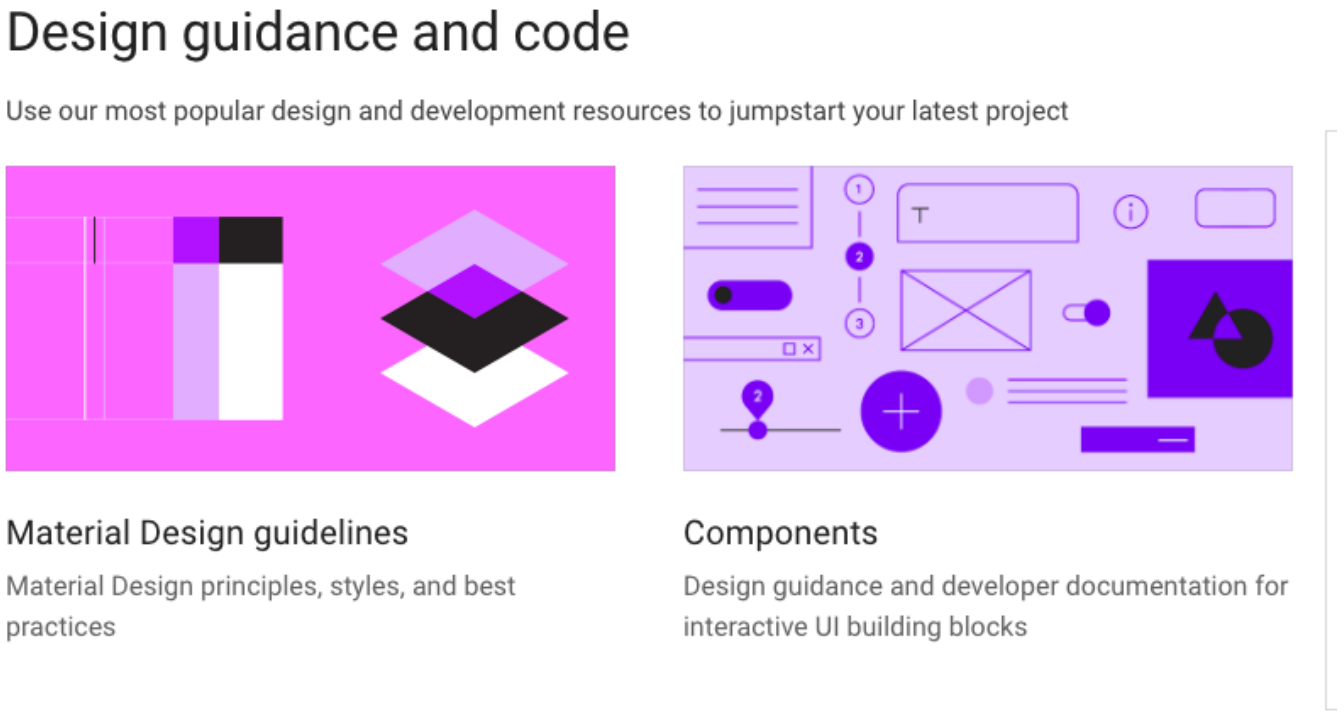


Catálogo de Widgets

Fonte: https://flutter.dev/docs

Material Design

- Criada em 2014 pela Google e muito utilizada
- É uma “linguagem de design” que propõem boas práticas para o desenho das telas dos aplicativos
- Disponível em : https://material.io/



MaterialApp ( )

- O Flutter fornece vários widgets que ajudam a construir aplicativos que segue o Material Design.
- MaterialApp é um widget de conveniência que envolve vários outros widgets que normalmente são necessários para definir o design do App
- Inclui uma série de widgets na sua raiz, incluindo o Navigator, que gerencia uma pilha de widgets identificados por strings, também conhecidos com “rotas”

MaterialApp ( )

Exemplo:

```
void main() {
  runApp(MaterialApp(
    title: 'Teste Material App',
    theme: ThemeData(
      primaryColor: Colors.blue,
    ), // ThemeData
    home: Container(
      color: Colors.white,
    ), // Container
  )); // MaterialApp
}
```

title: título da aplicação

theme: tema utilizado, possui outros atributos a serem configurados, no exemplo, usamos primaryColor, e a classe Colors que especifica internamente as cores do Material Desing

home: a primeira tela do App deve ser especificada dentro desse atributo. Telas flutuantes são distribuídas em rotas. Uma "rota" no Flutter é apenas uma única tela.

MaterialApp ( )

Ctrl + P ou Command + P para ver quais são os parâmetros, por exemplo do MaterialApp()

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
```

{**Key key**, navigatorKey, home, routes: const <String, WidgetBuilder>{}, initialRoute, onGenerateRoute, onGenerateInitialRoutes, onUnknownRoute, navigatorObservers: const <NavigatorObserver>[], builder, title: '', onGenerateTitle, color, theme, darkTheme, themeMode: ThemeMode.system, locale, localizationsDelegates, localeListResolutionCallback, localeResolutionCallback, supportedLocales: const <Locale>[Locale('en', 'US')], debugShowMaterialGrid: false, showPerformanceOverlay: false, checkerboardRasterCacheImages: false, checkerboardOffscreenLayers: false, showSemanticsDebugger: false, debugShowCheckedModeBanner: true, shortcuts, actions}

MaterialApp ( )

- Usar o widget MaterialApp é totalmente opcional, mas uma boa prática, sendo assim, vamos iniciar o exemplo com ele
- Veja o exemplo ao lado

Exemplo:

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

void main() {
  runApp(MaterialApp(
    title: 'Flutter Tutorial',
    home: TutorialHome(),
  ));
}
```

