



Aula 1 - Desenvolvimento mobile + Flutter

Objetivo do curso

Possibilitar que desenvolvedores possam desenvolver apps para Android e iOS utilizando o Framework Flutter.

Aula 1 - Index

1. Introdução ao desenvolvimento mobile.
2. Como instalar e preparar o ambiente de desenvolvimento?
3. Iniciando com Flutter
4. Instalando plugin para A.S
5. Iniciando um novo projeto
6. Conhecendo o Android Studio
7. Introdução a Dart
8. Primeiro app: Hello World
9. Widgets
10. Analisando o Hello World
11. Mais Widgets
12. Modificando o Hello World

Introdução ao desenvolvimento mobile

iPhone 2G

- Lançado em 2007.
- Tela Multi-touch.
- AppStore.
- SDK Lançada 2008.
- Tela: 320x480 pixels.
- iOS 3.1.3



Introdução ao desenvolvimento mobile

HTC Dream

- Lançado em setembro de 2008.
- Primeiro dispositivo comercial com o sistema Android.
- Não possuía um teclado virtual.
- Eclipse + plugin ADT
- Android Studio lançado apenas em 2015



Introdução ao desenvolvimento mobile - iOS vs Android



- IDE: XCode
- Object-C
- Swift
- ARC
- AppStore



- IDE: Android Studio (inteliJ)
- Java
- Kotlin (JetBrains)
- Garbage Collector
- GooglePlay

Introdução ao desenvolvimento mobile - iOS vs Android



- Anuidade de \$99.
- Necessário equipamento Apple.
- Revisão manual (1 semana).
- Rejeita 80% dos aplicativos.



- Pagamento único de \$25.
- Revisão automática (alguma horas)
- Revisão manual de termos de uso.

Como instalar e preparar o ambiente de desenvolvimento?

1. Instalar o Android Studio

<https://developer.android.com/studio>

2. Instalar um Dispositivo Virtual (emulador) ou Dispositivo físico

3. Instalar o Flutter

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

Obs. Podem ser utilizadas outras IDEs como Visual Code ou IntelliJ

Iniciando com Flutter.



Run flutter doctor

Rode o comando flutter doctor e corrija os erros caso existam.

Esse comando sempre indica problemas relacionados com instalação do framework e como resolvê-los

```
#####  ##      ##      ## ##### ##### ##### #####
##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##
#####  ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##
##      ##      ##      ##      ##      ##      ##      ##
##      #####  #####  ##      ##      #####  ##      ##

WELCOME to the Flutter Console.
=====

Use the console below this message to interact with the "flutter" command.
Run "flutter doctor" to check if your system is ready to run Flutter apps.
Run "flutter create <app_name>" to create a new Flutter project.

Run "flutter help" to see all available commands.

Want to use an IDE to interact with Flutter? https://flutter.dev/ide-setup/

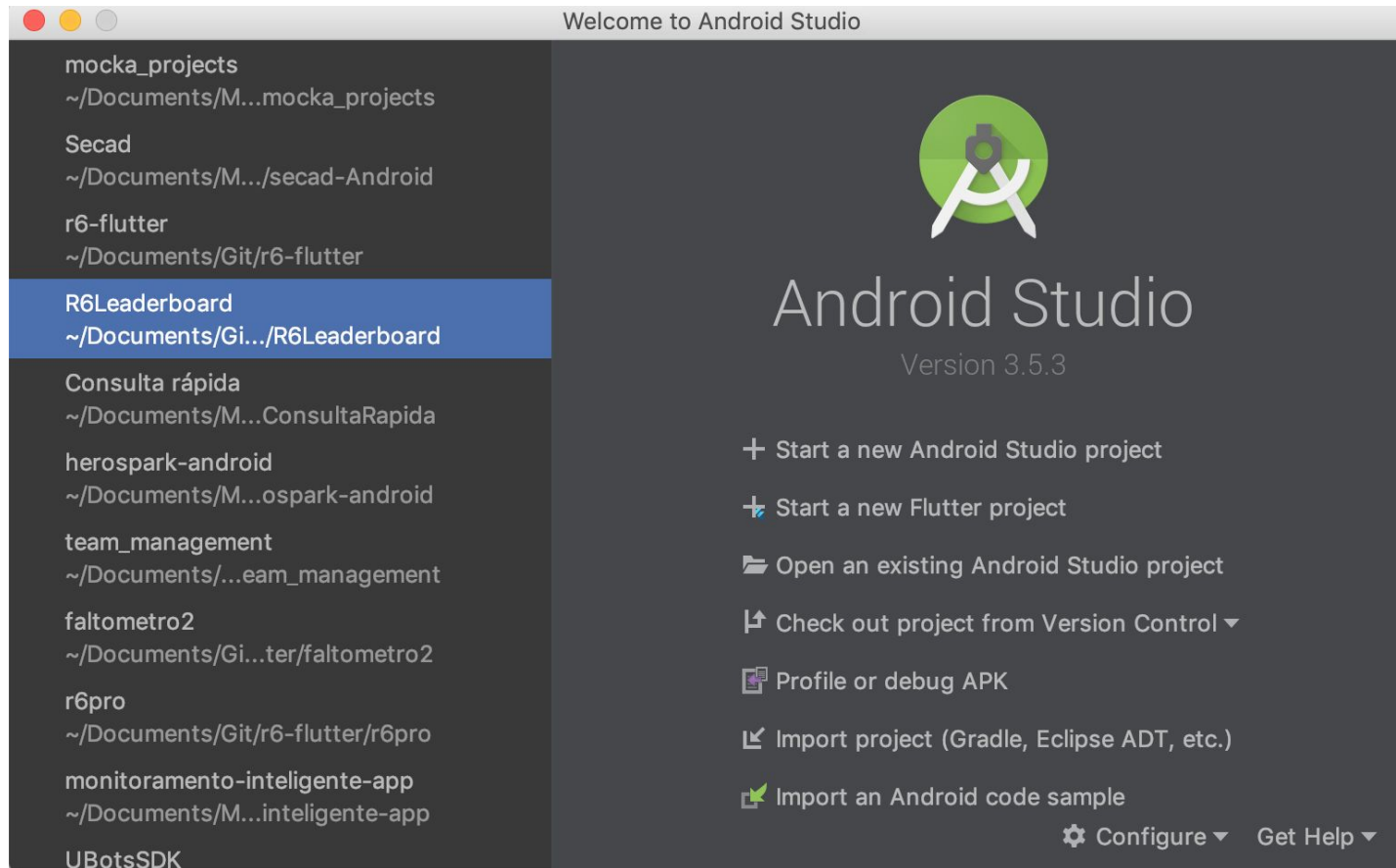
Want to run the "flutter" command from any Command Prompt or PowerShell window?
Add Flutter to your PATH: https://flutter.dev/setup-windows/#update-your-path

=====

C:\Users\Theo>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[v] Flutter (Channel stable, v1.7.8+hotfix.3, on Microsoft Windows [Version 10.0.18362.592], locale en-US)
[!] Android toolchain - develop for Android devices (Android SDK version 28.0.3)
    ! Some Android licenses not accepted.  To resolve this, run: flutter doctor --android-licenses
[v] Android Studio (version 3.4)
[v] VS Code, 64-bit edition (version 1.36.1)
[!] Connected device
    ! No devices available

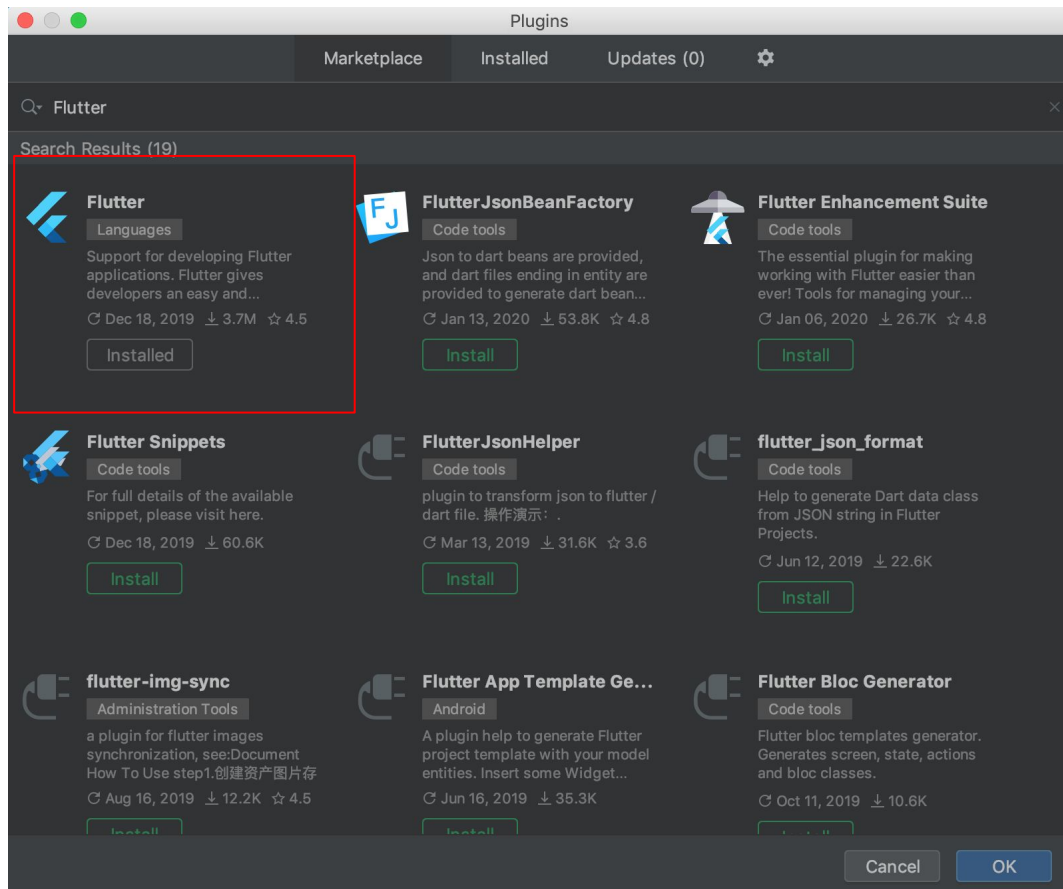
! Doctor found issues in 2 categories.
```

