ОНЛАЙН-ОБРАЗОВАНИЕ

Меня хорошо видно && слышно?

Ставьте + , если все хорошо
Напишите в чат, если есть проблемы

По предыдущим темам

Опрос

На что стоит обратить внимание преподавателям?

```yaml
# обратите внимание - я могу указать зависимость на репозиторий из git с указанием отдельной ветки
# и конкретный путь к фолдеру/директории
cocktailappmodels:
  git:
    url: git@github.com:guid-empty/otus-lesson-01-dart-basics-packages.git
    ref: master
    path: cocktail_app_models
```

Flutter Mobile Developer

Тема 4. Dart. Async и работа с сетью

Цель урока

- продолжить работать над приложением (на этом этапе получим реальные данные из RapidAPI).

- после занятия вы сможете:
- - использовать Generics;
- - писать Extentions;
- - писать асинхронный код;
- - делать запросы в сеть и обрабатывать результат;
- - парсить Json и использовать генераторы кода для создания классов.

О чем будем говорить

- Dart Language - Generics, Extensions
- Async
- Http
- Json Serialization
- Mobile App - Async Repository & HomeWork

Dart Language - Generics

Что такое generics - обобщенные типы?

Определение из спецификации:

14 Generics generics

A class declaration (10), type alias (19.3), or function (9) G may be generic, that is, G may have formal type parameters declared.

When an entity in this specification is described as generic, and the special case is considered where the number of type arguments is zero, the type argument list should be omitted

Dart Language - Generics

Что такое generics - обобщенные типы?

Обобщенный тип объявляет параметры типа (type parameters) - плейсхолдеры типов, которые будут заполнены в дальнейшем уже при использовании обобщенного типа, путем передачи в него аргументов типа (type arguments).

Обобщенные методы (Generic Methods)

Параметры типа могут задаваться не только для типа в целом, но и для отдельного метода.

Обобщенный метод объявляет параметры типа в сигнатуре метода.

Dart Language - Generics

- Demo

Demo

Dart Language - Generics

- Demo

Вы уже сталкивались с Generics - вспомните где?

Dart Language - Generics - Итоги

Generics позволяют нам

- Писать меньше кода
- Иметь единый контракт для работы с типами
- И одну имплементацию для множества типов
- Выполнить проверку кода на этапе компиляции (в отличие от dynamic)

Dart Language - Extensions

Определение из SDK:

Extension methods
Extension methods, introduced in Dart 2.7, are a way to add functionality to existing libraries.
You might use extension methods without even knowing it.

For example, when you use code completion in an IDE, it suggests extension methods alongside regular methods.

Dart Language - Extensions

Extension — это так называемый синтаксический сахар, syntax sugar.

Используя extension methods, мы можем расширить интерфейс (контракт) класса,
не затрагивая при этом его код.

При этом такой класс может быть даже не из нашего Dart Package.

Dart Language - Extensions

- Demo

Demo

Dart Language - Extensions

Extension methods не работают с dynamic

О чем будем говорить

Dart Language - Generics, Extensions

Async

Http

Json Serialization

Mobile App - Async Repository & HomeWork

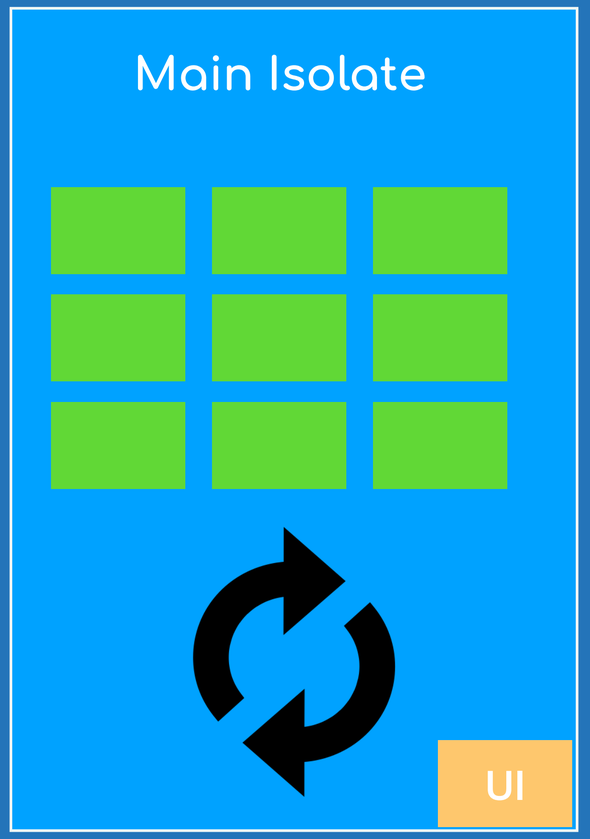
Flutter Application

Dart - это single threaded language. Однако, он поддерживает такие асинхронные понятия, как Streams, Futures, фоновые задачи (изоляты).

Dart & flutter приложение содержат минимум один изолят.

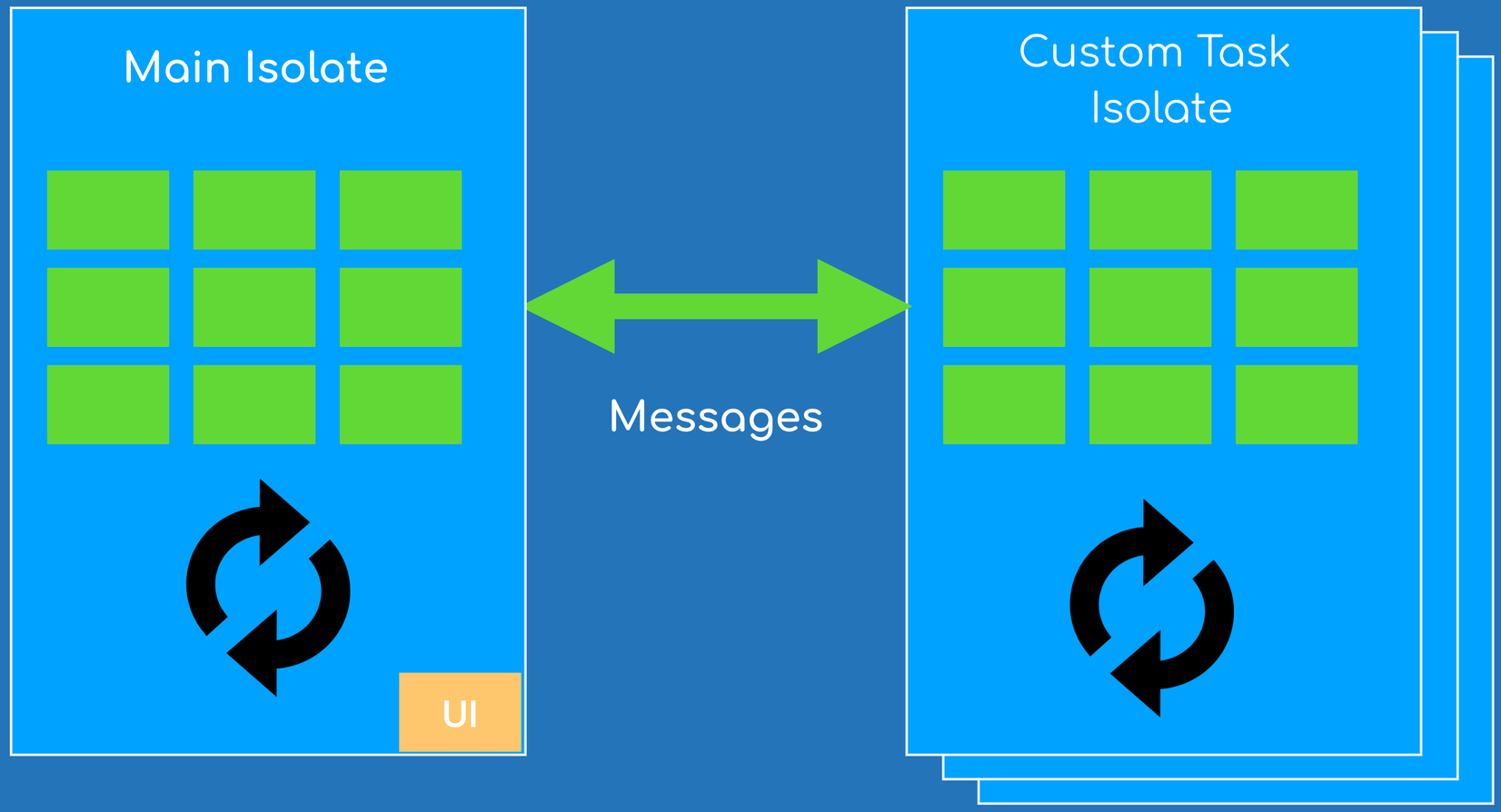
Главный isolate отвечает за UI.

Isolate - это среда выполнения Dart кода, которая имеет свою Изолированную память и поток обработки событий (event loop).



Flutter Application

В общем случае приложение может содержать несколько изолятов

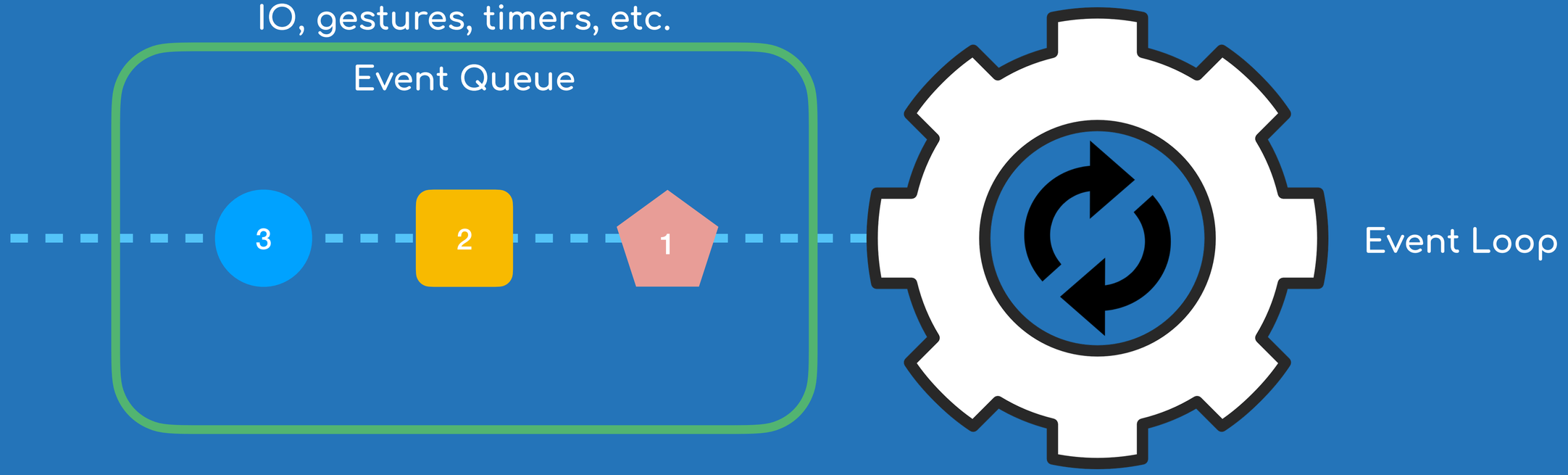


Isolate.spawn()
compute()

