Chapter 8:
Working with APIs

Plan

1. Introduction to APIs & REST APIs.

- APIs.
- REST APIs.
- JSON file.
- JsonPlaceHolder.

2. Using an API from a Flutter app.

- Setting up Flutter.
- Fetching data from an API and parse it to Dart data.
- Use model classes.
- Display data using *FutureBuilder* widget.
- Convert Dart data to JSON data and send it to an API.
- Handling errors.

Plan

1. Introduction to APIs & REST APIs.

- **APIs.**
- REST APIs.
- JSON file.
- JsonPlaceHolder.

2. Using an API from a Flutter app.

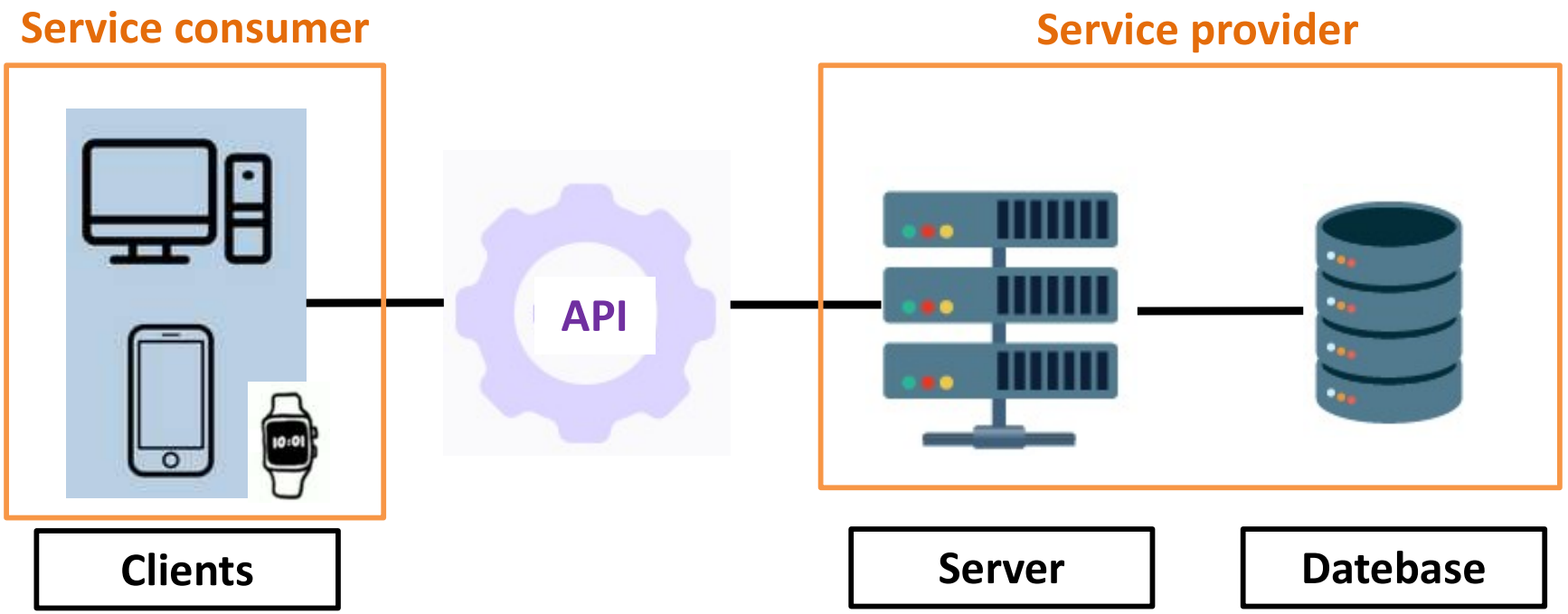
- Setting up Flutter.
- Fetching data from an API and parse it to Dart data.
- Use model classes.
- Display data using *FutureBuilder* widget.
- Convert Dart data to JSON data and send it to an API.
- Handling errors.

Introduction to APIs & REST APIs

- **Definition of API:** an **A**pplication **P**rogramming **I**nterface is an interface that establishes **communication** between a client and a server to obtain data.
- It is a set of rules and protocols that allows different software applications to communicate with each other defining the **methods** and **data formats** that applications can use to request and exchange information.
- An application **does not need to know** the internal workings of another application to use its services.

