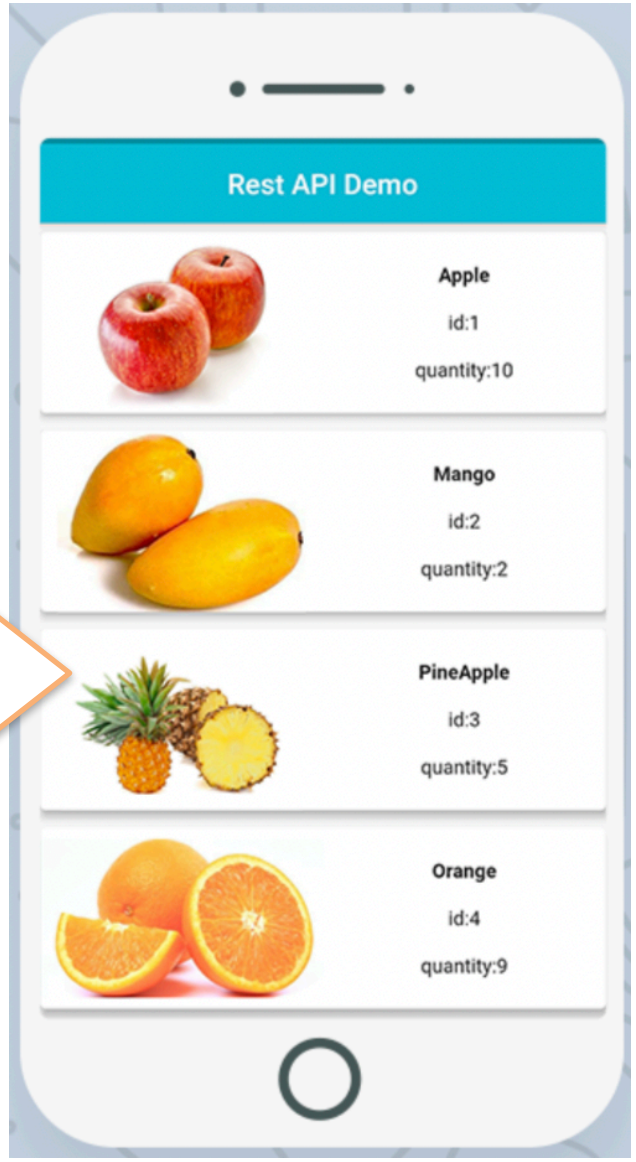
Resource Software Solution

Working with REST APIs

Introduction

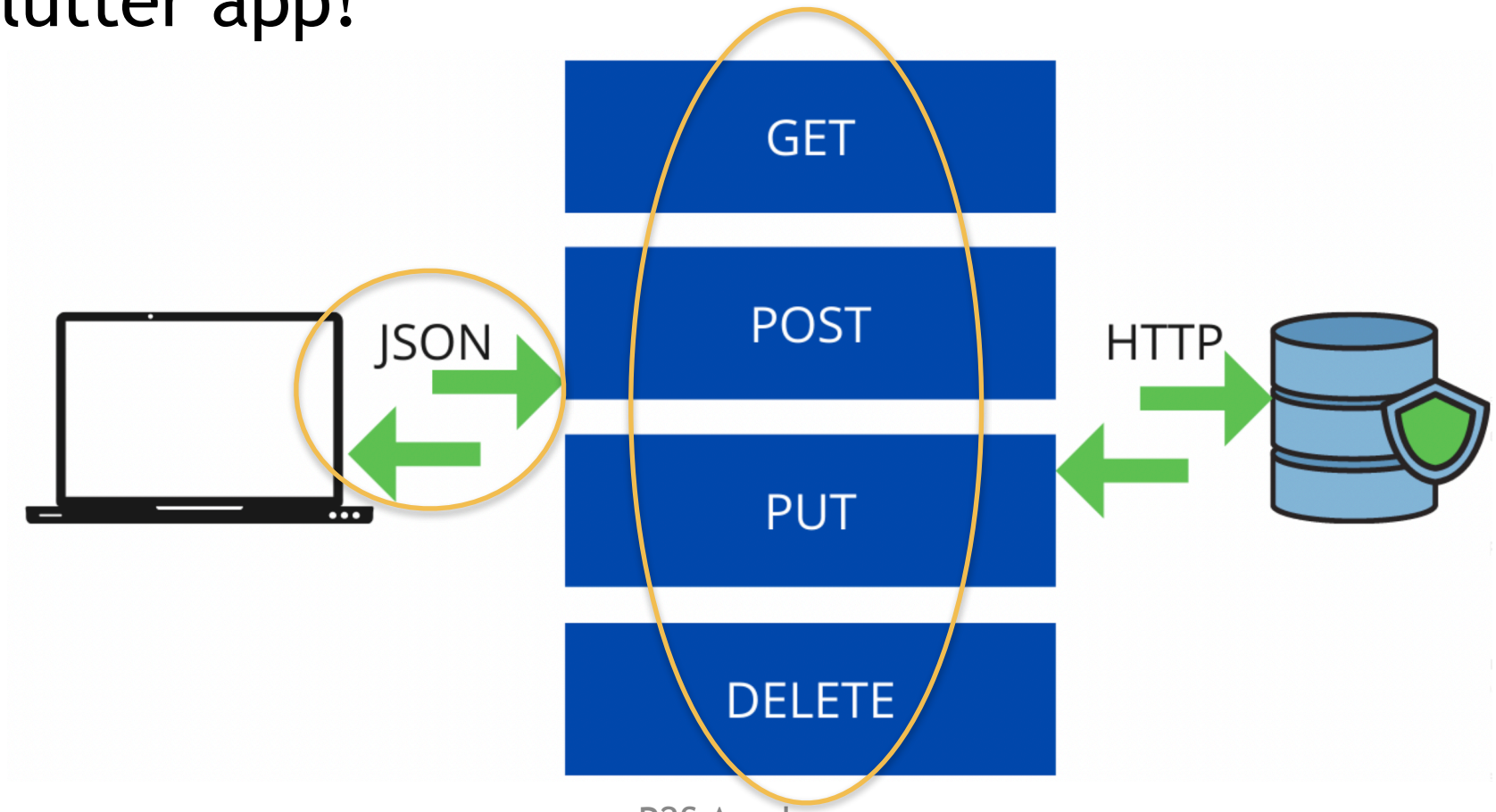
- Data, data and data. Everywhere we see in today's world is just data. So how can we get data in our app?
- We have a lot of ways to show data in our app:
  1. Static Data
  2. From a file
  3. From a database
  4. From public APIs


```
{
  "id": 1,
  "title": "Apple",
  "imageUrl": "https://produits.bienmanger.com/36433-0w470h470_Organic_Apples_Fuj.jpg",
  "quantity": 10
},
 {
  "id": 2,
  "title": "Mango",
  "imageUrl": "https://images-na.ssl-images-amazon.com/images/I/31KSJIUe16L._SX450_.jpg",
  "quantity": 2
},
 {
  "id": 3,
  "title": "PineApple",
  "imageUrl": "https://www.healthifyme.com/blog/wp-content/uploads/2015/12/shutterstock_1670265607-1.jpg",
  "quantity": 5
},
{
  "id": 4,
  "title": "Orange",
  "imageUrl": "https://i.ndtvimg.com/i/2016-08/orange_625x350_51471855916.jpg",
  "quantity": 9
}
```