Async & Event Loop

• Demo

Dart Event Loop Web Archive page

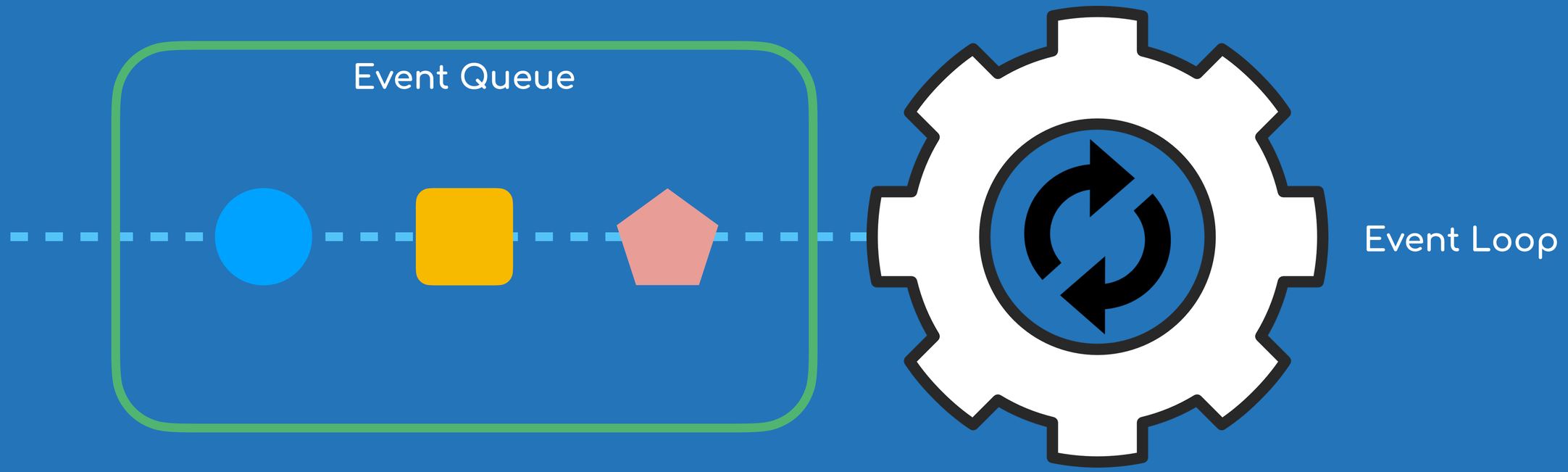


1.Жест на экране
2.Ввели символ
3.Запустили таймер

Async & Event Loop

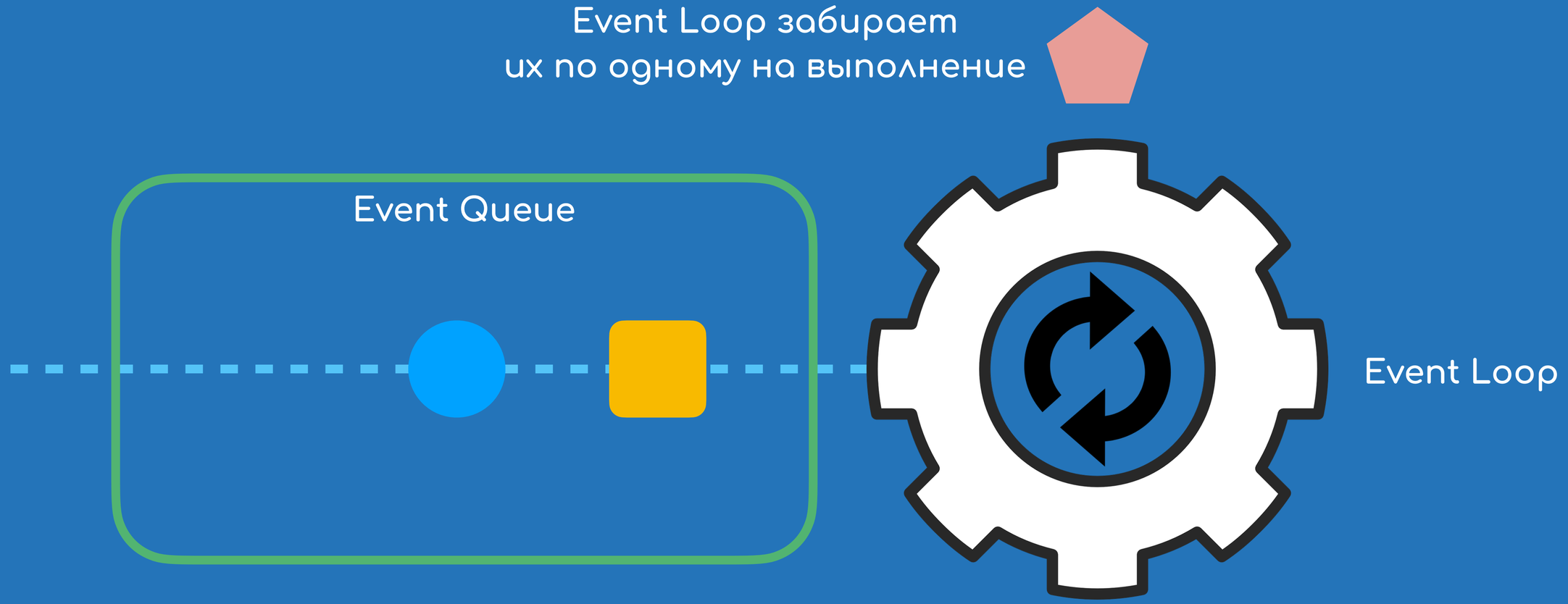
• Demo

В очереди событий есть события



Async & Event Loop

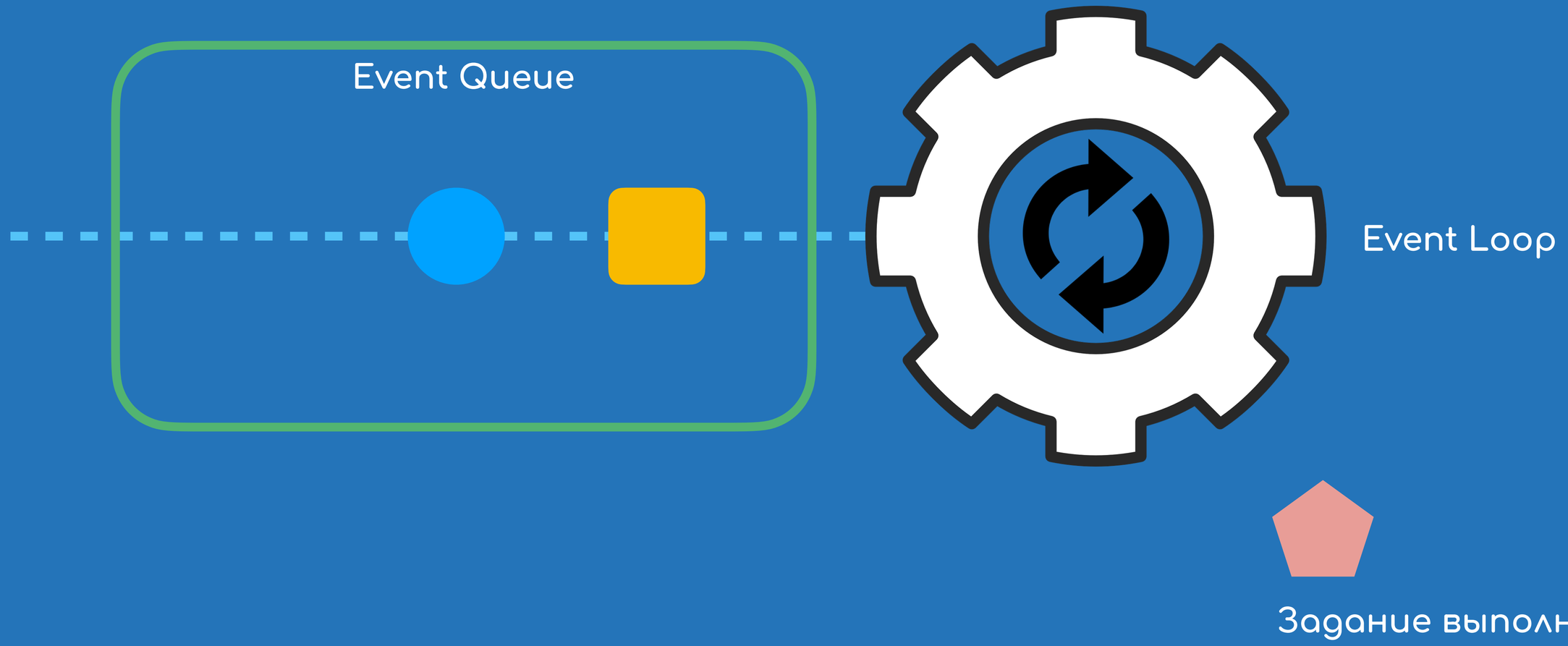
• Demo



Async & Event Loop

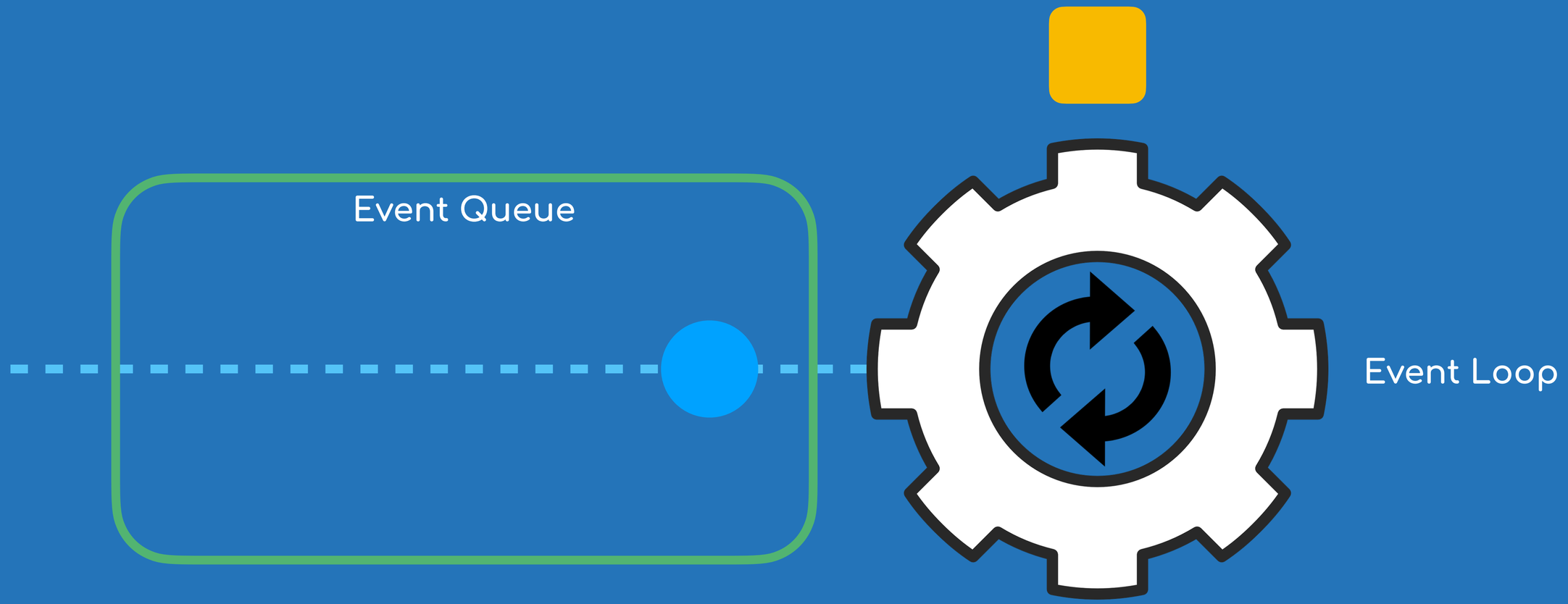
• Demo

После завершения переходит к следующему



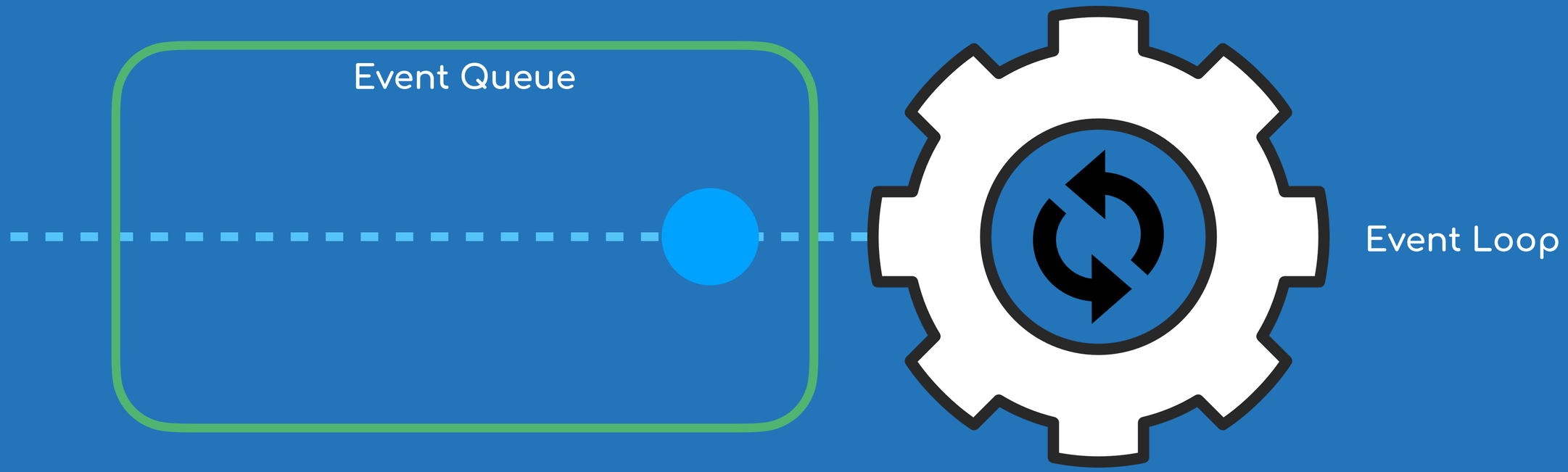
Async & Event Loop

• Demo



Async & Event Loop

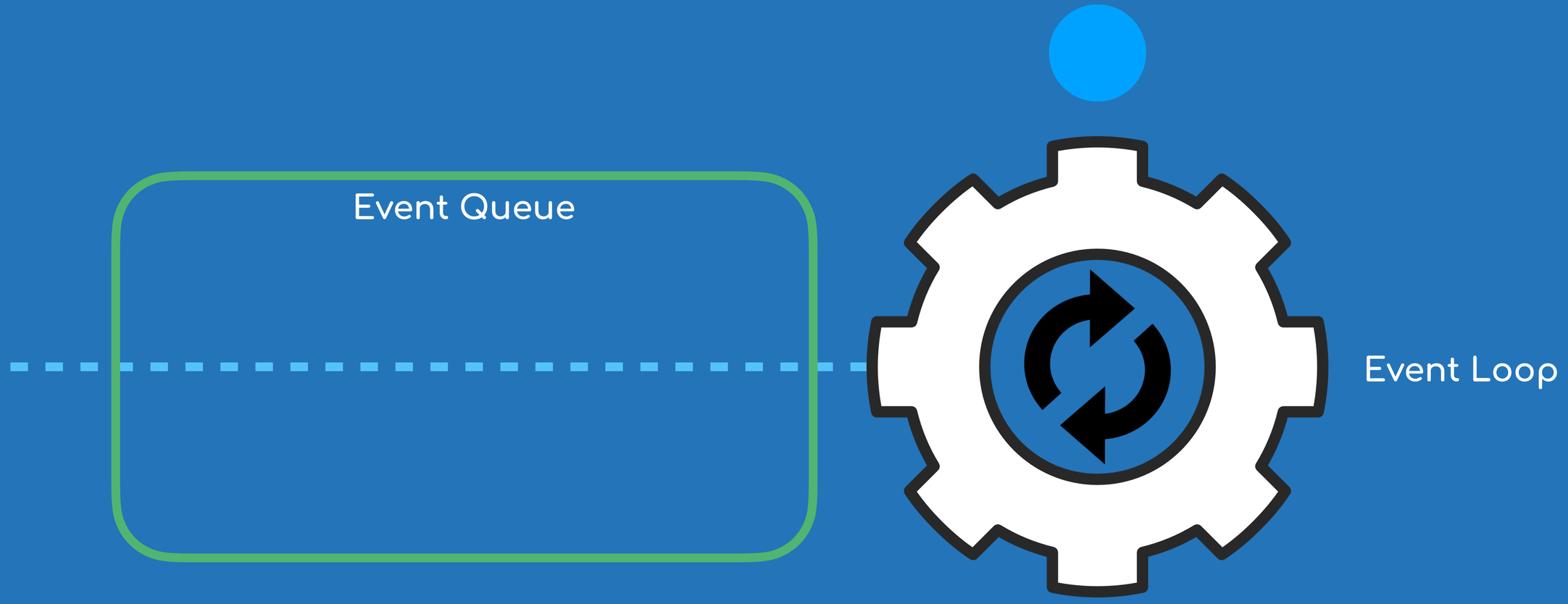
• Demo



Задание выполнено

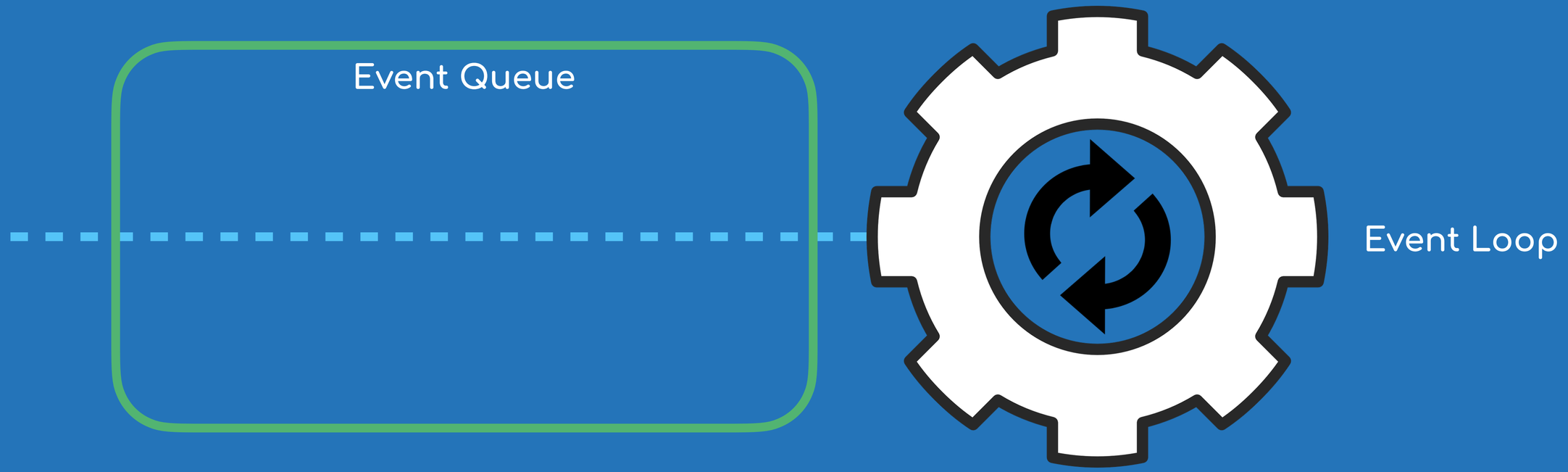
Async & Event Loop

• Demo



Async & Event Loop

• Demo



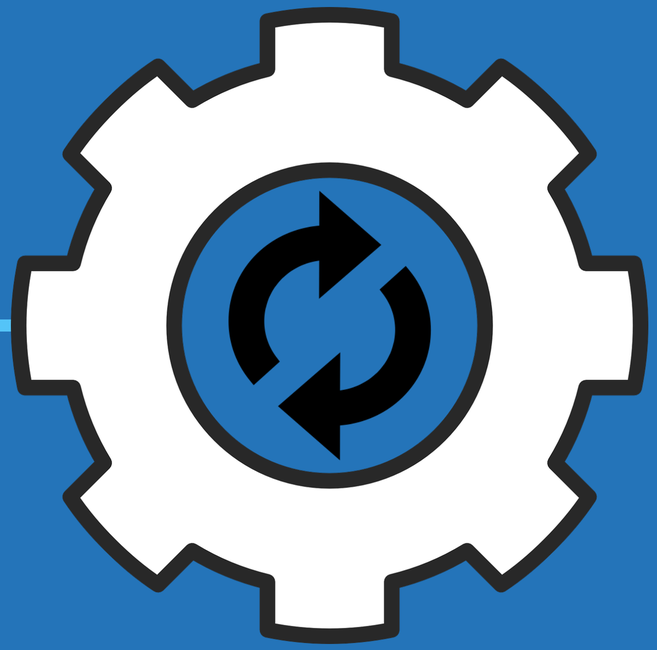
Задание выполнено

Async & Event Loop

• Demo

Ожидаем следующие события

Event Queue
Пустая



Event Loop

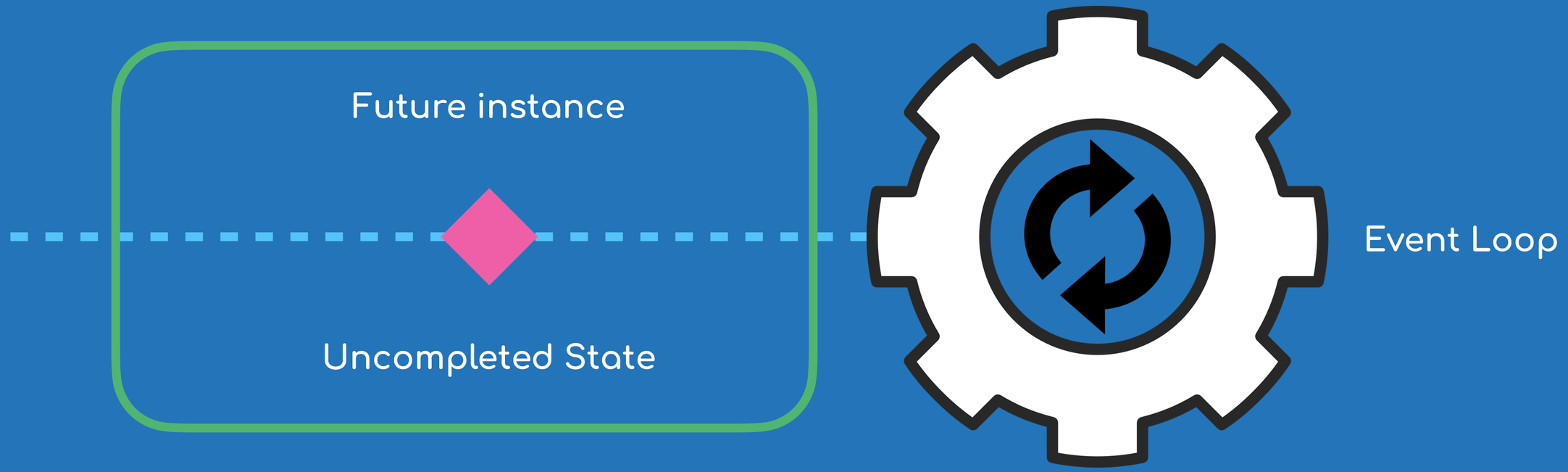
Async - Future - всего три состояния

Класс Future представляет результат отложенной операции, которая завершит свое выполнение в будущем.

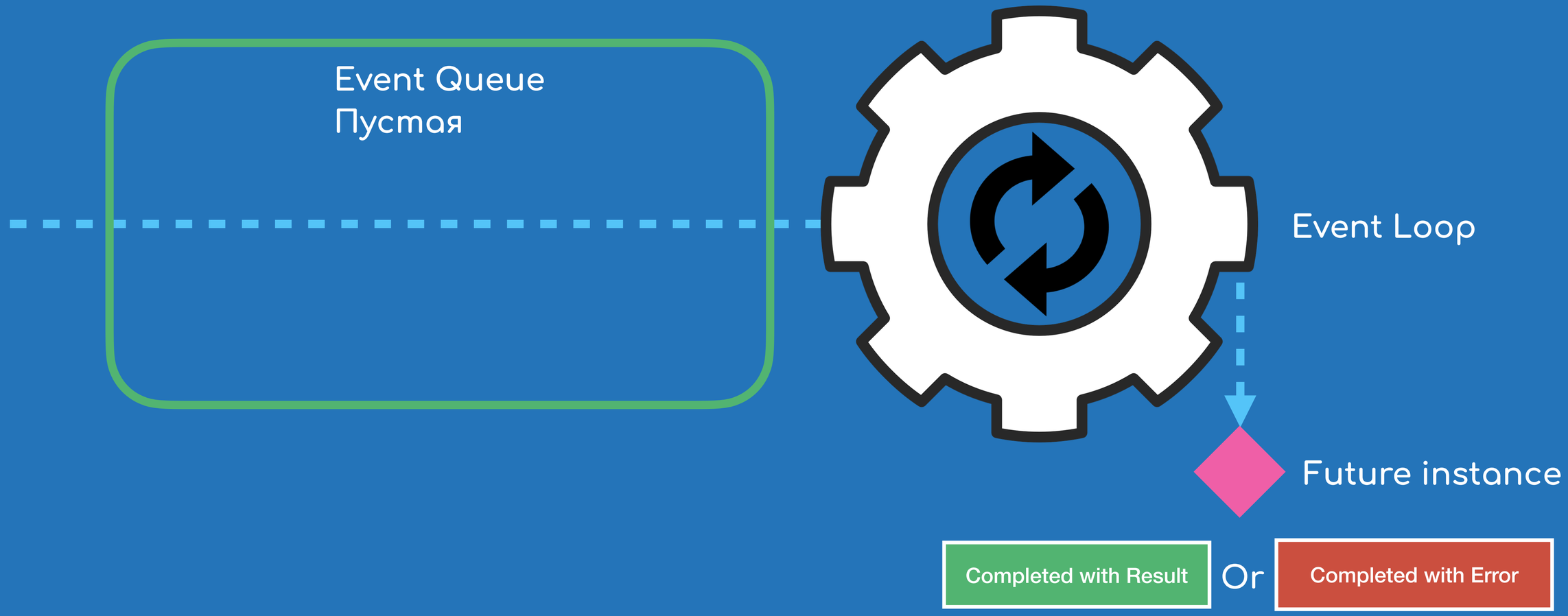
Эта отложенная операция сейчас или Незавершенна, или Завершена с определенным результатом.