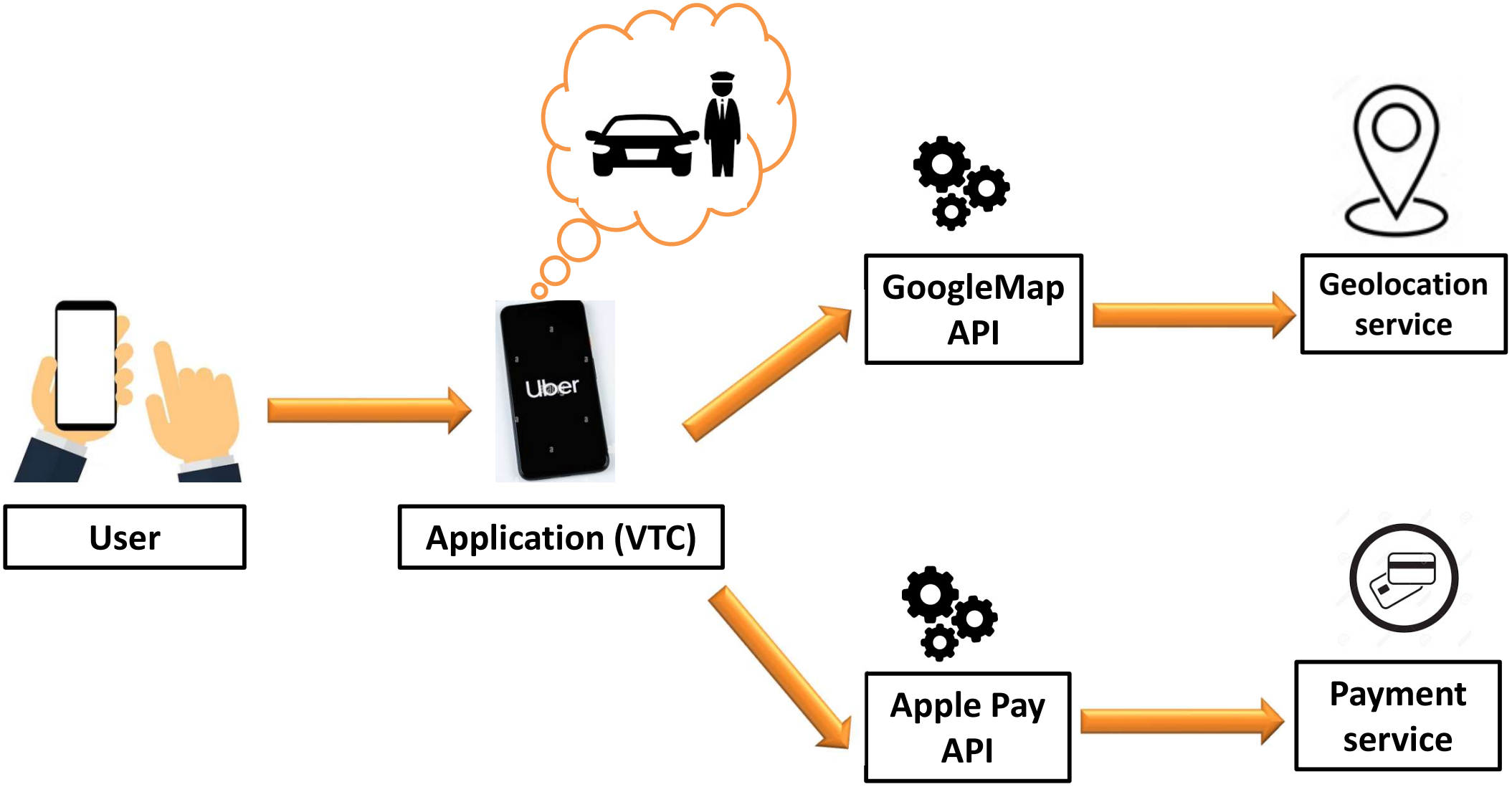
Introduction to APIs & REST APIs

- **Definition of API:**



Introduction to APIs & REST APIs

- **Example of API: VTC Uber app.**



Introduction to APIs & REST APIs

- **API advantages:**
  - Simplifies application development.
  - Extends the application by integrating existing features.
  - Saves time and money by avoiding reinventing what already exists.
- **API disadvantage:**
  - Requires precise knowledge of the documentation for each API.

Introduction to APIs & REST APIs

- **Key concepts of an API :**
  - **Endpoint**: the URL where the API can be accessed (query the server using the URL provided by the API creator).
  - **Request**: the action of querying the API.
  - **Response**: the data returned by the API.
  - **HTTP Methods**: common methods (GET, POST, PUT, DELETE).
  - **Authentication**: key or token required to authenticate requests.

Introduction to APIs & REST APIs

- **API types :**

  - **REST API**: uses HTTP requests to GET, POST, PUT, and DELETE data. It's the most common type of API due to its simplicity and scalability.

  - **SOAP**: uses XML for messaging and includes built-in error handling. It's more rigid and requires more setup than REST.

