

Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 1
Revision

Lecture 10



Chapter

10

Table Of Content

- Introduction
- Rest Api
- Pub.dev
- Dio
- FakeStore API Command-Line Application



Introduction

Introduction

What is REST API?

Definition:

A *REST* (Representational State Transfer) API lets two systems talk to each other over HTTP, like a waiter taking your order at a restaurant.





Rest Apis

Rest API

Key Concepts

Resources

- *What are they?*
Resources are like nouns: "Users," "Products," "Cats with funny hats."

URL for a resource:

`GET /users/123`

- *(Fetch details of user #123)*

Roast:

- REST resources act like VIPs:

You must approach with the exact ID and format, or they'll ignore you.



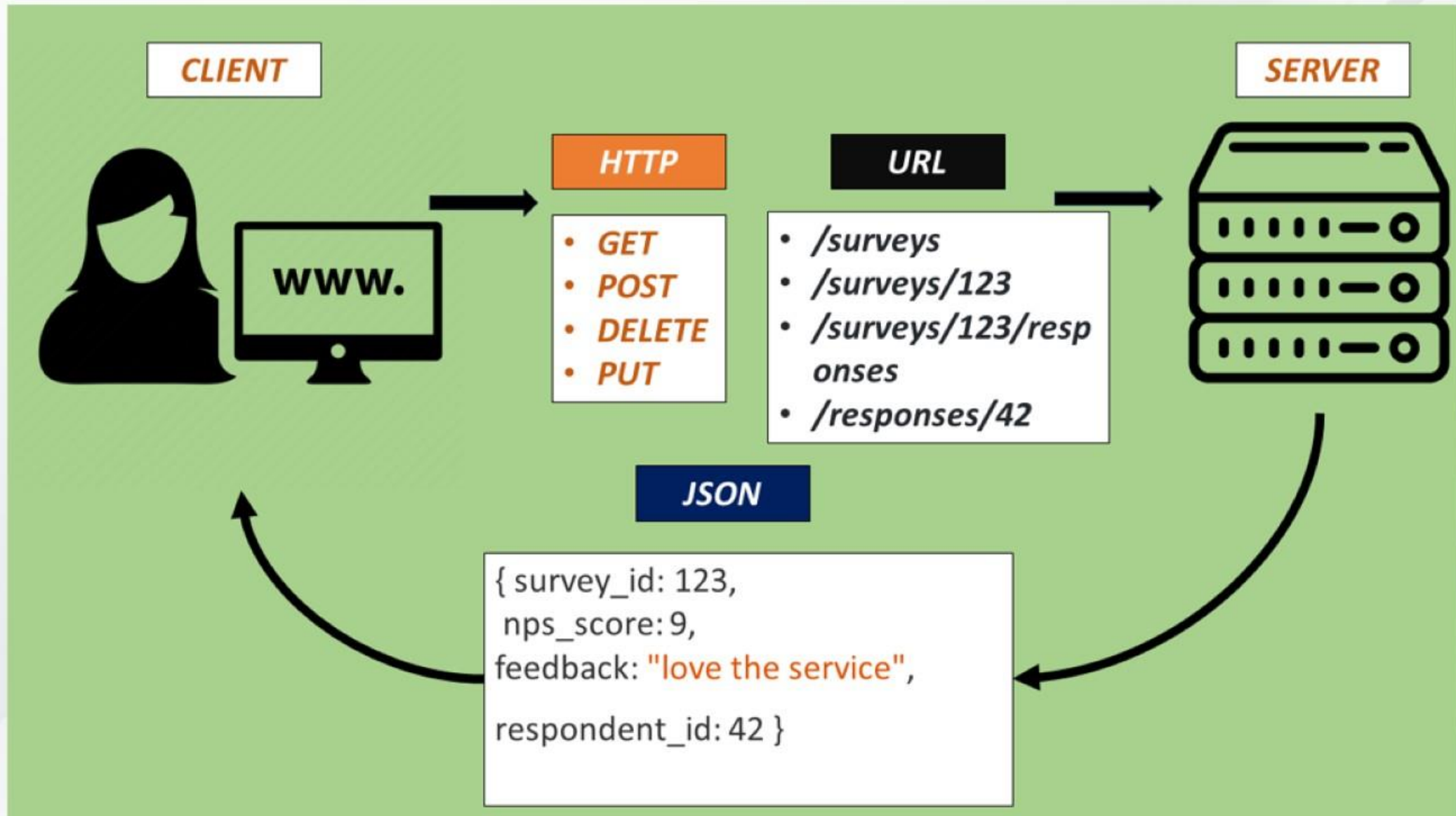
Rest API

HTTP Methods

REST uses HTTP methods to interact with resources. Think of them like verbs:

HTTP Method	Action	Example
GET	Fetch data (Read)	GET /users (Get all users)
POST	Create data	POST /users (Add a new user)
PUT	Replace data (Overwrite)	PUT /users/123 (Replace user #123)
DELETE	Remove data	DELETE /users/123 (Delete user #123)
PATCH	Update part of data	PATCH /users/123 (Change name only)





REST API

1. Fetch all books:

Client:

GET /books

Server Response:

```
json  
[  
  {"id": 1, "title": "1984"},  
  {"id": 2, "title": "The Catcher in the Rye"}  
]
```

Roast: "You asked for *all* books. Hope you have storage for the 10GB response."



REST API

Add a new book:

Client:

POST /books

```
{  
  "title": "Harry Potter"  
}
```

Roast: REST doesn't know how to spell "magic." It just blindly saves whatever you send.

Server Response:

```
json  
{  
  "id": 3,  
  "title": "Harry Potter"  
}
```



Status Codes

Code	Meaning	Roast
200	OK	<i>Congrats, you didn't screw up.</i>
201	Created	<i>Wow, you made something. Clap, clap!</i>
400	Bad Request	<i>Your input is garbage.</i>
401	Unauthorized	<i>No key? No entry.</i>
404	Not Found	<i>You lost something? So did the server.</i>
500	Internal Server Error	<i>You broke it. The server hates you now.</i>



Pub.dev

Pub Dev

Pub. dev is the official package repository for Dart and Flutter, developed and maintained by the Flutter team at Google. It serves as a central hub where developers can discover, share, and use packages in their Dart and Flutter projects.



Pub Dev

<https://pub.dev/>



Dio

Dio

A powerful HTTP networking package for Dart/Flutter, supports Global configuration, Interceptors, FormData, Request cancellation, File uploading/downloading, Timeout, Custom adapters, Transformers, etc.



Dio

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



Fake Store API Command Line Application

Project

Problem Story: The FakeStore API Command-Line Application

You have been hired as a **backend developer** for a new startup called **FakeStore CLI Services**. The company's mission is to create a command-line tool that interacts with the [FakeStore API](#) to enable users to manage their shopping experience directly from the terminal.

Your task is to create a **Dart command-line application** that provides users with the following features:



Project

1. Login:

Users should be able to log in using their username and password to authenticate with the FakeStore API. Upon successful login, they receive a token, which will be displayed.

2. View Product Categories:

Users can fetch and view a list of product categories available in the store.

3. View Products by Category:

Users can choose a specific category and view all products under that category. Each product should display its:

- a. ID
- b. Title
- c. Price



Project

User-Friendly Experience:

The application should use a **menu-based system** to guide users through the options.

Error Handling:

The application must handle errors gracefully. For example:

- Invalid credentials during login.
- Network issues or timeouts.
- Invalid inputs, such as choosing a non-existent category.

Code Requirements:

- Follow **OOP principles** by dividing responsibilities into classes like `ApiClient`, `AuthService`, and `ProductService`.
- Use the **Dio** package for HTTP requests.
- Ensure the application is modular, readable, and maintainable.



Project

User-Friendly Experience:

The application should use a **menu-based system** to guide users through the options.

Error Handling:

The application must handle errors gracefully. For example:

- Invalid credentials during login.
- Network issues or timeouts.
- Invalid inputs, such as choosing a non-existent category.

Code Requirements:

- Follow **OOP principles** by dividing responsibilities into classes like `ApiClient`, `AuthService`, and `ProductService`.
- Use the **Dio** package for HTTP requests.
- Ensure the application is modular, readable, and maintainable.