Результатом операции может быть некоторое значение или ошибка.

Future & Event Loop



Future & Event Loop



Async - Future

Определение из SDK:

What is a future?
A future (lower case “f”) is an instance of the Future (capitalized “F”) class.

A future represents the result of an asynchronous operation, and can have two states: uncompleted or completed.

Async - Future - как создать и использовать?

```dart
///
/// Create Future with computation
///
void main() {
  final future = Future(() {
    print('Hello From Async operation');
  });

  print('Main method is complete!');
}
```

```dart
///
/// Create Delayed Future
///
void main() {
  final future = Future.delayed(Duration(seconds: 3),  () {
    print('Hello From Delayed Async operation');
  }); // Future.delayed

  print('Main method is complete!');
}
```

Async - Future - как создать и использовать?

```dart
///
/// Create Sync Future
///
void main() {
  final future = Future.sync(() {
    print('Hello From Delayed Async operation');
  });

  print('Main method is complete!');
}
```

```dart
void main() {
  final future = Future.sync(() {
    Future(() {
      print('Hello From Delayed Async operation');
    }); // Future
  }); // Future.sync

  print('Main method is complete!');
}
```

Async - Future - как создать и использовать?

async_04_future_resolving.dart

```dart
///
/// Create Delayed Future & Resolve the result
///
const featureResult = 5;

void main() {
  Future<int>.delayed(Duration(seconds: 3), ourComputation).then((value) {
    print('current result is $value!');
  });

  print('Main method is complete!');
}
```

Async - Future - как создать и использовать?

Обработка ошибок и callback на завершение Future

```
///
// todo: uncomment this - catchError
///
void main() {
  Future<int>.delayed(Duration(seconds: 3), ourComputation).then((value) {
    print('current result is $value!');
  }).catchError((error) {
    print(error);
  }).whenComplete(() {
    print('Операция завершена - независимо от того, с ошибкой или нет');
  });

  print('Main method is complete!');
}
```

Async - IO операции и другие

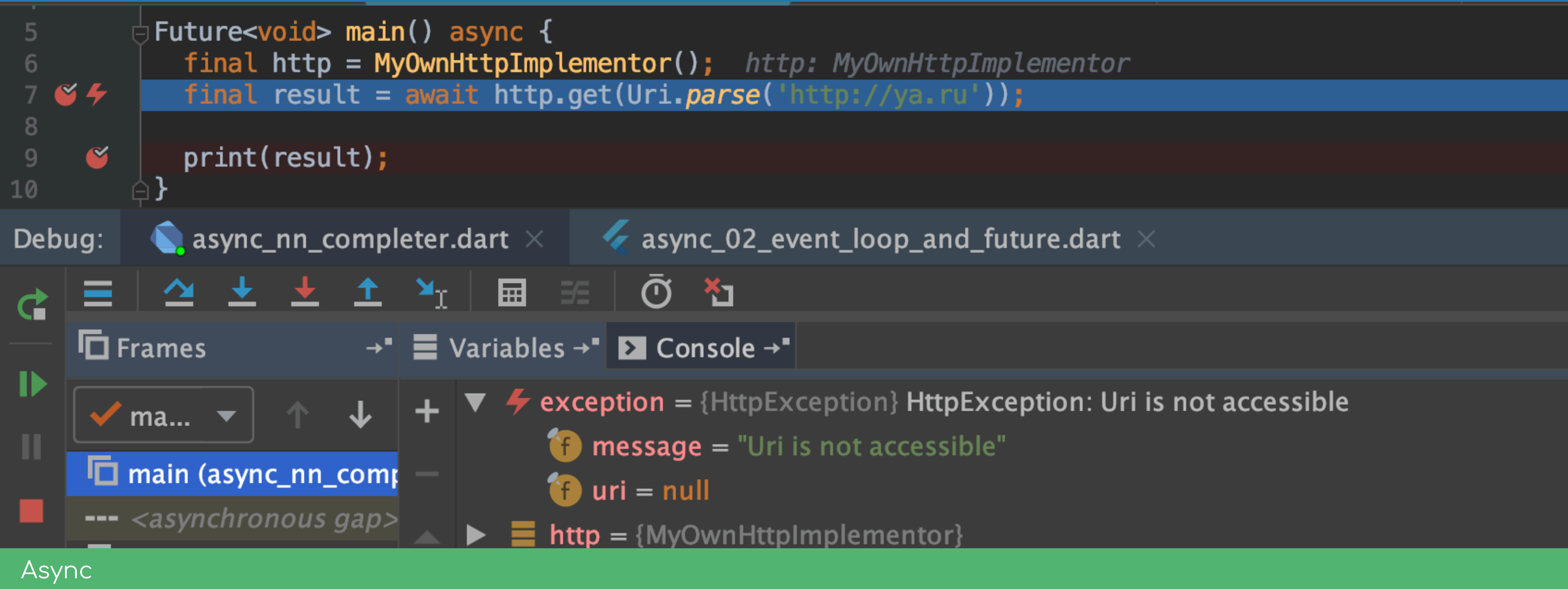
Какие типы асинхронных операций вы знаете?

```
myFuture = http.get('http://example.com');
```

```
myFuture = SharedPreferences.getInstance
```

Async - Future - как реализовать custom логику?

