



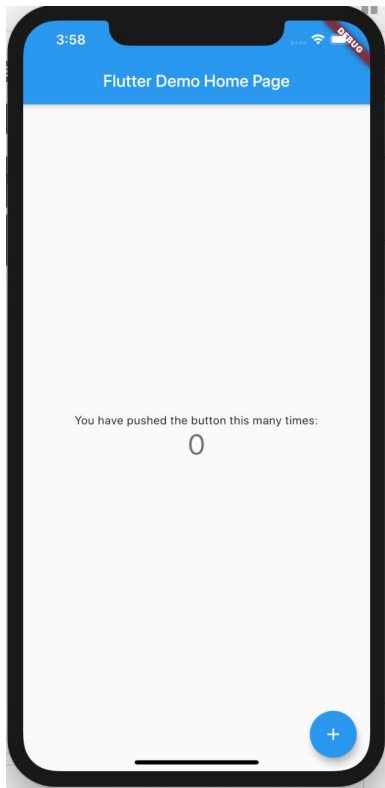
# Flutter

## **Aula 2 - Persistência + Plugins**

# Aula 2 - Index

1. Relembrando.
2. Melhorando o App C.V.
3. Criando widgets.
4. Navigator.
5. Mais ferramentas de desenvolvimento.
6. Release vs. Debug.
7. Plugins.
8. Shared preferences.
9. Serialização de dados.
10. ListView.
11. Await/async.
12. Objetivo: TODO list app

# Relembrando



- Introdução a Dart
  - Operador ternário
  - Listas
  - Nullability test
  - Classes
- Widgets
  - Stateless
  - Stateful
  - SetState
- App Hello World
  - Widgets como parâmetro
  - Estrutura em árvore

## Relembrando - Nullability test

```
//Para esse tipo de situação utilizamos o operador ?  
//que verifica se a a var é null antes de executar o método  
stringOrNull?.toLowerCase();|
```

stringOrNull é do tipo string, mas se não inicializado o valor inicial é null.

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

O operador ?? verifica se o que está à esquerda dele é não null.

Caso positivo retorna o que está à esquerda.

Caso negativo retorna o que está à direita.

Neste exemplo o print seria: 'Output text'

## Relembrando - 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';  
}
```

## Relembrando - Widgets: stateless

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

## Relembrando - Widgets: stateful

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

## Relembrando - Widgets como parâmetro

