Flutter Fundamentals
Async code

Peter Kassenaar –
info@kassenaar.com

Asynchronous code

Working with async code in your apps

Theory – short

- In Dart we use the `Future<T>` class to represent async operations/variables
  - A Future is like a Promise in JavaScript
  - We can use `.then()` if the Future is resolved
  - https://api.flutter.dev/flutter/dart-async/Future-class.html
  - Futures are non-blocking

```
Future<int> future = getFuture();
future
    .then((value) => handleValue(value))
    .catchError((error) => handleError(error));
```

Different syntax: `async/await`

```
void handleFuture() async {
  try {
    int value = await getFuture();
    handleValue(value);
  } catch (error) {
    handleError(error);
  }
}
```

1. The `async` keyword is added to the function, indicating that it contains asynchronous code and can use `await`.
2. `await` pauses execution until the future (here: `getFuture()`, a function defined elsewhere) completes and returns a value.
3. Since await can throw an exception, a `try-catch` block is used to handle errors.

This code is functionally equivalent to the original code but avoids chaining `.then()` and `.catchError()`, making it easier to follow

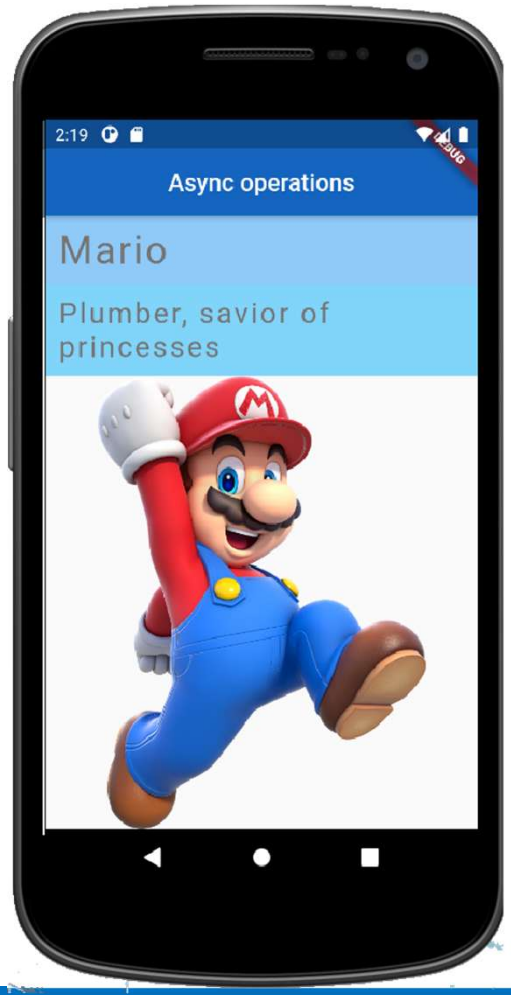
Future.delayed

- We can use `Future.delayed` to simulate a timeout
  - Like `setTimeout()` in JavaScript
  - The callback function is executed if the specified `Duration` has passed
- https://api.flutter.dev/flutter/dart-async/Future/Future.delayed.html

Example – a Home Screen

- Using static data, we can build a lay-out like this:

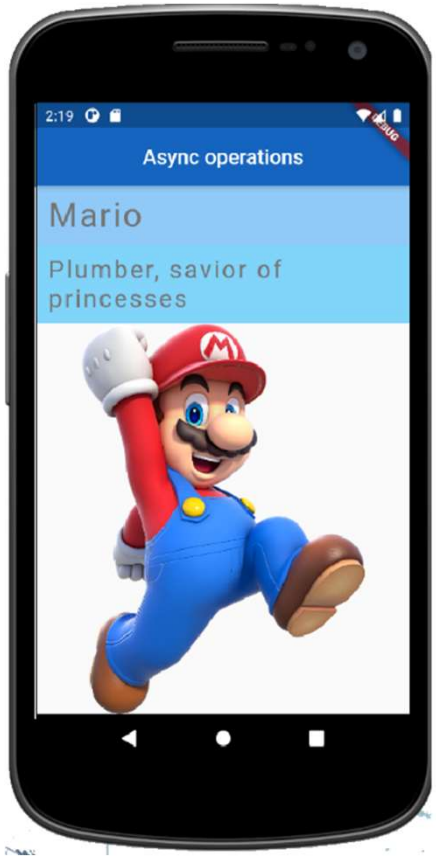
```
// 1. The data/state for our application
String name;
String occupation;
String image;


// 2. Lifecycle hook - it initializes our data
@override
void initState() {
  getPerson();
}

// 3. Fill the variables. If we update them, we have to
// use setState() of course.
void getPerson() {
  name = 'Mario';
  occupation = 'Plumber, savior of princesses';
  image = 'assets/mario.png';
}
```