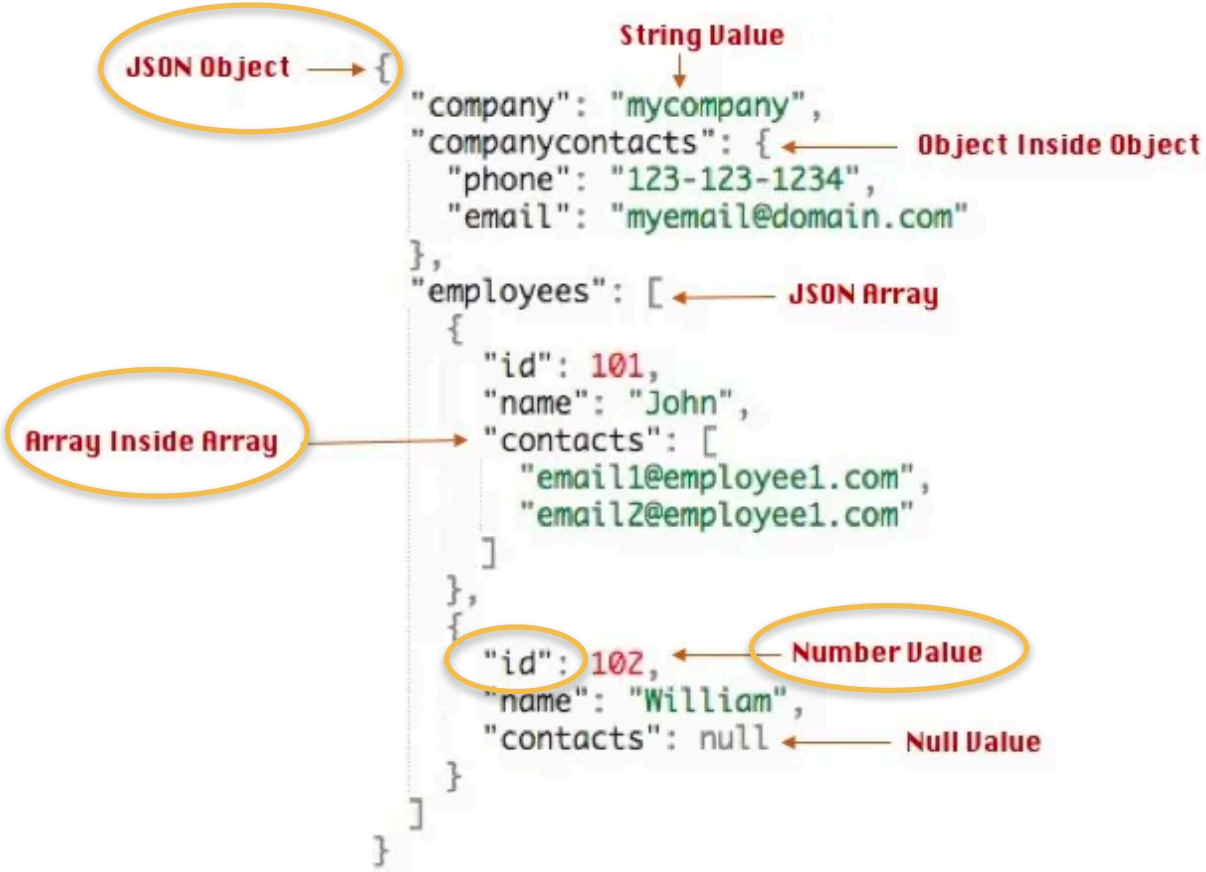
Working with APIs

- The most popularly used form is from **Database** or Public **APIs**.
- Let's see how we can integrate, fetch data from a API and use it in your flutter app!



JSON Overview

- JSON represents data in two ways:
  1. **Object**: a collection of name-value (or key-value) pairs. An object is defined within left ({) and right (}) braces. Each name-value pair begins with the name, followed by a colon, followed by the value. Name-value pairs are comma separated.
  2. **Array**: an ordered collection of values. An array is defined within left ([) and right (]) brackets. Items in the array are comma separated.



JSON Overview

- This is an example of a JSON file that captures multi-day **forecast data.**


```
{
  "longitude": 47.60,
  "latitude": 122.33,
  "forecasts": [
    {
      "date": "2015-09-01",
      "description": "sunny",
      "maxTemp": 22,
      "minTemp": 20,
      "windSpeed": 12,
      "danger": false
    },
    {
      "date": "2015-09-02",
      "description": "overcast",
      "maxTemp": 21,
      "minTemp": 17,
      "windSpeed": 15,
      "danger": false
    },
    {
      "date": "2015-09-03",
      "description": "raining",
      "maxTemp": 20,
      "minTemp": 18,
      "windSpeed": 13,
      "danger": false
    }
  ]
}
```

JSON Overview

- This is an example of a JSON file that data for **trainees** resource.

```
[
  {
    "name": "lam",
    "email": "lam@gmail.com",
    "phone": "0974111444",
    "gender": "Female",
    "id": "2"
  },
  {
    "name": "khanh",
    "email": "khanh@gmail.com",
    "phone": "0123456789",
    "gender": "1",
    "id": "5"
  },
  {
    "name": "Le Hong Dang",
    "email": "giasutinhoc.vn@gmail.com",
    "phone": "123",
    "gender": "Male",
    "id": "17"
  }
]
```

JSON Overview

- Link: https://jsonplaceholder.typicode.com/albums
- Result:


```
[
  {
    "userId": 1,
    "id": 1,
    "title": "quidem molestiae enim"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "sunt qui excepturi placeat culpa"
  },
  {
    "userId": 1,
    "id": 3,
    "title": "omnis laborum odio"
  },
```

JSON Overview

- Link: https://jsonplaceholder.typicode.com/photos
- Result:


```
[
  {
    "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",
    "url": "https://via.placeholder.com/600/24f355",
    "thumbnailUrl": "https://via.placeholder.com/150/24f355"
  },
```