Android Studio - Instalando o Plugin para Flutter



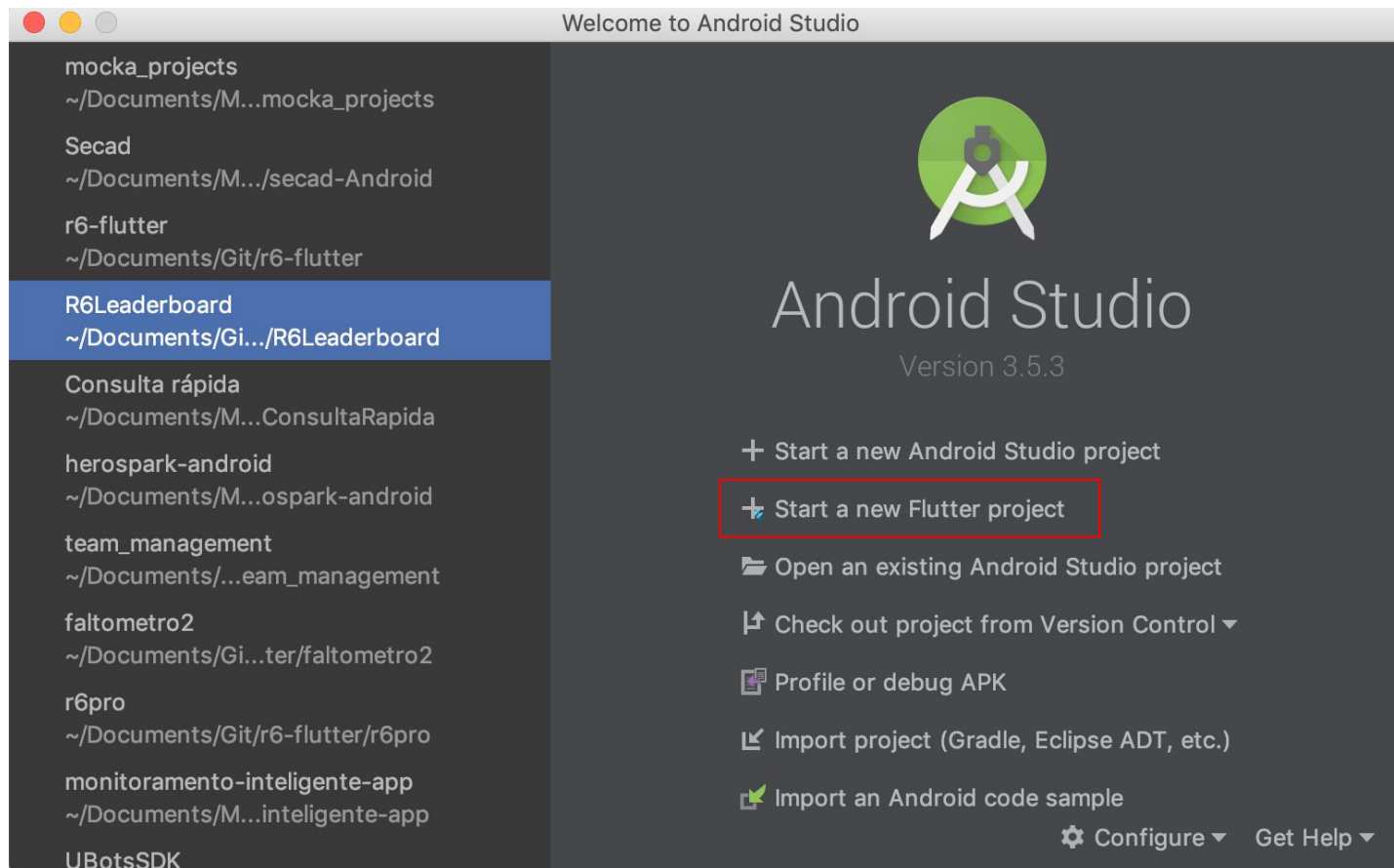
Android Studio - Instalando o Plugin para Flutter

Instalar o Plugin do Flutter e
reiniciar a IDE

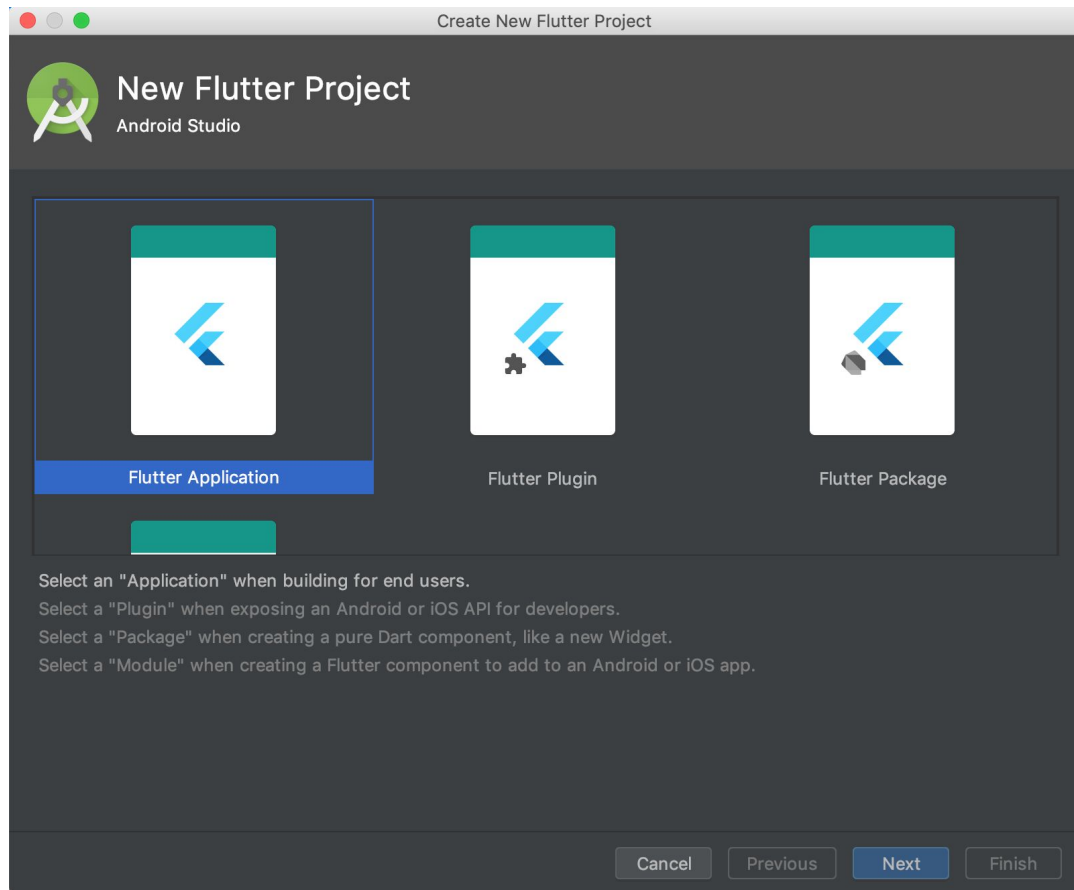


Iniciando um novo projeto

Android Studio - Novo projeto




Android Studio - Novo projeto




Android Studio - Novo projeto

Create New Flutter Project



New Flutter Application

Android Studio




Configure the new Flutter application

Project name

Flutter SDK path

... [↓ Install SDK...](#)

