

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

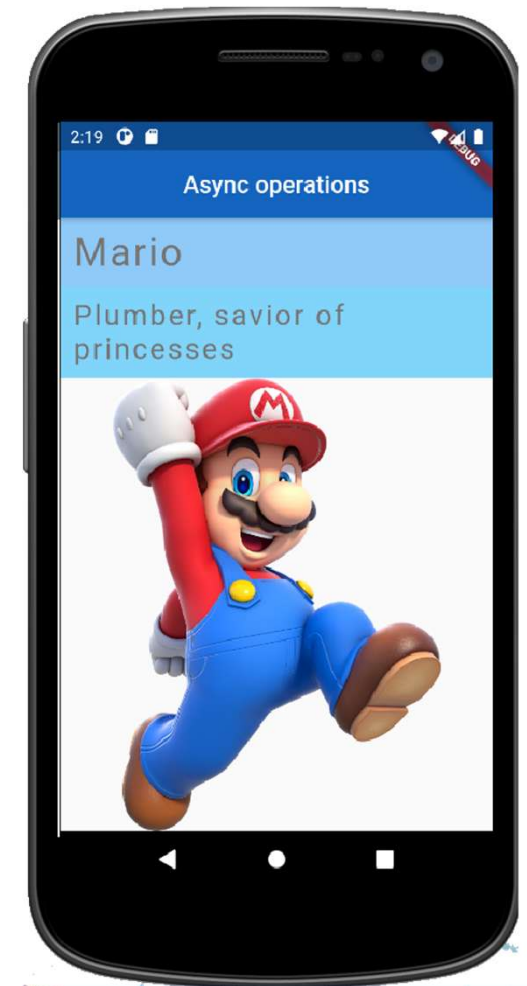


- Using **static data**, we can build a lay-out like this:

```
// 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

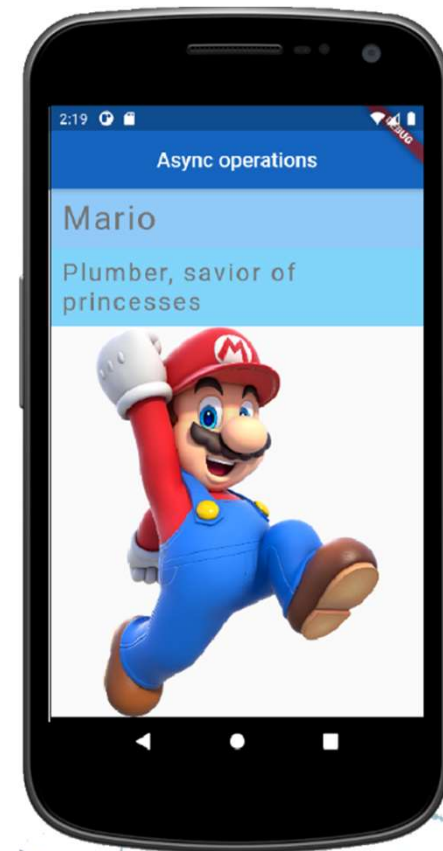


- We use `async/await` and `Future.delayed`

```
// 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>

A screenshot of the Dart documentation website. The top navigation bar includes links for 'Docs', 'Platforms', 'Community', 'Try Dart', and 'Get Dart'. A banner below the navigation bar reads 'Migrate your packages to null safety!'. The left sidebar contains a menu with 'Samples & tutorials', 'Language samples', 'Codelabs' (expanded to show 'List of Dart codelabs', 'Language cheatsheet', 'Iterable collections', and 'Asynchronous programming'), 'Tutorials', and 'Language'. The main content area features the title 'Asynchronous programming: futures, async, await' and a description: 'This codelab teaches you how to write asynchronous code using futures and the `async` and `await` keywords. Using embedded DartPad editors, you can test your knowledge by running example code and completing exercises.' Below this, it lists prerequisites: 'To get the most out of this codelab, you should have the following: Knowledge of basic Dart syntax. Some experience writing asynchronous code in another language.' The right sidebar shows a 'Contents' table of contents with links to 'Why asynchronous code matters', 'Example: Incorrectly using an asynchronous function', 'What is a future?', 'Uncompleted', 'Completed', 'Example: Introducing futures', 'Example: Completing with an error', and 'Working with futures:'.

Dart

Docs Platforms Community Try Dart Get Dart

Migrate your packages to null safety!