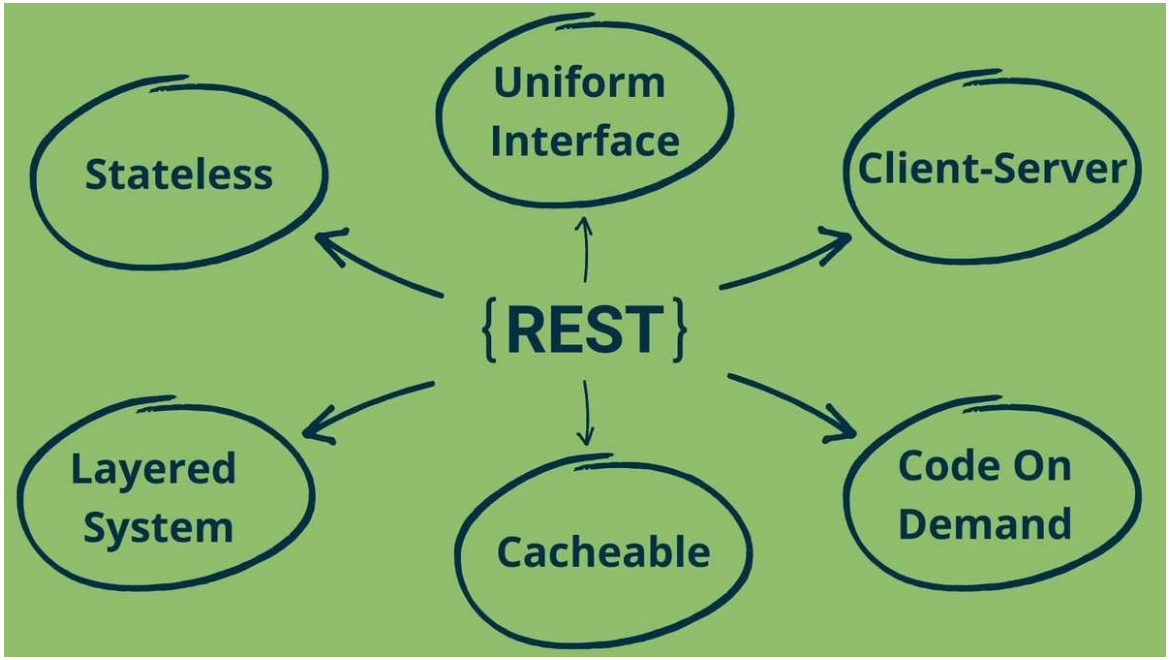
  - **GraphQL**: a query language for APIs that allows clients to request exactly the data they need, making it more efficient in some scenarios.

Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - **REST APIs.**
    - JSON file.
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Introduction to APIs & REST APIs

- **Definition of REST API :** is an API that adheres to the principles and constraints of the **RE**presentational **S**tate **T**ransfer architectural style.
- **Principles of REST :**

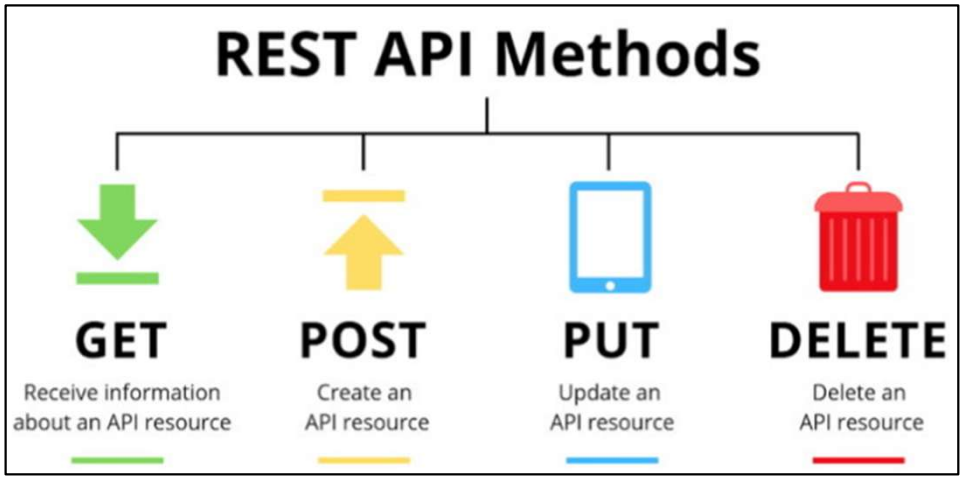


Introduction to APIs & REST APIs

- **Principles of REST :**

1. **Client-Server**: is a principle that describes the separation of responsibilities in an architecture.

   In a REST API the communication between a Client and a Server is based on **HTTP** protocol that uses standard methods such as GET, POST, PUT, DELETE, etc., to interact with resources.



Introduction to APIs & REST APIs

- **Principles of REST :**

2. **Stateless**: each request contains all the information necessary, and the server does not maintain state between requests.

3. **Layered system** : means that the API implementation can reside on one server but source its data from a separate database server. The client does not need to know whether it is directly connected to the end server or an intermediary server

Introduction to APIs & REST APIs

- **Principles of REST :**

4. **Cacheable**: implies that whenever possible, the client caches (stores) API responses and reuses them for future API requests. This improves the overall performance, reducing server load and saving bandwidth.

5. **Code On Demand**: states that if needed, the server can send *executable code* like applets or client-side scripts in JavaScript that can extend the functionality of a client (e.g. Customer Support Chat).