Project location

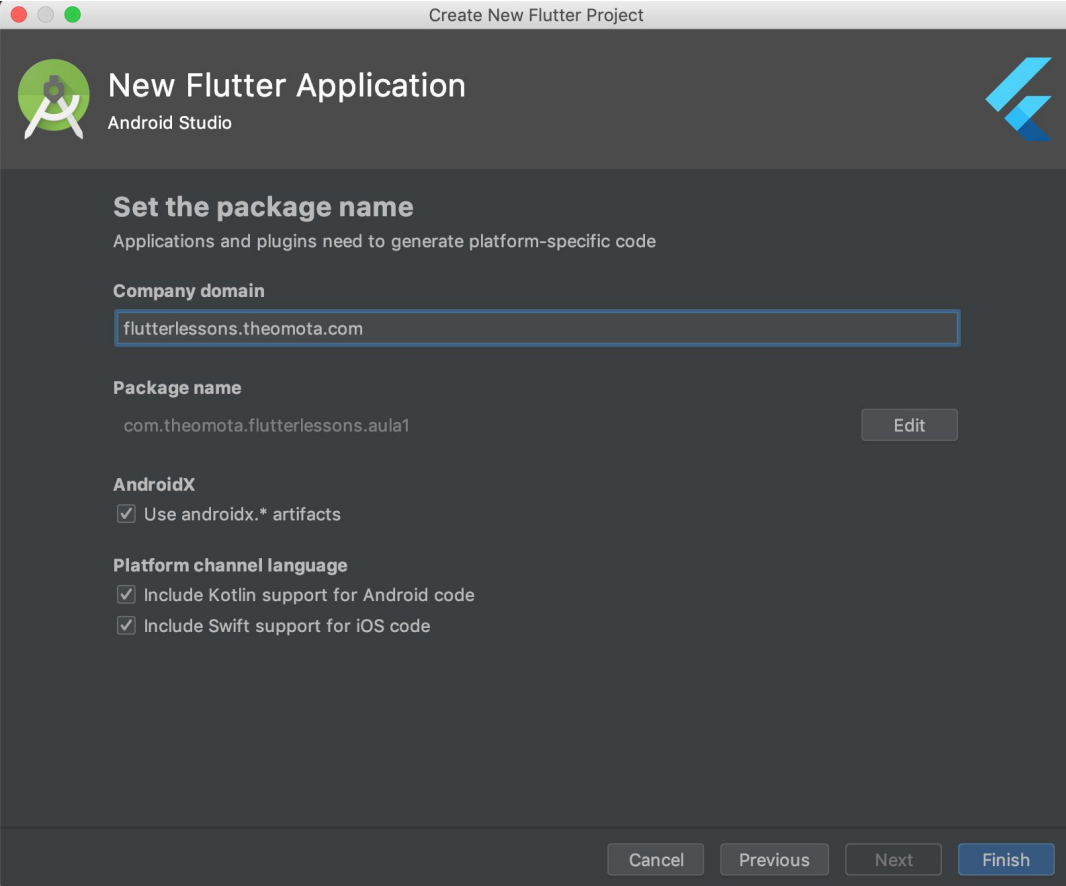


Description

☐ Create project offline



Cancel Previous Next Finish

Android Studio - Novo projeto



The screenshot shows the 'Create New Flutter Project' dialog in Android Studio. The dialog has a dark gray background and a title bar with the text 'Create New Flutter Project'. The top left corner features the Android Studio logo and the text 'New Flutter Application' and 'Android Studio'. The top right corner features the Flutter logo. The main content area is titled 'Set the package name' and includes a subtitle 'Applications and plugins need to generate platform-specific code'. Below this, there are three sections: 'Company domain' with a text input field containing 'flutterlessons.theomota.com'; 'Package name' with a text input field containing 'com.theomota.flutterlessons.aula1' and an 'Edit' button; and 'AndroidX' with a checked checkbox 'Use androidx.* artifacts'. Below these, there is a section titled 'Platform channel language' with two checked checkboxes: 'Include Kotlin support for Android code' and 'Include Swift support for iOS code'. At the bottom of the dialog, there are four buttons: 'Cancel', 'Previous', 'Next', and 'Finish'.

Create New Flutter Project

 **New Flutter Application**
Android Studio 

Set the package name
Applications and plugins need to generate platform-specific code

Company domain

Package name
 Edit

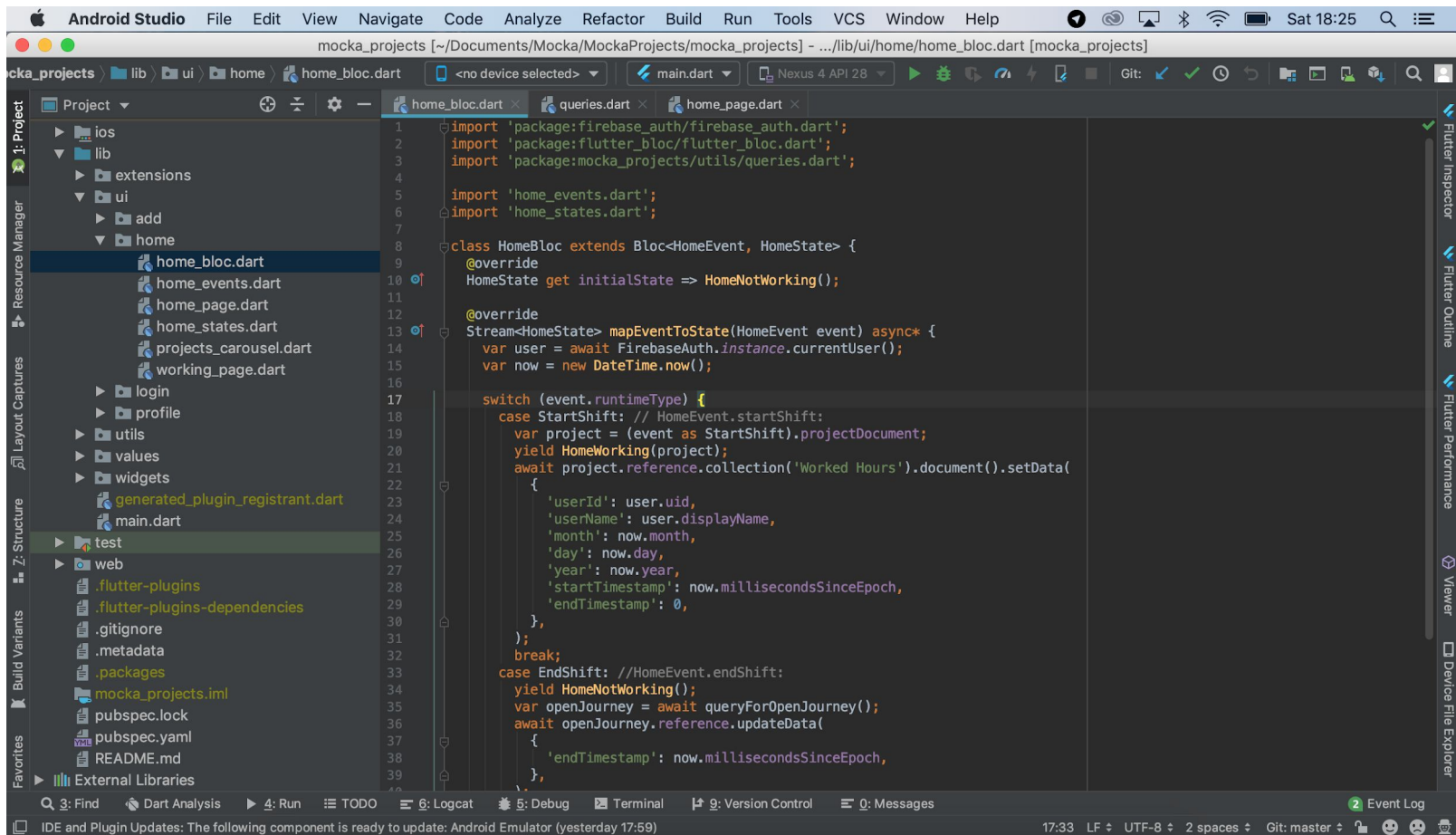
AndroidX
☒ Use androidx.* artifacts

Platform channel language
☒ Include Kotlin support for Android code
☒ Include Swift support for iOS code