Samples & tutorials ^

Language samples

Codelabs ^

List of Dart codelabs

Language cheatsheet

Iterable collections

Asynchronous programming

Tutorials

Language v

Asynchronous programming: futures, async, await

This codelab teaches you how to write asynchronous code using futures and the `async` and `await` keywords. Using embedded DartPad editors, you can test your knowledge by running example code and completing exercises.

To get the most out of this codelab, you should have the following:

- Knowledge of [basic Dart syntax](#).
- Some experience writing asynchronous code in another language.

This codelab covers the following material:

Contents

- Why asynchronous code matters
 - Example: Incorrectly using an asynchronous function
- What is a future?
 - Uncompleted
 - Completed
 - Example: Introducing futures
 - Example: Completing with an error
- Working with futures:



Communication: Using http

Example of using `http` to make API calls

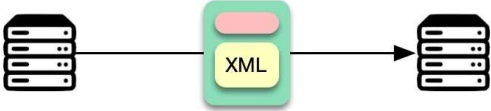

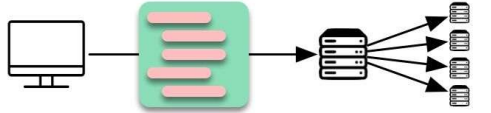
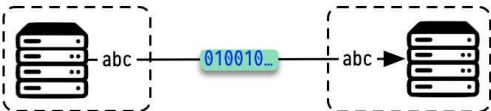
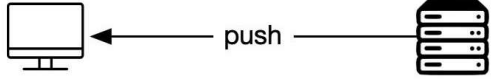
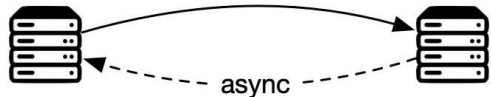
Communication – lots of options!