Introduction to APIs & REST APIs

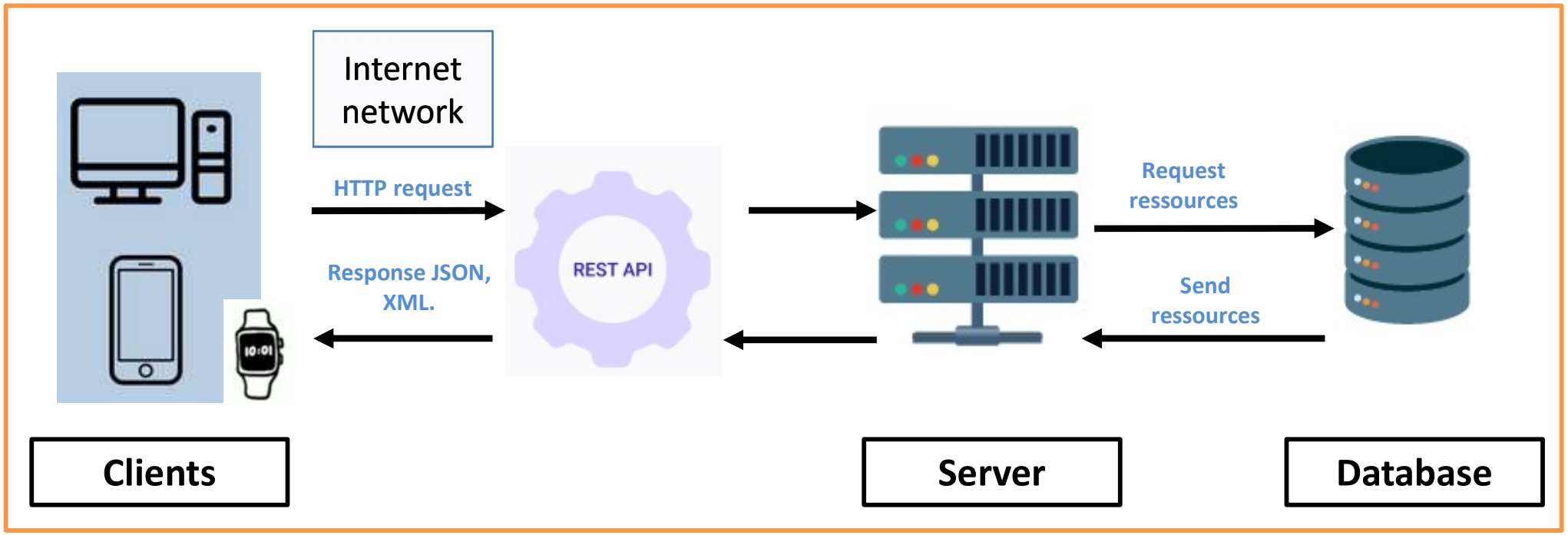
- **Principles of REST :**

6. **Uniformity**: a REST API follows a uniform, standardized structure so that the client can interact with different services in a consistent manner.

   - ✓ **Resource-Based Structure**: resources (e.g., users, products) are identified by URIs (Uniform Resource Identifiers).
   - ✓ **Data Representation**: data is sent as representations of resources, usually in **JSON** or **XML**.

Introduction to APIs & REST APIs

- **REST API model :**



Note: the response to an API request can have two states.

- ✓ Success with code **200**.
- × Failure with codes **403, 404, 500, 504, etc**.

Introduction to APIs & REST APIs

- **API vs REST API :**

Criterion	API	REST API
Protocol	Can use various protocols like HTTP, SOAP, etc.	Specifically uses **HTTP** and standard methods (GET, POST, PUT, DELETE).
Architecture	Can be of different types (RPC, SOAP, etc.)	Follows **REST** principles, which are focused on managing resources via HTTP.
Stateless	Not necessarily stateless, depends on the type of API	**Stateless**: each request must be independent.
Methods	Varies depending on the API (e.g., function calls via RPC or requests via SOAP)	Uses standardized HTTP methods (GET, POST, PUT, DELETE, etc.) to interact with resources.
Data Representation	Can be in various formats (JSON, XML, CSV, etc.)	Resource representations are usually in **JSON** or **XML**.

Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - **JSON file.**
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Introduction to APIs & REST APIs

- **JSON : JavaScript Object Notation**, is a standardized format for representing structured data.

- It is widely used for **exchanging data** between systems (across different platforms and technologies).

- It is **simple**, **compact** and **language-independent**.

Introduction to APIs & REST APIs

- **JSON structure :** JSON represents data as a collection of **key-value pairs** and **arrays** where,