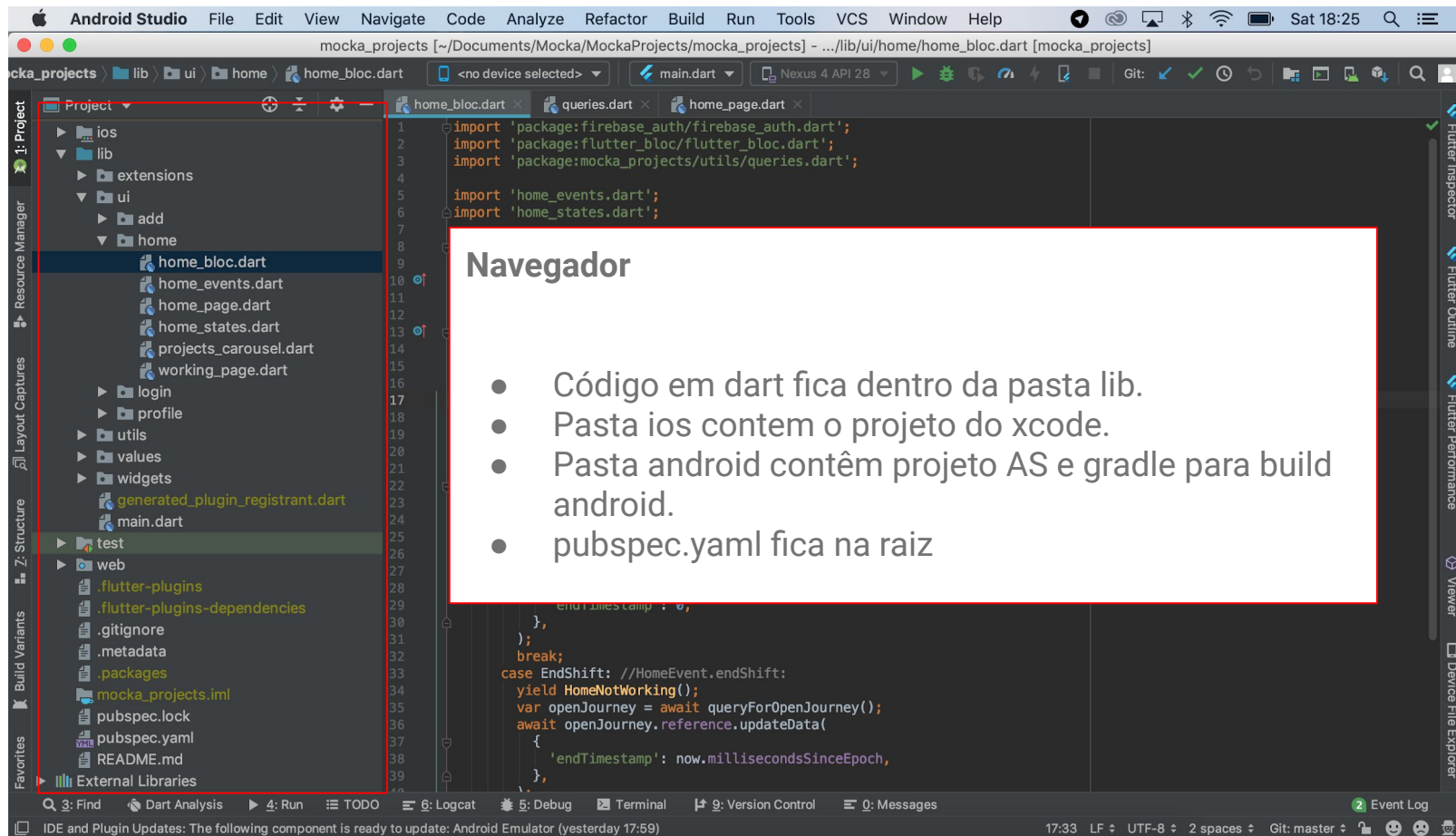
Cancel Previous Next Finish

Conhecendo a IDE

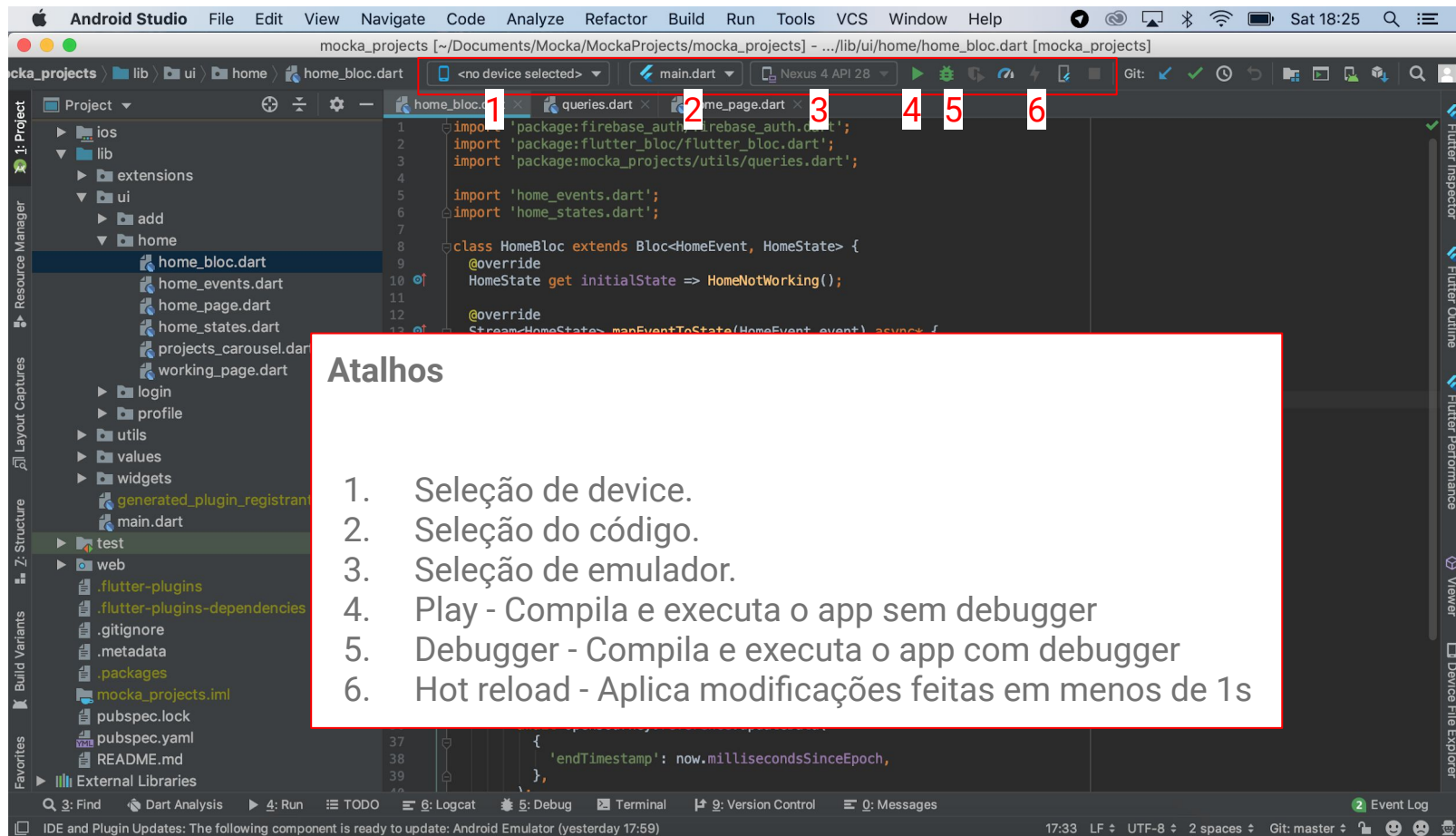
Android Studio



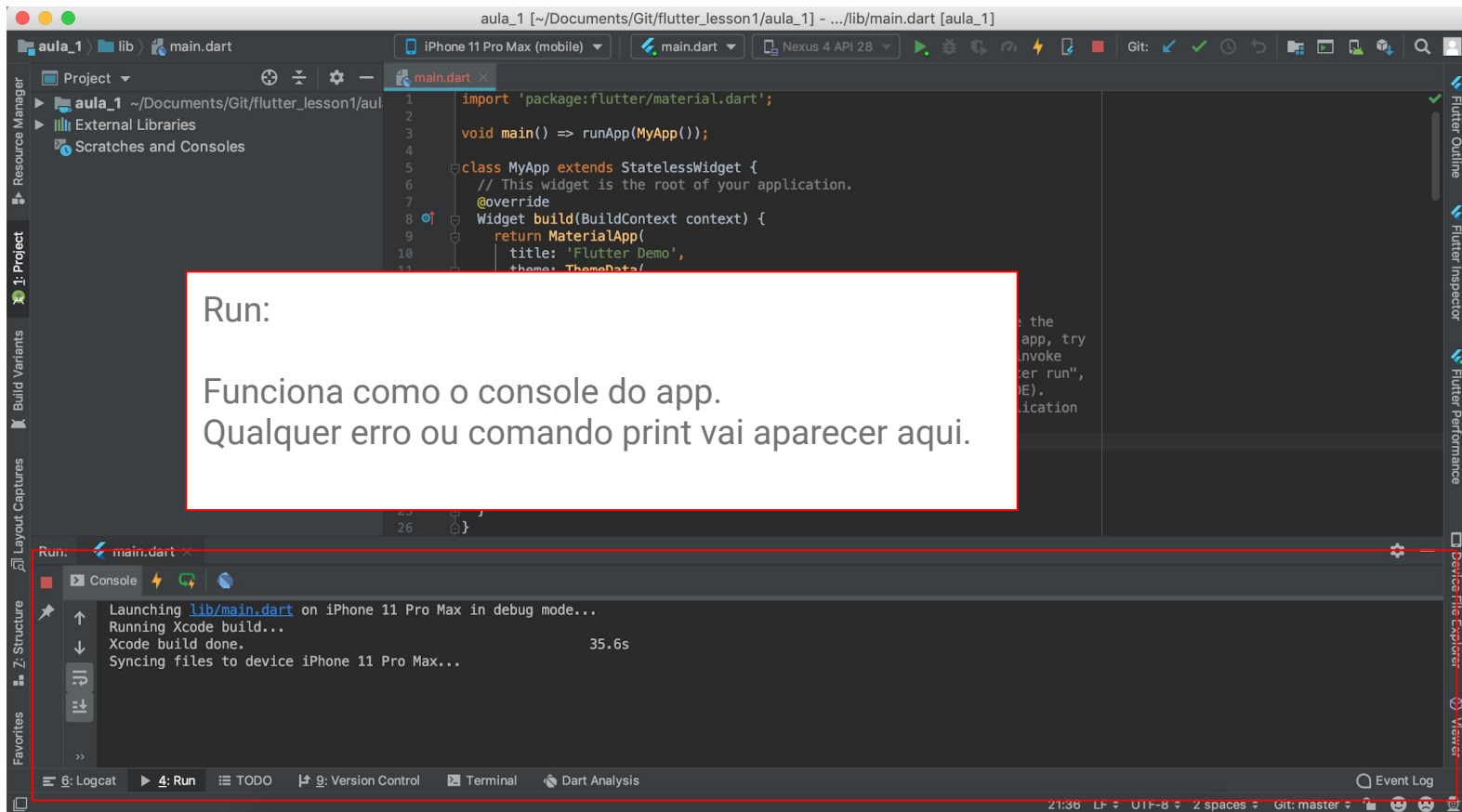
Android Studio



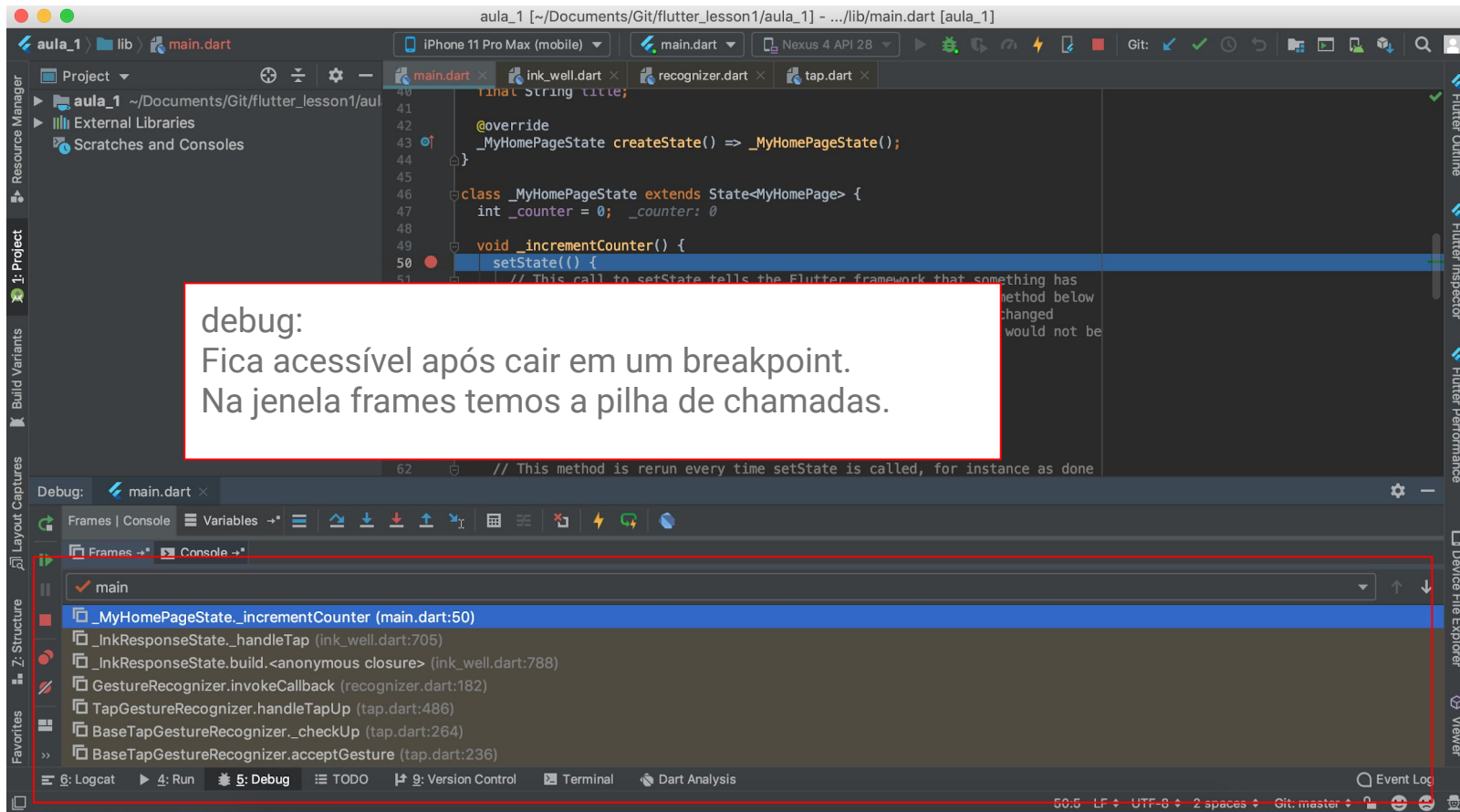
Android Studio



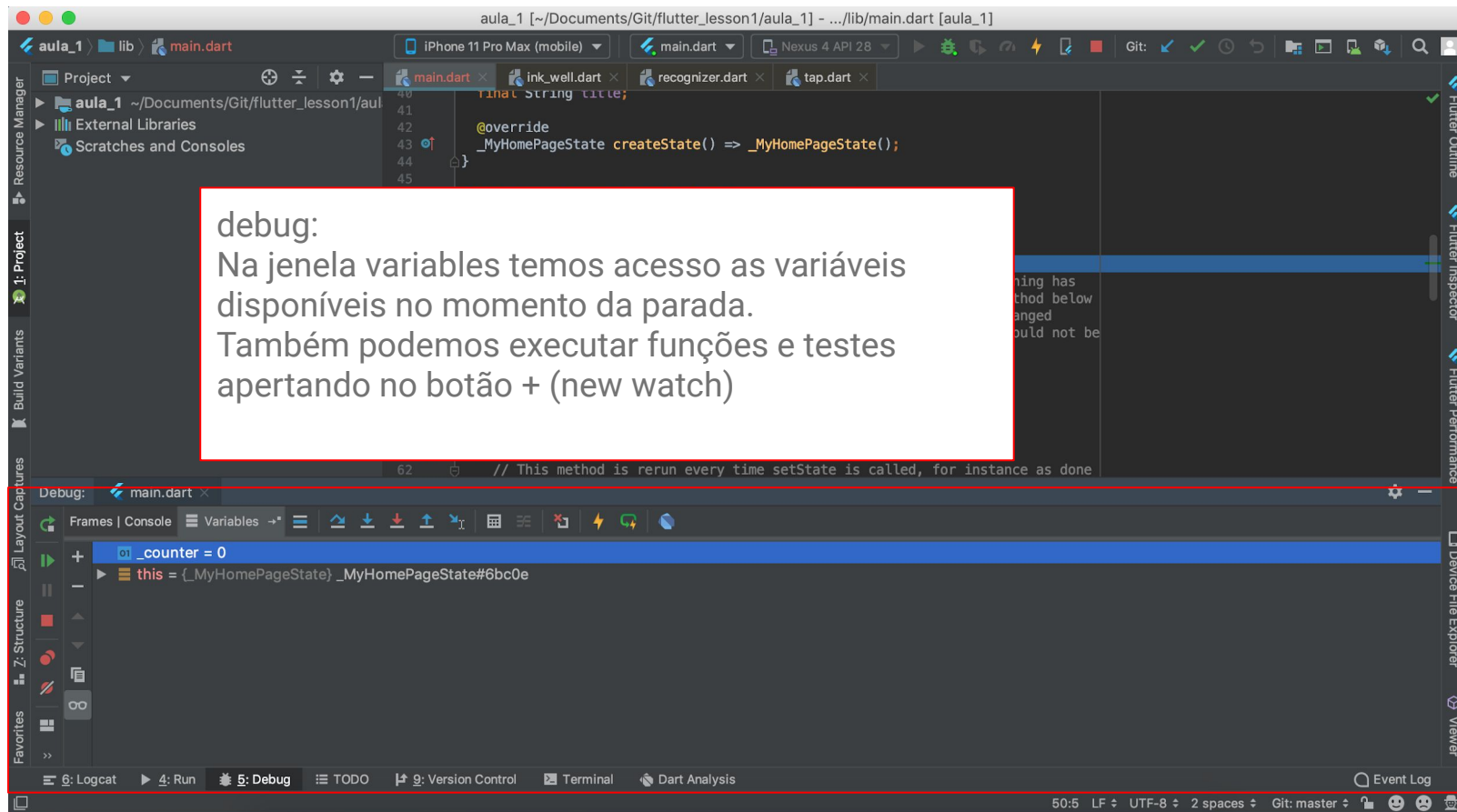
Android Studio



Android Studio



Android Studio



Introdução a Dart.



Introdução a Dart - Variáveis e constantes

```
var variavel; //Se não tipada pode assumir qualquer tipo.  
String variavelTipada; //Não inicializada é null, mas só pode receber outras strings.  
|  
const String constante = 'aula 2'; //Constante
```

Introdução a Dart - Nullability test

```
String stringOrNull;  
stringOrNull = null;  
  
//Método toLowerCase faz parte da classe String.  
//Como a variável stringOrNull é null ao tentar chamar o método vai resultar no seguinte erro:  
//    The method 'toLowerCase' was called on null.  
//    Receiver: null  
//    Tried calling: toLowerCase()  
stringOrNull.toLowerCase();  
  
//Para esse tipo de situação utilizamos o operador ?  
//que verifica se a var é null antes de executar o método  
stringOrNull?.toLowerCase();
```

Introdução a Dart - Nullability test

```
String stringNull;  
stringNull = null;  
  
var output = stringNull ?? 'Output text';  
print(output);
```

O operador **??** verifica se o que esta a esquerda dele é não null.
Caso positivo retorna o que esta a esquerda.
Caso negativo retorna o que esta a direita.