```
return Container(  
  child: Column(  
    children: <Widget>[  
      Text('filho 1'),  
      Text('filho 2'),  
      Text('filho 3'),  
      Text('filho 4'),  
      Row(  
        children: <Widget>[  
          Text('neto 1'),  
          Text('neto 2'),  
          Text('neto 3'),  
          Text('neto 4'),  
        ],  
      ),  
    ],  
  ),  
);
```

Widgets podem receber 1 ou mais widgets como filho.

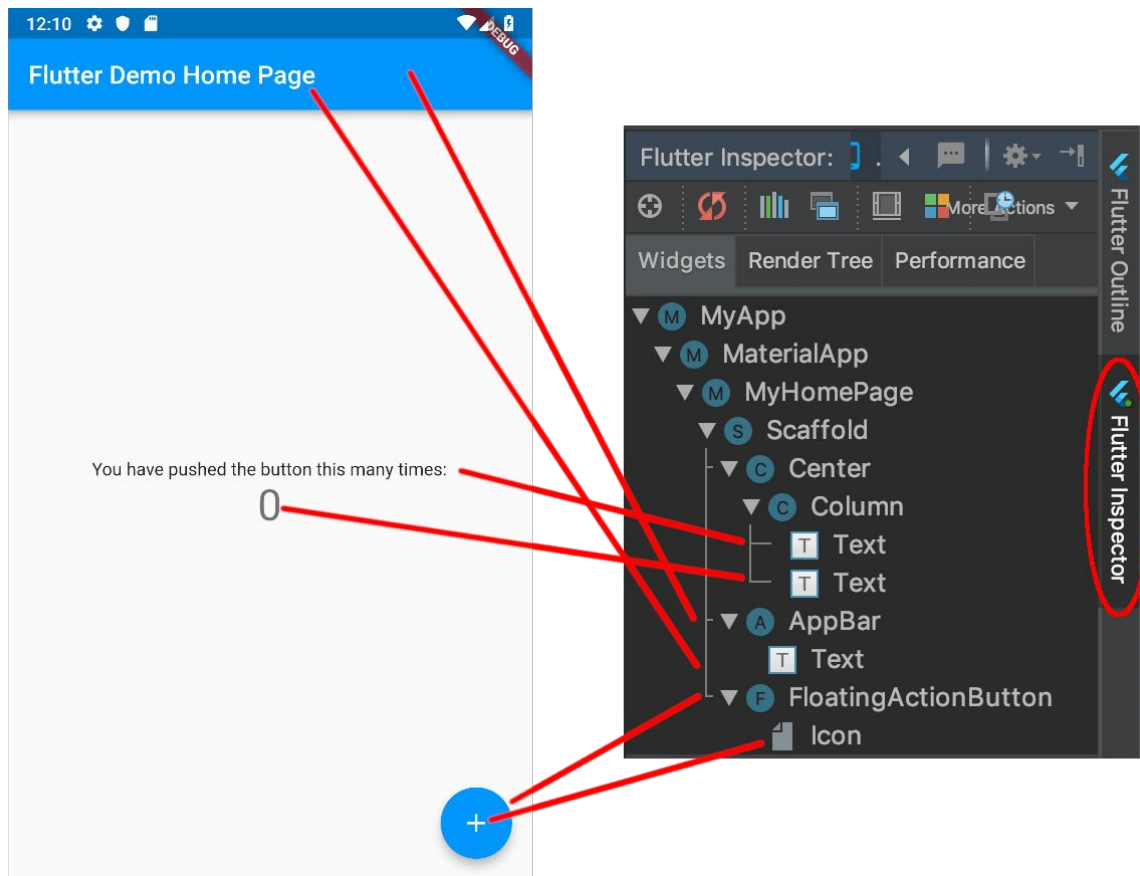
Tudo é passado no construtor.

Geralmente widgets tem diversos parâmetros opcionais, para verificar as opções utilize a documentação inclusa no próprio framework.

Ctrl + click no nome.



## Relembrando - Estrutura em árvore



# Melhorando o App C.V.



# Melhorando o App C.V.



## Problemas?

Tudo em um único arquivo.

