



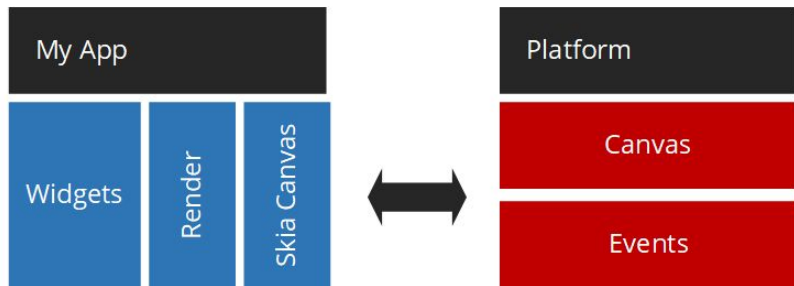
Aula 3 - Comunicação com API

Aula 3 - Index

1. Como funciona?
2. Relembrando.
3. O que é API?
4. Objetivo da aula: TODO list + API.
5. Endpoints.
6. Http Dart package.
7. Postman
8. Mais widgets: FutureBuilder
9. Loader

Como funciona?

O Flutter é construído de uma maneira totalmente nova, em comparação com outros frameworks, funcionando mais como um engine de jogo do que com uma estrutura de aplicativo tradicional.



- O aplicativo em Flutter é composto de widgets, que são renderizados em um canvas do Skia e enviados para a plataforma. A plataforma mostra a tela e envia os eventos de volta, conforme necessário.
- **Skia** é uma biblioteca gráfica 2D de código aberto que fornece APIs que funcionam em uma variedade de plataformas de hardware e software.

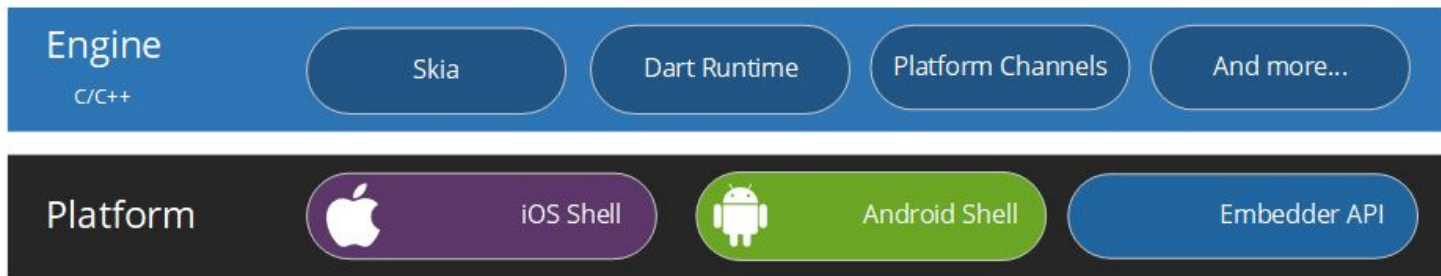
Como funciona?



Começando no nível da plataforma, o Flutter fornece um Shell, que hospeda a Dart VM. O Shell é específico da plataforma, fornecendo acesso às APIs da plataforma nativa e hospedando o canvas da plataforma.

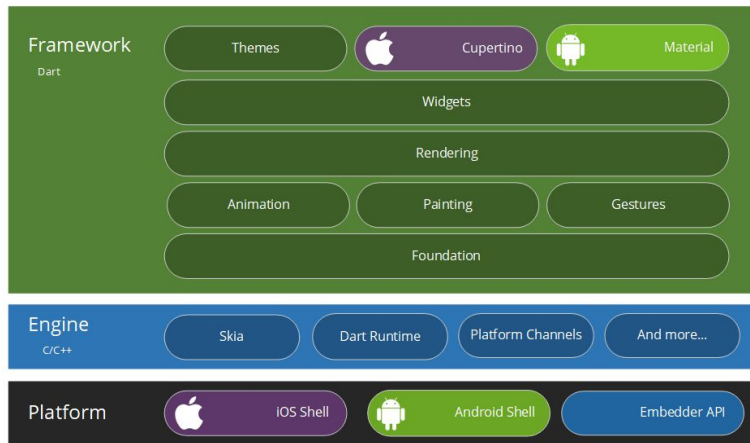
O Shell também ajuda a fornecer comunicação aos IMEs (input method editor) relevantes (e.g. Teclado) e aos eventos do ciclo de vida do aplicativo de sistemas.