Fonte: https://flutter.dev/docs/development/ui/widgets-intro

Definindo um StatelessWidget

- O StatelessWidget é um widget sem estado, sendo assim, não sofrerão alterações após serem renderizados
- São úteis quando a parte da interface do usuário que está sendo descrita não depende de nada além da informação de configuração no próprio objeto e do Contexto BuildContext no qual o widget é criado

Exemplo:

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

Scaffold

- Widget é “esqueleto” para montar o APP
- Permite adicionar outros widgets do Material Design dentro dele
- MaterialApp() é definido apenas uma vez no APP inteiro, mas o Scaffold será definido em cada tela que for feita

Exemplo:

```
return MaterialApp(
  debugShowCheckedModeBanner: false,
  theme: ThemeData(
    primaryColor: Colors.blue,
  ), // ThemeData
  home: Scaffold(
    appBar: AppBar(
      title: Text("Hello App"),
    ), // AppBar
    body: Container(
      color: Colors.white,
    ), // Container
  ), // Scaffold
); // MaterialApp
```

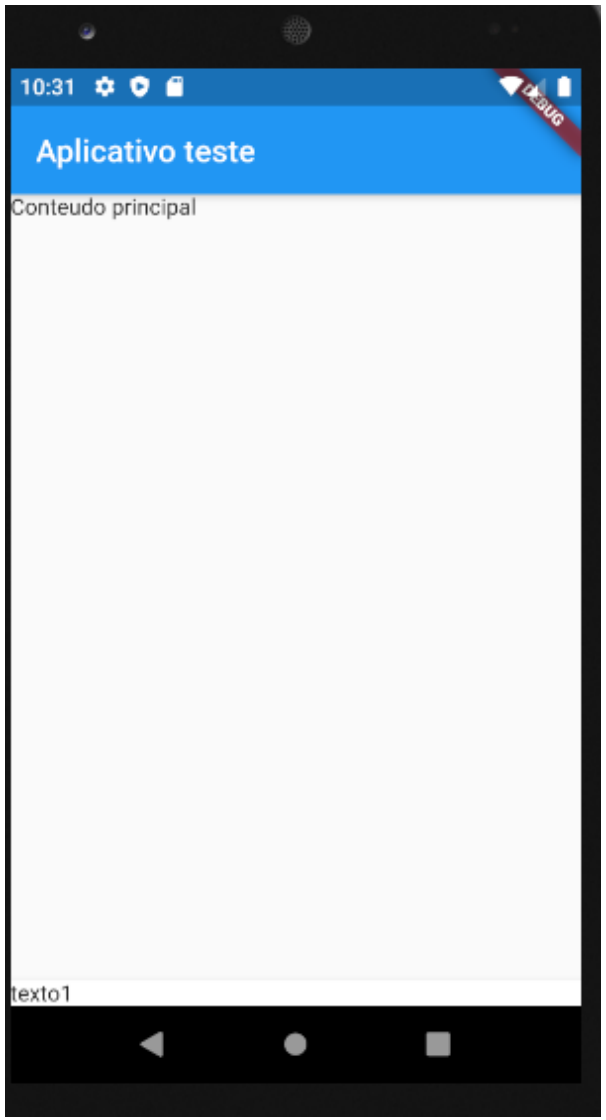
Scaffold

```dart
void main(){
  runApp(MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text("Aplicativo teste"),
      ), // AppBar
      body: Text("Conteudo principal"),
      bottomNavigationBar: BottomAppBar(
        child: Row(
          children: <Widget>[
            Text("texto1"),
          ], // <Widget>[]
        ), // Row
      ) // BottomAppBar
    ), // Scaffold
  )); // MaterialApp
}
```

appBar

body

bottomNavigation Bar



Scaffold - Parâmetros

{Key key, appBar, **body**, floatingActionButton, floatingActionButtonLocation, floatingActionButtonAnimator, persistentFooterButtons, drawer, endDrawer, bottomNavigationBar, bottomSheet, backgroundColor, resizeToAvoidBottomPadding, resizeToAvoidBottomInset, primary: true, drawerDragStartBehavior: DragStartBehavior.start, extendBody: false, extendBodyBehindAppBar: false, drawerScrimColor, drawerEdgeDragWidth, drawerEnableOpenDragGesture: true, endDrawerEnableOpenDragGesture: true}

Exemplo:

```
home: Scaffold(
  appBar: AppBar(
    title: Text("Hello App"),
  ), // AppBar
  body: Container(
    color: Colors.white,
  ), // Container
  drawer: Container(
    color: Colors.yellow,
  ), // Container
  floatingActionButton: FloatingActionButton(onPressed: (){}),
), // Scaffold
```

Alguns widgets que estão contidos no Scaffold: drawer, floatingActionButton

Scaffold - Parâmetros

- Podemos separar o código em pequenos componentes, como no exemplo, que criamos o home()
- Dessa forma, o resultado será o mesmo