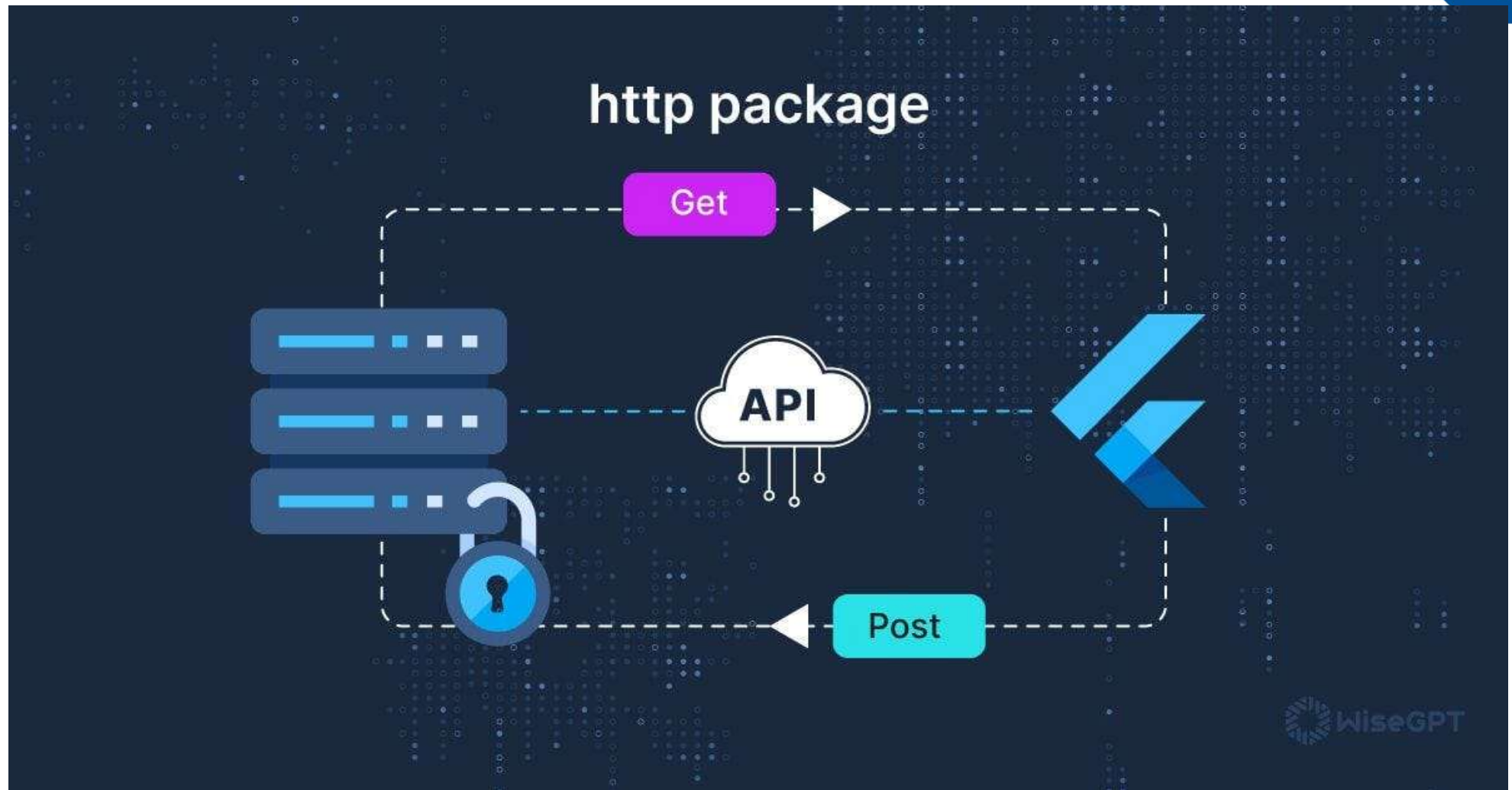
API Architecture Styles

 ByteByteGo.com



Style	Illustration	Use Cases
SOAP		XML-based for enterprise applications
RESTful		Resource-based for web servers
GraphQL		Query language reduce network load
gRPC		High performance for microservices
WebSocket		Bi-directional for low-latency data exchange
Webhook		Asynchronous for event-driven application

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>

The screenshot shows the REST Countries API website. At the top, there are links for 'View on GitHub' and 'Issues'. The main content area is divided into two columns. The left column contains a sidebar with links for 'REST COUNTRIES', 'Donate!', 'Users', 'Stay up-to-date', 'API ENDPOINTS', and 'FILTER RESPONSE'. The right column contains the main content, which includes the title 'REST COUNTRIES', a description 'Get information about countries via a RESTful API', a version indicator 'build unknown', and the current version '2.0.5'. Below this, there is a 'DONATE!' section with a message about the project's acquisition by apilayer and the decision to continue supporting the API as a free solution for developers.

REST Countries | View on GitHub | Issues

REST COUNTRIES

Donate!

Users

Stay up-to-date

API ENDPOINTS

All

Name

Full Name

Code

List of codes

Currency

Language

Capital city

Calling code

Region

Regional Bloc

Response Example

FILTER RESPONSE

REST COUNTRIES

Get information about countries via a RESTful API

build unknown

Current version: 2.0.5

DONATE!

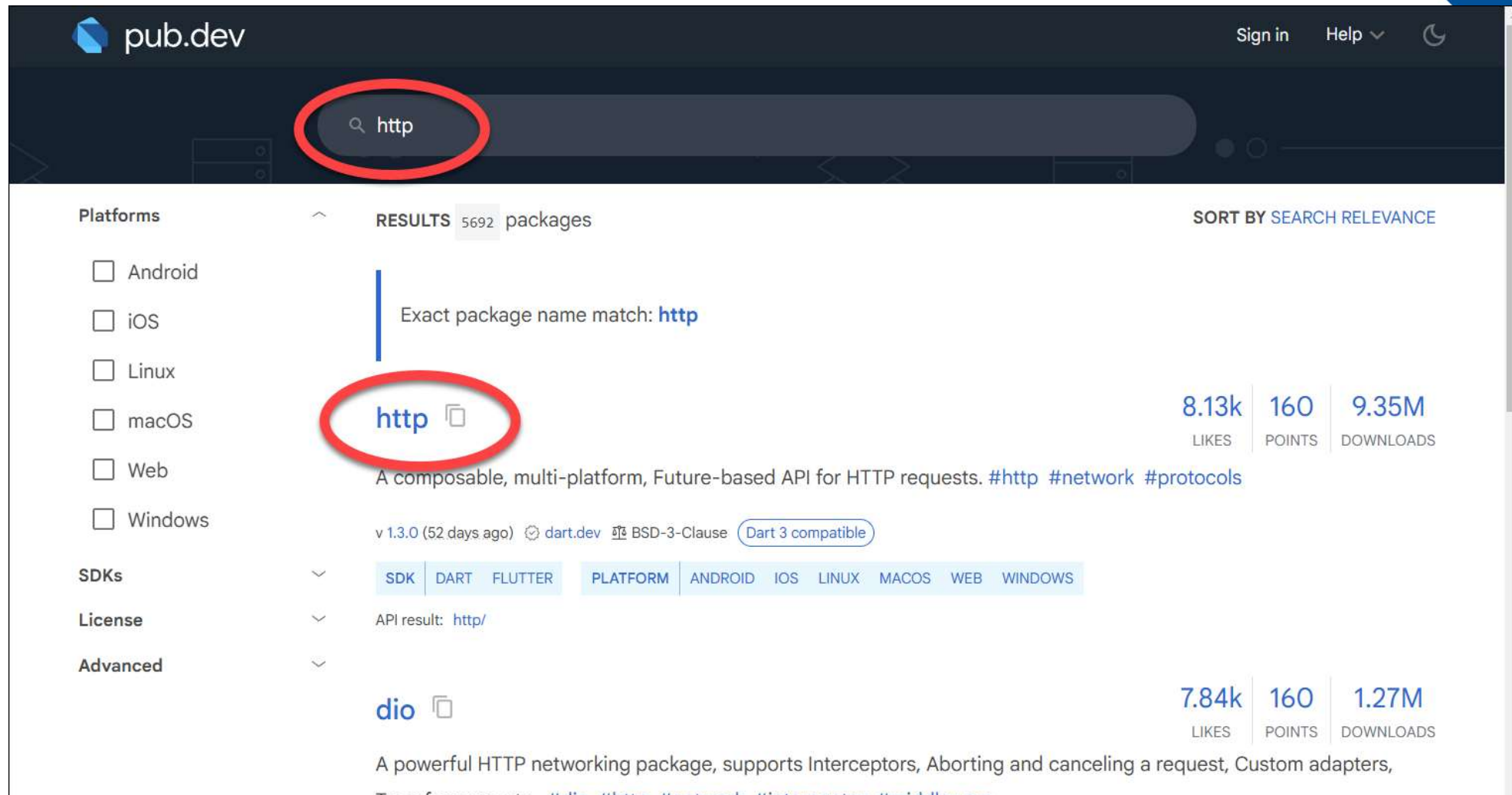
The restcountries project has been acquired by apilayer, one of the leading providers of API microservices. We will keep supporting restcountries and providing it as a free solution for developers. We will finance this project fully and have turned off the donations feature.