Como funciona?



O Flutter Engine é a próxima camada, fornecendo Dart Runtime, Skia, Platform Channels (sistema de arquivos e rede, suporte à acessibilidade, arquitetura de plugins, etc).

Como funciona?



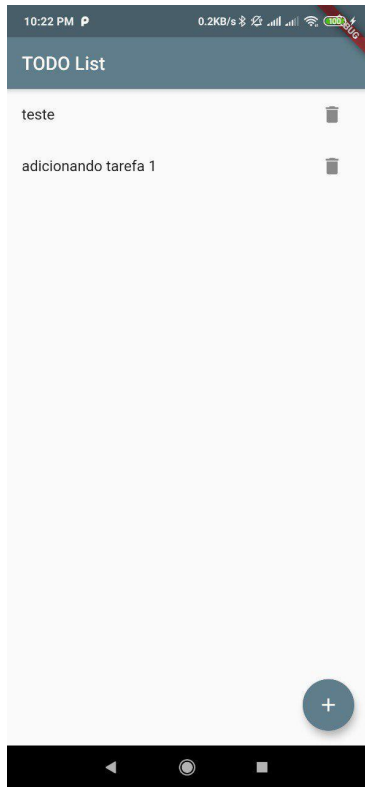
O Flutter framework é a camada mais relevante para o desenvolvedor. Ela contém tudo o que você interagirá ao desenvolver seu aplicativo.

Como funciona?

O Flutter funciona mais como uma engine de jogo do que com uma estrutura de aplicativo android ou iOS.

A interface do usuário é criada e renderizada em um Skia Canvas e atualizada a 60fps. A GPU faz praticamente todo o trabalho e é o motivo pelo qual a interface do usuário funciona rápida e sem travamentos.

Relembrando



- Navigator
 - Push
 - Pop
- Release vs. Debug
- Packages
- Serialização de dados
- ListView
- Async & Await