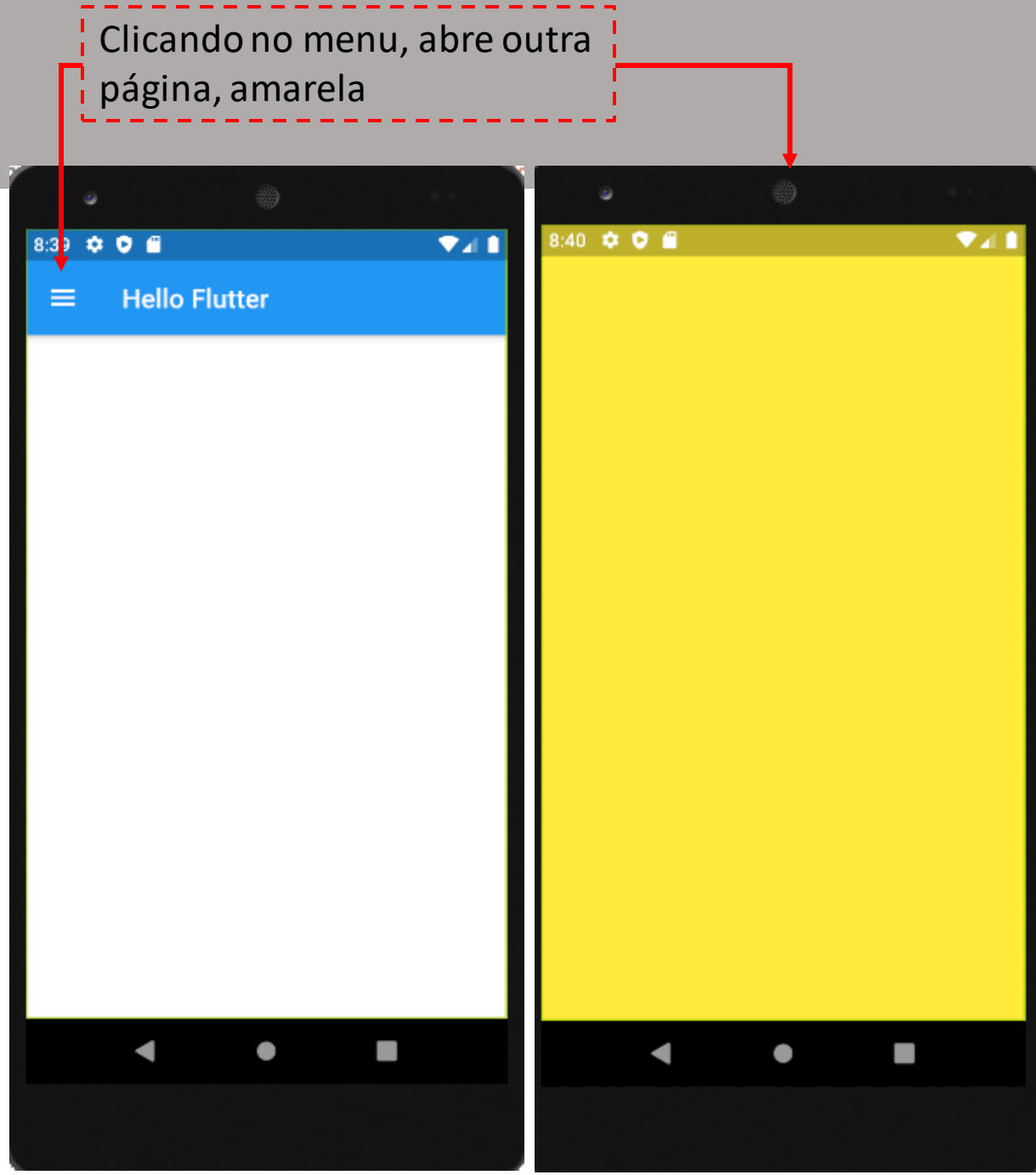
Exemplo:

```
        home: home(),
      ); // MaterialApp
    }
    //Separar o código em pequenos componentes
    home(){
      return Scaffold(
        appBar: AppBar(
          title: Text("Hello Flutter"),
        ),//propriedade // AppBar
        body: Container(
          color: Colors.white,
        ), // Container
        drawer: Container(
          color: Colors.yellow,
        ), // Container
      ); // Scaffold
    }
```

Scaffold - Parâmetros

Exemplo:

```
        home: home(),
    ); // MaterialApp
  }
  //Separar o código em pequenos componentes
  home(){
    return Scaffold(
      appBar: AppBar(
        title: Text("Hello Flutter"),
      ),//propriedade // AppBar
      body: Container(
        color: Colors.white,
      ), // Container
      drawer: Container(
        color: Colors.yellow,
      ), // Container
    ); // Scaffold
  }
```



Scaffold - Widget customizado

- É possível criar um widget customizado, para isso, vamos criar uma classe, por exemplo, a classe HomePage
- E nesse caso, vamos chamar HomePage como a seguir:

  home: HomePage(),
- Todo o widget possui um método **build** que retorna um Widget que é responsável pela tela
- Cada tela vai ter um Scafoold com seu AppBar