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



The screenshot shows the pub.dev website interface. At the top, the 'pub.dev' logo is on the left, and 'Sign in' and 'Help' links are on the right. A search bar in the center contains the text 'http', which is circled in red. Below the search bar, the results section shows 'RESULTS 5692 packages'. On the left side, there are filters for 'Platforms' (Android, iOS, Linux, macOS, Web, Windows) and 'SDKs' (DART, FLUTTER, PLATFORM, ANDROID, IOS, LINUX, MACOS, WEB, WINDOWS). The main content area displays the search results. The first result is the 'http' package, which is circled in red. It shows 'Exact package name match: http' and 'A composable, multi-platform, Future-based API for HTTP requests. #http #network #protocols'. It also shows 'v 1.3.0 (52 days ago)' and 'dart.dev' as the source. To the right of the package name, there are statistics: '8.13k' likes, '160' points, and '9.35M' downloads. Below the first result, the 'dio' package is partially visible, showing 'A powerful HTTP networking package, supports Interceptors, Aborting and canceling a request, Custom adapters, Transformers, etc. #dio #http #network #interceptors #middleware'.

pub.dev

Sign in Help

http

Platforms

- ☐ Android
- ☐ iOS
- ☐ Linux
- ☐ macOS
- ☐ Web
- ☐ Windows

SDKs

License

Advanced

RESULTS 5692 packages

SORT BY SEARCH RELEVANCE

Exact package name match: [http](#)

[http](#)

8.13k 160 9.35M
LIKES POINTS DOWNLOADS

A composable, multi-platform, Future-based API for HTTP requests. #http #network #protocols

v 1.3.0 (52 days ago) dart.dev BSD-3-Clause Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

API result: [http/](#)