Navigator

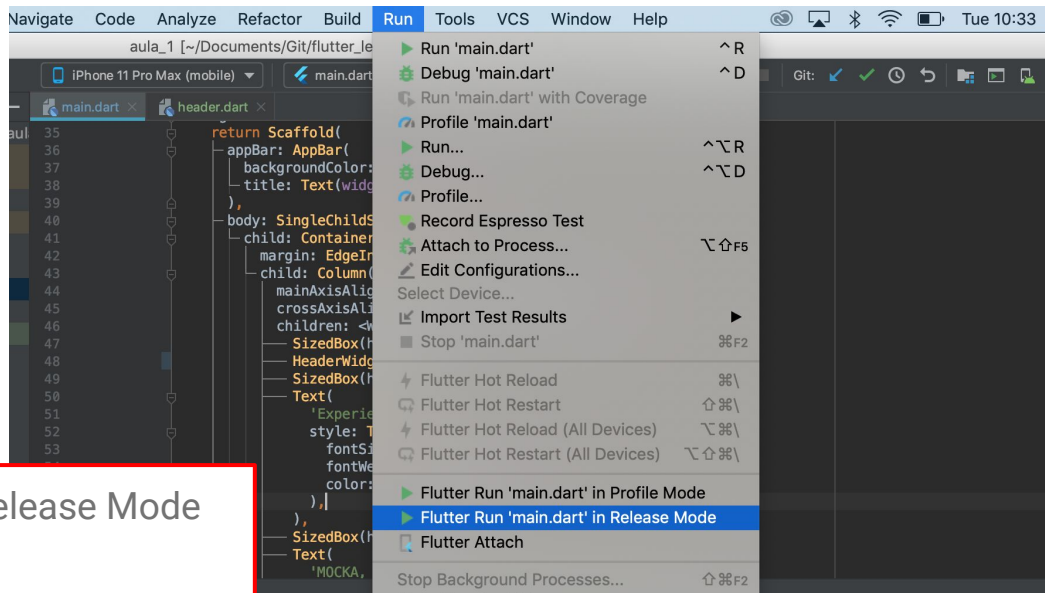
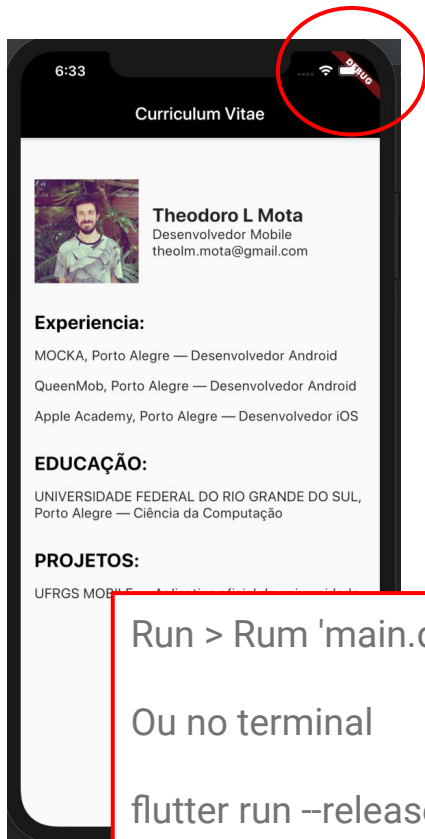
Crie um novo widget que será a nova tela (de preferência em um novo arquivo).