Project

- Create a new Dart project:

```
dart create -t console dio_fakestore  
cd dio_fakestore
```



Project

- Add the **dio** package to your **pubspec.yaml**:

```
dependencies:  
  dio: ^5.0.0
```



Project

- Then Run

```
dart pub get
```



User login

```
fetch('https://fakestoreapi.com/auth/login',{
  method:'POST',
  body:JSON.stringify({
    username: "mor_2314",
    password: "83r5^_"
  })
})
.then(res=>res.json())
.then(json=>console.log(json))
```

Hide output

```
//output
{
  token: "eyJhbGciOiJIUzI1NiIsInR"
}
```

Products

Get all products

```
fetch('https://fakestoreapi.com/products')  
  .then(res=>res.json())  
  .then(json=>console.log(json))
```

Hide output

```
//output  
[  
  {  
    id:1,  
    title:'...',  
    price:'...',  
    category:'...',  
    description:'...',  
    image:'...'  
  },  
  /*...*/  
  {  
    id:30,  
    title:'...',  
    price:'...',  
    category:'...',  
    description:'...',  
    image:'...'  
  }  
]
```

Get all categories

```
fetch('https://fakestoreapi.com/products/categories')  
  .then(res=>res.json())  
  .then(json=>console.log(json))
```

Hide output

```
//output  
[  
  "electronics",  
  "jewelery",  
  "men's clothing",  
  "women's clothing"  
]
```

```
class Product {  
    final int id;  
    final String title;  
    final double price;  
  
    Product({  
        required this.id,  
        required this.title,  
        required this.price,  
    });  
  
    static Product fromJson(Map<String, dynamic> json) {  
        return Product(  
            id: json['id'],  
            title: json['title'],  
            price: (json['price'] as num).toDouble(),  
        );  
    }  
  
    @override  
    String toString() {  
        return 'ID: $id\nTitle: $title\nPrice: \${price.toStringAsFixed(2)}\n';  
    }  
}
```

```
/// A class to handle API requests using Dio
class ApiClient {

  /// Sends a POST request
  Future<dynamic> post(String endpoint, Map<String, dynamic> data) async {
    try {
      final response = await Dio().post(endpoint, data: data);
      return response.data;
    } on DioException catch (e) {
      throw _handleDioError(e);
    }
  }

  /// Sends a GET request
  Future<dynamic> get(String endpoint) async {
    try {
      final response = await Dio().get(endpoint);
      return response.data;
    } on DioException catch (e) {
      throw _handleDioError(e);
    }
  }

  ApiException _handleDioError(DioException error) {
    if (error.type == DioExceptionType.connectionTimeout) {
      return ApiException('Connection timeout.');
```

```
class ApiException implements Exception {  
    final String message;  
  
    ApiException(this.message);  
  
    @override  
    String toString() => 'ApiException: $message';  
}
```

```
class ProductService {  
    final ApiClient apiClient;  
  
    ProductService({required this.apiClient});  
  
    Future<List<String>> fetchCategories() async {  
        final data = await apiClient.get('https://fakestoreapi.com/products/categories');  
        return List<String>.from(data);  
    }  
  
    Future<List<Product>> fetchProductsByCategory(String category) async {  
        final data = await apiClient.get('https://fakestoreapi.com/products/category/$category');  
        return (data as List).map((item) => Product.fromJson(item)).toList();  
    }  
}
```

```
class ProductService {  
    final ApiClient apiClient;  
  
    ProductService({required this.apiClient});  
  
    Future<List<String>> fetchCategories() async {  
        final data = await apiClient.get('https://fakestoreapi.com/products/categories');  
        return List<String>.from(data);  
    }  
  
    Future<List<Product>> fetchProductsByCategory(String category) async {  
        final data = await apiClient.get('https://fakestoreapi.com/products/category/$category');  
        return (data as List).map((item) => Product.fromJson(item)).toList();  
    }  
}
```

```
class AuthService {  
    final ApiClient apiClient;  
  
    AuthService({required this.apiClient});  
  
    Future<String> login({required String username, required String password}) async {  
        final response = await apiClient.post('https://fakestoreapi.com/auth/login', {  
            'username': username,  
            'password': password,  
        });  
        return response['token'];  
    }  
}
```

```
Welcome to FakeStore CLI!
```

- ```
1. Login
2. View Categories
3. View Products by Category
4. Exit
```

```
Choose an option: 2
```

```
Fetching categories...
```

```
Categories:
```

- ```
1. electronics  
2. jewelery  
3. men's clothing  
4. women's clothing
```

What is Next ?

<https://www.youtube.com/watch?v=mMeQhLGD-og>

<https://docs.flutter.dev/get-started/install>



Thank You