Let’s simulate the results are async

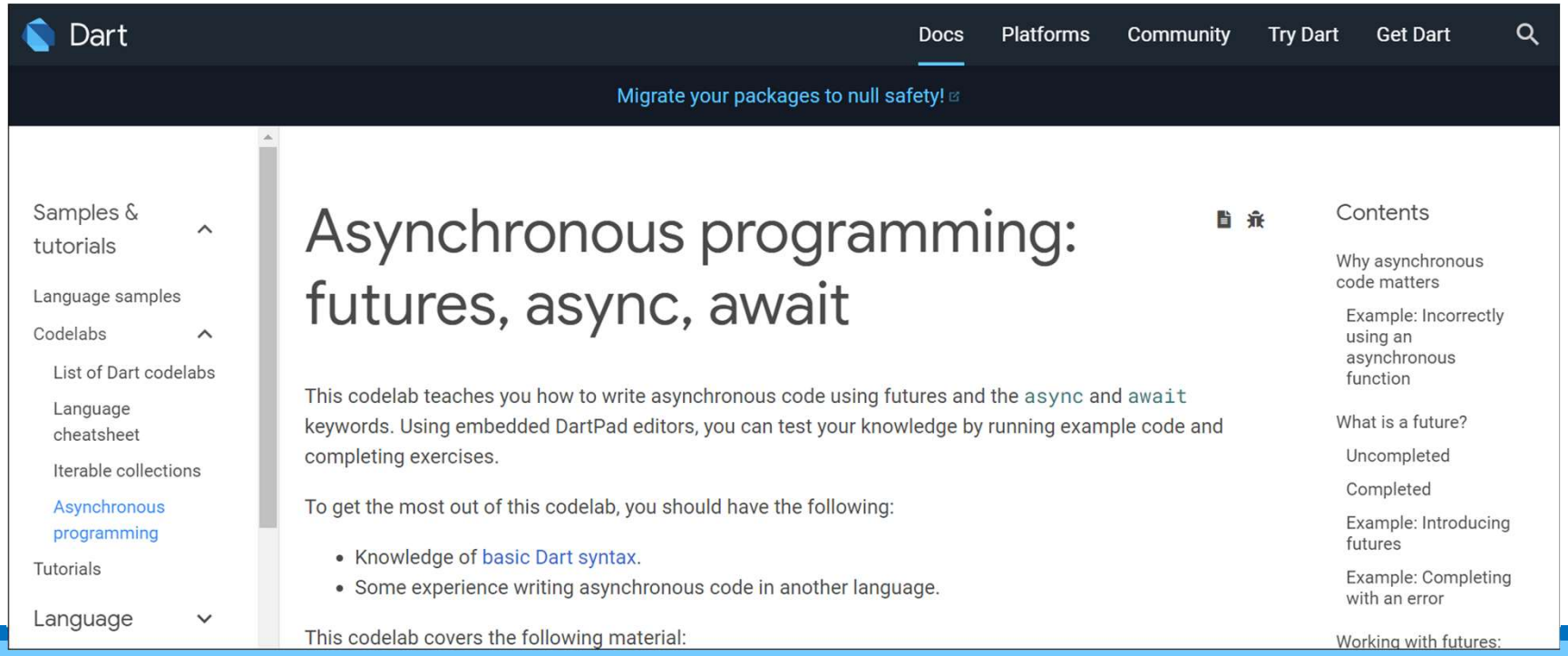
- We use `async/await` and `Future.delayed`

```
// 3. Fill the variables.
void getPerson() async {
  // Simulate 2 seconds delay, then continue
  name = await Future.delayed(Duration(seconds: 2), () {
    return 'mario';
  });

  // Simulate another second delay, then continue
  occupation = await Future.delayed(Duration(seconds: 1), () {
    // IRL - do lookup for ‘mario’, and find that he's a plumber.
    return 'Plumber, savior of princesses';
  });

  // set the state with the now retrieved values
  setState(() {
    name;
    occupation;
    image = 'assets/$name.png';
  });
}
```