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);  
}
```

Release vs. Debug



Run > Run 'main.dart' in Release Mode

Ou no terminal

`flutter run --release`

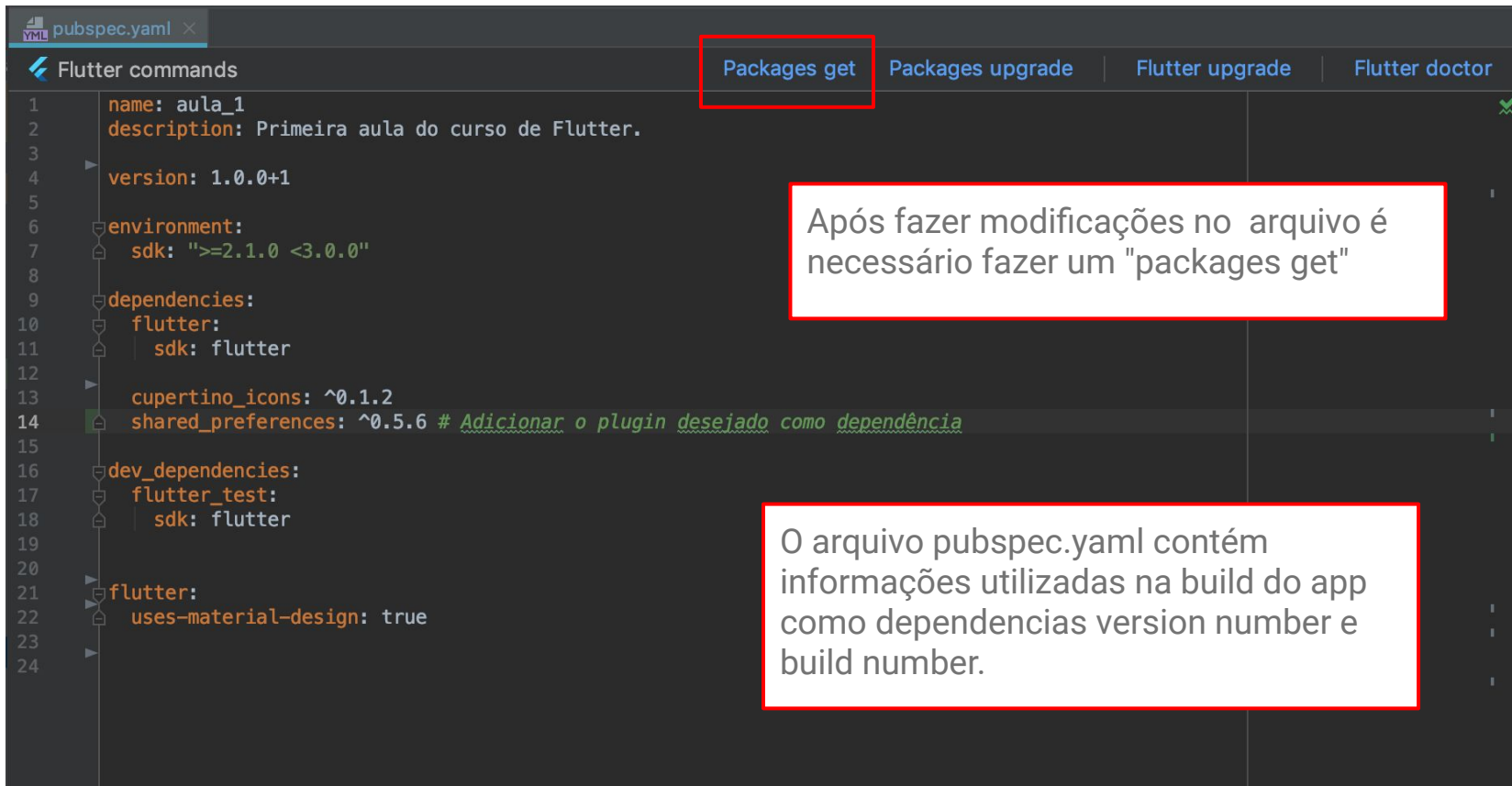
Packages

Pacotes permitem a criação de código modular que pode ser compartilhado facilmente. Existem 2 tipos de Packages:

Dart Packages: Pacote contendo apenas código dart. Como uma lib.

Plugin Packages: Um pacote Dart que contém uma API escrita em código Dart combinada com uma implementação específica da plataforma para Android (usando Java ou Kotlin) e / ou para iOS (usando ObjC ou Swift)

Packages



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 includes metadata (name, description, version), environment (sdk), dependencies (flutter, cupertino_icons, shared_preferences), dev_dependencies (flutter_test), and flutter-specific settings (uses-material-design). 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 'packages get' is necessary after modifications, and the other states that the 'pubspec.yaml' file contains build information like version 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.

Serialização de dados

Objeto em dart

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



String (json)

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

Nesta aula vai ser muito útil para transformar o retorno da api em uma classe Dart que possa ser facilmente manipulada dentro do app.

ListView

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

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

Dinâmica e explícita.

Além disso existe um builder com a separated, que permite incluir divisórias entre os itens.

Async & Await

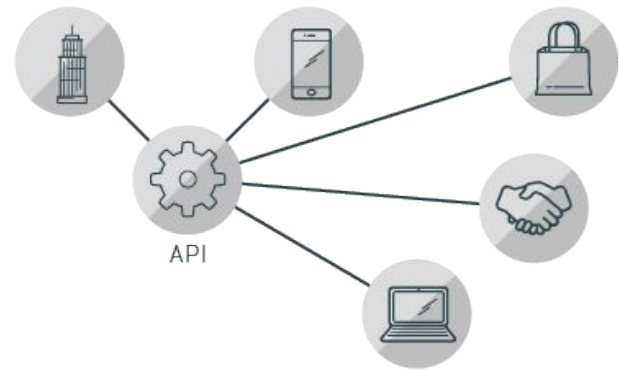
```
Future<String> _retornaStringAposSegundos() async {  
  await Future.delayed(Duration(seconds: 10));  
  return "Essa string retorna após 10 segundos";  
}
```

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.

API