Neste exemplo o print seria: 'Output text'

Introdução a Dart - Listas

```
var list = List();  
list.add('list dinâmica');  
list.add(1);  
list.add(1.4);  
print(list); //output -> flutter: [list dinâmica, 1, 1.4]  
  
var secondList = ['Segunda list', 1, 1.5];  
print(secondList); //output -> flutter: [Segunda list, 1, 1.5]  
  
var listaTipada = List<int>();  
listaTipada.add(1);  
listaTipada.add('teste');  
listaTipada.add(1.4);
```

No exemplo acima o código não compila pois estamos tentando adicionar uma string e um double em uma lista do tipo int.

Introdução a Dart - Map

```
///LinkedHashMap não tipado.  
///Aceita qualquer tipo tanto na chave quanto no valor.  
var map = Map();  
map[1] = 'Teste';  
map['chave'] = 1;  
print(map); //output-> flutter: {1: Teste, chave: 1}  
  
///Outra forma de inicialização  
var hashmap = {'chave' : 1, 123: 'valor'};  
print(hashmap); //output-> flutter: {chave: 1, 123: valor}  
  
///HashMap com chave do tipo String e valor do tipo int  
var mapTipado = Map<String, int>();  
mapTipado['chave'] = 1;  
print(mapTipado);
```

Introdução a Dart - Iterations

```
for(var i=0 ; i < 100 ; i++) {  
  | print(i);  
}
```

```
var list = [1,2,3,4,5];
```

```
list.forEach((element) {  
  | print(element);  
});
```

```
for(var element in list) {  
  | print(element);  
}
```

Introdução a Dart - Conditionals

```
var teste = 'String teste';

if(teste == 'String teste') {
  print('Verdade');