More info on async operations

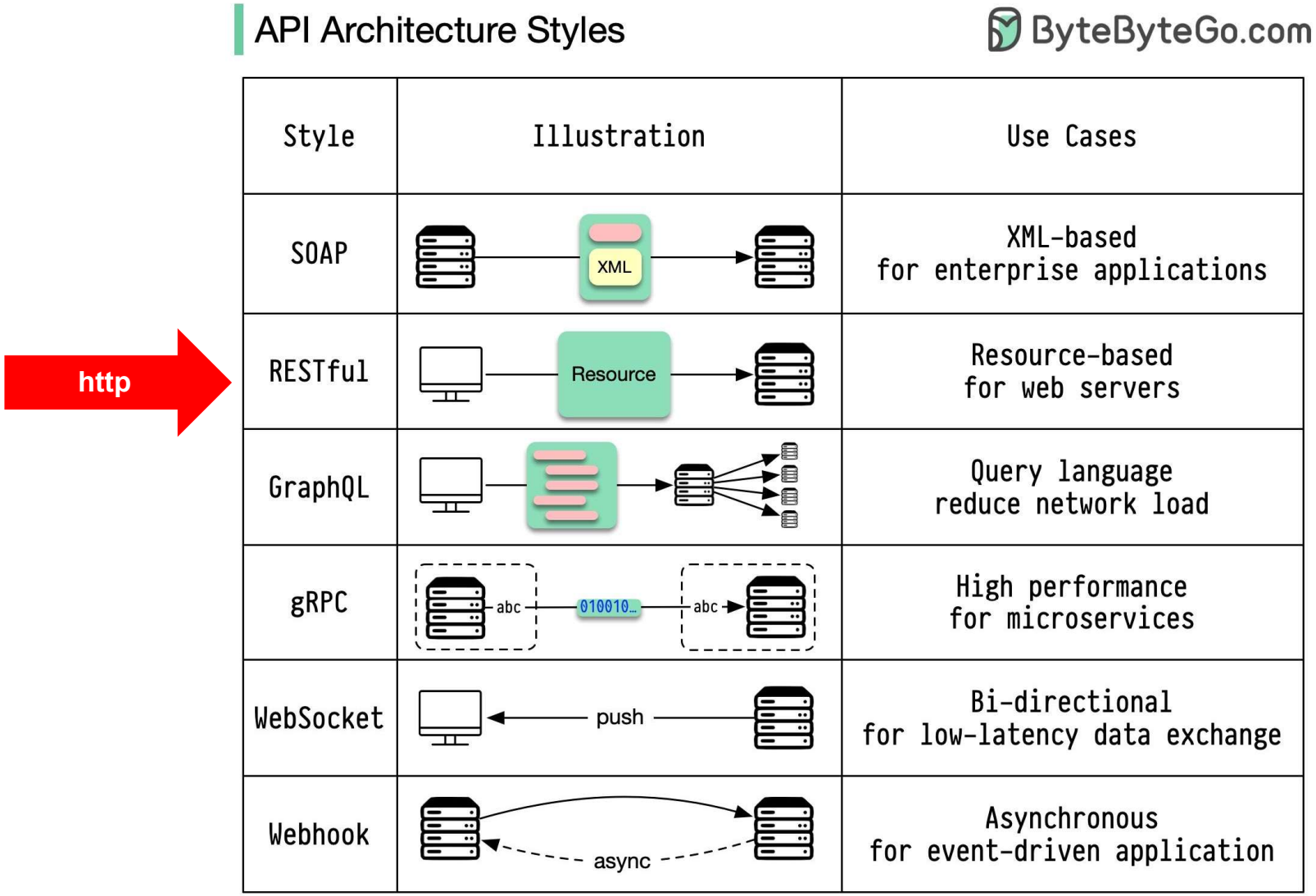
- https://api.flutter.dev/flutter/dart-async/Future/Future.delayed.html
- https://dart.dev/codelabs/async-await



Communication: Using http

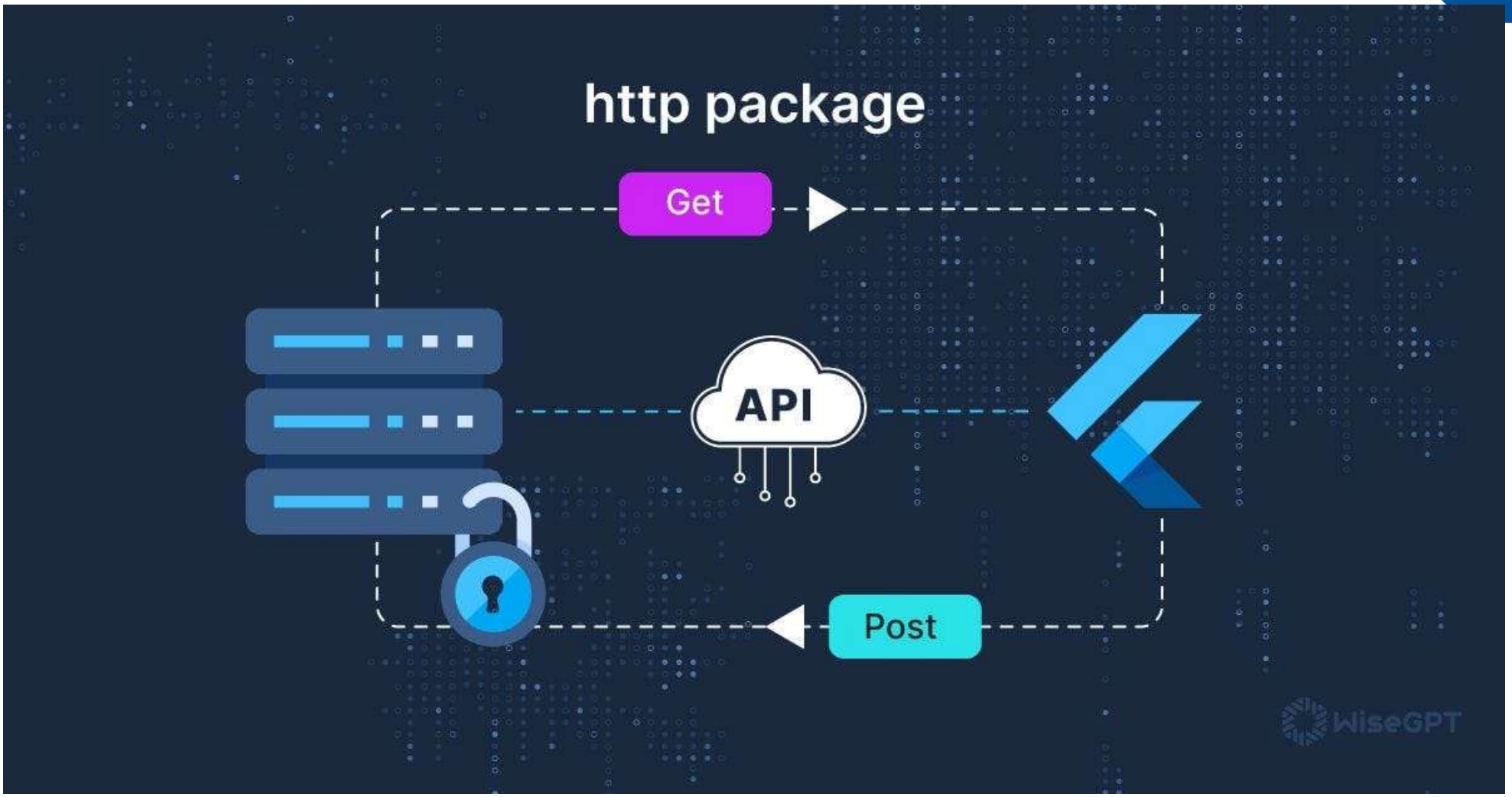
Example of using `http` to make API calls