Completer



Future - разбираем пример с FAB

async_01_event_loop.dart | async_nn_completer.dart | async_nn_on_preset_handler.dart | future.dart

```
32      floatingActionButton: FloatingActionButton(
33        onPressed: () {
34          final http = MyOwnHttpImplementor();
35          final imageFuture = http.get(Uri.parse('http://some-image-from-network.com'));
36          imageFuture.then((Image image) {
37            setState(() {
38              _imageWidget = image;
39            });
40          });
41        },
42        tooltip: 'Increment',
43        child: Icon(Icons.file_download),
```

Future - разбираем пример с FAB

```
floatingActionButton: FloatingActionButton(
  onPressed: () {
    final http = MyOwnHttpImplementor();
    final imageFuture = http.get(Uri.parse('http://some-image-from-network.com'));
    imageFuture.then((Image image) {
      setState(() {
        _imageWidget = image;
      });
    });
  },
```

Future - Future Builder

async_05_completer.dart | async_07_on_preset_handler.dart | async_10_future_builder.dart | async.dart | future.dart | future_impl.dart

```
24          child: FutureBuilder(
25            future: _fetchNetworkData(),
26            builder: (BuildContext context, AsyncSnapshot<Image> snapshot) {
27              if (snapshot.hasData) {
28                return snapshot.data;
29              } else if (snapshot.hasError) {
30                return Text(snapshot.error.toString(), textAlign: TextAlign.center);
31              } else {
32                return Text('Ждем выполнения асинхронной операции', textAlign: TextAlign.center);
33              }
34            },
35          ), // FutureBuilder
36        ), // Container
37      ), // Center
```

Future - await async

Await Async нотация является синтаксическим сахаром, который позволяет воспринимать ваш асинхронный код выполняемый последовательно.

```dart
void main() {
  final http = MyOwnHttpImplementor();
  http
      .get(Uri.parse('http://ya.ru'))
      .then((value) {
        //  <<  Continuation >>

        print(value);
      });
  }
```

```dart
Future<void> main() async {
  final http = MyOwnHttpImplementor();
  final value = await http.get(Uri.parse('http:

  // <<  Continuation >>

  print(value);

}
```

Future - try catch

```dart
void main() {
  final http = MyOwnHttpImplementor();
  http
      .get(Uri.parse('http://ya.ru'))
      .then((value) {
        //  <<  Continuation >>

        print(value);
      })
          .catchError((error) {
        print(error);
      });
  }
```

```dart
Future<void> main() async {
  final http = MyOwnHttpImplementor();
  try {
    ///
    ///  ожидаем получение результата
    ///  после await весь код становится областью Continuati
    ///
    final value = await http.get(Uri.parse('http://ya.ru'));

    // <<  Continuation >>

    print(value);
  } catch (error) {
    print(error);
  }
}
```

Future - try catch finally

```dart
Future<void> main() async {
  final http = MyOwnHttpImplementor();
  try {
    ///
    ///  ожидаем получение результата
    ///  после await весь код становится областью Continuation
    ///
    final value = await http.get(Uri.parse('http://ya.ru'));

    // <<  Continuation >>

    print(value);
  } catch (error) {
    print(error);
  } finally {
    print('Операция завершена - независимо от того, с ошибкой или нет');
  }
}
```

Run: async_11_future_try_catch.dart

```
dart async_11_future_try_catch.dart [Observatory: http://127.0.0.1:57765/5TezPJpv52Y=/]

lib/async/async_11_future_try_catch.dart: Warning: Interpreting this as package URI, 'package:lesson04async/async/async_11_futur
HttpException: Uri is not accessible
Операция завершена - независимо от того, с ошибкой или нет

Process finished with exit code 0
```

Event Loop - все гораздо сложнее

Для самостоятельного изучения

https://webdev.dartlang.org/articles/performance/event-loop

О чем будем говорить

Dart Language - Generics, Extensions

Async

Http

Json Serialization

Mobile App - Async Repository & HomeWork

Http - Использование во Flutter

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

1. Add the `http` package

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

To install the `http` package, add it to the dependencies section of the `pubspec.yaml` file. You can find the latest version of the `http` package the pub.dev.

```
dependencies:
  http: <latest_version>
```

Import the http package.

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

Additionally, in your AndroidManifest.xml file, add the Internet permission.

```
<!-- Required to fetch data from the internet. -->
<uses-permission android:name="android.permission.INTERNET" />
```

Contents

Http - практический пример получения данных

http_using.dart

```dart
import 'dart:convert' as convert;
import 'dart:io';

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

Future<void> main() async {
  final client = http.Client();
  try {
    const url = 'https://the-cocktail-db.p.rapidapi.com/popular.php';

    //    список доступных операций
    //
    //    If you're planning on making multiple requests to
    //    the same server, you should use a single [Client] for all of those requests.
    //
    //    Future<String> response = http.get(url, ...);
    //    Future<String> response = http.put(url, ...);
    //    Future<String> response = http.post(url, ...);
    //    Future<String> response = http.delete(url, ...);
    //    Future<String> response = http.patch(url, ...);
    //    Future<String> response = http.read(url, ...);

    var response = await client.get(
      url,
```

Backend & Http Server

http_other_scenarios.dart

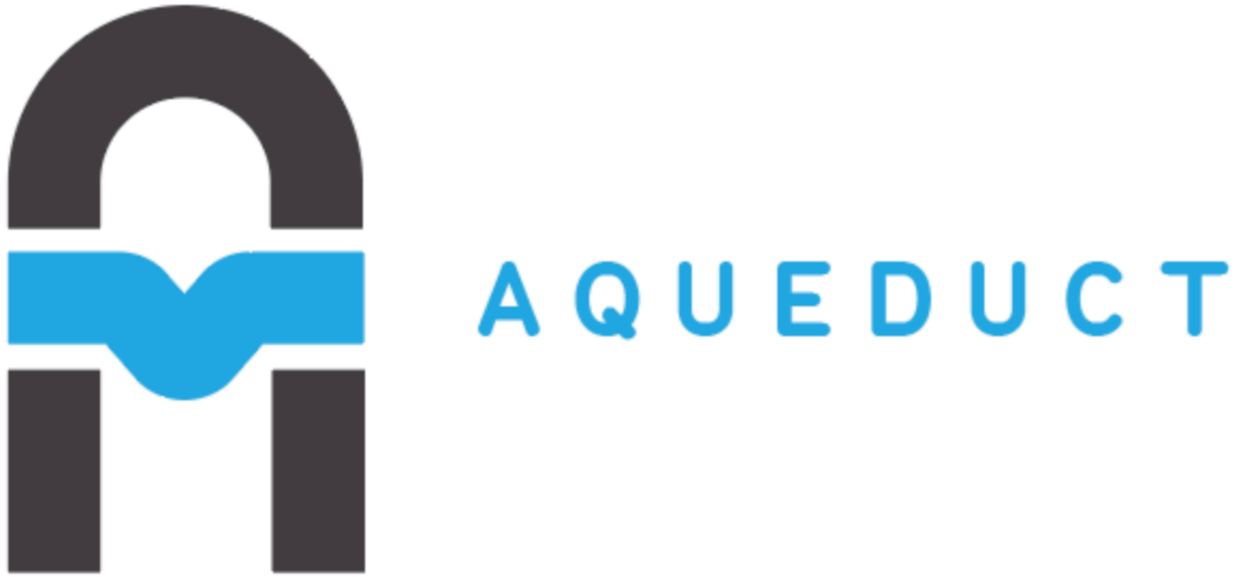
```dart
import 'dart:io';

Future main() async {
  var server = await HttpServer.bind(
    InternetAddress.loopbackIPv4,
    4040,
  );
  print('Listening on localhost:${server.port}');

  await for (HttpRequest request in server) {
    request.response.write('Hello, world!');
    await request.response.close();
  }
}
```

```
❯ curl 127.0.0.1:4040/test.html
Hello, world!%
 ~/otus/flutter-examples/json
```

Backend & Http Server



build passing build passing codecov 88%

slack 896

Aqueduct is a modern Dart HTTP server framework. The framework is composed of libraries for handling and routing HTTP requests, object-relational mapping (ORM), authentication and authorization (OAuth 2.0 provider) and documentation (OpenAPI). These libraries are used to build scalable REST APIs that run on the Dart VM.

If this is your first time viewing Aqueduct, check out the tour.

A modern HTTP server application framework, ORM and OAuth2 provider with OpenAPI 3.0 integration. Foundation for REST, RPC or GraphQL services.

Homepage

Documentation

API reference

Uploader

joe.conway@stablekernel.com

License

BSD (LICENSE)

Dependencies

analyzer, args, crypto, isolate_executor, logging, meta, open_api, password_hash, path, postgres, pub_cache,

Backend & Http Server

Not Secure | aqueduct.io/docs/tut/getting-started/

Tutorial / 1. Getting Started

Search

GitHub
2.1k Stars · 156 Forks


```dart
import 'package:aqueduct/aqueduct.dart';
import 'package:heroes/heroes.dart';

class HeroesController extends Controller {
  final _heroes = [
    {'id': 11, 'name': 'Mr. Nice'},
    {'id': 12, 'name': 'Narco'},
    {'id': 13, 'name': 'Bombasto'},
    {'id': 14, 'name': 'Celeritas'},
    {'id': 15, 'name': 'Magneta'},
  ];

  @override
  Future<RequestOrResponse> handle(Request request) async {
    return Response.ok(_heroes);
  }
}
```

Notice that `HeroesController` is a subclass of `Controller`; this is what makes it a controller object. It overrides its `handle` method by returning a `Response` object. This response object has a 200 OK status code, and it body contains a JSON-encoded list of hero objects. When a controller returns a `Response` object from its `handle` method, it is sent to the client.