Scaffold - Exemplo

Apenas um MaterialApp para o APP

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: Colors.blue
      ), // ThemeData
      home: HomePage(),
    ); // MaterialApp
  }
}
```

Um Scaffold para cada tela

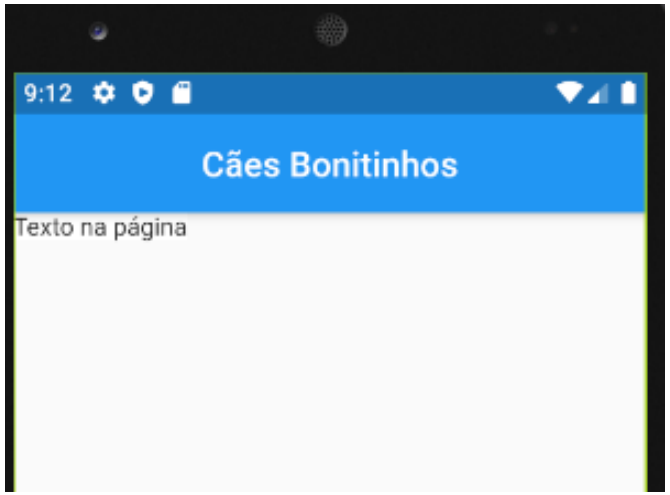
```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Cães Bonitinhos"),
        centerTitle: true,
      ),//propriedade // AppBar
      body: Container(
        color: Colors.white,
      ), // Container
    ); // Scaffold
  }
```

Text / TextStyle

- Widget que representa um texto no aplicativo
- No body criamos o layout em si da página
- Vamos criar uma propriedade filha dentro do Container e, em seguida, incluir um Text, para colocar um texto

Exemplo:

```
body: Container(
  color: Colors.white,
  child: Text("Texto na página"),
), // Container
```



Text / TextStyle

Exemplo:

```
body: Container(
  color: Colors.white,
  child: Center(
    child: Text("Texto na pagina no centro") ,
  ), // Center
), // Container
```

Widget dentro de Widget
Text dentro de Center dentro de Container

Text / TextStyle

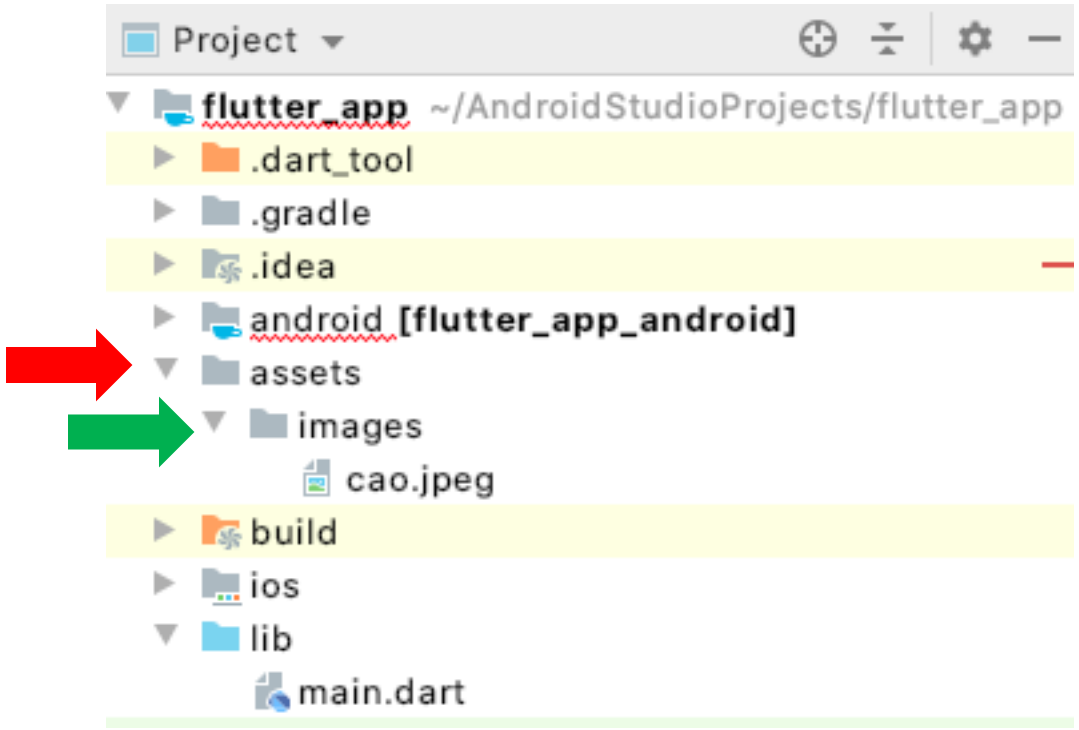
- **style:** propriedade que permite formatar um texto
- TextStyle possui diversas propriedades que permitem a formatação do texto, desde cor, tamanho, negrito, até mesmo sublinhado, entre outras
- Vale a pena brincar um pouco com essas propriedades

Exemplo:

```
body: Container(
  color: Colors.white,
  child: Center(
    child: Text("Texto na pagina",
      style: TextStyle(
        color: Colors.blue,
        fontSize: 30,
        fontWeight: FontWeight.bold,
        fontStyle: FontStyle.italic,
        decoration: TextDecoration.underline,
        decorationColor: Colors.red,
        decorationStyle: TextDecorationStyle.dashed,
      ), // TextStyle
      ), // Text
```

Image

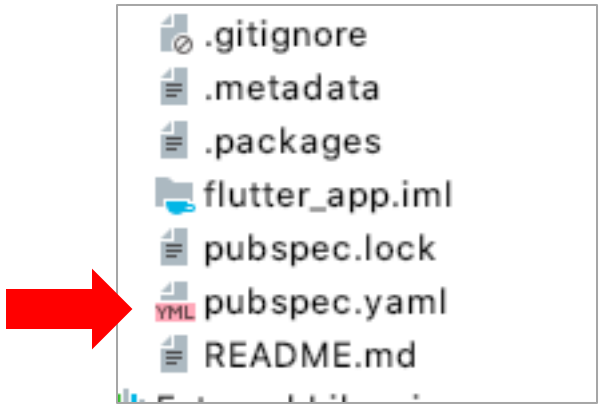
- Crie o diretório assets e o subdiretório imagens, para isso, clique com o botão direito no nome do projeto e vá em New -> Directory, em seguida, botão direito em assets, New-> Directory
- Arrastar as imagens para a pasta images



Image

- Em seguida, abra o arquivo pubspec.yaml e faça as seguintes alterações
- Tire o comentário # de frente de assets e altere o caminho para as imagens, caso tenha mais de uma pode deixar assim: assets/images/

Localizando o arquivo:



Arquivo pubspec.yaml