} else {
  print('False');
}

///Operador ternário
teste == 'String teste' ? print('Verdade') : print('False');

switch(teste) {
  case 'teste':
    print('oi');
    break;
  case 'String teste':
    print('hello');
    break;
}
```


Introdução a Dart - Funções

```
void teste1() {  
  print('teste1');  
}
```

```
bool verifica2(int value) {  
  if(value == 2){  
    return true;  
  } else {  
    return false;  
  }  
}
```

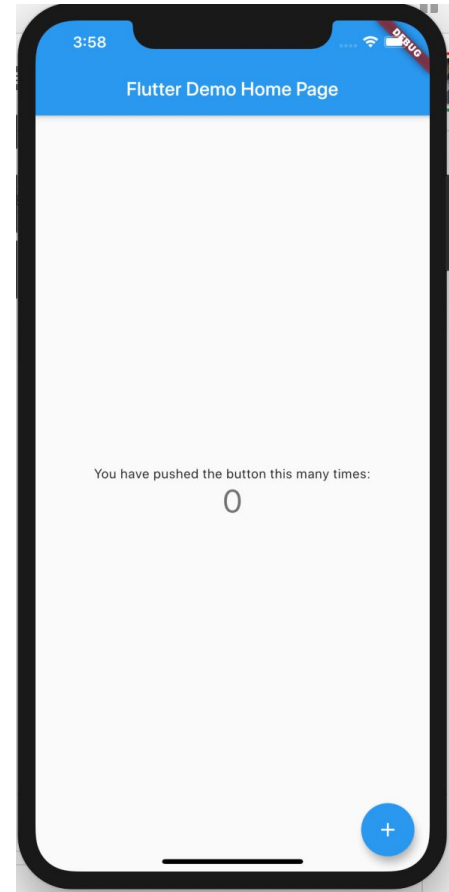
```
bool oneLineFunction(int value) => value == 2;
```

Introdução a Dart - Classes

```
class Person {  
  ///Modificador final faz com que depois de atribuido o valor ele não possa ser alterado.  
  ///É diferente de const.  
  final String name;  
  final String lastName;  
  final int age;  
  
  ///Construtor da classe  
  Person(this.name, this.lastName, this.age);  
  
  String getFullName() => '$name $lastName';  
}
```

Se não for declarado um construtor para a classe o compilador vai criar automaticamente um default.

Hello World



Hello World

1. Após criar um novo projeto como demonstrado anteriormente teremos disponível um arquivo `main.dart` que contém o nosso app inicial, o counter, que é o equivalente ao Hello World do flutter.
2. Conecte um dispositivo físico em modo de desenvolvedor ou inicie um emulador pelo próprio Android Studio.
3. Rode o aplicativo apertando `ctrl+r` ou na seta verde na barra de atalhos.
4. Explore o app no dispositivo conectado.