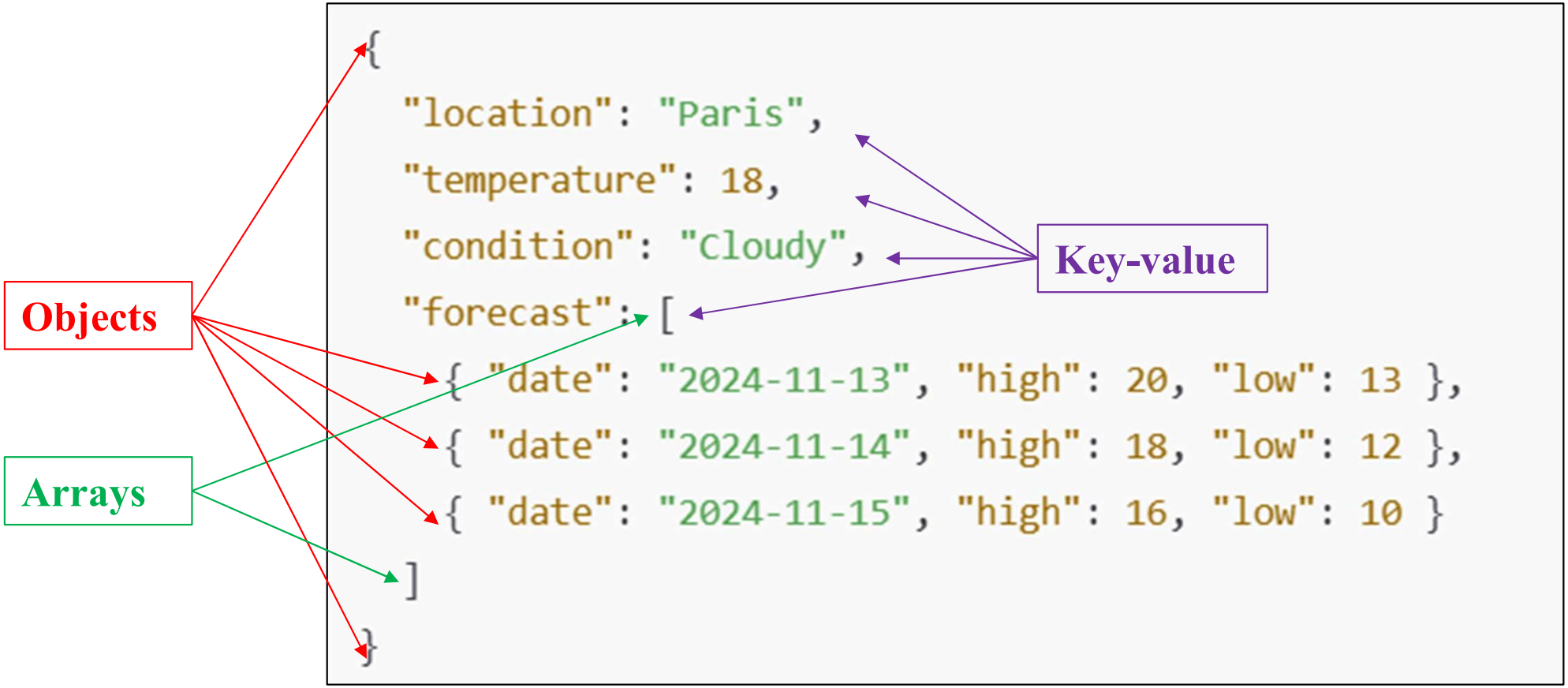
  - ✓ **Keys** are strings, and **values** can be strings, numbers, booleans, arrays, or other JSON objects.

  - ✓ **Objects** are enclosed in curly braces {} and consist of key-value pairs.

  - ✓ **Arrays** are enclosed in square brackets [] and contain ordered lists of objects.

Introduction to APIs & REST APIs

- **JSON structure :** example.



Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - JSON file.
    - **JsonPlaceHolder.**
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Introduction to APIs & REST APIs

- **JSONPlaceholder :**

Is a **free online service** that provides a **fake** REST API for testing applications without needing to set up a real backend (database or server).

Introduction to APIs & REST APIs

- **JSONPlaceholder :** offers common resources with sample data.
  - /posts (blog posts or articles),
  - /comments: (associated with posts),
  - /albums: (photo albums),
  - /photos: (within albums),
  - /todos: (to-do lists) and
  - /users.
- Each resource is accessible via predefined **URLs** and supports typical **HTTP** operations (GET, PUT, POST, DELETE).

Introduction to APIs & REST APIs

- **JSONPlaceholder :**

  For instance, "jsonplaceholder.typicode.com/users" is the endpoint used to access user data. It provides a list of sample users, including fields like name, username, email, etc.


```
[
{       "id": 3,
        "name": "Clementine Bauch",
        "username": "Samantha",
        "email": "Nathan@yesenia.net",
        "address": { "street": "Douglas Extension", "suite": "Suite 847", "city":
          "McKenziehaven","zipcode": "59590-4157",  "geo": { "lat": "-68.6102", "lng": "-
           47.0653“}  },
        "phone": "1-463-123-4447", ……
},
……]
```

Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - JSON file.
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - JSON file.
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Using an API from a Flutter app

- **Setting up Flutter :**

  - Many packages can be used to integrate APIs in Flutter. The most widely used are:

    - ✓ http.
    - ✓ dio.
    - ✓ chopper, … and many more.

  - **http** is the basic one, while the others provide additional functionalities.

Using an API from a Flutter app

- **Setting up Flutter :** add the **HTTP** package.