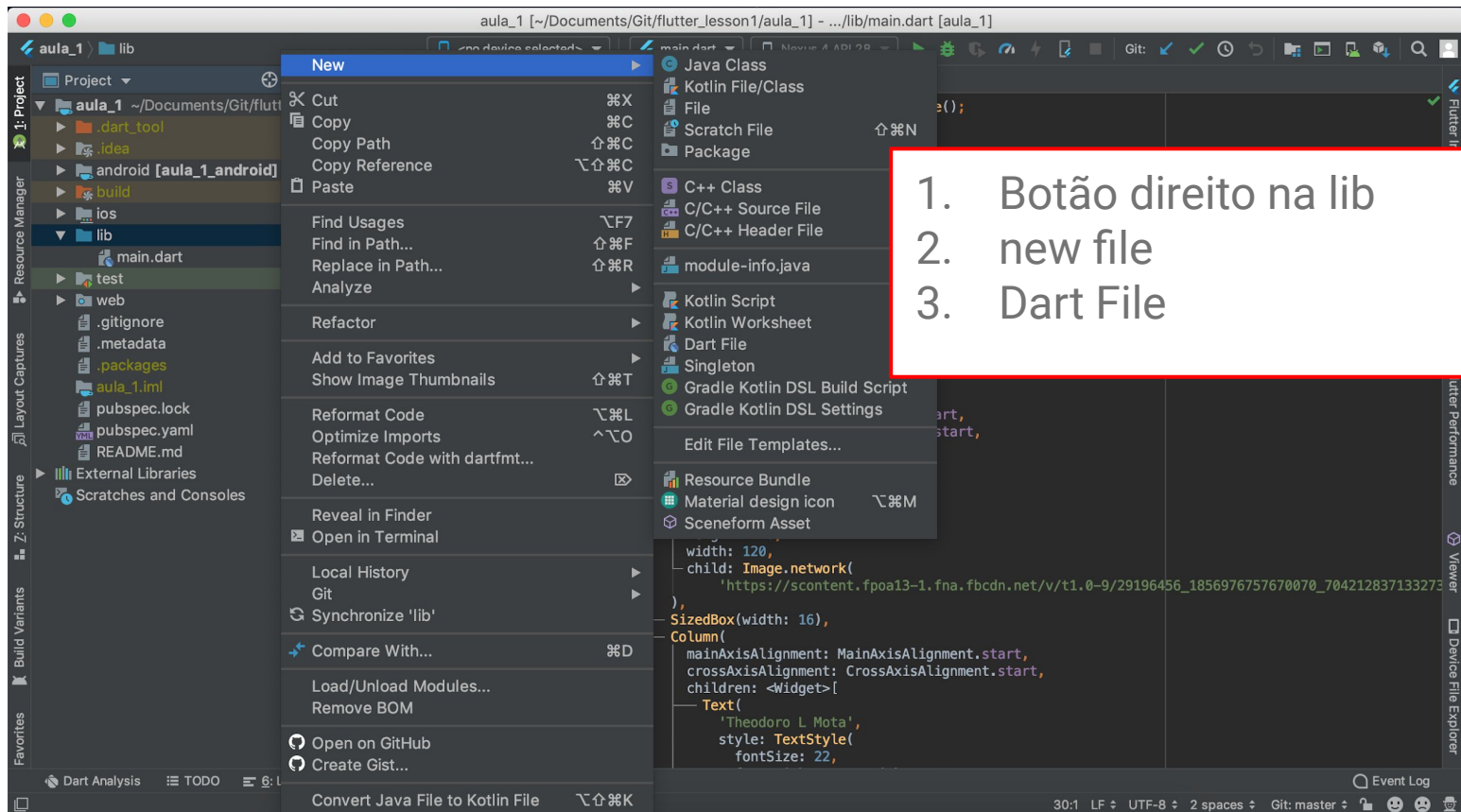
Muitos Widgets aninhado.

Muito código repetido.

## Solução:

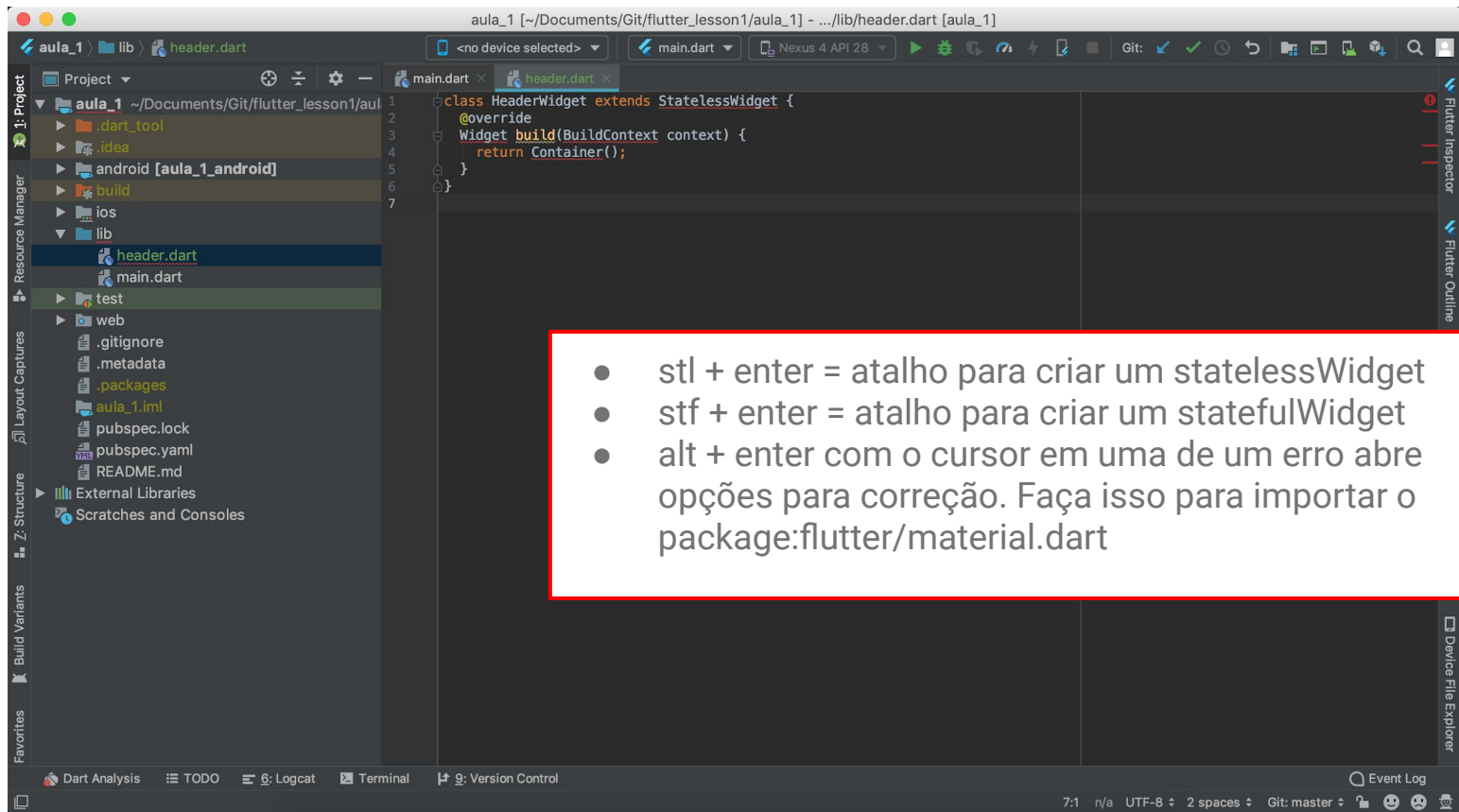
Criar novos widgets customizados.

## Criando widgets



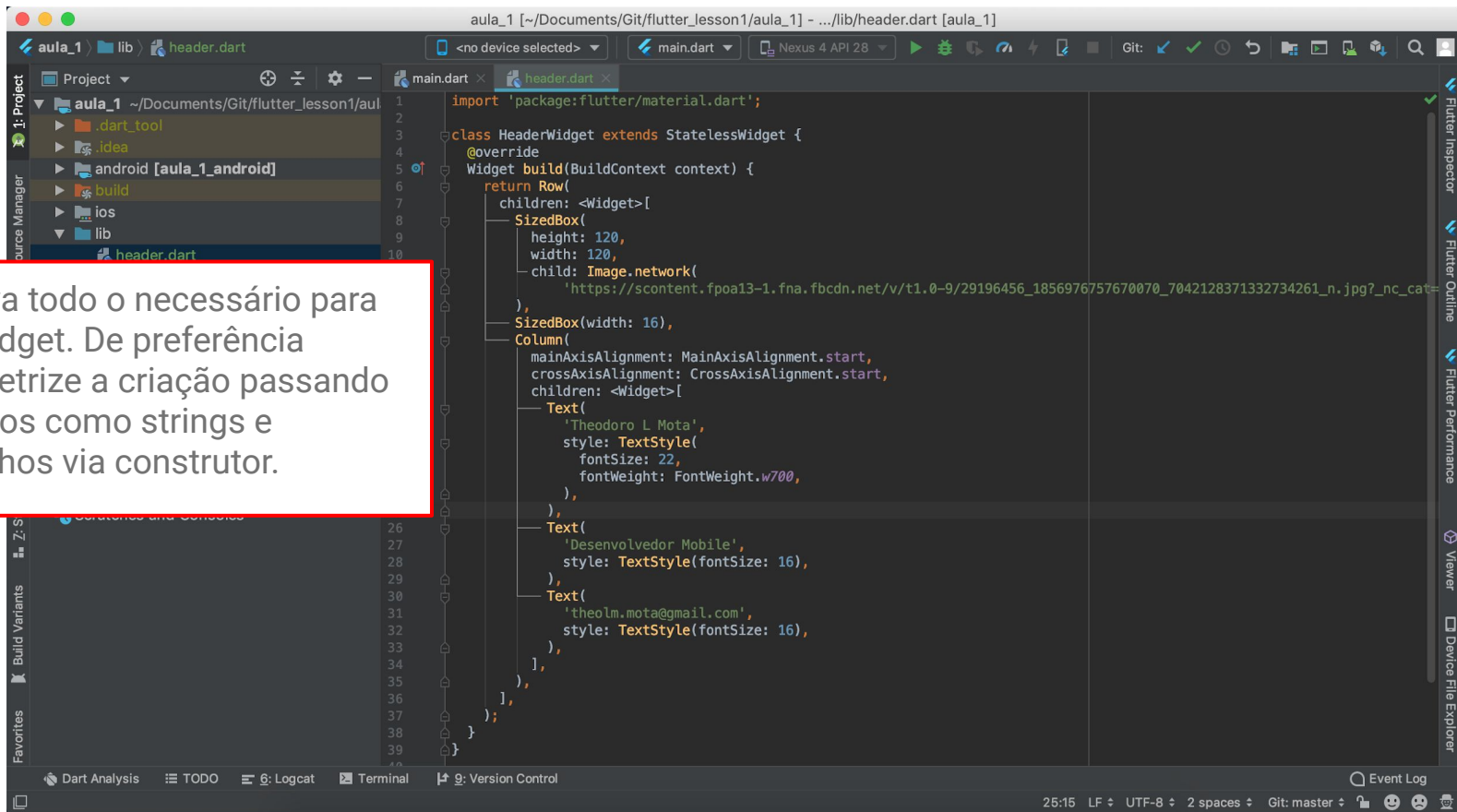
1. Botão direito na lib
2. new file
3. Dart File

# Criando widgets



# Criando widgets

Escreva todo o necessário para seu widget. De preferência parametrize a criação passando atributos como strings e tamanhos via construtor.



The screenshot shows an IDE window titled 'aula\_1' with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with folders for 'android', 'ios', and 'lib', and a file 'header.dart' under 'lib'. The code editor shows the following Dart code:

```
1 import 'package:flutter/material.dart';
2
3 class HeaderWidget extends StatelessWidget {
4   @override
5   Widget build(BuildContext context) {
6     return Row(
7       children: <Widget>[
8         SizedBox(
9           height: 120,
10          width: 120,
11          child: Image.network(
12            'https://scontent.fpoa13-1.fna.fbcdn.net/v/t1.0-9/29196456_1856976757670070_7042128371332734261_n.jpg?_nc_cat=1',
13          ),
14        ),
15        SizedBox(width: 16),
16        Column(
17          mainAxisAlignment: MainAxisAlignment.start,
18          crossAxisAlignment: CrossAxisAlignment.start,
19          children: <Widget>[
20            Text(
21              'Theodoro L Mota',
22              style: TextStyle(
23                fontSize: 22,
24                fontWeight: FontWeight.w700,
25              ),
26            ),
27            Text(
28              'Desenvolvedor Mobile',
29              style: TextStyle(fontSize: 16),
30            ),
31            Text(
32              'theolm.mota@gmail.com',
33              style: TextStyle(fontSize: 16),
34            ),
35          ],
36        ),
37      ],
38    );
39  }
40 }
```

The IDE interface includes a top toolbar with icons for running, debugging, and other actions. The bottom status bar shows the time as 25:15, the file encoding as UTF-8, and the Git status as master.

## Criando widgets

Utilize o widget criado no código principal do app.

Não esqueça de importar o arquivo que contém o widget criado.