Communication – lots of options!



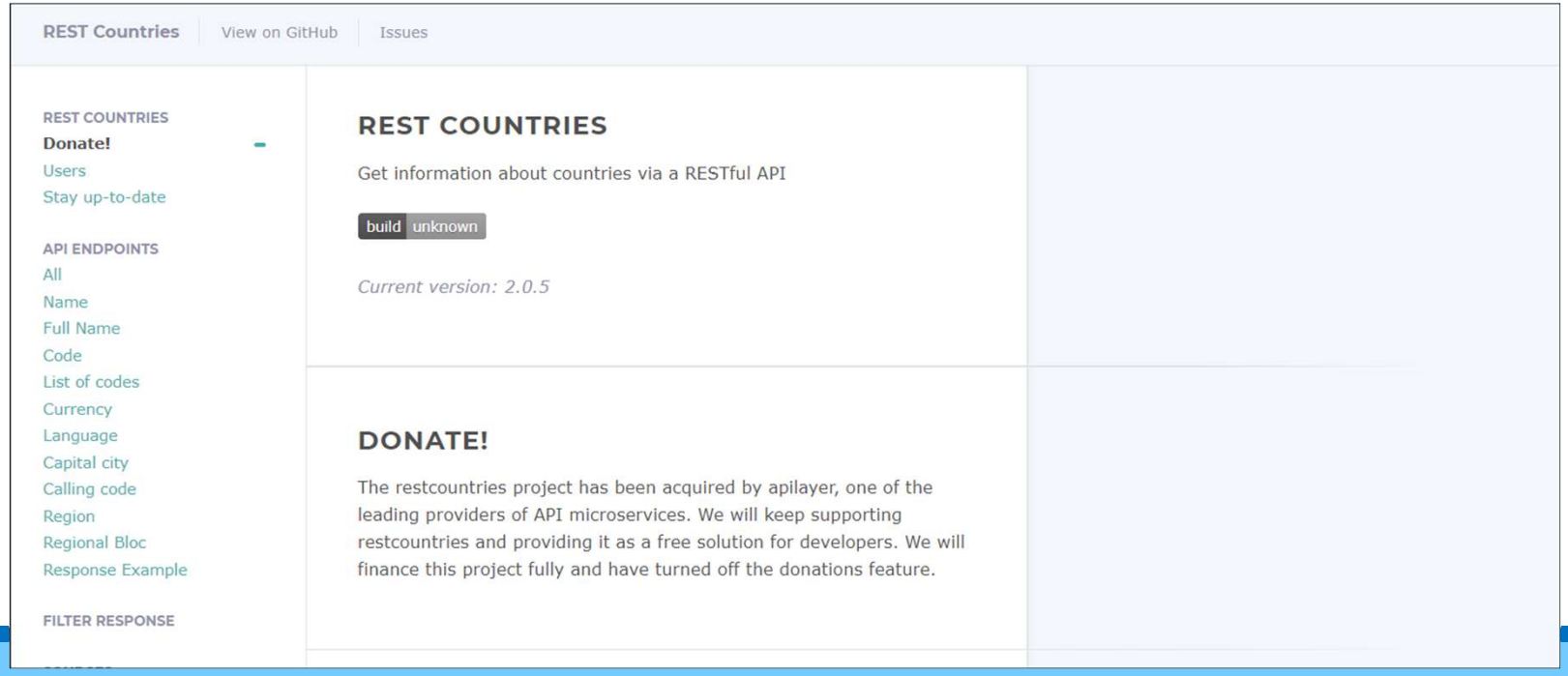
http

Choice: for learning purposes, using `http`



Using the RestCountries API

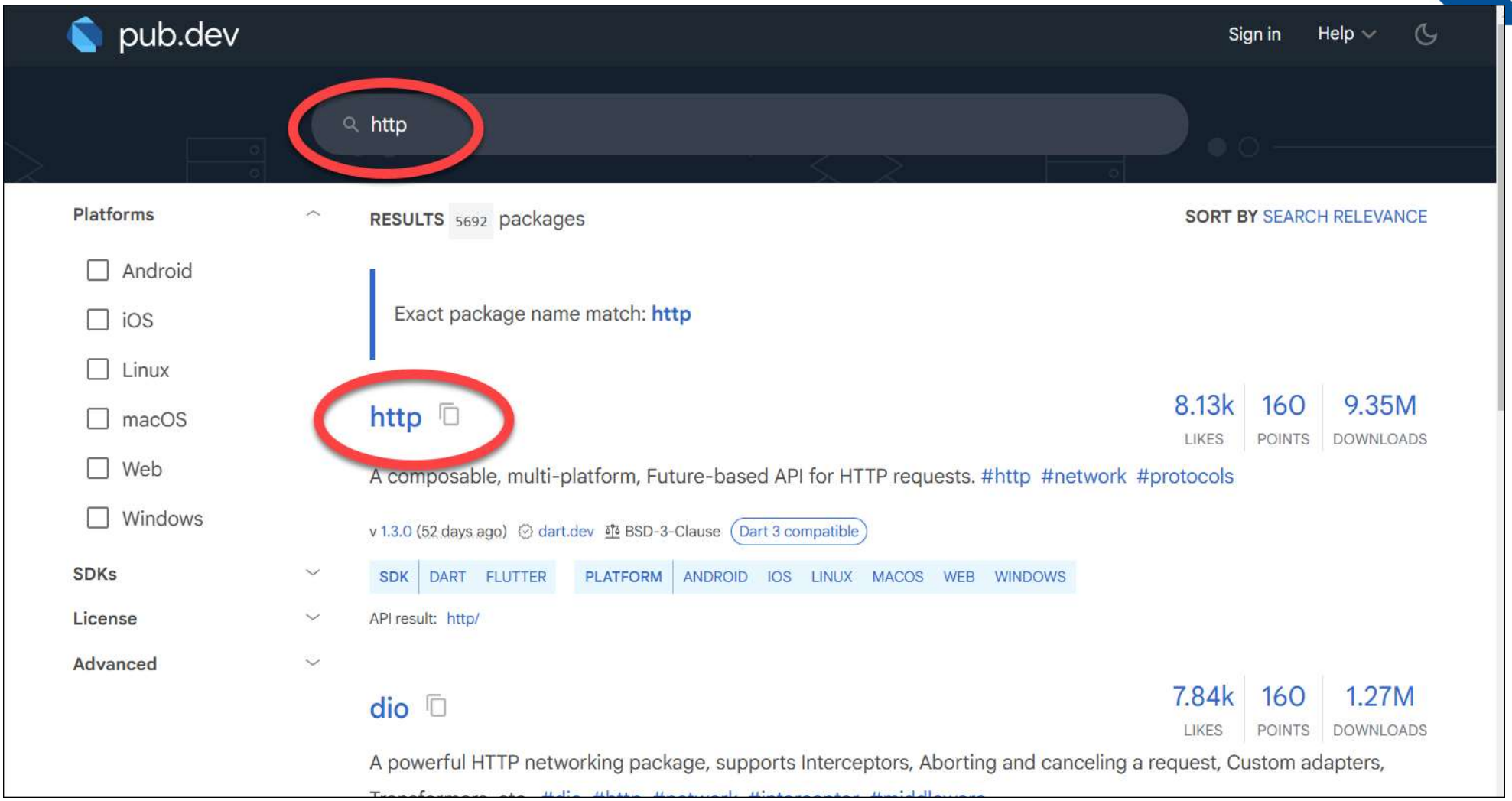
- There are *tons* of APIs available
- We use the RestCountries API here
  - https://restcountries.com/
  - https://restcountries.com/v3.1/all



Adding http to your project

- Flutter can't do `http` by itself
- It needs a package for that
- A *package* acts like a plugin
- When in doubt, *"There is a package for that"*
- So, look at the big picture
  - We're using `http` here, but recognise the pattern for all other kinds of stuff!
  - (Datepicker, slider menu, emojis, firebase, puppeteer, etc!)

Packages for Dart & Flutter



https://pub.dev/flutter/packages

Search for package ‘http’

pub.dev

Sign in Help

http 1.3.0

Published 52 days ago • dart.dev Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

8.1K

Readme Changelog Example Installing Versions Scores

pub v1.3.0 publisher dart.dev