```
# To add assets to your application,
assets:
  - assets/images/
```

Image

- Para incluir a imagem no App, incluir o widget Image e o método auxiliar asset e indicar qual é a imagem que será incluída como no exemplo.

Exemplo:

```
_body() {
  return Container(
    color: Colors.white,
    child: Center(
      child: Image.asset('assets/images/cao.jpeg'),
    ), // Center
  ); // Container
}
```

Image

- É possível alterar o tamanho da imagem, com width e heigth
- Podemos usar também o fit, com o BoxFit cover ou Fill

Exemplo:

```
_img() {
  return Image.asset("assets/images/cao.jpeg",
    width: 350,
    height: 350,
    fit: BoxFit.cover,
  );
}
```

Image

- BoxFit.cover corta a imagem como no exemplo 1
- BoxFit.fill aumenta a imagem distorcendo, como no exemplo 2

Exemplo 1:

Exemplo 2:

Image

- É possível também incluir uma imagem a partir de uma URL, para isso, usaremos o método auxiliar network, como no exemplo, ao lado

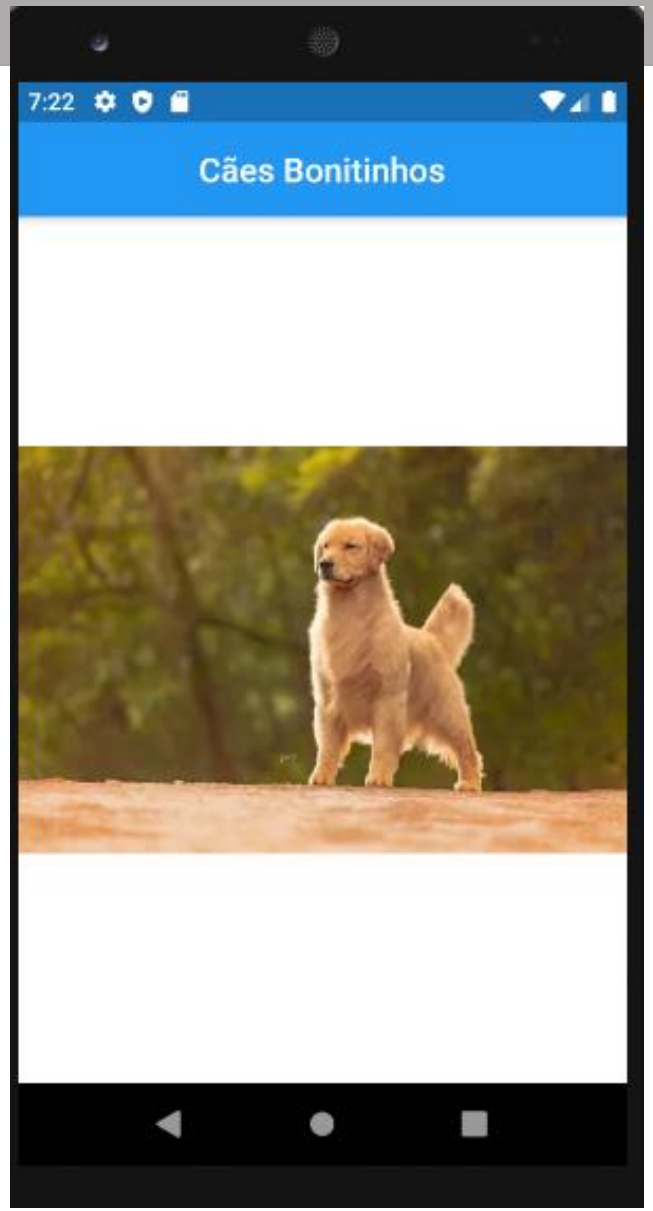
Exemplo:

```
_body() {
  return Container(
    color: Colors.white,
    child: Center(
      child: Image.network("https://static.wixstatic.com/media/
      ), // Image.network
    ), // Center
  ); // Container
}
```

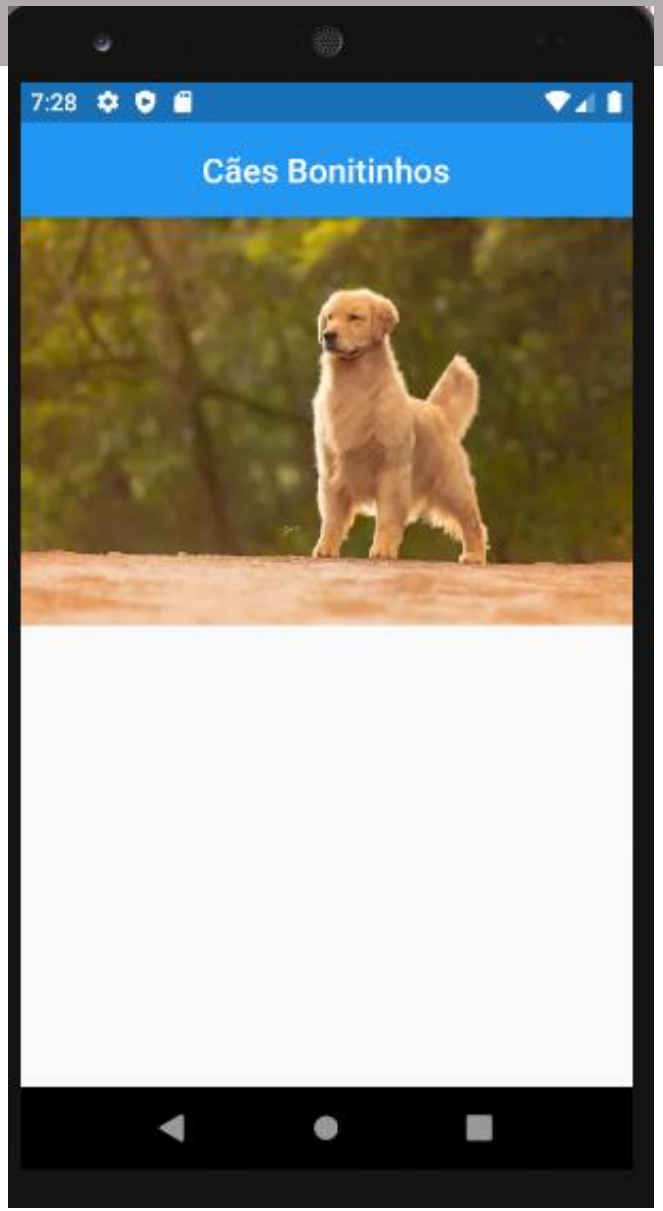
Image

- Para o exemplo, o widget **Image** foi incluído dentro do widget **Center**, por isso, a imagem do cachorro está no centro (exemplo1)
- Se retirar o widget **Center** ela ficará bem no início da tela (exemplo2)

Exemplo 1:



Exemplo 2:



SizeBox

- O widget SizeBox permite que seus filhos ocupem o tamanho inteiro da tela se for desejado
- SizeBox.expand expandi na tela inteira

Exemplo:

```
_body() {
  return Container(
    color: Colors.white,
    child: SizedBox.expand(
      child: _img(),
    ), // SizedBox.expand
  ); // Container
}
```

SizeBox

```
_body() {
  return Container(
    color: Colors.white,
    child: SizedBox.expand(
      child: _img(),
    ), // SizedBox.expand
  ); // Container
}
_img() {
  return Image.asset("assets/images/cao.jpeg",
    fit: BoxFit.fill,
  );
```

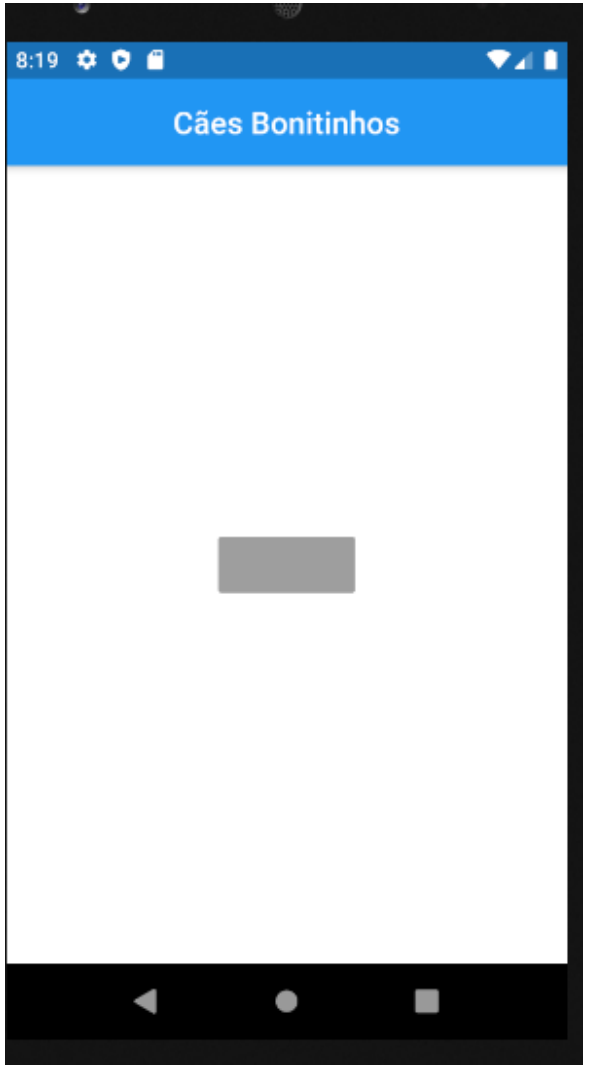
Use tanto o fill, quanto o cover, para ver a diferença

Button

- O widget RaisedButton, por padrão fica com a cor cinza, mas não trata evento e não faz nada
- Sendo assim, vamos começar incluindo a função onPressed()

Exemplo:

```
_body() {
  return Container(
    color: Colors.white,
    child: Center(
      child: _button(),
    ), // Center
  ); // Container
}
_button() {
  return RaisedButton(
  );
}
```

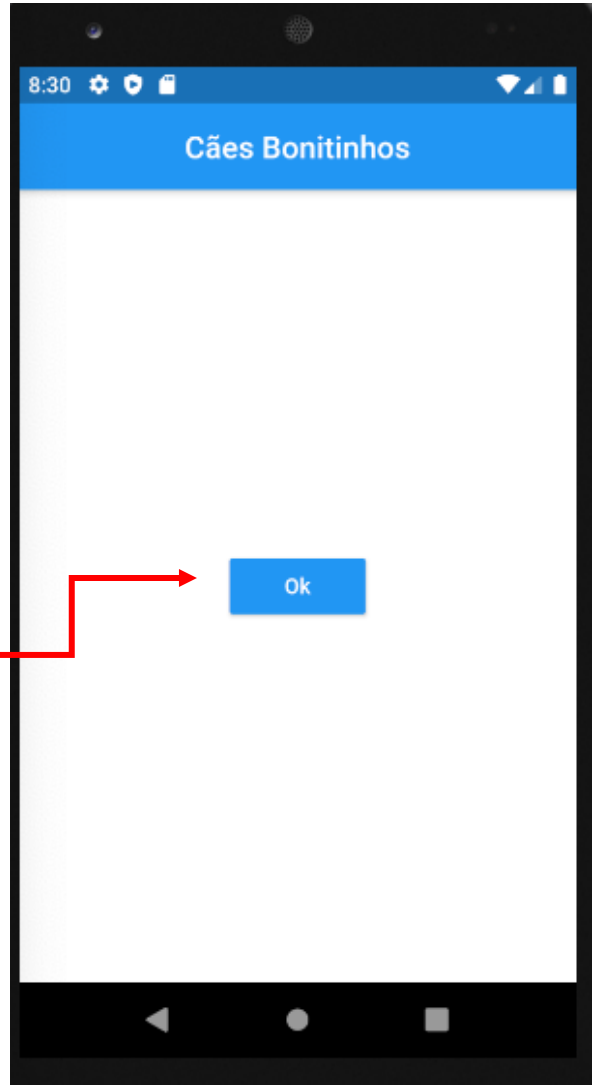


Button

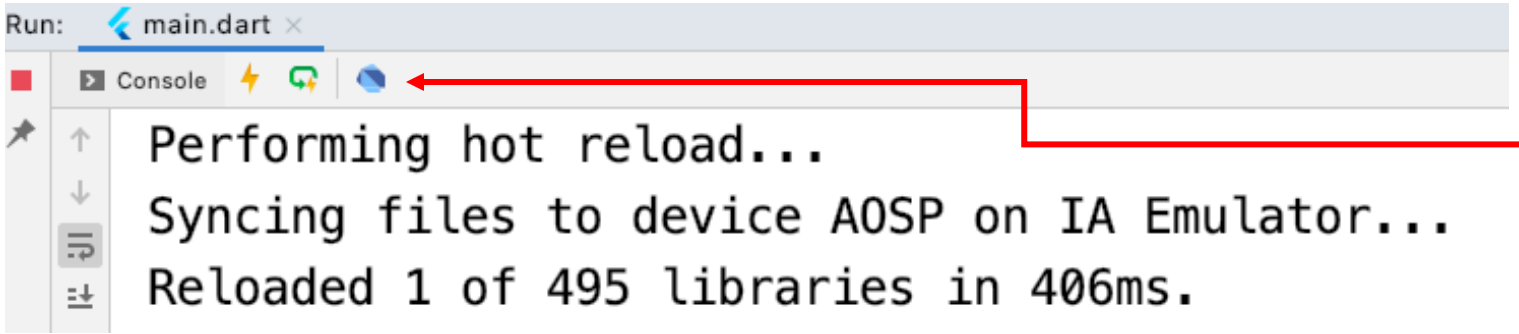
```
_button() {
  return RaisedButton(
    color: Colors.blue,
    child: Text(
      "Ok",
      style: TextStyle(
        color: Colors.white,
      ) , // TextStyle
    ), // Text
    onPressed: () => print("Clicou no botão ok!")
  ); // RaisedButton
}
```