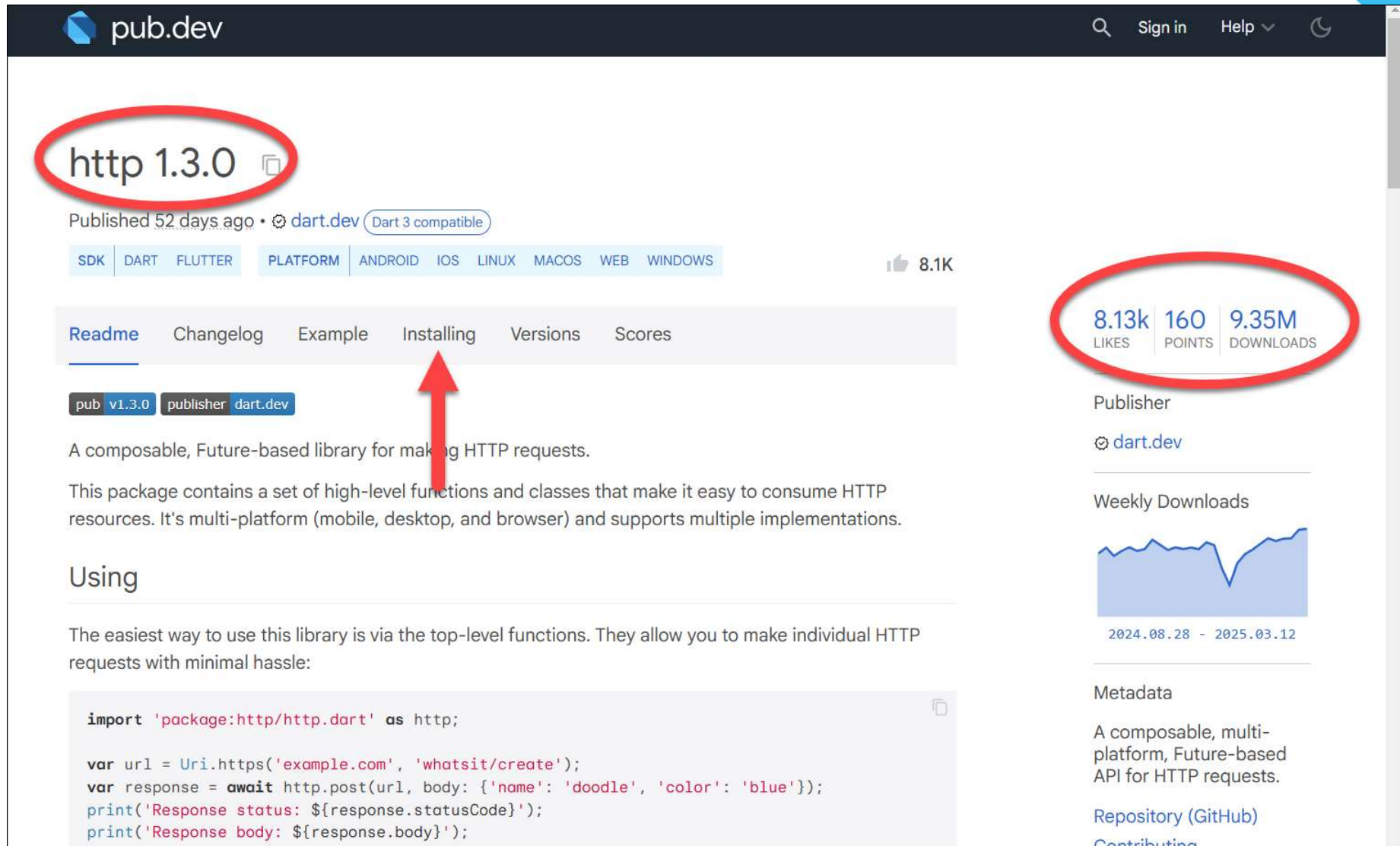
[dio](#)

7.84k 160 1.27M
LIKES POINTS DOWNLOADS

A powerful HTTP networking package, supports Interceptors, Aborting and canceling a request, Custom adapters, Transformers, etc. #dio #http #network #interceptors #middleware

<https://pub.dev/flutter/packages>

Search for package 'http'



The screenshot shows the pub.dev package page for 'http 1.3.0'. The package name and version are circled in red. The page includes a navigation bar with 'SDK', 'DART', 'FLUTTER', 'PLATFORM', 'ANDROID', 'IOS', 'LINUX', 'MACOS', 'WEB', and 'WINDOWS'. The 'Installing' tab is highlighted with a red arrow. The right sidebar shows statistics: 8.13k likes, 160 points, and 9.35M downloads, all circled in red. The 'Weekly Downloads' chart shows a peak in late 2024. The 'Metadata' section describes the package as a composable, multi-platform, Future-based API for HTTP requests.