```
margin: EdgeInsets.all(10),
child: Column(
  mainAxisAlignment: MainAxisAlignment.start,
  crossAxisAlignment: CrossAxisAlignment.start,
  children: <Widget>[
    SizedBox(height: 32),
    HeaderWidget(),
    SizedBox(height: 32),
    Text(
      'Experiencia: ',
      style: TextStyle(
        fontSize: 22,
        fontWeight: FontWeight.w700,
        color: colors[colorPos],
      ),
    ),
    SizedBox(height: 16),
    Text(
      'MOCKA, Porto Alegre – Desenvolvedor Android',
      style: TextStyle(fontSize: 16),
    ),
    SizedBox(height: 16),
    Text(
      'QueenMob, Porto Alegre – Desenvolvedor Android',
      style: TextStyle(fontSize: 16),
    ),
    SizedBox(height: 16),
    Text(
      'Apple Academy, Porto Alegre – Desenvolvedor iOS',
      style: TextStyle(fontSize: 16),
    ),
    SizedBox(height: 32),
    Text(
      'EDUCAÇÃO: ',
      style: TextStyle(
        fontSize: 22,
        fontWeight: FontWeight.w700,
        color: colors[colorPos],
      ),
    ),
  ],
),
```

## Navigator

Crie um novo widget que será a nova tela (de preferência em um novo arquivo).

Utilize `Navigator.push` para abrir a nova tela e `Navigator.pop` para fecha-la.

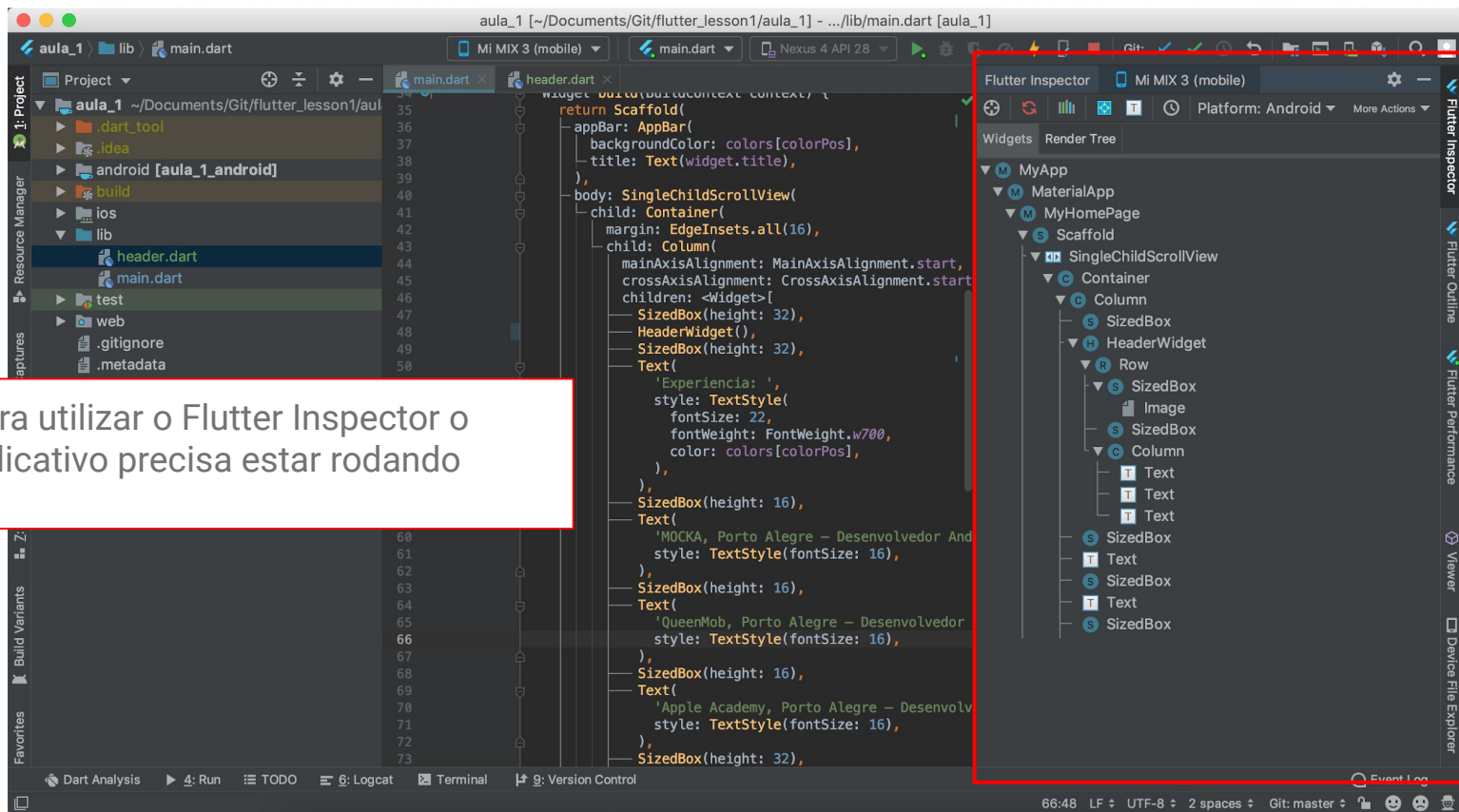
```
onPressed: () {  
  Navigator.push(  
    context,  
    MaterialPageRoute(builder: (context) => SecondRoute()),  
  );  
}
```