  - To install it, run the following command from a terminal.

```
$ flutter pub add http
```

  - Or, add dependency directly to *pubspec.yaml*

```
dependencies:
  http:^1.2.2
```

  - Then, import the package from the project:

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

Plan

1. **Introduction to APIs & REST APIs.**
   - APIs.
   - REST APIs.
   - JSON file.
   - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
   - Setting up Flutter.
   - Fetching data from an API and parse it to Dart data.
   - Use model classes.
   - Display data using *FutureBuilder* widget.
   - Convert Dart data to JSON data and send it to an API.
   - Handling errors.

Using an API from a Flutter app

- **Fetching data from an API :**

```
Future fetchData() async{
// Specify the API endpoint (the URL used to access the API)
   string URL= "https://jsonplaceholder.typicode.com/photos";
// Parse the URL into an Uri object.
   var URI = Uri.parse (URL);
// Use the URI object in HTTP Get request to load data from the API.
      final var response = await get (URI);
}
```

It takes some time to load the data

Headers: {'Authorization': 'your_token',},

Using an API from a Flutter app

- **Fetching data from an API :**

```
final var response = await get (URI);
```

Is of type "Instance of Response class"

➢ To access the **contents** of the response, we need to retrieve the **body** field:

```
var data = response.body;
```

Is of type "String"

Using an API from a Flutter app

- **Fetching data from an API :**

➢ The value of **data** is the following string:


```
[
    {
        "albumId": 1,
        "id": 1,
        "title": "accusamus beatae ad facilis cum similique qui sunt",
        "url": "https://via.placeholder.com/600/92c952",
        "thumbnailUrl": "https://via.placeholder.com/150/92c952"
    },
    {
        "albumId": 1,
        "id": 2,
        "title": "reprehenderit est deserunt velit ipsam",
        "url": "https://via.placeholder.com/600/771796",
        "thumbnailUrl": "https://via.placeholder.com/150/771796"
    },
    {
        "albumId": 1,
        "id": 3,
        "title": "officia porro iure quia iusto qui ipsa ut modi",
```

Using an API from a Flutter app

- **Parse Json data to a Map:**

```
// Parse Json string to Dart Map
Var jsonData = JsonDecode (data);
```

```
// Get the first element of the Map
 jsonData[0] ;
```


```
[
    {
        "albumId": 1,
        "id": 1,
        "title": "accusamus beatae ad facilis cum similique qui sunt",
        "url": "https://via.placeholder.com/600/92c952",
        "thumbnailUrl": "https://via.placeholder.com/150/92c952"
    },
    {
        "albumId": 1,
        "id": 2,
        "title": "reprehenderit est deserunt velit ipsam",
        "url": "https://via.placeholder.com/600/771796",
        "thumbnailUrl": "https://via.placeholder.com/150/771796"
    },
    {
        "albumId": 1,
        "id": 3,
        "title": "officia porro iure quia iusto qui ipsa ut modi",
```

Using an API from a Flutter app

- **Parse Json data to a Map:**

```
// Parse Json string to Dart Map
Var jsonData = JsonDecode (data);

// Get the first element of the M
 jsonData[0] ;
```

"dart:convert" Dart library

JsonEncode :
Dart object (Map, List…) → JSON string.