pub.dev

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

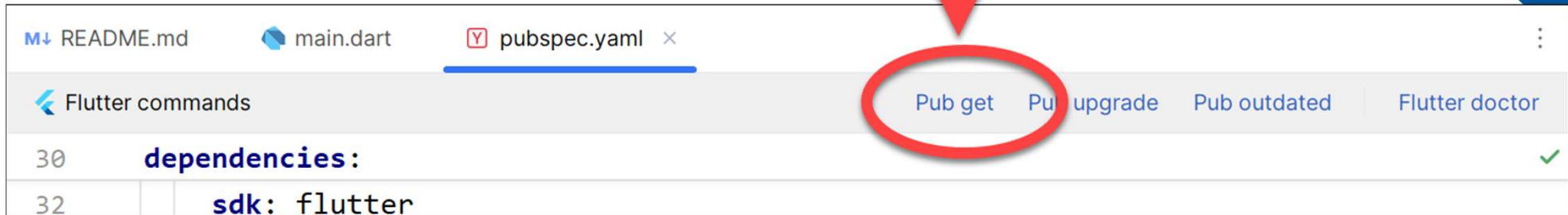
Adding package to project



- 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)
+ tuned_data 1.4.0
```

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","Afgānistān"],"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":"Oʻzbek"}, {"iso639_1":"tk","iso639_2":"tuk","name":"Turkmen","nativeName":"Türkmen"}],  
  "translations":{"de":"Afghanistan","es":"Afganistán","fr":"Afghanistan","ja":"アフガニスタン","it":"Afghanistan","kn":"ಆಫಘಾನಿಸ್ತಾನ್","pt":"Afeganistão"}
```

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<Country> 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

The screenshot shows the Flutter documentation website. The top navigation bar includes the Flutter logo, links to Docs, Showcase, and Community, a search icon, social media icons, and a 'Get started' button. A blue banner below the navigation bar reads 'Migrate your packages to null safety!'. The left sidebar contains a table of contents with sections like 'Get started', 'Samples & tutorials', 'Cookbook', 'Codelabs', 'Tutorials', and 'Development'. The main content area is titled 'Fetch data from the internet' and includes a breadcrumb trail: 'Docs > Cookbook > Networking > Fetch data from the internet'. The text explains that fetching data from the internet is necessary for most apps and that Dart and Flutter provide tools like the `http` package. It lists four steps: 1. Add the `http` package, 2. Make a network request using the `http` package, 3. Convert the response into a custom Dart object, and 4. Fetch and display the data with Flutter. The first step is expanded, showing the text: 'The `http` package provides the simplest way to fetch data from the internet.'

Flutter

Docs Showcase Community

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

1. Add the `http` package
2. Make a network request
3. Convert the response into a custom Dart object
 - Create an Album class
 - Convert the `http.Response` to an Album
4. Fetch the data
5. Display the data

Why is `fetchAlbum()` called in `initState()`?