```
onPressed: () {  
  Navigator.pop(context);  
}
```

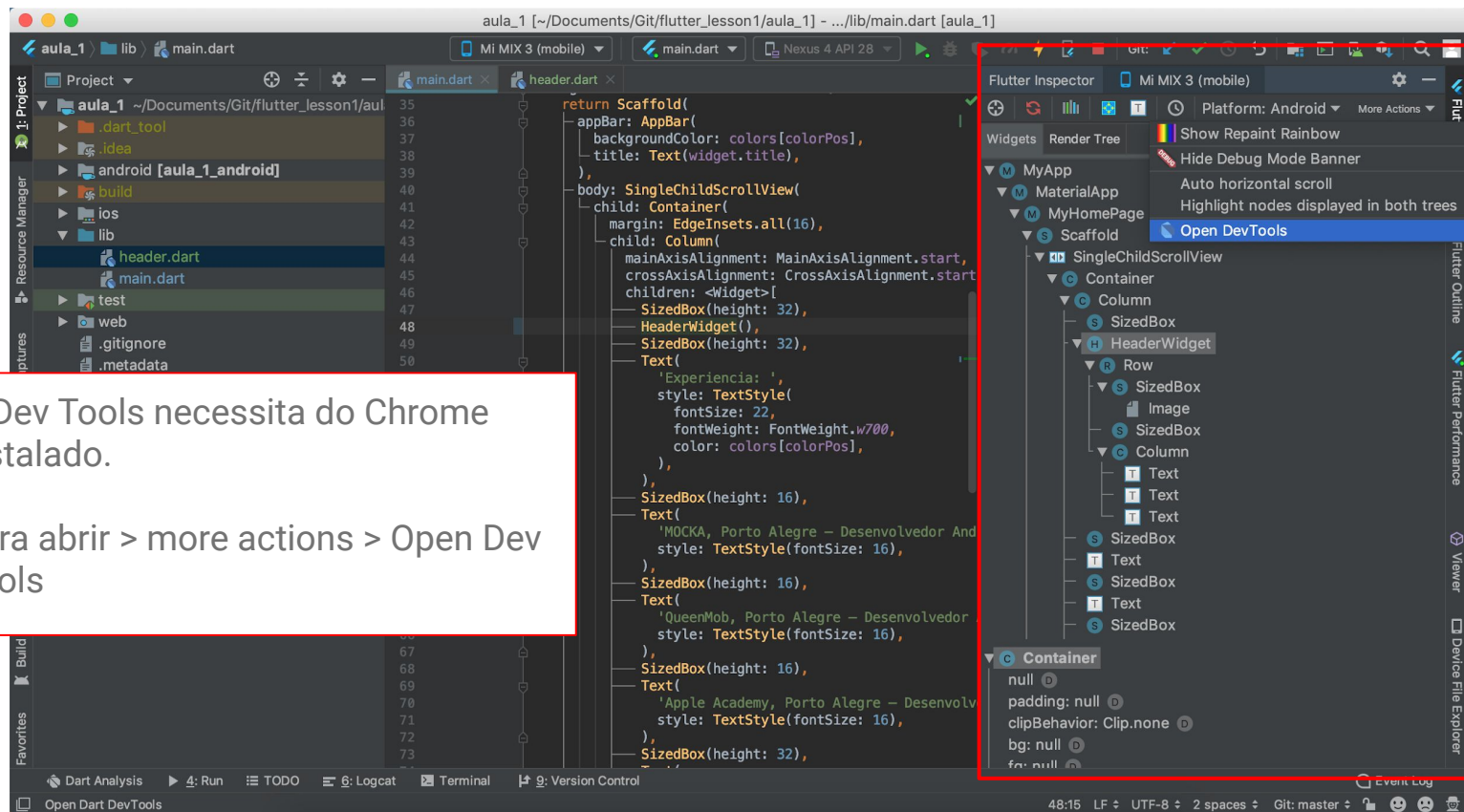


# Mais ferramentas de desenvolvimento - Flutter Inspector

Para utilizar o Flutter Inspector o aplicativo precisa estar rodando



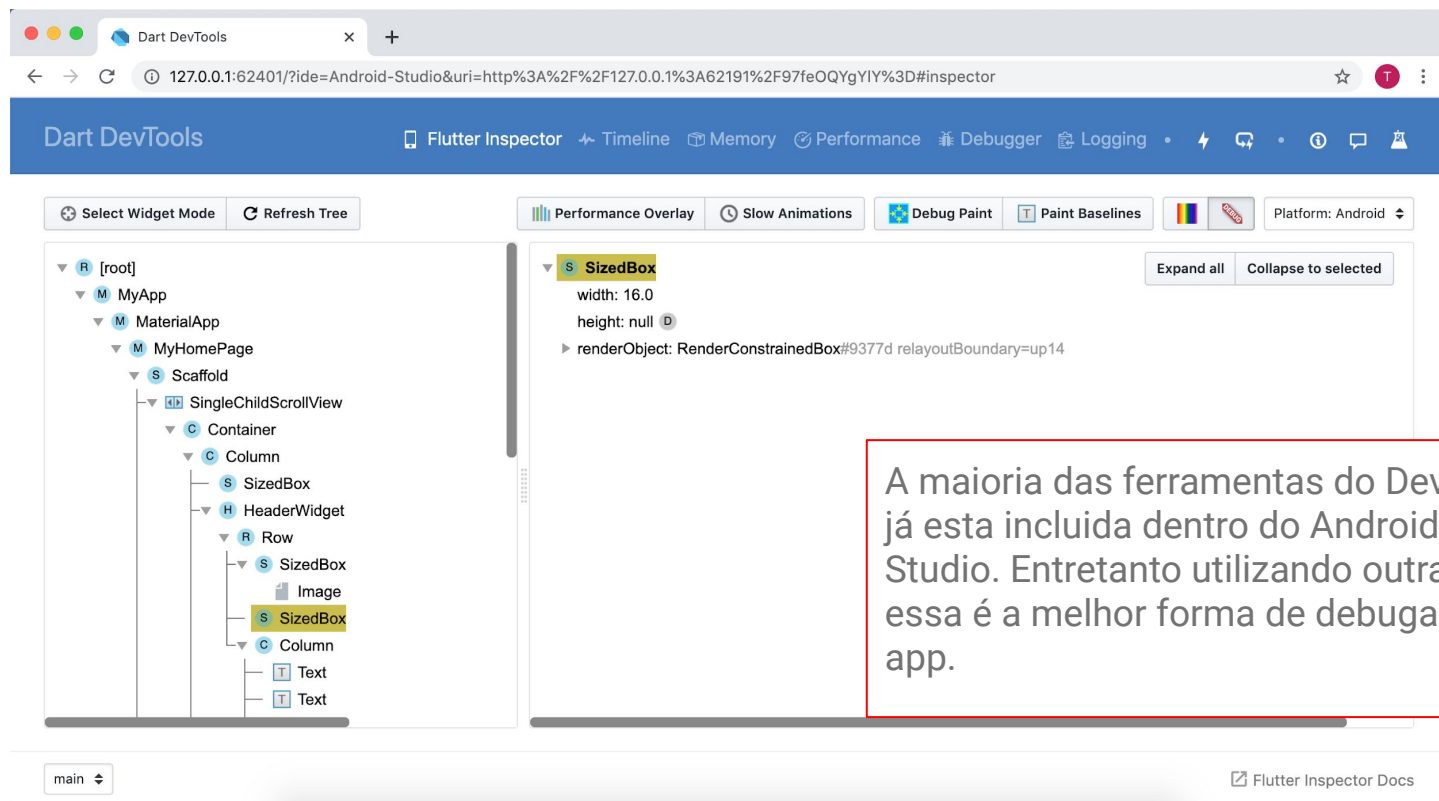
# Mais ferramentas de desenvolvimento - Flutter Inspector



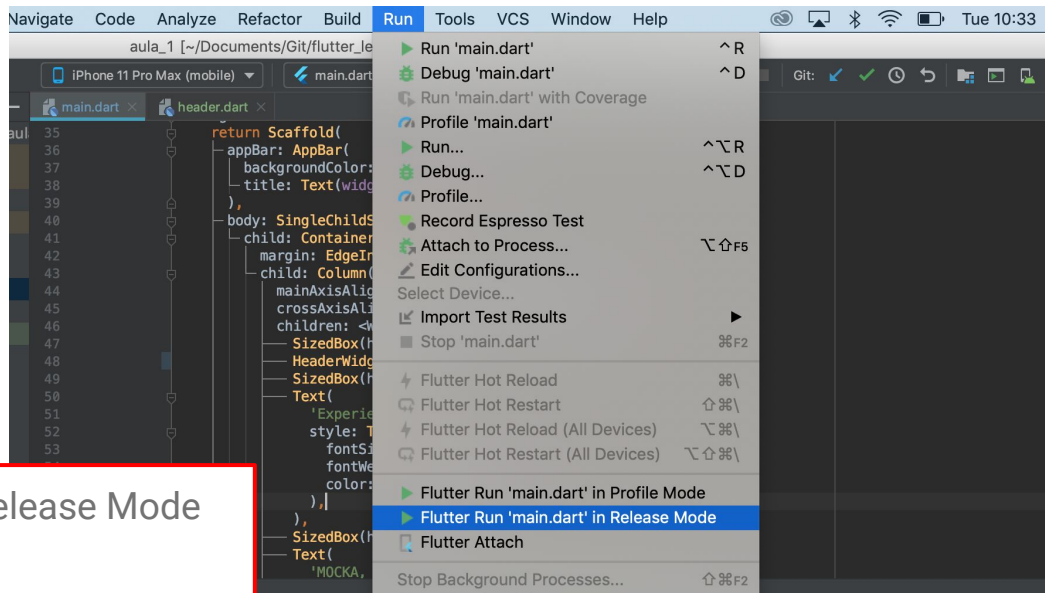
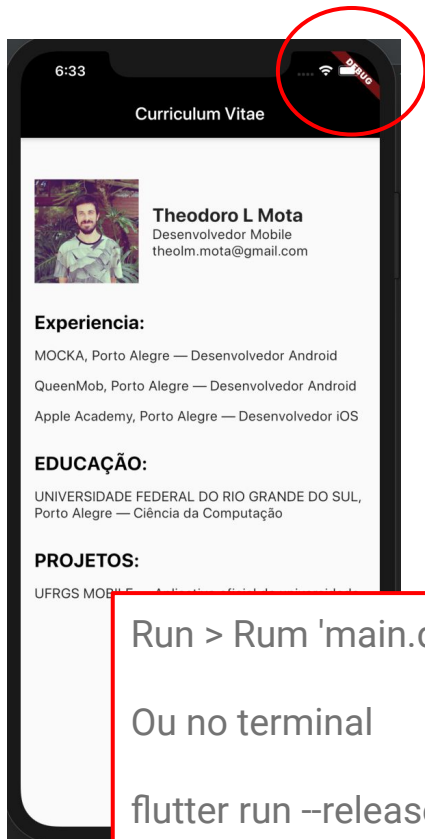
O Dev Tools necessita do Chrome instalado.

Para abrir > more actions > Open Dev Tools

# Mais ferramentas de desenvolvimento - Dev Tools



# Release vs. Debug

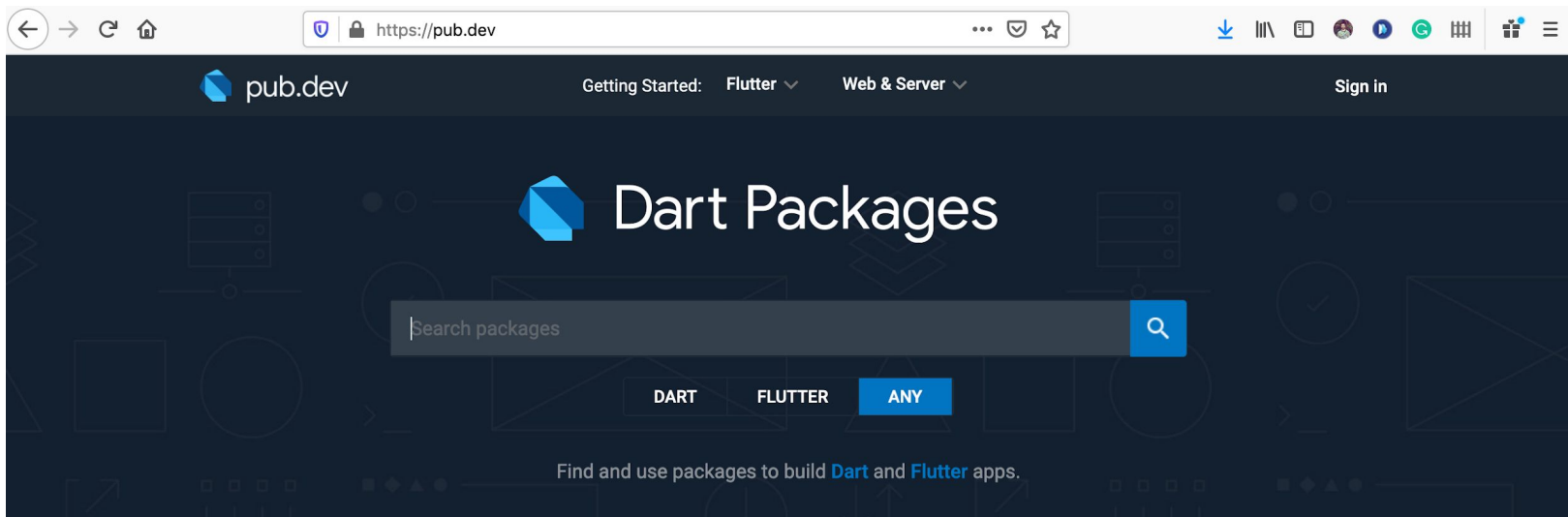


Run > Run 'main.dart' in Release Mode

Ou no terminal

`flutter run --release`

# Plugins



## Flutter Favorites

[flutter\\_slidable](#)

[romainrastel.com](#)

FLUTTER

A Flutter implementation of slidable list item with directional slide actions that can be dismissed.

[battery](#)

[flutter.dev](#)

FLUTTER

Flutter plugin for accessing information about the battery state (full, charging, discharging) on Android and iOS.

[sqflite](#)

[tekartik.com](#)

FLUTTER

Flutter plugin for SQLite, a self-contained, high-reliability, embedded, SQL database engine.

# Plugins

The screenshot shows the web interface for the `shared_preferences` package version 0.5.6 on pub.dev. The browser address bar shows the URL `https://pub.dev/packages/shared_preferences#-installing-tab-`. The page header includes the pub.dev logo, navigation links for 'Getting Started: Flutter' and 'Web & Server', and a 'Sign in' button. A search bar is present with the text 'Search Flutter packages'. The package name 'shared\_preferences 0.5.6' is prominently displayed, along with its publication date 'Dec 11, 2019', the publisher 'flutter.dev', and '282 likes'. A circular 'Flutter Favorite' badge is on the right. Below the package name are tabs for 'FLUTTER', 'ANDROID', 'IOS', and 'WEB'. A secondary row of tabs includes 'Readme', 'Changelog', 'Example', 'Installing' (which is highlighted with a red box and an underline), 'Versions', and a '100' badge (also highlighted with a red box). The main content area is titled 'Use this package as a library' and contains a section '1. Depend on it' with the instruction 'Add this to your package's pubspec.yaml file:'. A code block, highlighted with a red box, shows the following YAML snippet:

```
dependencies:  
  shared_preferences: ^0.5.6
```

On the right side, a 'Publisher' section identifies 'flutter.dev' as the publisher. An 'About' section describes the package as a 'Flutter plugin for reading and writing simple key-value pairs. Wraps NSUserDefaults on iOS and SharedPreferences on Android.' and provides links to the 'Homepage', 'Repository (GitHub)', 'View/report issues', and 'API reference'.

# Plugins

The screenshot shows an IDE window titled 'pubspec.yaml'. The 'Flutter commands' tab is active, displaying a list of commands: 'Packages get' (highlighted with a red box), 'Packages upgrade', 'Flutter upgrade', and 'Flutter doctor'. The main editor area shows the contents of the 'pubspec.yaml' file, which is a YAML configuration for a Flutter project. The file includes fields for 'name', 'description', 'version', 'environment', 'dependencies', 'dev\_dependencies', and 'flutter'. A red box highlights the 'shared\_preferences' dependency on line 14, with a comment in Portuguese: '# Adicionar o plugin desejado como dependência'. Two text boxes provide additional context: one explains that after modifications to the file, a 'packages get' command is necessary, and the other states that the 'pubspec.yaml' file contains information used in the app build, such as dependencies, version number, and build number.

```
1  name: aula_1
2  description: Primeira aula do curso de Flutter.
3
4  version: 1.0.0+1
5
6  environment:
7    sdk: ">=2.1.0 <3.0.0"
8
9  dependencies:
10   flutter:
11     sdk: flutter
12
13   cupertino_icons: ^0.1.2
14   shared_preferences: ^0.5.6 # Adicionar o plugin desejado como dependência
15
16  dev_dependencies:
17   flutter_test:
18     sdk: flutter
19
20
21  flutter:
22    uses-material-design: true
23
24
```

Após fazer modificações no arquivo é necessário fazer um "packages get"

O arquivo pubspec.yaml contém informações utilizadas na build do app como dependencias version number e build number.

## Shared preferences - Salvando

```
_incrementCounter() async {  
  SharedPreferences prefs = await SharedPreferences.getInstance();  
  int counter = (prefs.getInt('counter') ?? 0) + 1;  
  print('Pressed $counter times.');
```

```
  await prefs.setInt('counter', counter);  
}
```

Primeiro passo é obter uma instância do SharedPreferences.

O método getInstance retorna um Future, para obter a instância do sharedPreferences é necessário usar await.

O sharedPreferences fornece os seguintes métodos: setString, setBool, setDouble, setInt, setStringList



## Shared preferences - Recuperando dados

```
SharedPreferences prefs = await SharedPreferences.getInstance();  
var counter = prefs.getInt('counter');
```

Tendo acesso a uma instância do SharedPreferences basta utilizar os métodos apropriados para obter o elemento salvo.

**Cuidado: caso o valor não exista o retorno vai ser null.**

O sharedPreferences fornece os seguintes métodos: getString, getBool, getDouble, getInt, getStringList.

## Shared preferences

Como salvar um objeto ou uma lista de objetos?

Por exemplo: classe **Person** ou **List<Person>**

O SharedPreferences não é preparado para salvar dados dessa forma.

Como fazer?

# Serialização de dados

## Serialização de dados

### Objeto em dart

```
Servidor(  
  "Thiago",  
  38,  
  "Programador"  
);
```



### String (json)

```
{"Nome": "Thiago",  
  "Idade": 38,  
  "Cargo": "programador"  
};
```

## Serialização de dados

### Objeto em dart

```
Servidor(  
  "Thiago",  
  38,  
  "Programador"  
);
```

????

### String (json)

```
{  
  "Nome" : "Thiago",  
  "Idade": 38,  
  "Cargo": "programador"  
};
```

## Serialização de dados - Método 1

Crie um arquivo dart e nele  
a classe que você deseja  
Serializar/desserializar

```
class Servidor {  
  final String name;  
  final int idade;  
  final String cargo;  
  
  Servidor(this.name, this.idade, this.cargo);  
}
```

## Serialização de dados - Método 1

Adicione na classe o método toJson() como indicado ao lado.

O método toJson retorna o Map que contém o método toString().

```
class Servidor {  
    final String name;  
    final int idade;  
    final String cargo;  
  
    Servidor(this.name, this.idade, this.cargo);  
  
    Map<String, dynamic> toJson() => {  
        "name": name,  
        "idade": idade,  
        "cargo": cargo,  
    };  
}
```

# Serialização de dados - Método 2

← → ↺ 🏠

app.quicktype.io

★ 🔴 🟢 🟠

Show apps

Apps ★ Bookmarks 📁 IPTV 📁 Flutter 📁 3D 📁 Android 📁 Netflix 📁 FileWarez 📁 The Red King, List1 | 📁 GPS Mount on Lipo... 📁 Ublox M8N GPS ba... » 📁 Other Bookmarks

QT quicktype

Please Share! 👁 Options 👤 Sign In ? ⋮

Name

servidor

Source type

JSON

```
{
  "name": "Thiago",
  "idade": 132,
  "cargo": "programador"
}
```

```
// To parse this JSON data, do
//
//      final servidor = servidorFromJson(jsonString);

import 'dart:convert';

Servidor servidorFromJson(String str) => Servidor.fromJson(json.decode(str));

String servidorToJson(Servidor data) => json.encode(data.toJson());

class Servidor {
  String name;
  int idade;
  String cargo;

  Servidor({
    this.name,
    this.idade,
    this.cargo,
  });

  factory Servidor.fromJson(Map<String, dynamic> json) => {
    name: json["name"],
    idade: json["idade"],
    cargo: json["cargo"],
  });

  Map<String, dynamic> toJson() => {
    "name": name,
    "idade": idade,
    "cargo": cargo,
  });
}
```

Language Other

Dart

☐ Types only

☐ Put encoder & decoder in Class

☐ Use method names from Map() & toMap()

☐ Make all properties required

☐ Make all properties final

☐ Generate CopyWith method

☐ Make all properties optional ?

Share Link to Code

Copy Code

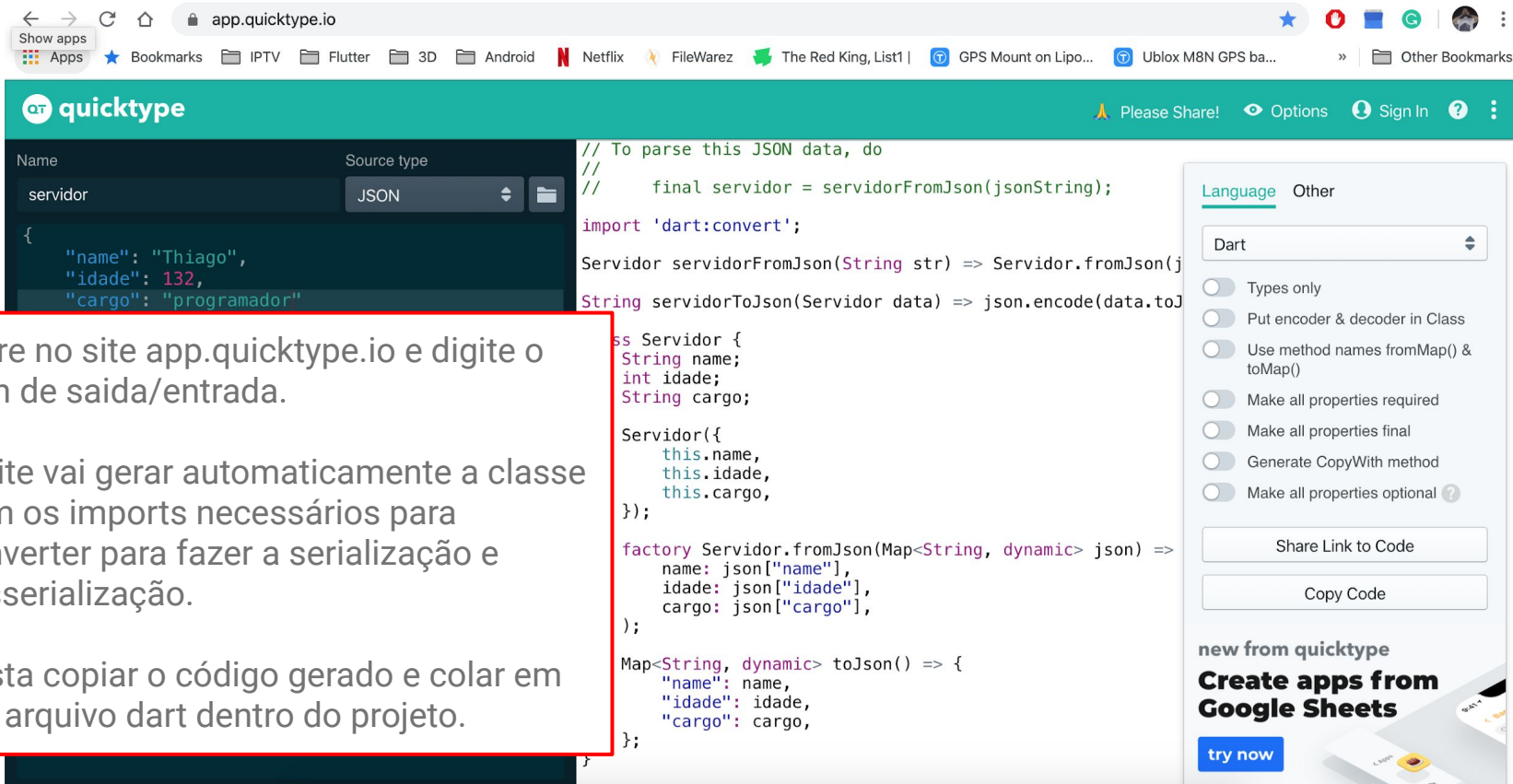
new from quicktype

Create apps from Google Sheets

try now



# Serialização de dados - Método 2



The screenshot shows the app.quicktype.io website. In the top left, there's a navigation bar with 'Show apps' and 'Apps'. Below it, a list of categories: 'Apps', 'Bookmarks', 'IPTV', 'Flutter', '3D', 'Android', 'Netflix', 'FileWarez', 'The Red King, List1', 'GPS Mount on Lipo...', 'Ublox M8N GPS ba...', and 'Other Bookmarks'. The main header is teal with the 'quicktype' logo and links for 'Please Share!', 'Options', 'Sign In', and a help icon. The interface is split into three main sections. On the left, a 'Name' field contains 'servidor' and a 'Source type' dropdown is set to 'JSON'. Below this, a JSON object is displayed: 