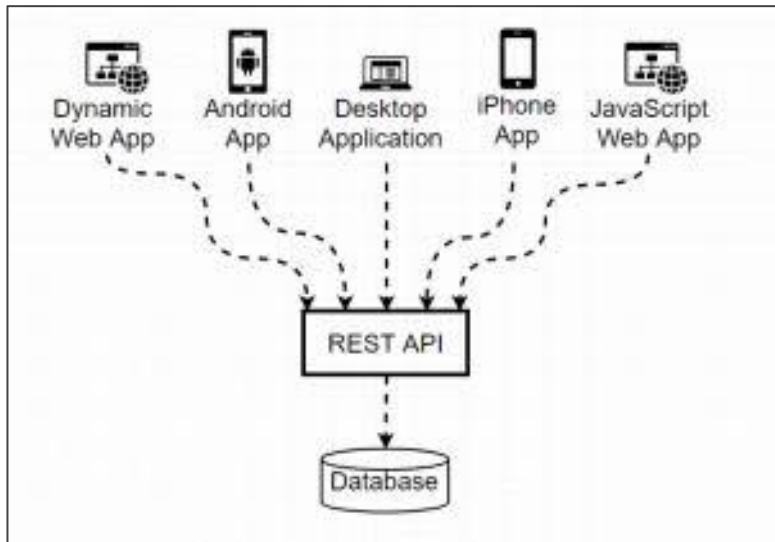


Application Programming Interface

API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web.

Uma API é criada quando uma empresa de software tem a intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço.

REST API

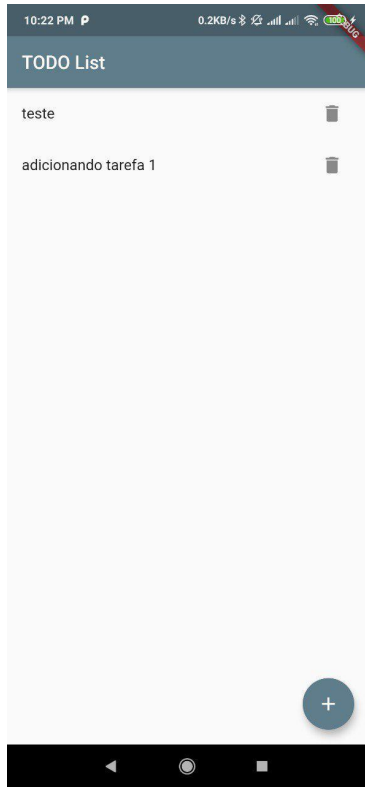


A figura ao lado ilustra a importância de uma API REST no desenvolvimento mobile.

Em geral não se armazena (ou processa) muitas informações no app, isso é delegado para um servidor que cuida do "trabalho pesado".

TODO list + API

TODO list + API



Modificar o aplicativo feito na aula anterior e substituir o armazenamento local por armazenamento remoto utilizando a API fornecida.

Endpoints

Documentação:

<https://documenter.getpostman.com/view/476996/SWTA9xjh?version=latest#52dec620-2231-4d5f-a3d7-a2cea9d4844f>

Listar tarefas:

POST: miaula.tsmotta.com/todolist/list

Criar tarefa:

POST: miaula.tsmotta.com/todolist/task

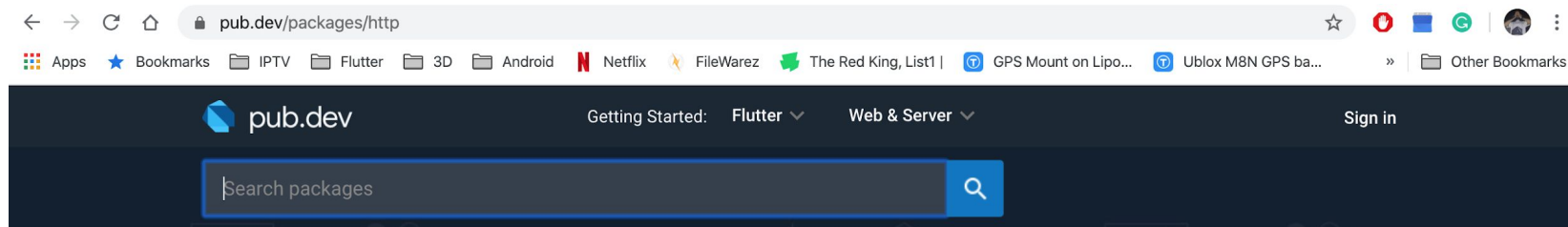
Editar tarefa:

POST: miaula.tsmotta.com/todolist/done

Apagar tarefa:

POST: miaula.tsmotta.com/todolist/delete

Http Dart package



http 0.12.0+4

Published Jan 7, 2020 • dart.dev 160 likes

DART NATIVE JS **FLUTTER** ANDROID IOS WEB

Readme Changelog Example Installing Versions

A composable, Future-based library for making HTTP requests.

pub **v0.12.0+4** build **passing**

This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's platform-independent, and can be used on both the command-line and the browser.

Using

The easiest way to use this library is via the top-level functions. They allow you to make

Instalar o package http.
Ele vai ser o responsável por fazer as chamadas a API.

Não esquecer de dar package get.

Publisher

dart.dev

About

A composable, multi-platform, Future-based API for HTTP requests.

[Repository \(GitHub\)](#)

[View/report issues](#)

[API reference](#)

License

BSD (4-clause)

Http Dart package - GET

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