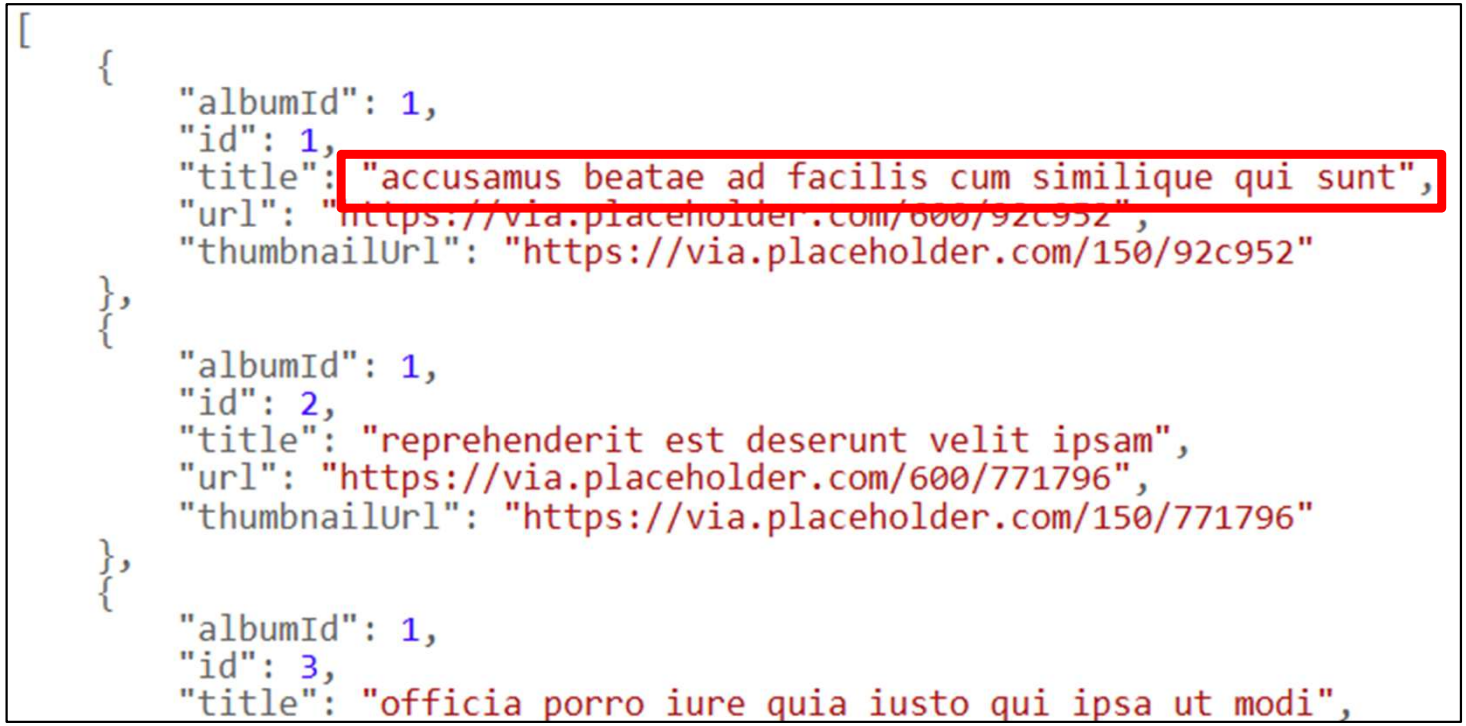
JsonDecode :
JSON string → Dart object (Map, List…)


```
[
    {
        "albumId": 1,
        "id": 1,
        "title": "accusamus beatae ad facilis cum similique qui sunt",
        "url": "https://via.placeholder.com/600/92c952",
        "thumbnailUrl": "https://via.placeholder.com/150/92c952"
    },
    {
        "albumId": 1,
        "id": 2,
        "title": "reprehenderit est deserunt velit ipsam",
        "url": "https://via.placeholder.com/600/771796",
        "thumbnailUrl": "https://via.placeholder.com/150/771796"
    },
    {
        "albumId": 1,
        "id": 3,
        "title": "officia porro iure quia iusto qui ipsa ut modi",
```

Using an API from a Flutter app

- **Parse Json data to a Map:**

```
// Retrieve the value of the 'title' key from the first element of the Map
                    jsonData[0]['title'] ;
```



```
[
    {
        "albumId": 1,
        "id": 1,
        "title": "accusamus beatae ad facilis cum similique qui sunt",
        "url": "https://via.placeholder.com/600/92c952",
        "thumbnailUrl": "https://via.placeholder.com/150/92c952"
    },
    {
        "albumId": 1,
        "id": 2,
        "title": "reprehenderit est deserunt velit ipsam",
        "url": "https://via.placeholder.com/600/771796",
        "thumbnailUrl": "https://via.placeholder.com/150/771796"
    },
    {
        "albumId": 1,
        "id": 3,
        "title": "officia porro iure quia iusto qui ipsa ut modi",
```

Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - JSON file.
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Using an API from a Flutter app

- **Use model classes :** create the Dart class.

```
class Photo {

// Attributes

final int id;
final int albumID;
final String title;
final String url;

// Constructor

Photo ({ required this.id, required this.albumID, required
this.title, required this.url });
```

Using an API from a Flutter app

- **Use model classes :** define **fromJson** method.

```
class Photo {
.....

// A method to convert JSON data to Photo objects.

fromJson(Map<String, dynamic> inputJson) {
return Photo( id: inputJson['id'],
                albumID:inputJson['albumId']),
                title: inputJson['title'],
                url: inputJson['url']);
        }
}
```

Using an API from a Flutter app

- **Use model classes :** define **toJson** method.

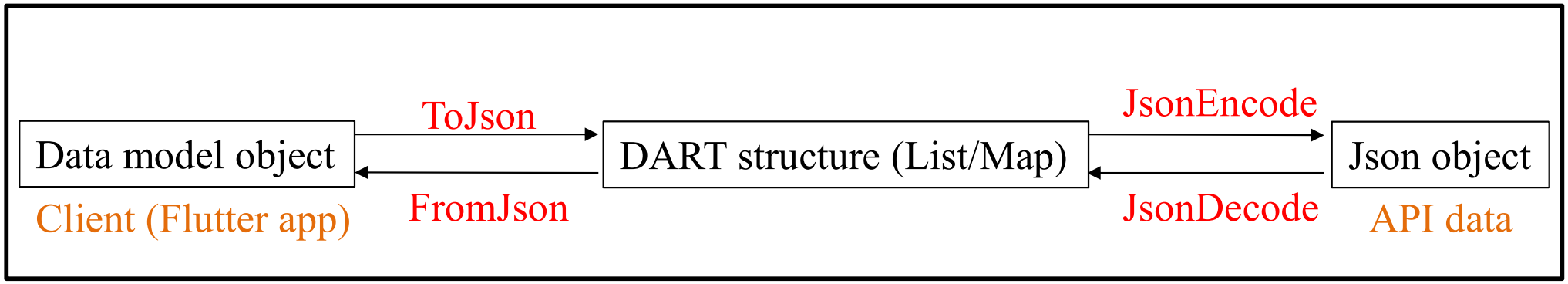
```
class Photo {
.....

// A method to convert Photo objects to JSON data.

Map<String, dynamic> toJson( ) {
return { 'id': id ,
          'albumId' : albumID,
          'title' : title,
          'url' : url);
         }
    }
}
```