Testing

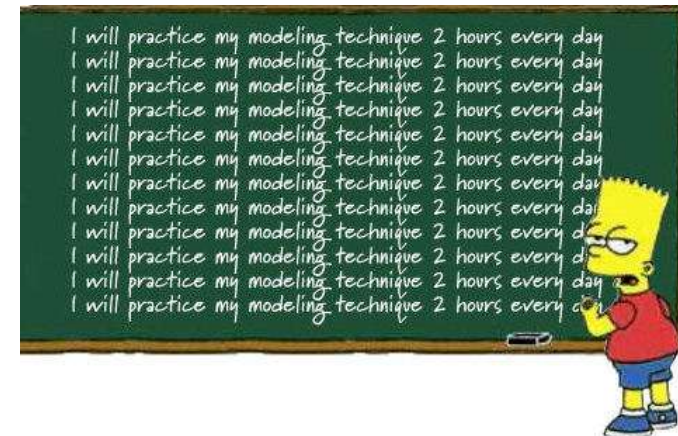
Complete example

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

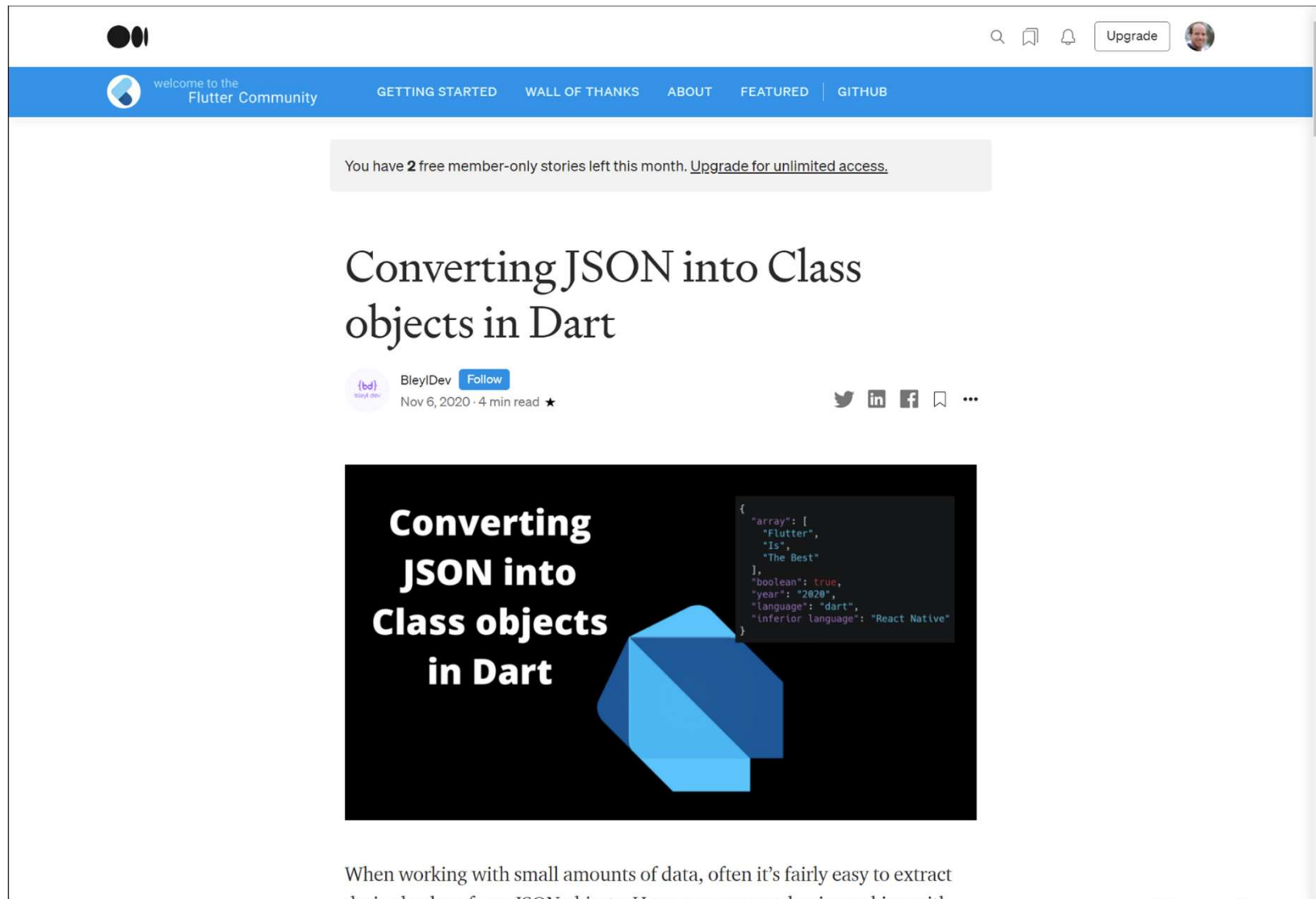
Workshop




- 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>



welcome to the
Flutter Community






GETTING STARTEDWALL OF THANKSABOUTFEATUREDGITHub

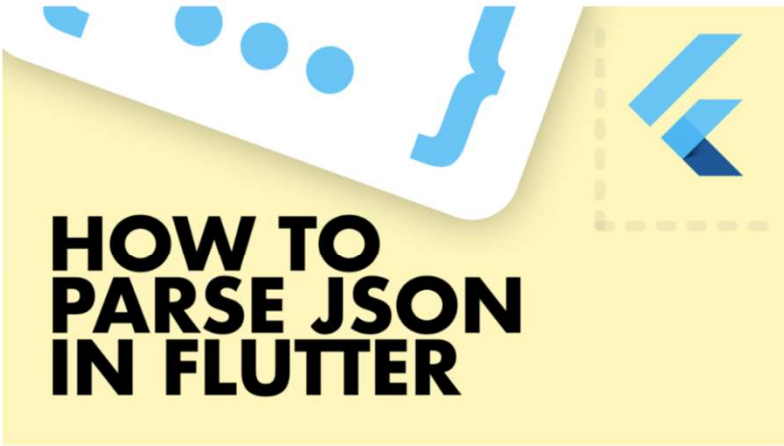
How to parse Json in Flutter for Beginners