```
void getApi() async {  
  print('Fez chamada');  
  var url = 'https://jsonplaceholder.typicode.com/posts';  
  var response = await http.get(url);  
  print('Response status: ${response.statusCode}');  
  print('Response body: ${response.body}');  
}
```

1. Importar o package como mostrado ao lado.
2. Criar função assíncrona para tratar o request.
3. Obter a resposta do endpoint.
4. No response.body tem uma string que representa o retorno. No caso de uma API Rest essa string vai ser um json.

Http Dart package - GET

```
import 'dart:convert' as JSON;  
  
final json=JSON.jsonDecode(myJsonAsString);
```

Podemos utilizar o package convert para transformar a string de retorno em um Map (ou List<Map>) e acessar os atributos conforme foi visto na primeira aula.

Entretanto a maneira mais "elegante" seria desserializar a string de retorno e trata-la como um objeto dart.

Http Dart package - POST

```
import 'package:http/http.dart' as http;

var url = 'https://example.com/whatsit/create';
var response = await http.post(url, body: {'name': 'doodle', 'color': 'blue'});
print('Response status: ${response.statusCode}');
print('Response body: ${response.body}');
```

Quando enviamos um post também podemos enviar um body que nada mais é que um Map representando um json. No exemplo acima o json enviado no body é:

```
{
  "name": "doodle",
  "color": "blue"
}
```

Postman

The screenshot displays the Postman application window. The top bar includes a 'New' button, 'Import', 'Runner', and 'My Workspace' dropdown. The left sidebar shows a 'Collections' tab with a list of collections: 'New REST API' (4 requests), 'Parse R6' (1 request), 'R6Pro' (4 requests), 'Secad' (17 requests), 'Todo-list' (4 requests), and 'Twitter' (2 requests). The 'Todo-list' collection is expanded, showing a list of requests. The main panel shows a POST request to 'localhost/miaula/todolist/list?id=0&pass=123'. The request body is set to 'form-data' and contains two parameters: 'id' with value '5' and 'pass' with value '123'. The response is displayed in the bottom panel, showing a JSON object with two items: {id: 3, task: 'Testar API', done: false} and {id: 4, task: 'Testar API', done: false}. The status is 200 OK, time is 628ms, and size is 329 B.

Postman

New Import Runner +

My Workspace Invite

Filter

History Collections APIs BETA

+ New Collection Trash

POST localhost/miaula/todolist/list?id=0&pass=123

POST localhost/miaula/todolist/task

POST localhost/miaula/todolist/done

POST localhost/miaula/todolist/done

Twitter 2 requests

localhost/miaula/todolist/list

Send Save

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL BETA

| KEY | VALUE | DESCRIPTION |
|--|-------|-------------|
| <input checked="" type="checkbox"/> id | 5 | |
| <input checked="" type="checkbox"/> pass | 123 | |
| Key | Value | Description |

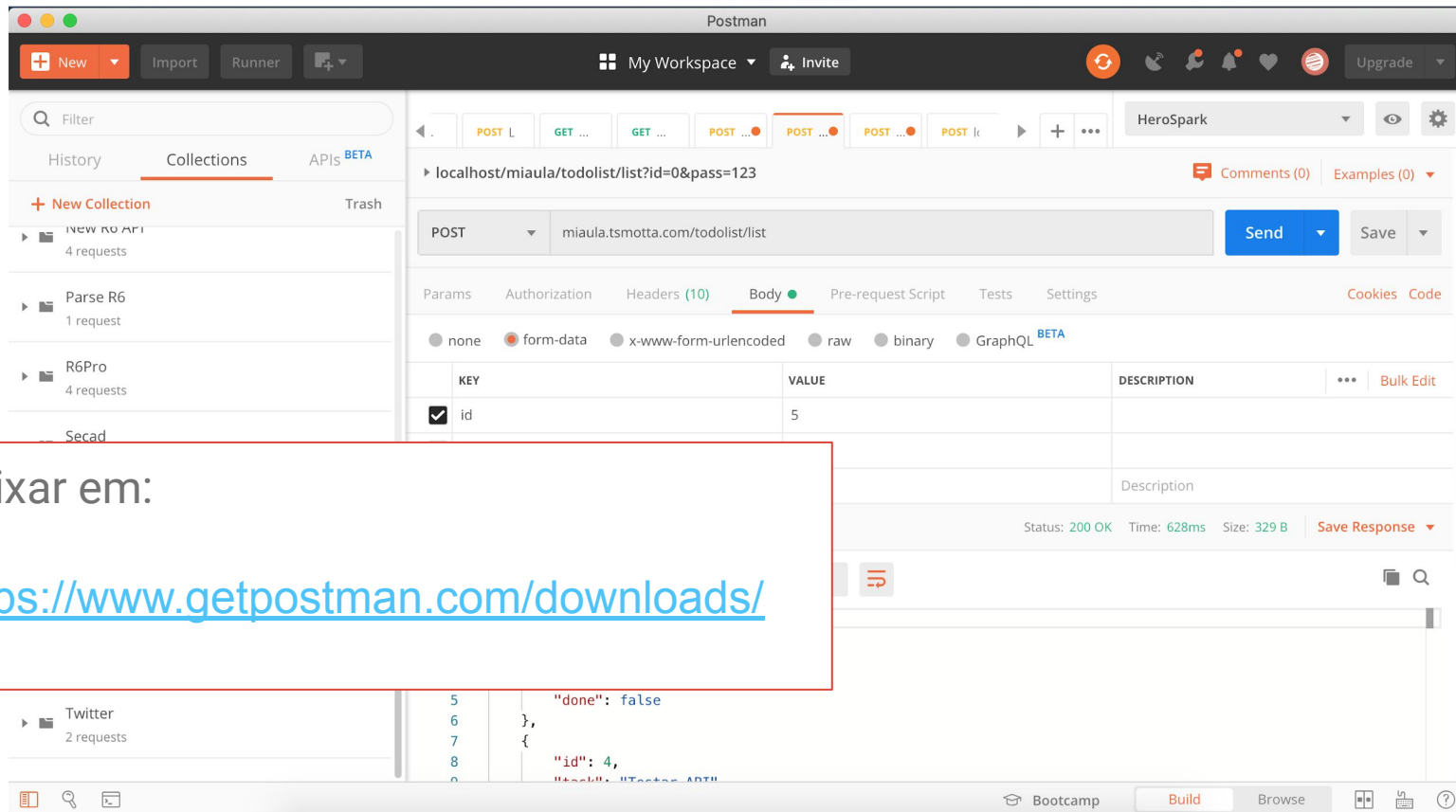
Body Cookies (1) Headers (9) Test Results Status: 200 OK Time: 628ms Size: 329 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 [
2   {
3     "id": 3,
4     "task": "Testar API",
5     "done": false
6   },
7   {
8     "id": 4,
9     "task": "Testar API",
10    "done": false
11  }
12 ]
```

Bootcamp Build Browse ?

