

Flutter & Dart Complete Guide

Prepared by Eng. Yousef Salah Kassem

DAFL 3
API

Lecture 4



Chapter
04

Table Of Content

- Introduction
- Rest Api
- Dio
- Authentication and Authorization
- JWT
- FakeStore API 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.



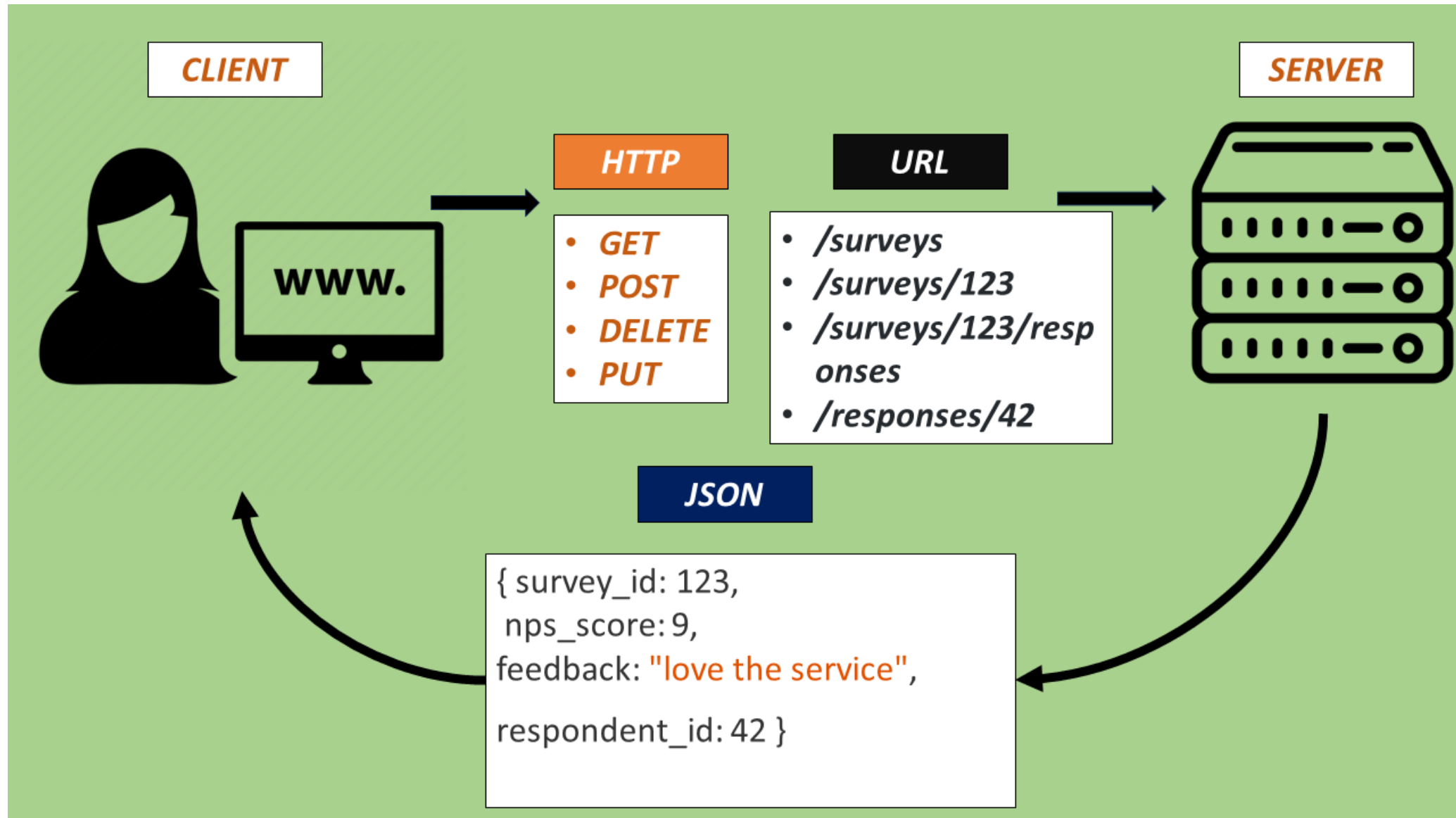
Rest API

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>



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>



Authentication And Authorization

Authentication and authorization are critical components of any application, particularly for those handling sensitive user information. In Flutter, implementing secure authentication and authorization requires a comprehensive understanding of various technologies and best practices to ensure data privacy and prevent unauthorized access. This article will guide you through the steps of setting up secure authentication and authorization in a Flutter application, focusing on best practices to protect your users and their data.



Understanding Authentication vs. Authorization

Before diving into implementation, it's important to differentiate between authentication and authorization



Authentication is the process of verifying a user's identity. It involves collecting credentials, such as usernames and passwords, and determining whether they match stored records. Authentication ensures that users are who they claim to be and is the first step in securing an application.



Authorization is the process of granting or denying access to resources based on the user's identity and permissions.

Authorization occurs after authentication and determines what users can or cannot do within the application



In a Flutter application, both are essential for ensuring that the right users have appropriate access to resources, enhancing both security and the user experience. Secure authentication verifies the identity of users, while secure authorization ensures that users can only access the data and features relevant to their roles.



Authentication Methods in Flutter

Flutter supports several ways to implement authentication, such as Firebase Authentication, or custom backend solutions. Below, we will explore some of the most popular methods and how they can be implemented securely.



Firestore Authentication

[Firestore Authentication](#) is a popular solution for implementing authentication in mobile apps, including Flutter. It provides easy integration for multiple authentication methods, such as email/password, Google Sign-In, Facebook, Apple Sign-In, and biometric authentication.



Firebase handles much of the complexity behind the scenes, such as password hashing and token management, making it a convenient choice for developers who want to implement secure authentication without extensive backend development.



JWT

What is a JWT Token?

A JWT token is a compact, URL-safe token used to securely transmit information between parties as a JSON object.

It's widely used for authentication purposes. A JWT token consists of three parts: Header, Payload, and Signature.



Header: This typically consists of two parts: the type of token (JWT) and the signing algorithm (e.g., HMAC SHA256).

```
{  
  
  "alg": "HS256",  
  
  "typ": "JWT"  
}
```



Payload: This contains the claims, which are statements about an entity (typically, the user) and additional data.

```
{  
  
  "sub": "1234567890",  
  
  "name": "John Doe",  
  
  "iat": 1516239022  
}
```



Signature: This is created by taking the encoded header, the encoded payload, a secret, and the algorithm specified in the header, and signing them.



Fake Store API Flutter Application

Project

Problem Story: The FakeStore API 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 **Flutter 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

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

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



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"  
]
```

What is Next ?



Thank You