Using an API from a Flutter app

- **Summary of conversion flow:**



Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - JSON file.
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Using an API from a Flutter app

- **FutureBuilder :** is a widget used for handling **asynchronous** data in the **UI**. It listens to a **Future** and rebuilds the UI based on the state of that future.
- A **Future** can have several states:
  - ✓ **waiting**: the Future is still waiting for completion.
  - ✓ **done**: the Future is completed, either successfully with data or with an error.
  - ✓ **No data**: the Future is completed, but it didn't return any data.

Using an API from a Flutter app

- **FutureBuilder :** basic structure.

```
FutureBuilder  < T > (
  future: myFuture, // The Future you want to wait for
  builder: (context, snapshot) {
    // Build the UI based on the Future's state
    if (snapshot.connectionState == ConnectionState.waiting) {
                        return CircularProgressIndicator(); } // Loading state
    if (snapshot.connectionState == ConnectionState.done) {
                        if (snapshot.hasError) {
                        return Text('Error: ${snapshot.error}'); } // Error state
                        else if (snapshot.hasData) {
                        return Text('Data: ${snapshot.data}'); } // Success state
    else { return Text('No data available'); } // Default state
}, );
```

Using an API from a Flutter app

- **FutureBuilder :** example of **myFuture** function.

```
Future  <List<dynamic>> fetchData() async{
   String URL= "https://jsonplaceholder.typicode.com/photos";
   var URI = Uri.parse (URL); .
   var response = await get (URI);
   List<dynamic>  responseBody = jsonDecode(response.body);
return responseBody
}
```

Plan

1. **Introduction to APIs & REST APIs.**
    - APIs.
    - REST APIs.
    - JSON file.
    - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
    - Setting up Flutter.
    - Fetching data from an API and parse it to Dart data.
    - Use model classes.
    - Display data using *FutureBuilder* widget.
    - Convert Dart data to JSON data and send it to an API.
    - Handling errors.

Using an API from a Flutter app

- **Convert Dart data to JSON data and send it to an API :**

```
// Convert Photo object to Json Map
    String convertPhotoToJson(Photo photo) {

        // Step 1: convert model object (photo) to Dart Map
        Map<String, dynamic> photoDart = photo.toJson();

         // Step 2: Encode Dart Map to Json string
          String photoJson = jsonEncode(photoDart);
     return photoJson;
}
```

Using an API from a Flutter app

- **Convert Dart data to JSON data and send it to an API :**

```
// Sends a POST request to add new data to the API
Future <void> postData(Photo photo) async{
await post ( URI, Headers: {'Content-Type':
'application/json',  'Authorization': 'your_token',},
            body: convertPhotoToJson(photo)  ); }
// Create a photo object
Photo  myPhoto = Photo ( id : 1 , albumId : 10,
                    title : "photo1",
                    url : "https://example.com/photo.jpg" ) ;
// Call the postData method to send the object myPhoto
postData(myPhoto);
```

Plan

1. **Introduction to APIs & REST APIs.**
   - APIs.
   - REST APIs.
   - JSON file.
   - JsonPlaceHolder.
2. **Using an API from a Flutter app.**
   - Setting up Flutter.
   - Fetching data from an API and parse it to Dart data.
   - Use model classes.
   - Display data using *FutureBuilder* widget.
   - Convert Dart data to JSON data and send it to an API.
   - Handling errors.

Using an API from a Flutter app

- **Handling errors :**

```
Future fetchData() async{
……
try{
 final var response = await get (URI);
// Handle success status
if (response.statusCode ==200) {
          var data = jsonDecode (response.body)
}// Handle error status
else {
        throw Exception "Failed to load data";
       }
} catch (e) {….}
```

✓Success code : **200, 201**.
× Failure codes : **403, 404, 500, 504, etc**.

Using an API from a Flutter app

- **Handling errors :**

```
Future <void> postData(Photo myPhoto) async{
try{
final var response = await post ( URI,
                Headers: {'Content-Type': 'application/json'},
                body: convertPhotoToJson(myPhoto) );
// Handle success status
if (response.statusCode ==200 || response.statusCode ==201){
          print("Data added successfully");
         }
// Handle error status
else { throw Exception("Failed to add data"); }
} catch (e) { … } }
```

Conclusion

- **OpenWeatherMap:** provides real-time weather data, hourly forecasts, and multi-day forecasts.
- **REST Countries:** provides information about countries, such as names, capitals, regions, and populations.
- **NewsAPI:** allows you to fetch news articles from various news sources.
- **CoinGecko:** provides real-time cryptocurrency prices, market data, and more financial information.
- **NASA API:** allows access to various NASA data, including space images, planet information, and space missions.