Right now, our `HeroesController` isn't hooked up to the application channel. We need to link it

Backend & Http Server

Not Secure | aqueduct.io/docs/tut/getting-started/

Tutorial / 1. Getting Started

Search

GitHub
2.1k Stars · 156 Forks


```
@override
Controller get entryPoint {
  final router = Router();

  router
    .route('/heroes')
    .link(() => HeroesController());

  router
    .route('/example')
    .linkFunction((request) async {
      return Response.ok({'key': 'value'});
    });

  return router;
}
```

We now have a application that will return a list of heroes. In the project directory, run the following command from the command-line:

```
aqueduct serve
```

This will start your application running locally. Reload the browser page `http://aqueduct-tutorial.stablekernel.io`. It will make a request to `http://localhost:8888/heroes`

О чем будем говорить

Dart Language - Generics, Extensions

Async

Http

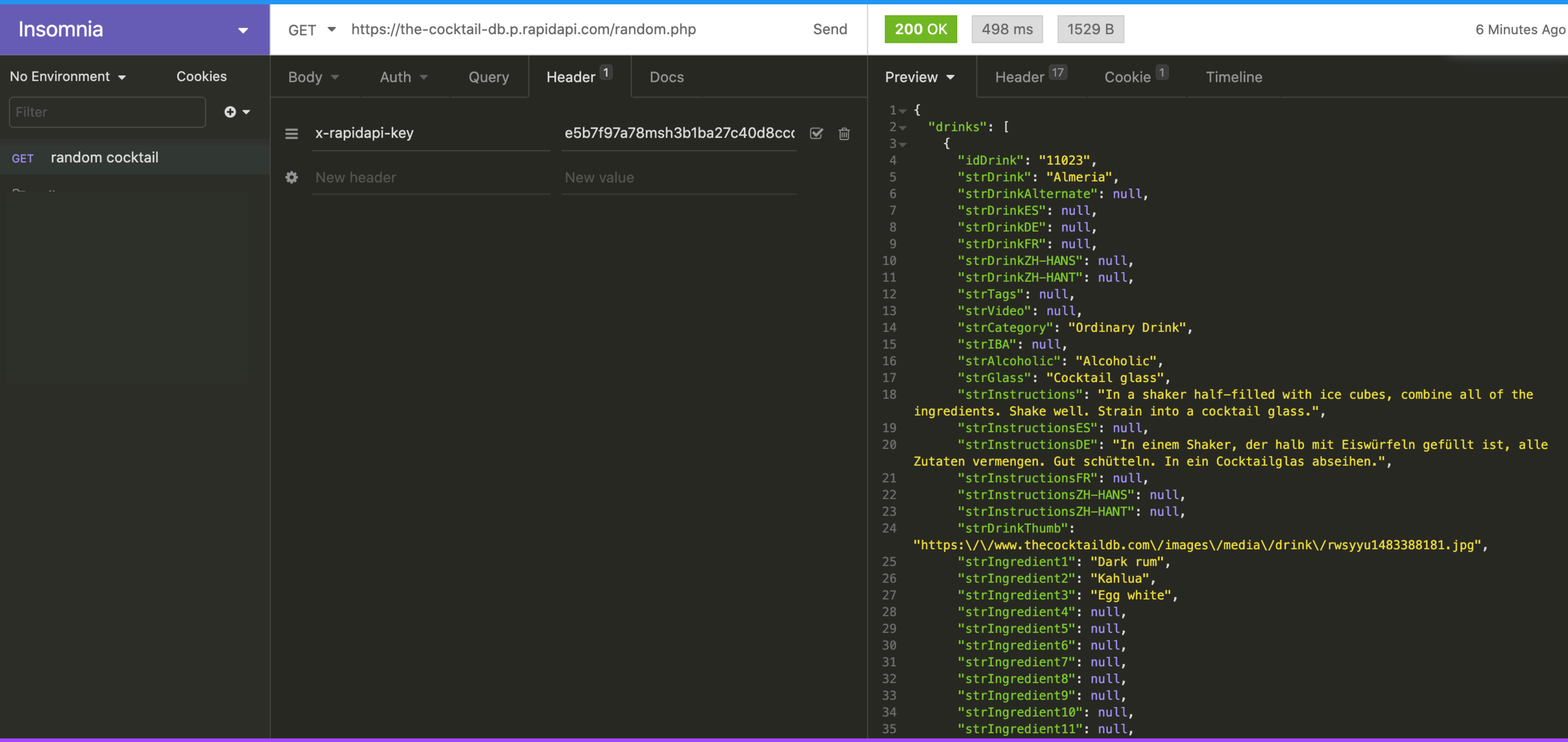
Json Serialization

Mobile App - Async Repository & HomeWork

Http Response - Json Structure

```
❯ curl https://the-cocktail-db.p.rapidapi.com/random.php -H "x-rapidapi-key : e5b7f97a78msh3b1ba27c40d8ccdp105034jsn34e2da32d50b"| jora -p
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1616  100  1616    0     0   4941      0 --:--:-- --:--:-- --:--:--  4926
{
    "drinks": [
        {
            "idDrink": "14195",
            "strDrink": "Snowball",
            "strDrinkAlternate": null,
            "strDrinkES": null,
            "strDrinkDE": null,
            "strDrinkFR": null,
            "strDrinkZH-HANS": null,
            "strDrinkZH-HANT": null,
            "strTags": null,
            "strVideo": null,
            "strCategory": "Ordinary Drink",
            "strIBA": null,
            "strAlcoholic": "Alcoholic",
            "strGlass": "Highball glass",
            "strInstructions": "Place one ice cube in the glass and add 1 1/2 oz of Advocaat. Fill up the glass with lemonade and decorate with
 a slice of lemon. Serve at once.",
            "strInstructionsES": null,
            "strInstructionsDE": "Einen Eiswürfel in das Glas geben und 4,5 cl. Advocaat hinzufügen. Das Glas mit Limonade füllen und mit einer
 Scheibe Zitrone dekorieren. Sofort servieren.",
            "strInstructionsFR": null,
```

Http Response - Json Structure



Json de|serialization

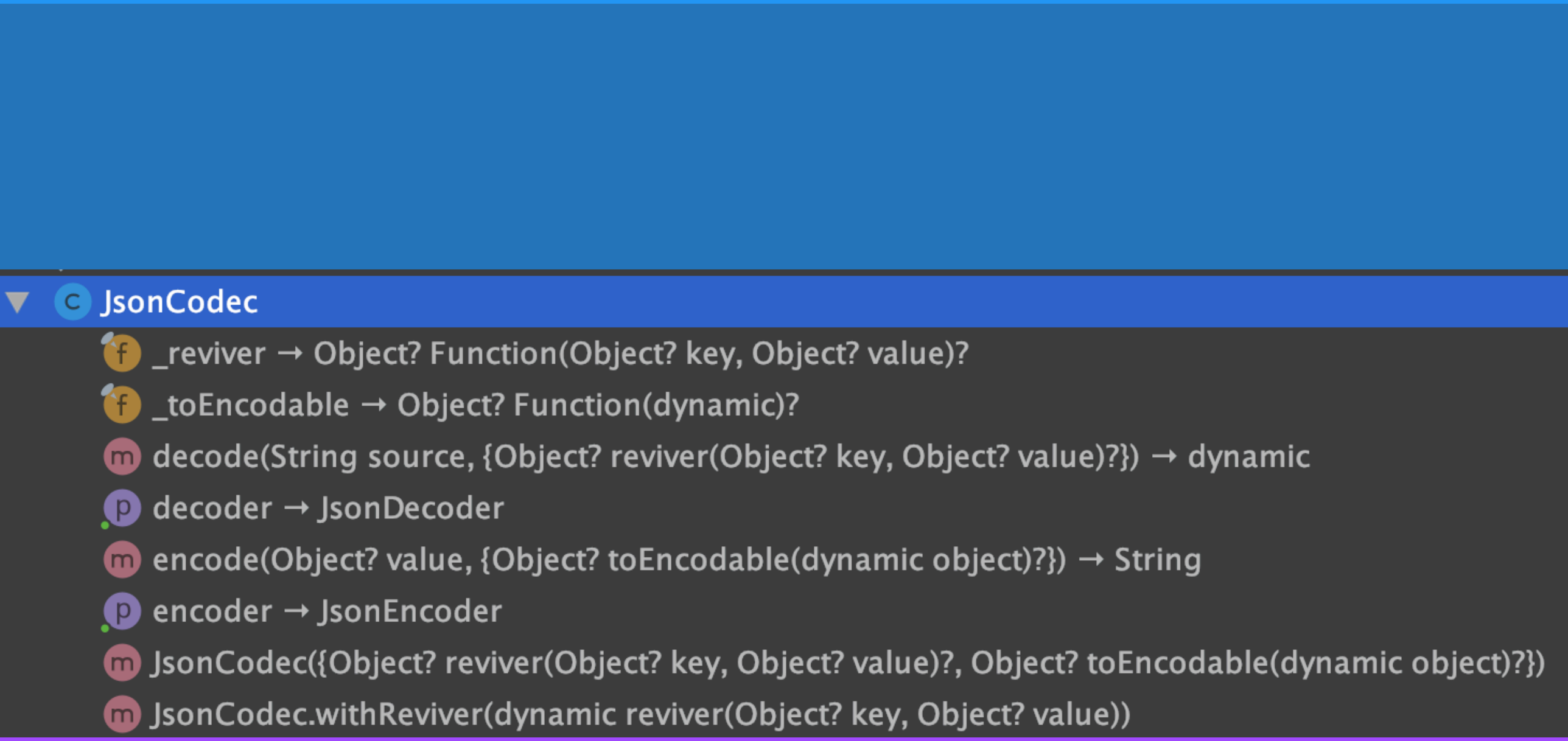
Три способа выполнить десериализацию данных

1. Вручную, используя dart:convert
2. Используя json_annotation & json_serializable
3. Используя built_value

1. Json de|serialization - dart:convert

```dart
/// An instance of the default implementation of the [JsonCodec].
///
/// This instance provides a convenient access to the most common JSON
/// use cases.
///
/// Examples:
///
///     var encoded = json.encode([1, 2, { "a": null }]);
///     var decoded = json.decode('["foo", { "bar": 499 }]');
///
/// The top-level [jsonEncode] and [jsonDecode] functions may be used instead if
/// a local variable shadows the [json] constant.
const JsonCodec json = JsonCodec();
```

1. Json de|serialization - dart:convert



1. Json de|serialization - dart:convert

json_01_example.dart

```dart
import 'dart:convert';

const String httpResponse = '''
{
    "idDrink": "11408",
    "strDrink": "Gin Daisy",
    "strCategory": "Ordinary Drink",
    "strAlcoholic": "Alcoholic",
    "strGlass": "Old-fashioned glass"
}
''';

void main() {
  Map<String, dynamic> data = jsonDecode(httpResponse);
  final cocktail = Cocktail.fromJson(data);
  assert(cocktail.id == "11408");
}

class Cocktail {
  final String id;
  final String drink;
  final String category;
  final String alcoholic;
  final String glass;

  Cocktail(this.id, this.drink, this.category, this.alcoholic, this.glass);

  Cocktail.fromJson(Map<String, dynamic> json)
```

Другие кодеки в dart:convert

```
54  library dart.convert;
55
56  import 'dart:async';
57  import 'dart:typed_data';
58  import 'dart:_internal' show CastConverter, parseHexByte;
59
60  part 'ascii.dart';
61  part 'base64.dart';
62  part 'byte_conversion.dart';
63  part 'chunked_conversion.dart';
64  part 'codec.dart';
65  part 'converter.dart';
66  part 'encoding.dart';
67  part 'html_escape.dart';
68  part 'json.dart';
69  part 'latin1.dart';
70  part 'line_splitter.dart';
71  part 'string_conversion.dart';
72  part 'utf.dart';
```

2. Json de|serialization - json_annotation & json_serializable

github.com/google/json_serializable.dart

build passing

Provides source_gen `Generator`s to create code for JSON serialization and deserialization.

json_serializable pub v3.5.0

- Package: https://pub.dev/packages/json_serializable
- Source code

The core package providing Generators for JSON-specific tasks.

Import it into your pubspec `dev_dependencies:` section.

json_annotation pub v3.1.0

- Package: https://pub.dev/packages/json_annotation
- Source code

The annotation package which has no dependencies.

Import it into your pubspec `dependencies:` section.

+ 8 contributors

Languages

Dart 99.5% Shell 0.5%

json_annotation 3.1.0

DART | NATIVE | JS | FLUTTER | ANDROID | IOS | WEB

117

Readme | Changelog | Installing | Versions | Scores

pub v3.1.0

Defines the annotations used by json_serializable to create code for JSON serialization and deserialization.