Hello World

```
1  import 'package:flutter/material.dart';
2
3  void main() => runApp(MyApp());
4
5  class MyApp extends StatelessWidget {
6    // This widget is the root of your application.
7    @override
8    Widget build(BuildContext context) {
9      return MaterialApp(
10        title: 'Flutter Demo',
11        theme: ThemeData(
12          primarySwatch: Colors.blue,
13        ),
14        home: MyHomePage(title: 'Flutter Demo Home Page'),
15      );
16    }
17  }
18
19  class MyHomePage extends StatefulWidget {...}
26
27  class _MyHomePageState extends State<MyHomePage> {...}
64
```

Como podemos ver o aplicativo é formado apenas por 3 classes e 64 linhas de código.

A função main() chama o método runApp passando a raiz do app como parâmetro. Neste caso o root do app é a classe MyApp.

Widgets: Tudo é um widget.

Widgets

Tudo é um widget.

Os widgets são os blocos de construção básicos da interface de usuário de um aplicativo Flutter.

Cada widget é uma declaração imutável de parte da interface do usuário. Diferente de outros frameworks que separam visualizações, controladores de exibição, layouts e outras propriedades, o Flutter possui um modelo de objeto unificado e consistente: o widget.

Em Flutter existem apenas 2 tipos de widgets: Stateless e Stateful.

Widgets: Stateless

Stateless widgets são widgets que não requerem um estado mutável, ou seja são imutáveis.

Esse tipo de widget é muito útil quando a parte da interface do usuário que você está descrevendo não depende de nada além das informações de configuração do próprio objeto, como por exemplo um texto que não muda ou uma imagem estática.

Ao observarmos o app Hello World podemos encontrar alguns stateless widgets como por exemplo o `Text()`, `Icon()`, `FloatingActionButton()`...

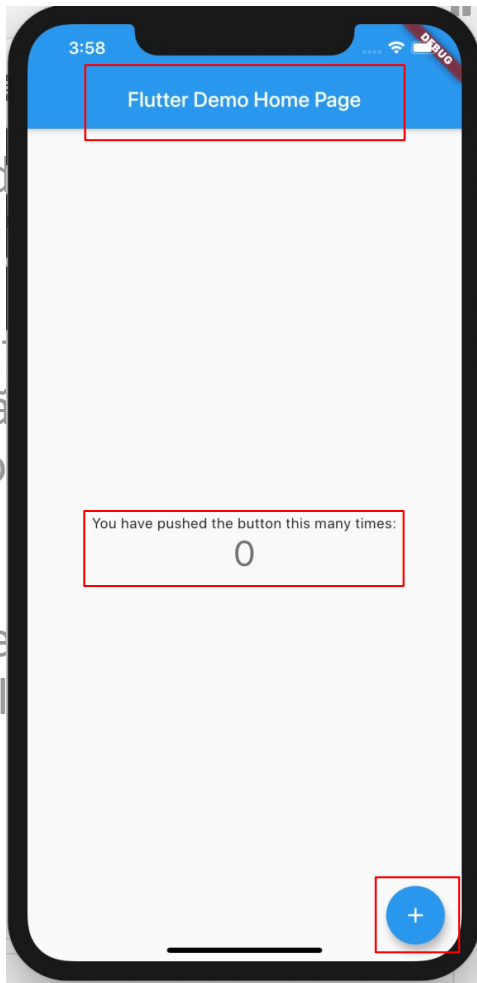
Até mesmo a raiz da aplicação (`MyApp`) é um `StatelessWidget`.

Widgets: Stateless

Stateless widgets são widgets que não possuem um estado mutável, ou seja, são imutáveis.

Esse tipo de widget é muito útil para descrever elementos da interface do usuário que não dependem das informações de configuração do próprio widget, como um texto que não muda ou uma imagem estática.

Ao observarmos o app Hello Flutter, encontramos alguns stateless widgets como por exemplo:



um estado mutável, ou seja

da interface do usuário que
em das informações de
lo um texto que não muda ou

contrar alguns stateless
ingActionButton()...

Widgets: Stateless

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

Widgets: Stateful

Stateful widgets são widgets que requerem um estado mutável, ou seja podem se reconstruir sempre que necessário.

Esse tipo de widget é muito útil quando a parte da interface do usuário que você está descrevendo muda dinamicamente como por exemplo um relógio que muda a todo segundo.

Ao observarmos o app Hello World podemos ver que apesar da raiz do app ser stateless a MyHomePage é stateful. Isso é necessário pois sempre que apertamos o botão azul o contador é incrementado. Para isso parte desse widget precisa ser reconstruída refletindo seu novo estado.

Widgets: Stateful

Um Stateful widget é composto por duas classes: O statefulWidget e o State do mesmo.

```
class TesteDeStateful extends StatefulWidget {  
  @override  
  _TesteDeStatefulState createState() => _TesteDeStatefulState();  
}  
  
class _TesteDeStatefulState extends State<TesteDeStateful> {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

Widgets: Stateful

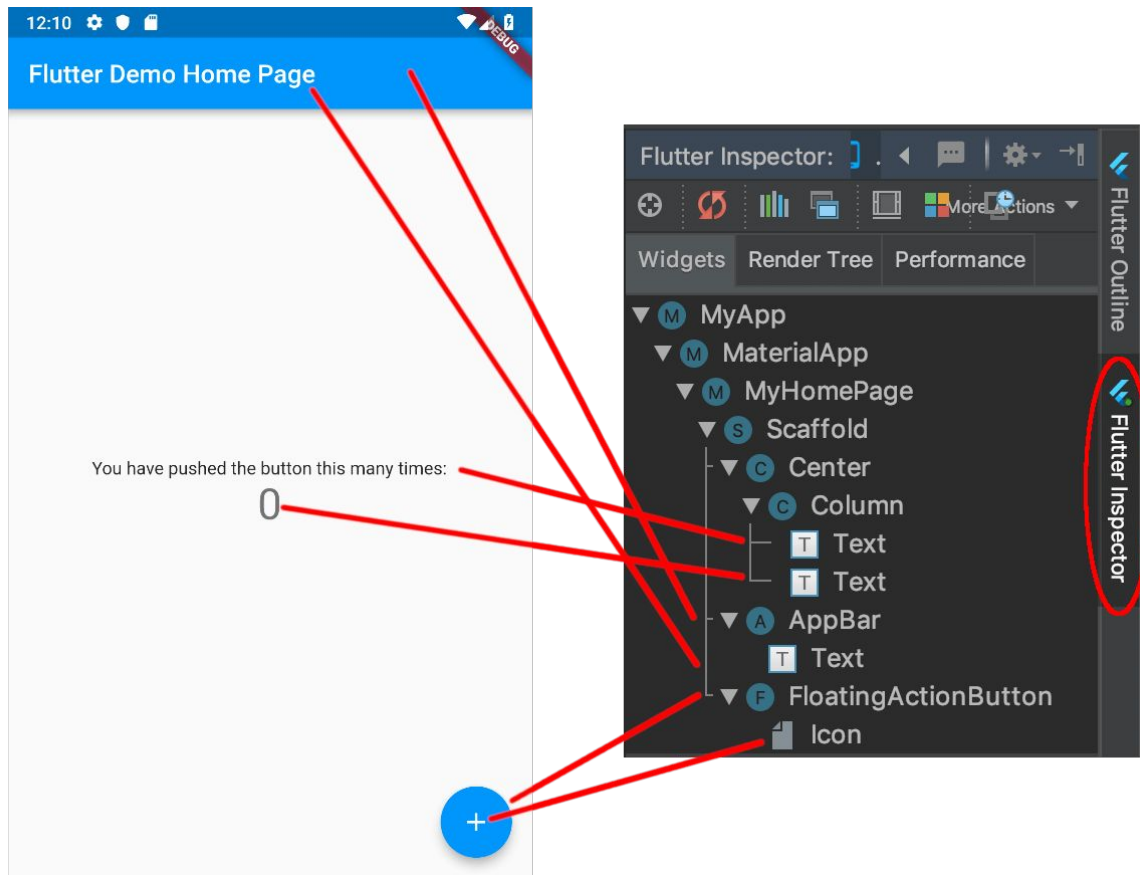
Um Stateful widget é composto por duas classes: O `statefulWidget` e o `State` do mesmo.