```
{  "name": "Thiago",  "idade": 132,  "cargo": "programador"}
```

. The middle section shows the generated Dart code. It starts with a comment: `// To parse this JSON data, do`, followed by `// final servidor = servidorFromJson(jsonString);`. Then, it imports `'dart:convert';` and defines `Servidor` as a class with fields `String name;`, `int idade;`, and `String cargo;`. The class has a constructor `Servidor({ this.name, this.idade, this.cargo, });`. There are two factory methods: `factory Servidor.fromJson(Map<String, dynamic> json) => { name: json["name"], idade: json["idade"], cargo: json["cargo"], };}` and `Map<String, dynamic> toJson() => { "name": name, "idade": idade, "cargo": cargo, };}`. The right section has a 'Language' dropdown set to 'Dart' and a list of options: 

- ☐ Types only
- ☐ Put encoder & decoder in Class
- ☐ Use method names from Map() & toMap()
- ☐ Make all properties required
- ☐ Make all properties final
- ☐ Generate CopyWith method
- ☐ Make all properties optional ?

At the bottom of this section are buttons for 'Share Link to Code' and 'Copy Code'. A promotional banner at the bottom right says 'new from quicktype Create apps from Google Sheets try now'.

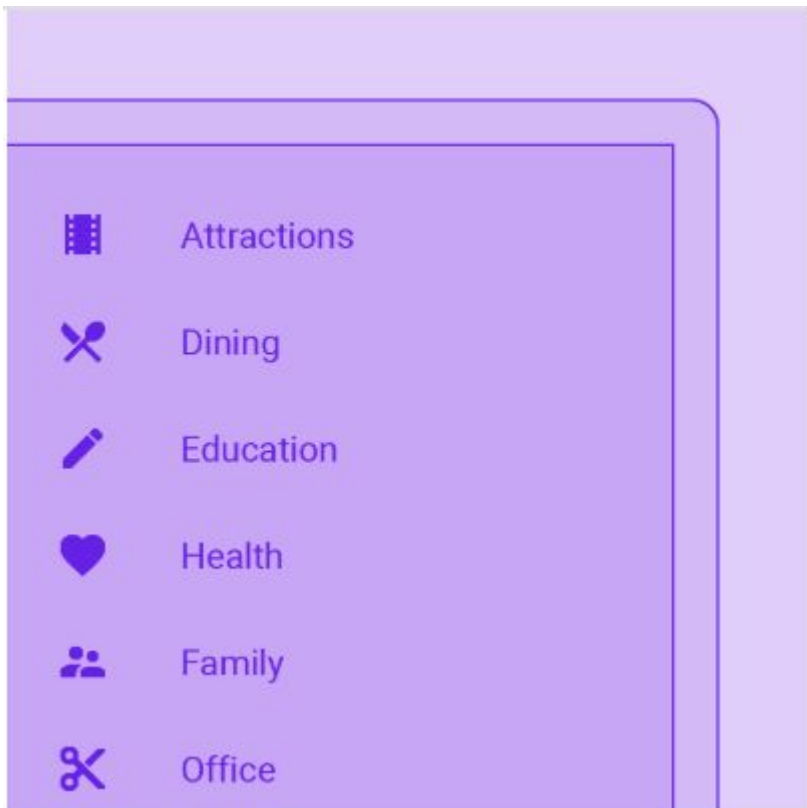
Entre no site app.quicktype.io e digite o json de saída/entrada.

O site vai gerar automaticamente a classe com os imports necessários para converter para fazer a serialização e desserialização.

Basta copiar o código gerado e colar em um arquivo dart dentro do projeto.

**Mais widgets!!**

## ListView



Provavelmente o widget mais importante até agora.

Existem diversas formas de utilizá-lo.

Com uma lista explícita de widgets ou dinamicamente através de um builder.

## ListView

```
Widget _createListView() {  
  return ListView(  
    children: <Widget>[  
      Text('linha 1'),  
      Text('linha 2'),  
      Text('linha 3'),  
      Text('linha 4'),  
    ],  
  );  
}
```

No exemplo ao lado criamos uma listView passando dinamicamente 4 widgets pelo construtor. Esses widgets não precisam ser iguais e podem ser tanto stateless quanto stateful.

## ListView

```
Widget _createListView() {  
  var list = ['Thiago', 'Devanir', 'Abel', 'Victor'];  
  return ListView.builder(  
    itemCount: list.length,  
    itemBuilder: (BuildContext context, int position) {  
      return Text(list[position]);  
    }  
  );  
}
```

No exemplo ao lado criamos uma listView utilizando um builder.

Esse método é muito utilizado quando precisamos criar widgets dinamicamente.

No exemplo temos uma lista de nomes e no itemBuilder recebemos o contexto e a posição da linha sendo construída. Assim podemos gerar os widgets buscando as informações da lista.

# Async & Await

## Async & Await

### Para que serve?

**async** é o indicativo para o framework de que essa função vai rodar assincronamente.

**await** é o indicador que faz esperar o resultado de uma função assíncrona.

### O que é Future?

Future representa o resultado de uma função assíncrona e pode ter dois estados: completo e incompleto.

## Async & Await

```
Future<String> _retornaStringAposSegundos() async {  
  await Future.delayed(Duration(seconds: 10));  
  return "Essa string retorna após 10 segundos";  
}
```

No exemplo acima temos uma função que se invocada diretamente vai retornar um **Future<String> incompleto**.

Entretanto se fizermos a chamada com **await** a execução vai esperar por 10 segundos (delay da função) e retornar a string esperada.

Isso é extremamente útil para fazer requisições na internet, escrever em um DB ou ler um arquivo.



# Objetivo: TODO list app

Código disponível no git:

[https://github.com/theolm/flutter\\_lesson2](https://github.com/theolm/flutter_lesson2)