See the example to understand how to configure your package.

Features and bugs

Please file feature requests and bugs at the issue tracker.

json_serializable: ^3.5.0

Readme | Changelog | Example | Installing | Versions | Scores

pub v3.5.0

Provides Dart Build System builders for handling JSON.

The builders generate code when they find members annotated with classes defined in package:json_annotation.

- To generate to/from JSON code for a class, annotate it with `@JsonSerializable`. You can provide arguments to `JsonSerializable` to configure the generated code. You can also customize individual fields by annotating them with `@JsonKey` and providing custom arguments. See the table below for details on the annotation values.

- To generate a Dart field with the contents of a file containing JSON, use the `JsonLiteral` annotation.

2. json_annotation - описываем модель

cocktail.dart

```dart
import 'package:json_annotation/json_annotation.dart';


///
/// это будет сгенерированный файл
///
part 'cocktail.g.dart';

@JsonSerializable(createToJson: true, createFactory: true)
class Cocktail {

  @JsonKey(name: 'idDrink', nullable: false, required: true)
  final String id;

  @JsonKey(name: 'strDrink', nullable: false, required: true)
  final String drink;

  @JsonKey(name: 'strCategory', nullable: false, required: true)
  final String category;

  @JsonKey(name: 'strAlcoholic', nullable: false, required: true)
  final String alcoholic;
```

2. Json de|serialization - json_annotation & json_serializable

```
flutter packages pub run build_runner build
```

2. Json de|serialization - pub run build_runner build

```
❯ flutter packages pub run build_runner build
[INFO] Generating build script...
[INFO] Generating build script completed, took 371ms

[INFO] Creating build script snapshot......
[INFO] Creating build script snapshot... completed, took 12.2s

[INFO] Initializing inputs
[INFO] Building new asset graph...
[INFO] Building new asset graph completed, took 658ms

[INFO] Checking for unexpected pre-existing outputs....
[INFO] Checking for unexpected pre-existing outputs. completed, took 1ms

[INFO] Running build...
[INFO] Generating SDK summary...
[INFO] 3.7s elapsed, 0/16 actions completed.
[INFO] Generating SDK summary completed, took 3.7s

[INFO] 4.7s elapsed, 2/18 actions completed.
[INFO] 5.9s elapsed, 2/18 actions completed.
[INFO] 7.1s elapsed, 3/19 actions completed.
[INFO] 8.2s elapsed, 5/21 actions completed.
[INFO] 9.3s elapsed, 7/23 actions completed.
[INFO] 10.3s elapsed, 7/23 actions completed.
[INFO] 13.9s elapsed, 7/23 actions completed.
[INFO] 18.5s elapsed, 18/23 actions completed.
[INFO] Running build completed, took 18.8s
```

2. json_annotation - результат кодогенерации

cocktail.g.dart

```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'cocktail.dart';

// **************************************************************************
// JsonSerializableGenerator
// **************************************************************************

Cocktail _$CocktailFromJson(Map<String, dynamic> json) {
  $checkKeys(json, requiredKeys: const [
    'idDrink',
    'strDrink',
    'strCategory',
    'strAlcoholic',
    'strGlass'
  ]);
  return Cocktail(
    json['idDrink'] as String,
    json['strDrink'] as String,
    json['strCategory'] as String,
    json['strAlcoholic'] as String,
    json['strGlass'] as String,
  );
}

Map<String, dynamic> _$CocktailToJson(Cocktail instance) => <String, dynamic>{
```

3. Json de|serialization - built_value

pub.dev/packages/built_value

Flutter Favorite

built_value 7.1.0

Published Apr 23, 2020 • dart.dev

DART | NATIVE | JS | FLUTTER | ANDROID | IOS | WEB

186

Readme | Changelog | Example | Installing | Versions | Scores

Built Values for Dart

build failing

Introduction

Built Value provides:

- Immutable value types;
- EnumClass, classes that behave like enums;
- JSON serialization.

Immutable collections are from built_collection.

186 LIKES | 110 PUB POINTS | 99% POPULARITY

Publisher

dart.dev

Metadata

Value types with builders, Dart classes as enums, and serialization. This library is the runtime dependency.

Repository (GitHub)

View/report issues

Documentation

API reference

3. Json de|serialization - built_value

For an end to end example see the chat example, which was demoed at the Dart Summit 2016. The data model, used both client and server side, uses value types, enums and serialization from built_value.

Simple examples are here.

Since `v5.2.0` codegen is triggered by running `pub run build_runner build` to do a one-off build or `pub run build_runner watch` to continuously watch your source and update the generated output when it changes. Note that you need a dev dependency on `built_value_generator` and `build_runner`. See the example pubspec.yaml.

If using Flutter, the equivalent command is `flutter packages pub run build_runner build`. Alternatively, put your `built_value` classes in a separate Dart package with no dependency on Flutter. You can then use `built_value` as normal.

If using a version before v5.2.0, codegen is triggered via either a build.dart to do a one-off build or a watch.dart to continuously watch your source and update generated output.

3. Json de|serialization - built_value_generator 7.1.0

Readme | Changelog | Example | Installing | Versions | Scores

Built Values for Dart

build failing

Introduction

Built Value provides:

- Immutable value types;
- EnumClass, classes that behave like enums;
- JSON serialization.

3. BuiltValue - описываем модель

built_value_approach/cocktail.dart

```dart
import 'package:built_value/built_value.dart';
import 'package:built_value/serializer.dart';

part 'cocktail.g.dart';

///
/// Класс модели абстрактный
///
abstract class Cocktail implements Built<Cocktail, CocktailBuilder> {
  static Serializer<Cocktail> get serializer => _$cocktailSerializer;

  ///
  /// нет полей, только геттеры
  ///
  String get idDrink;
  String get strDrink;
  String get strCategory;
  String get strAlcoholic;
  String get strGlass;

  ///
  /// Конструктор делаем приватным
  ///
  Cocktail._();

  ///
  /// Объявляем внешнюю фабрику для создания объектов
  ///
  factory Cocktail([updates(CocktailBuilder b)]) = _$Cocktail;
```

3. Json de|serialization - built_value_generator 7.1.0

```
flutter packages pub run build_runner build
```

3. BuiltValue - результаты кодогенерации

cocktail.g.dart

```
11  class _$CocktailSerializer implements StructuredSerializer<Cocktail> {
12    @override
13    final Iterable<Type> types = const [Cocktail, _$Cocktail];
14    @override
15    final String wireName = 'Cocktail';
16
17    @override
18    Iterable<Object> serialize(Serializers serializers, Cocktail object,
19        {FullType specifiedType = FullType.unspecified}) {...}
40
41    @override
42    Cocktail deserialize(Serializers serializers, Iterable<Object> serialized,
43        {FullType specifiedType = FullType.unspecified}) {
44      final result = new CocktailBuilder();
45
46      final iterator = serialized.iterator;
47      while (iterator.moveNext()) {
48        final key = iterator.current as String;
49        iterator.moveNext();
50        final dynamic value = iterator.current;
51        switch (key) {
52          case 'idDrink':
53            result.idDrink = serializers.deserialize(value,
54                specifiedType: const FullType(String)) as String;
55            break;
56          case 'strDrink':
57            result.strDrink = serializers.deserialize(value,
```

3. BuiltValue - используем сериализатор

```dart
import 'package:lesson04async/json/built_value_approach/cocktail.dart';
import 'package:lesson04async/json/built_value_approach/cocktail_serializers.dart';

const String httpResponse = '''
{
    "idDrink": "11408",
    "strDrink": "Gin Daisy",
    "strCategory": "Ordinary Drink",
    "strAlcoholic": "Alcoholic",
    "strGlass": "Old-fashioned glass"
}
''';

void main() {
  Map<String, dynamic> data = jsonDecode(httpResponse);

  var cocktail = serializers.deserializeWith(Cocktail.serializer, data);
  assert(cocktail.idDrink == "11408");
```

3. BuiltValue - используем сериализатор

github.com/google/built_value.dart#serialization

Serialization

Built Values comes with JSON serialization support which allows you to serialize a complete data model of Built Values, Enum Classes and Built Collections. The chat example shows how easy this makes building a full application with Dart on the server and client.

Here are the major features of the serialization support:

It *fully supports object oriented design*: any object model that you can design can be serialized, including full use of generics and interfaces. Some other libraries require concrete types or do not fully support generics.

It *allows different object oriented models over the same data*. For example, in a client server application, it's likely that the client and server want different functionality from their data model. So, they are allowed to have different classes that map to the same data. Most other libraries enforce a 1:1 mapping between classes and types on the wire.

It *requires well behaved types*. They must be immutable, can use interface but not concrete inheritance, must have predictable nullability, `hashCode`, `equals` and `toString`. In fact, they must be Enum Classes, Built Collections or Built Values. Some other libraries allow badly behaved types to be serialized.

It *supports changes to the data model*. Optional fields can be added or removed, and fields can be switched from optional to required, allowing your data model to evolve without breaking compatbility. Some other libraries break compatibility on any change to any serializable class.

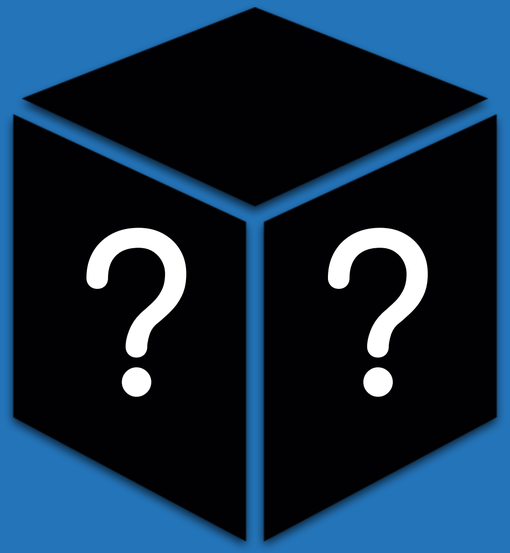
It's *modular*. Each endpoint can choose which classes to know about; for example, you can have multiple clients

Json de|serialization - некоторые выводы

Choosing a library: You might have noticed two Flutter Favorite packages on pub.dev that generate JSON serialization code, `json_serializable` and `built_value`. How do you choose between these packages? The `json_serializable` package allows you to make regular classes serializable by using annotations, whereas the `built_value` package provides a higher-level way of defining immutable value classes that can also be serialized to JSON.

Немного про кодогенерацию, и про bin-пакеты

flutter packages pub run build_runner build



build_runner https://pub.dev/packages/build

build_runner 1.10.3

Published Sep 22, 2020 • dart.dev

DART | NATIVE

206

Readme Changelog Installing Versions Scores

Standalone generator and watcher for Dart using `package:build`.

The `build_runner` package provides a concrete way of generating files using Dart code, outside of tools like `pub`. Unlike `pub serve/build`, files are always generated directly on disk, and rebuilds are *incremental* - inspired by tools such as Bazel.

build_runner commands

Built-in Commands

The `build_runner` package exposes a binary by the same name, which can be invoked using `pub run build_runner <command>`.