Postman



Mais widgets!

FutureBuilder

É um widget que se reconstrói baseado no último snapshot disponibilizado por um Future.

```
class Teste extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      backgroundColor: Colors.white,  
      body: Center(  
        child: FutureBuilder(  
          future: getApi(),  
          builder: (context, snapshot) {  
            if(snapshot.hasData) {  
              ///Recebeu resposta do Future.  
              return Text('Recebeu info');  
            } else {  
              ///Ainda não recebeu resposta do Future.  
              return Text('Esperando');  
            }  
          },  
        ),  
      ),  
    );  
  }  
}
```

Mesmo sendo um Stateless widget, quando o estado do Future muda ele chama o builder novamente, podendo assim redesenhar a parte de si que depende de uma chamada assíncrona.

FutureBuilder

É um widget que se reconstrói baseado no último snapshot disponibilizado por um Future.

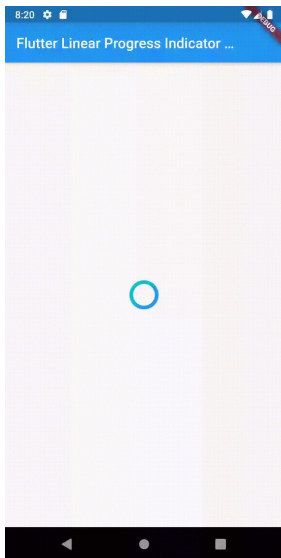
```
class Teste extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      backgroundColor: Colors.white,  
      body: Center(  
        child: Text('Esperando');  
      ),  
    );  
  }  
}
```

Esse widget é especialmente importante quando montamos telas baseadas em retorno de chamadas a APIs.

Mesmo sendo um Stateless widget, quando o estado do Future muda ele chama o builder novamente, podendo assim desenhar a parte de si que depende de uma chamada assíncrona.

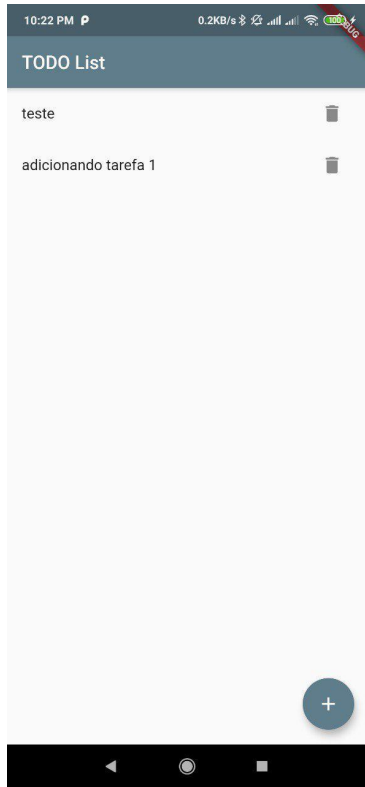
Loader

Sempre que temos uma chamada assíncrona que pode demorar devemos dar algum feedback para o usuário. Nestes casos uma combinação do FutureBuilder com o CircularProgressIndicator pode ser bastante interessante.



```
class Teste extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      backgroundColor: Colors.white,  
      body: Center(  
        child: FutureBuilder(  
          future: getApi(),  
          builder: (context, snapshot) {  
            if(snapshot.hasData) {  
              //Recebeu resposta do Future.  
              return Text('Recebeu info');  
            } else {  
              //Ainda não recebeu resposta do Future.  
              return CircularProgressIndicator();  
            }  
          },  
        ),  
      ),  
    );  
  }  
}
```

TODO list + API



Modificar o aplicativo feito na aula anterior e substituir o armazenamento local por armazenamento remoto utilizando a API fornecida.