Dane MackierFollow

May 11, 2019 · 3 min read





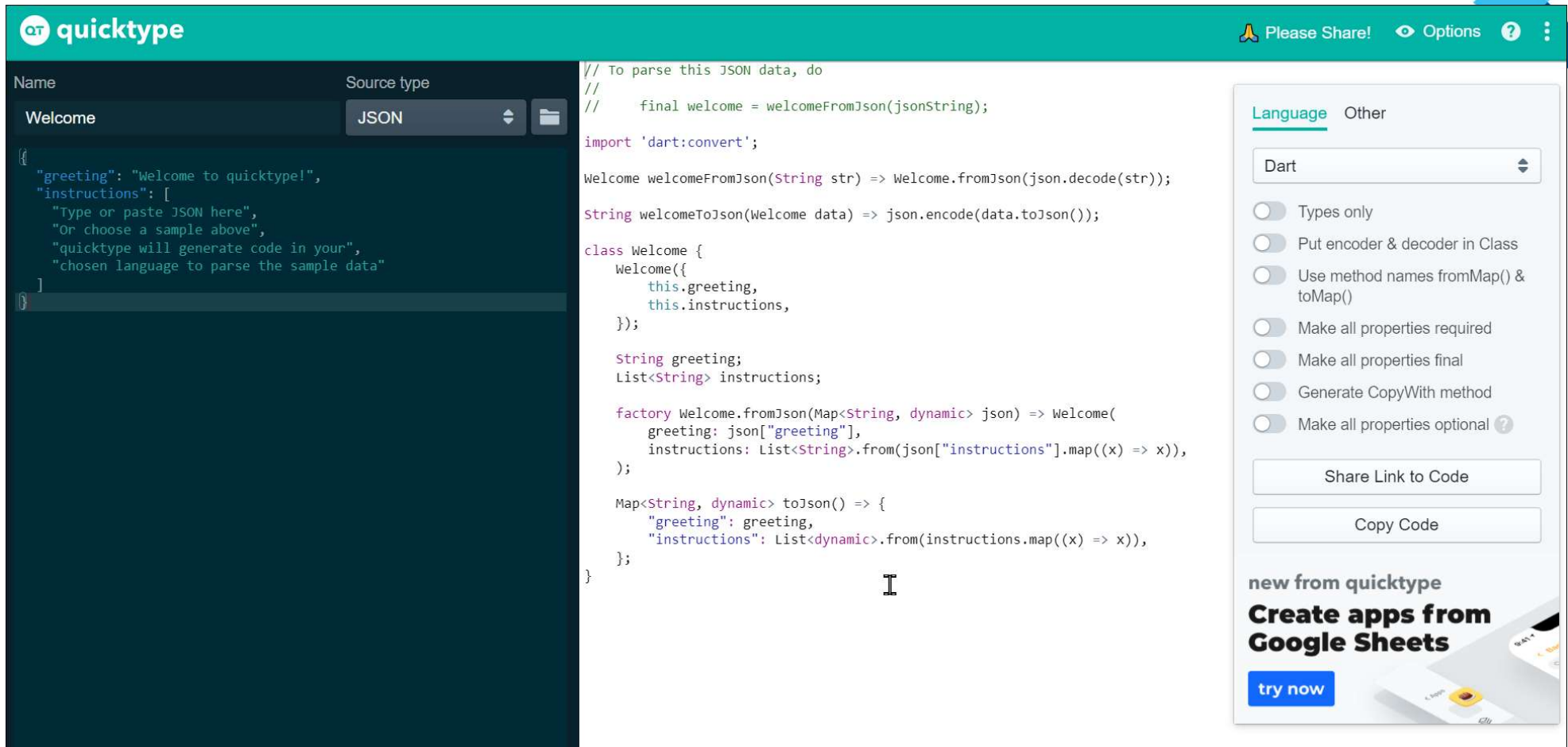
Json parse in flutter

Originally posted [here](#)

Dart has built in support for parsing json. Given a String you can use the `dart:convert` library and convert the Json (if valid json) to a Map with

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

Instantly parse JSON to any language



The screenshot displays the quicktype web application interface. The top header is teal with the 'quicktype' logo on the left and links for 'Please Share!', 'Options', and a help icon on the right. The main area is divided into three sections:

- Left Panel:** A dark-themed editor with a 'Name' field containing 'Welcome' and a 'Source type' dropdown set to 'JSON'. Below this, a JSON object is pasted:

```
{  "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"  ]}
```
- Middle Panel:** A light-themed code editor showing the generated Dart code. The code includes a comment, an import statement, a function, a class definition with a constructor, and a factory method.

```
// 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)),  
  };  
}
```
- Right Panel:** A settings sidebar with a 'Language' tab selected. A dropdown menu shows 'Dart'. Below are several toggle switches for options like 'Types only', 'Put encoder & decoder in Class', 'Use method names from Map() & toMap()', 'Make all properties required', 'Make all properties final', 'Generate CopyWith method', and 'Make all properties optional'. At the bottom of the sidebar are buttons for 'Share Link to Code' and 'Copy Code'. A promotional banner at the very bottom right says 'new from quicktype Create apps from Google Sheets try now'.

<https://app.quicktype.io/>