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

This package contains a set of high-level functions and classes that make it easy to consume HTTP resources. It's multi-platform (mobile, desktop, and browser) and supports multiple implementations.

Using

The easiest way to use this library is via the top-level functions. They allow you to make individual HTTP requests with minimal hassle:

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

var url = Uri.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}');
```

8.13k LIKES | 160 POINTS | 9.35M DOWNLOADS

Publisher

dart.dev

Weekly Downloads

2024.08.28 - 2025.03.12

Metadata

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

Repository (GitHub)

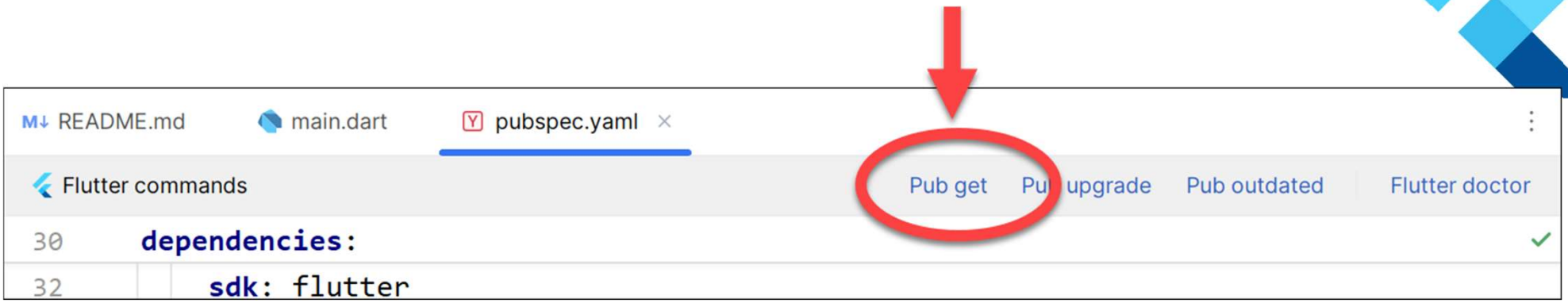
Contributing

Adding package to project

- Using command line: `flutter pub add http`
- OR: manually add it to `pubspec.yaml`:
  - Copy the install command from the site
  - Add dependency to `pubspec.yaml`
  - `Pub get` – to get/install the package
- Anyhow, you get this:

```
dependencies:
  flutter:
    sdk: flutter
  http: ^1.3.0
```

Get dependencies




```
PS C:\Users\Gebruiker\Desktop\flutter_example> flutter pub add http
Resolving dependencies...
Downloading packages...
  async 2.12.0 (2.13.0 available)
  fake_async 1.3.2 (1.3.3 available)
+ http 1.3.0
+ http_parser 4.1.2
  leak_tracker 10.0.8 (10.0.9 available)
  material_color_utilities 0.11.1 (0.12.0 available)
```

Import http package

- Import the `http` package in the file where you want to do the API request
- Use the `Response` object to store the data in
- Use `response.body` object to retrieve the actual data

Performing an http-request

```
import 'package:http/http.dart’;
…
class _HomeCountriesState extends State<HomeCountries> {
…
  String url = 'https://restcountries.com/v3.1/all';
  String countries = '';

  void initState() {
    getCountries();
  }

  void getCountries() async{
    Response response = await get(url);
    setState(() {
      countries = response.body;
    });
  }
…
```

Convert to object

- The `response.body` *looks* like a JSON array, or object, but is actually a string!
- If you `print(countries)`, it looks like this:


```
I/flutter (19461): [{"name":"Afghanistan","topLevelDomain":[".af"],"alpha2Code":"AF","alpha3Code":"AFG","callingCodes":["93"],"capital":"Kabul",
 "altSpellings":["AF","Afġānistān"],"region":"Asia","subregion":"Southern Asia","population":27657145,"latlng":[33.0,65.0],"demonym":"Afghan",
 "area":652230.0,"gini":27.8,"timezones":["UTC+04:30"],"borders":["IRN","PAK","TKM","UZB","TJK","CHN"],"nativeName":"افغانستان","numericCode":"004",
 "currencies":[{"code":"AFN","name":"Afghan afghani","symbol":"؋"}],"languages":[{"iso639_1":"ps","iso639_2":"pus","name":"Pashto","nativeName":"پښتو"},
 {"iso639_1":"uz","iso639_2":"uzb","name":"Uzbek","nativeName":"O‘zbek"},{"iso639_1":"tk","iso639_2":"tuk","name":"Turkmen","nativeName":"Türkmen"}],
```

Converting http responses: `dart:convert`

- Import the package `dart:convert`, which has the `jsonDecode()` method
  - We can now decode the response to a `Map` (if response is object)
  - We can decode the response to a `List` (if response is an array)

```
import 'package:http/http.dart'; // to do http requests
import 'dart:convert'; // to convert response body to JSON objects/arrays
…
class _HomeCountriesState extends State<HomeCountries> {
…
  List countries =[];

  // Get countries
  void getCountries() async{
    Response response = await get(url);
    setState(() {
      countries = jsonDecode(response.body);
    });
    print (countries[0]['name’][‘common’]); // Afghanistan
  }
```

Better practice – use classes

- Create a .fromJson() method on the class which returns a factory

```
// country class
class Country {
  String flag;
  String name;
  String capital;

  Country({this.name, this.flag, this.capital});