```
class TesteDeStateful extends StatefulWidget {  
  @override  
  _TesteDeStatefulState createState() => TesteDeStatefulState();  
}  
  
class _TesteDeStatefulState extends State<TesteDeStateful> {  
  @override  
  Widget build(BuildContext context) {  
    return Container();  
  }  
}
```

Sugestão: analisar o app HelloWorld e o porquê precisamos utilizar um StatefulWidget.

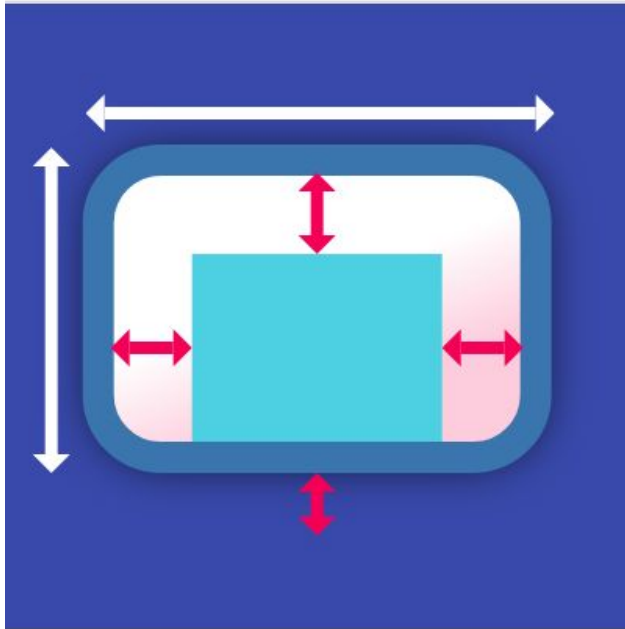
Atenção para o método `setState()`

Analizando o HelloWorld



Layout widgets

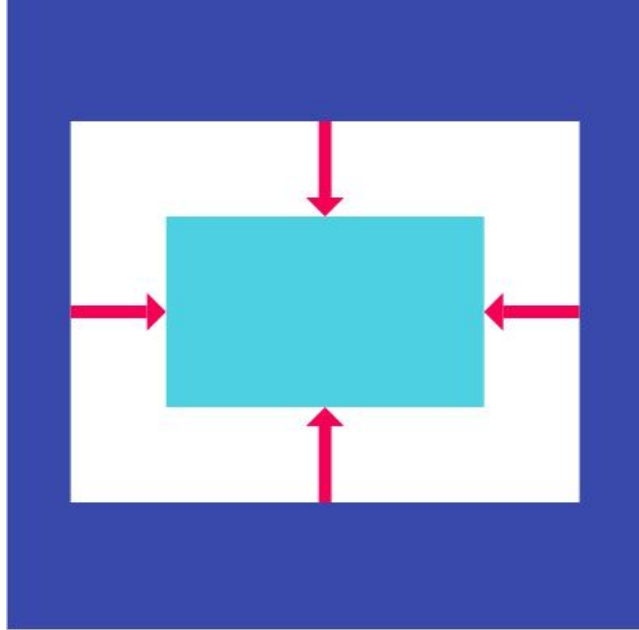
Container



É um dos widgets mais utilizados. Ele pode receber como parâmetros: padding, alignment, color, width, height, margin e uma child.

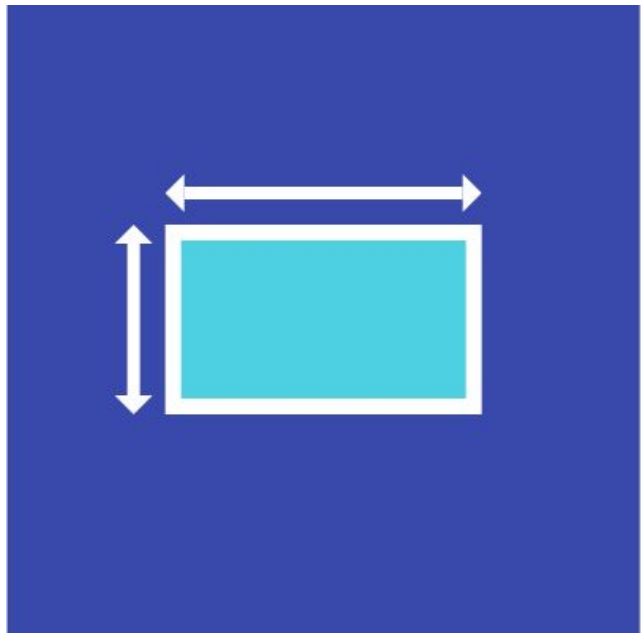
Como ele recebe um único filho (outro widget) ele é bastante utilizado para adicionar margens, alinhamento ou até mesmo mudar a cor do fundo.

Center



Recebe um único filho e o centraliza dentro de si mesmo.

SizedBox



Recebe um único filho, altura e largura. Cria uma caixa do tamanho informado com o filho dentro.

Esse widget é muito utilizado para fazer espaçamentos em listas. Por exemplo:

```
SizedBox(height: 16)
```

Vai criar um espaço vazio de altura 16.

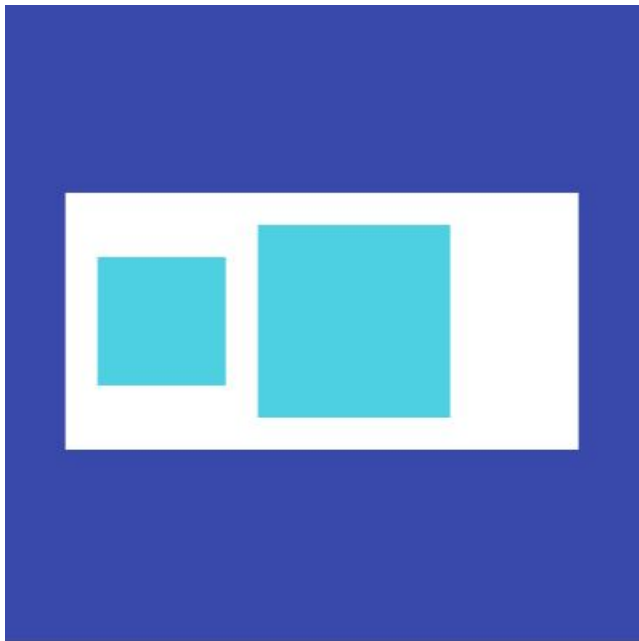
Column



Cria uma coluna com uma lista de widgets filhos.

Também pode ser informado no construtor alinhamento e tamanho.

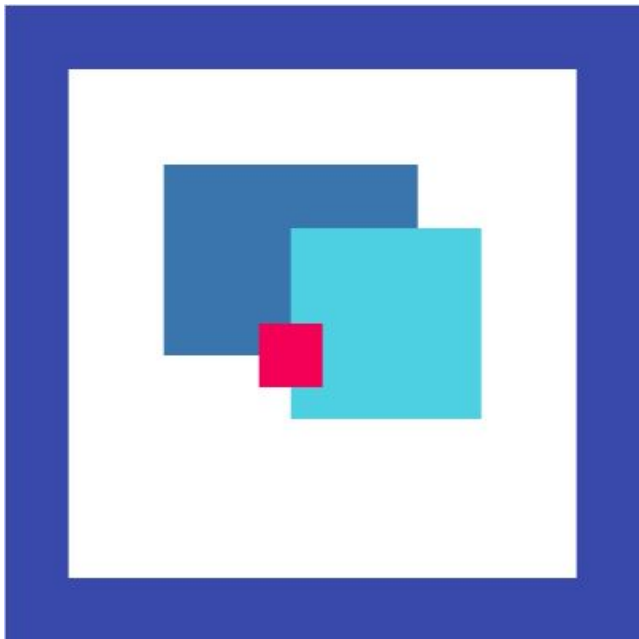
Row



Cria uma linha com uma lista de widgets filhos.

Também pode ser informado no construtor alinhamento e tamanho.

Stack



Cria um stack de widgets.

Nos filhos pode ser informado o alinhamento utilizando o widgets Align.

Muito mais

<https://flutter.dev/docs/development/ui/widgets>

Modificando o HelloWorld

Objetivo:

Modificar o app Hello World e transforma-lo em um app de curriculum vitae.

Tente utilizar ao menos um stateful widget.



Documentação

Git da aula 1:

https://github.com/theolm/flutter_lesson1

Catálogo de widgets:

<https://flutter.dev/docs/development/ui/widgets>

Layouts in Flutter:

<https://flutter.dev/docs/development/ui/layout>

Flutter channel:

<https://www.youtube.com/channel/UCwXdFgeE9KYzIDdR7TG9cMw>

Widget of the week:

https://www.youtube.com/watch?v=b_sQ9bMltGU&list=PLjxrf2q8roU23XGwz3Km7sQZFTdB996iG



Theodoro Loureiro Mota
theolm.mota@gmail.com