```
Console
Performing hot reload...
Syncing files to device AOSP on IA Emulator...
Reloaded 1 of 495 libraries in 406ms.
I/flutter ( 5892): Clicou no botão ok!
```

Ao clicar no botão Ok, será apresentado no console o texto Clicou no botão ok

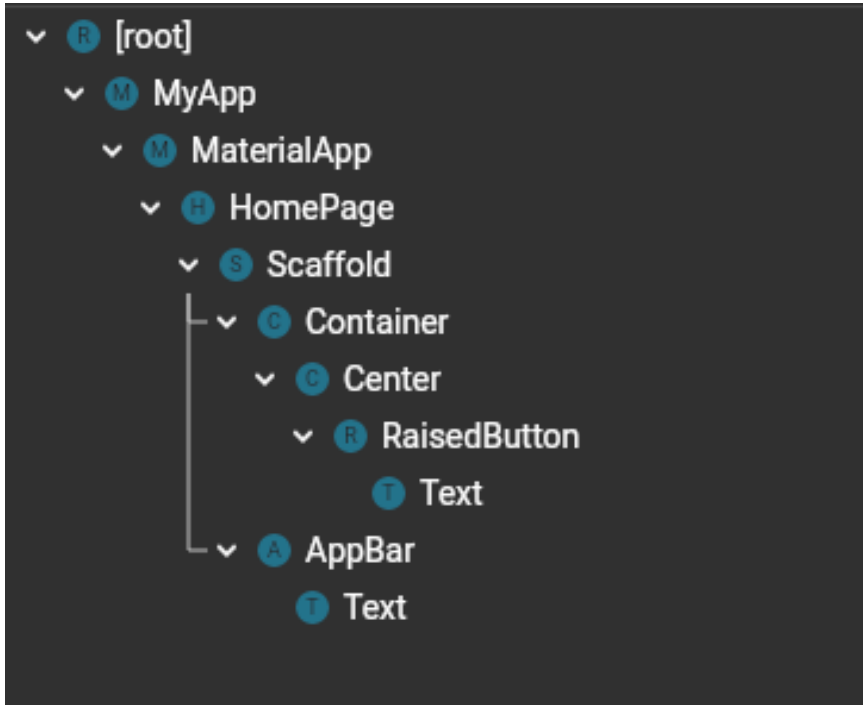


Dev Tools

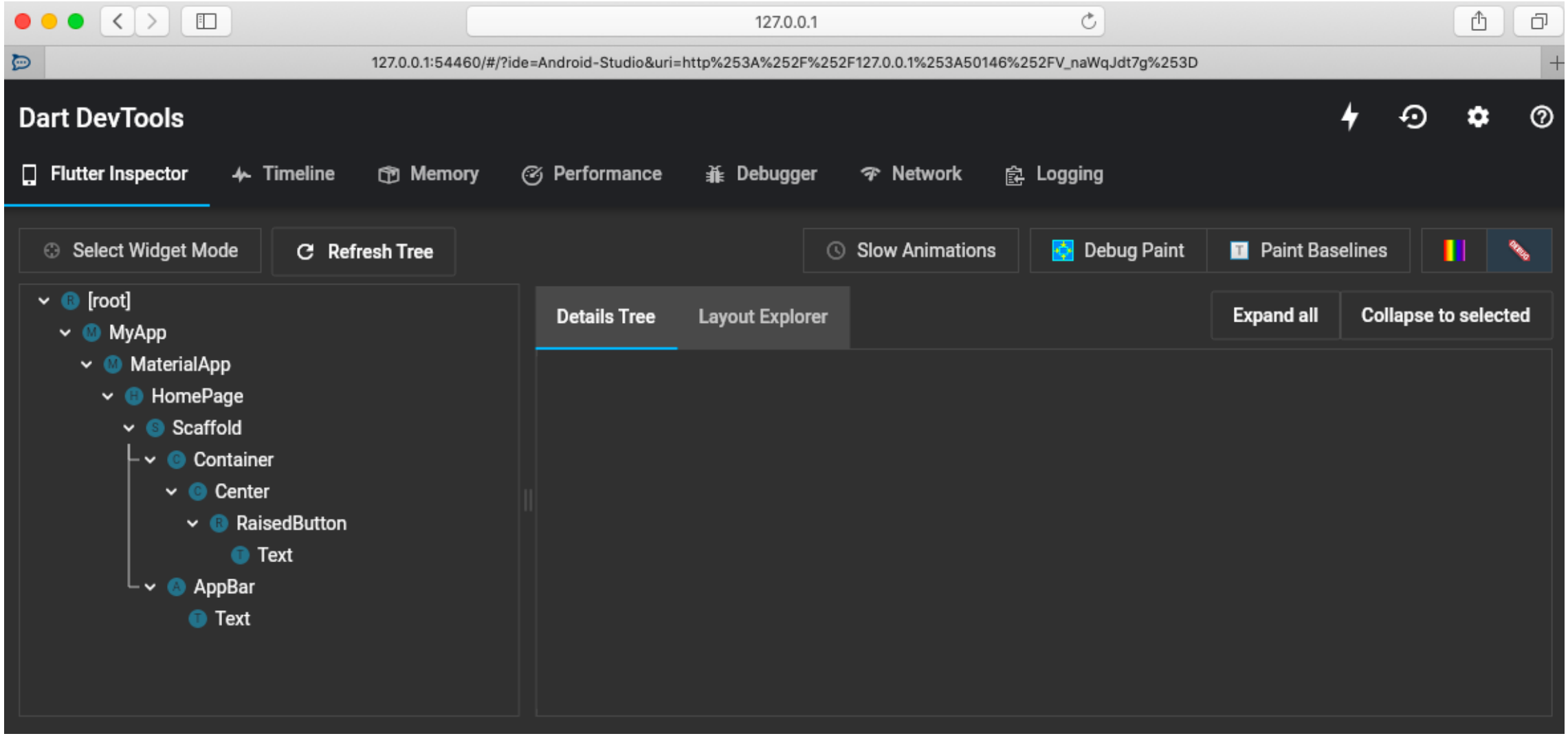


Ao clicar no botão Open Dev Tools

Apresenta a árvore do projeto que está sendo construído, de acordo com os widgets utilizados



Dev Tools


127.0.0.1
127.0.0.1:54460/#/?ide=Android-Studio&uri=http%253A%252F%252F127.0.0.1%253A50146%252FV_naWqJdt7g%253D
Dart DevTools
Flutter Inspector
Timeline
Memory
Performance
Debugger
Network
Logging
Select Widget Mode
Refresh Tree
Slow Animations
Debug Paint
Paint Baselines
[root]
MyApp
MaterialApp
HomePage
Scaffold
Container
Center
RaisedButton
Text
AppBar
Text
Details Tree
Layout Explorer
Expand all
Collapse to selected

Referências Bibliográficas

- Flutter - Apresentando o widget MaterialApp. Disponível em: http://www.macoratti.net/19/06/flut_matapp1.htm#:~:text=O%20MaterialApp%20%C3%A9%20um%20widget,para%20aplicativos%20de%20material%20design.&text=home%20%2D%20A%20primeira%20tela%20a,caso%20home%20representa%20%22%2F%22.
- Curso da Udemy – Flutter Essencial do professor Ricardo Lecheta.