  factory Country.fromJson(Map<String, dynamic> parsedJson) {
    return Country(
        name: parsedJson['name'].toString(),
        capital: parsedJson['capital'].toString(),
        flag: parsedJson['flag'].toString());
  }
}
```

Adding a factory to a class to convert JSON

Flutter

Docs Showcase Community Get started

Migrate your packages to null safety!

Get started

Samples & tutorials

Flutter Gallery [running app]

Flutter Gallery [repo]

Sample apps on GitHub

Cookbook

Codelabs

Tutorials

Development

User interface

Data & backend

Accessibility & internationalization

Platform integration

Packages & plugins

Add Flutter to existing app

⟨ Delete data on the internet

Make authenticated requests ⟩

Fetch data from the internet

Docs > Cookbook > Networking > Fetch data from the internet

Fetching data from the internet is necessary for most apps. Luckily, Dart and Flutter provide tools, such as the `http` package, for this type of work.

This recipe uses the following steps:

1. Add the `http` package.
2. Make a network request using the `http` package.
3. Convert the response into a custom Dart object.
4. Fetch and display the data with Flutter.

1. Add the `http` package

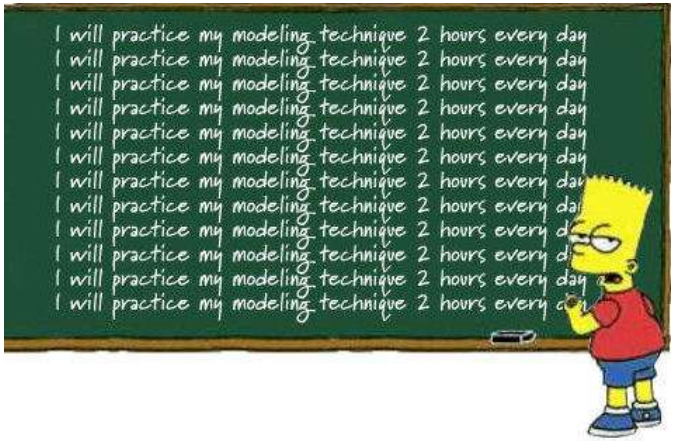
The `http` package provides the simplest way to fetch data from the internet.

Contents

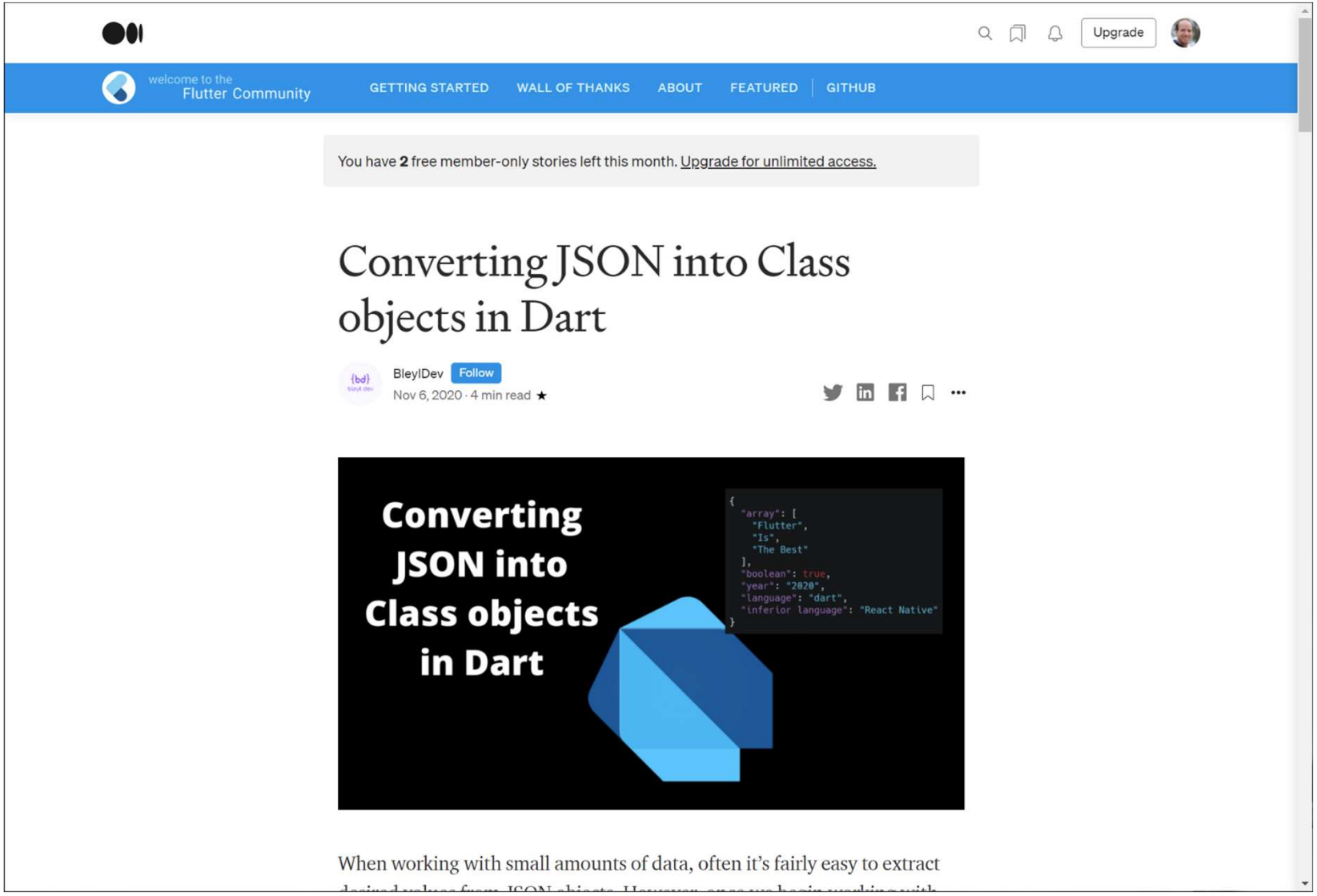
https://flutter.dev/docs/cookbook/networking/fetch-data

Workshop

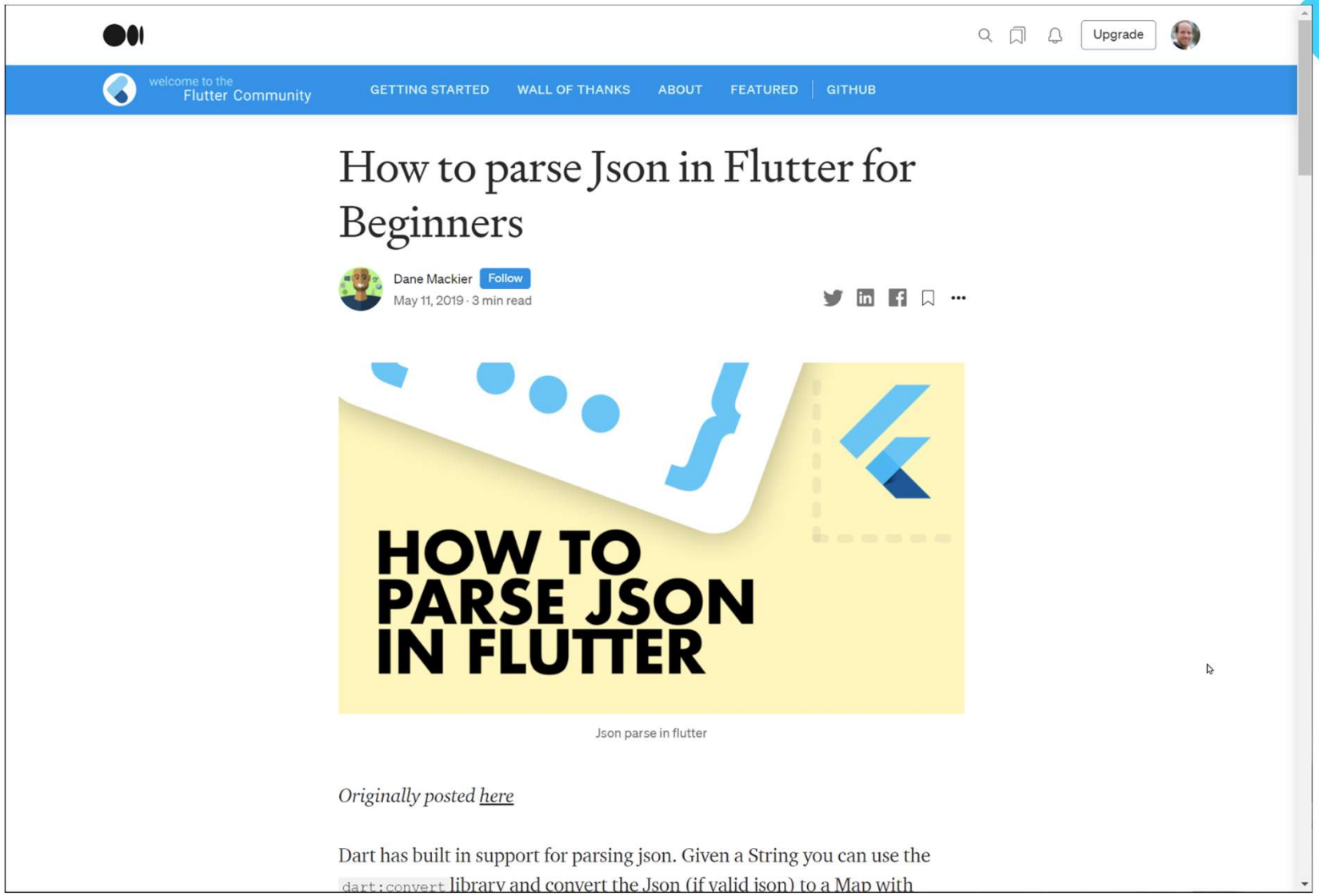
- Study the example on how to fetch data using http communication
- Optional: Create a Flutter app, talking to the dummy user API, at https://jsonplaceholder.typicode.com/users
  - Fetch data,