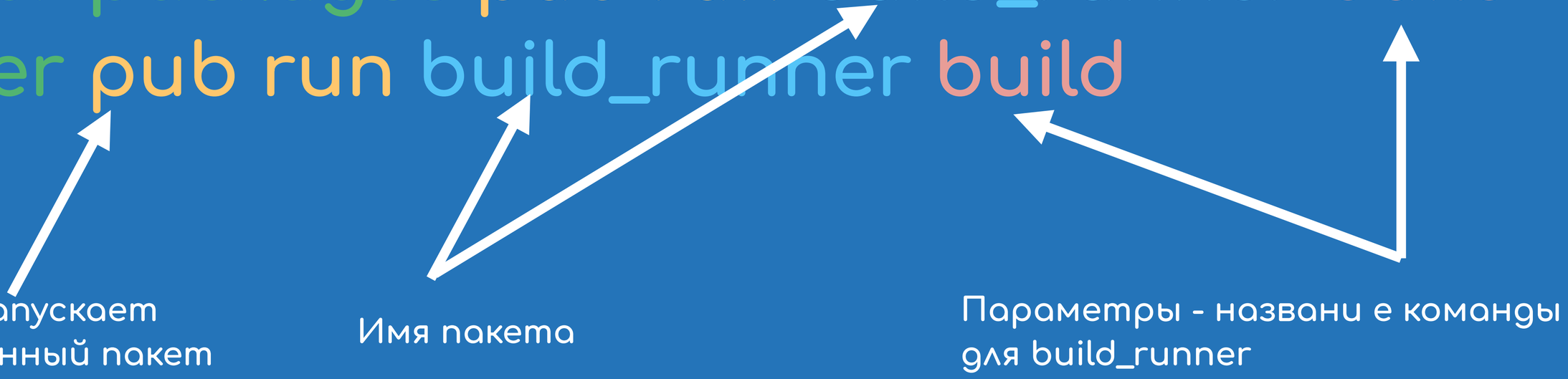
The available commands are `build`, `watch`, `serve`, and `test`.

- `build`: Runs a single build and exits.
- `watch`: Runs a persistent build server that watches the files system for edits and does rebuilds as necessary.
- `serve`: Same as `watch`, but runs a development server as well.
  - By default this serves the `web` and `test` directories, on port `8080` and `8081` respectively. See below for how to configure this.
- `test`: Runs a single build, creates a merged output directory, and then runs `pub run test --precompiled <merged-output-dir>`. See below for instructions on passing custom args to the test command.

build_runner commands - генерация кода

2. Json de|serialization - json_annotation & json_serializable

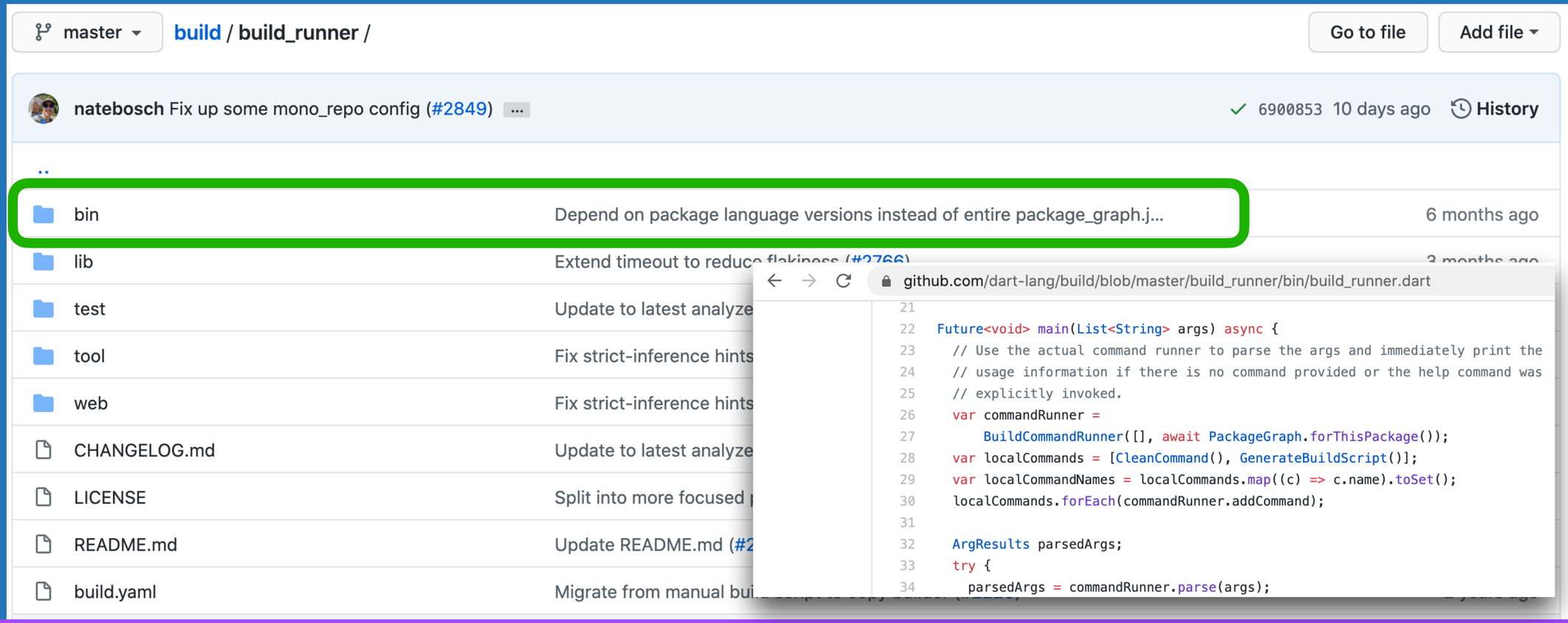
```
flutter packages pub run build_runner build
flutter pub run build_runner build
```



build_runner - это запускаемый пакет (/bin)

```
flutter pub run build_runner build
```

Запускаем



Как написать свой custom generator

source_gen 0.9.7+1

Published Sep 22, 2020

DART | NATIVE

20

Readme | Changelog | Installing | Versions | Scores

Overview

`source_gen` provides utilities for automated source code generation for Dart:

- A **framework** for writing Builders that consume and produce Dart code.
- A **convention** for human and tool generated Dart code to coexist with clean separation, and for multiple code generators to integrate in the same project.

It's main purpose is to expose a developer-friendly API on top of lower-level packages like the analyzer or build. You don't *have* to use `source_gen` in order to generate source code; we also expose a set of library APIs that might be useful in your generators.

20 LIKES | 80 PUB POINTS | 99% POPULARITY

Metadata

Source code generation builders and utilities for the Dart build system

Repository (GitHub)

View/report issues

Documentation

API reference

Uploaders

Source Gen

Quick Start Guide for writing a Generator

Add a dependency on `source_gen` in your pubspec.

```
dependencies:
  source_gen:
```

If you're only using `source_gen` in your own project to generate code and you won't publish your Generator for others to use, it can be a `dev_dependency`:

```
dev_dependencies:
  source_gen:
```

Once you have `source_gen` setup, you should reference the examples below.

Writing a generator to output Dart source code

Extend the `Generator` or `GeneratorForAnnotation` class and `source_gen` will call your generator for a Dart library or for each element within a library tagged with the annotation you are interested in.

nbosch@google.com

jakemac@google.com

matanl@google.com

License

BSD (LICENSE)

Dependencies

analyzer, async, build, dart_style, glob, meta, path, pedantic, source_span

More

Packages that depend on source_gen

Source Gen & Json Serializeble

json_serializable_generator.dart

```dart
/.../

import ...

class JsonSerializableGenerator
    extends GeneratorForAnnotation<JsonSerializable> {
  static const _coreHelpers = <TypeHelper>[
    IterableHelper(),
    MapHelper(),
    EnumHelper(),
    ValueHelper(),
  ]; // <TypeHelper>[]

  static const _defaultHelpers = <TypeHelper>[
    BigIntHelper(),
    DateTimeHelper(),
    DurationHelper(),
    JsonHelper(),
    UriHelper(),
  ]; // <TypeHelper>[]

  final List<TypeHelper> _typeHelpers;

  Iterable<TypeHelper> get _allHelpers => const <TypeHelper>[
        ConvertHelper(),
        JsonConverterHelper(),
        GenericFactoryHelper(),
```

Немного про кодогенерацию

[Part 1] Code generation in Dart: the basics

https://github.com/dart-lang/source_gen

О чем будем говорить

Dart Language - Generics, Extensions

Async

Http

Json Serialization

Mobile App - Async Repository & HomeWork

Mobile App - Async Repository & HomeWork

Dart. Async и работа с сетью. Реализуем свой API Service.

Mobile App - Async Repository & HomeWork

Dart. Async и работа с сетью. Реализуем свой API Service.

Цель домашнего задания - получить навык работы с HTTP, Json сериализацией/десериализацией, навык работы с dart:async (futures, async, await).

Пишем сервис CocktailDBApiService.

Данный сервис будет дорабатываться в последующих уроках в процессе разработки нашего мобильного приложения.

Целью данного сервиса будет получение реальных данных от публично доступного API (https://rapidapi.com/theapiguy/api/the-cocktail-db/details)

Mobile App - Async Repository & HomeWork

Dart. Async и работа с сетью. Реализуем свой API Service.

- Склонировать github репозиторий с заготовкой приложения
- Внести изменения в класс CocktailDBApiService в соот-щем месте, отмеченном комментарием с /// TODO
- TODO будет содержать Enpoint URL используемого АПИ и формат обращения к нему
- Реализовать класс модели для описания Ингредиента
- Реализовать класс соот-щего DTO (Data Transfer Object), используемого для десериализации состояния Ингредиента
- Реализовать код для получения полного описания используемого ингредиента из стороннего API

На усмотрение студента:
- Использовать фабричные конструкторы для получения модели Ингредиента из соот-щего DTO
- Можно выполнить рефакторинг кода, внеся логгинг для логгирования операций и ошибок HTTP
- Можно выполнять любую декомпозицию в коде обращения к API

Mobile App - Async Repository & HomeWork

Dart. Async и работа с сетью. Реализуем свой API Service.

Форма сдачи:
- ДЗ Сдается в виде ссылки на github репозиторий с проектом

Куда сдать ДЗ:
- Отправляется напрямую в slack преподавателям (Андрей, Никита)

Куда и кому задавать вопросы, если они возникнут
- По всем вопросам можно обращаться в Slack к студентам, преподавателям в канал группы

Итоги

Dart Language - Generics, Extensions

Async

Http

Json Serialization

Mobile App - Async Repository & HomeWork

Заполните, пожалуйста,
опрос о занятии по ссылке в чате

Спасибо за внимание!
Приходите на следующие вебинары

Смирнов Андрей wrike

Курс **Мобильная разработка на Flutter**