  - Show every user with some details in a `Card()` in your app.
  - Optional: create a `User` class and use this (tip: transform the response json to a Dart class quickly, using app.quicktype.io
- Example: `../_270-http`



Google for articles on converting JSON into Dart-objects



https://medium.com/flutter-community/converting-json-into-class-objects-in-dart-abcc3cc05478



https://medium.com/flutter-community/how-to-parse-json-in-flutter-for-beginners-8074a68d7a79

Instantly parse JSON to any language

quicktype

Please Share! Options

Name: Welcome

Source type: JSON

```
{
  "greeting": "Welcome to quicktype!",
  "instructions": [
    "Type or paste JSON here",
    "Or choose a sample above",
    "quicktype will generate code in your",
    "chosen language to parse the sample data"
  ]
}
```

```
// To parse this JSON data, do
//
//     final welcome = welcomeFromJson(jsonString);

import 'dart:convert';

Welcome welcomeFromJson(String str) => Welcome.fromJson(json.decode(str));

String welcomeToJson(Welcome data) => json.encode(data.toJson());

class Welcome {
    Welcome({
        this.greeting,
        this.instructions,
    });

    String greeting;
    List<String> instructions;

    factory Welcome.fromJson(Map<String, dynamic> json) => Welcome(
        greeting: json["greeting"],
        instructions: List<String>.from(json["instructions"].map((x) => x)),
    );

    Map<String, dynamic> toJson() => {
        "greeting": greeting,
        "instructions": List<dynamic>.from(instructions.map((x) => x)),
    };
}
```

Language | Other

Dart

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

Share Link to Code

Copy Code

https://app.quicktype.io/