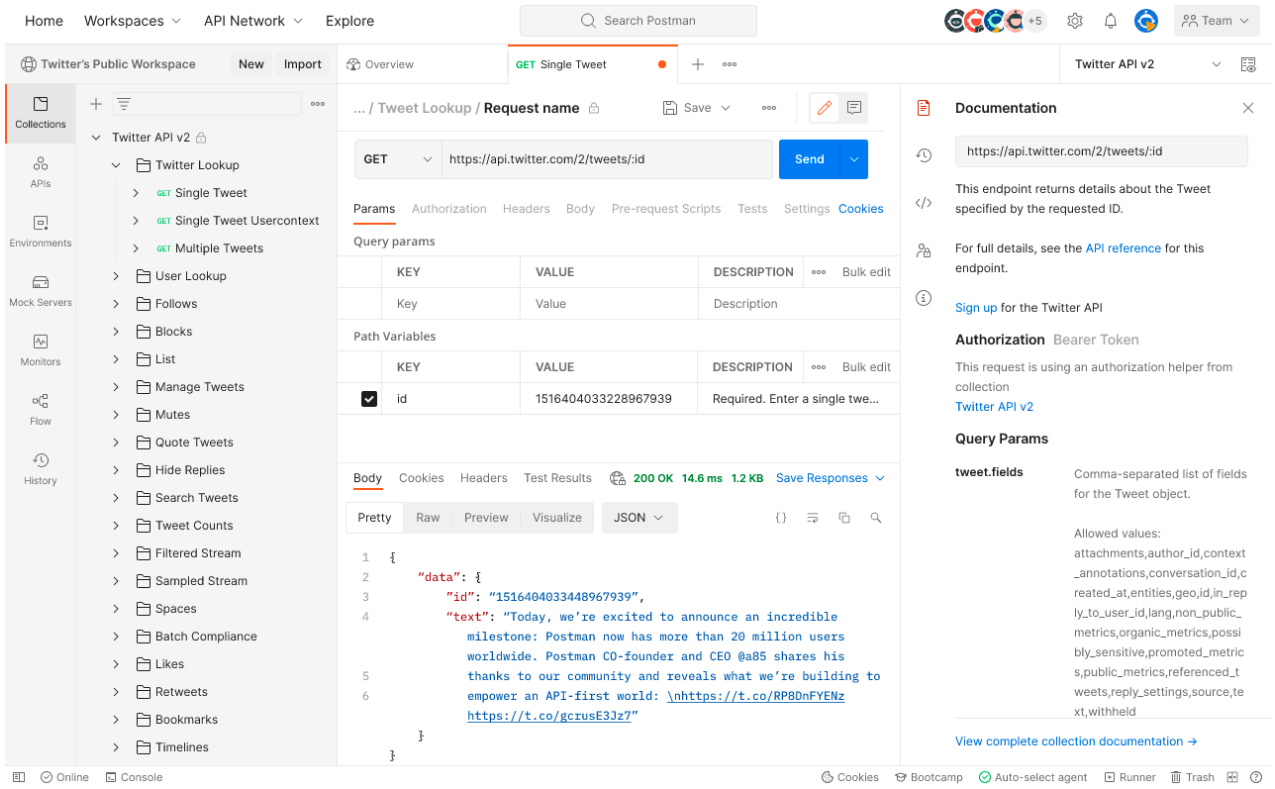
REST API Overview

- API (Application Programming Interface)
- For example, the **Task REST API** below

Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}

API Testing using Postman

- The Postman app: https://www.postman.com/downloads/
- How to use for API Testing
  1. Working with **GET** requests
  2. Working with **POST** requests
  3. Working with **UPDATE** requests
  4. Working with **DELETE** requests



Working with GET Requests

- Get requests are used to retrieve information from the given URL. There will be no changes done to the endpoint.
- We will use the following URL **https://60db1a79801dcb0017290e61.mockapi.io/todos**

GET | https://60db1a79801dcb0017290e61.mockapi.io/todos | Send

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings | Cookies

Query Params

KEY	VALUE	DESCRIPTION	ooo	Bulk Edit
Key	Value	Description		

Body | Cookies | Headers (12) | Test Results | Status: 200 OK | Time: 838 ms | Size: 476 B | Save Response

Pretty | Raw | Preview | Visualize | JSON


```
[
    {
        "title": "First todo",
        "description": "First description",
        "id": "1"
    }
]
```

Working with POST Requests

- Post requests are different from Get request as there is data manipulation with the user adding data to the endpoint.
- We will use the following URL **https://60db1a79801dcb0017290e61.mockapi.io/todos**

POST | https://60db1a79801dcb0017290e61.mockapi.io/todos | Send

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify

```
1 {
2     "title":"First todo",
3     "description": "First description"
4 }
```

Body | Cookies | Headers (11) | Test Results | Status: 201 Created | Time: 1078 ms | Size: 459 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "title": "First todo",
3     "description": "First description",
4     "id": "1"
5 }
```

Working with PUT Requests

- PUT is used to send data to a server to update a resource.
- We will use the following URL **https://60db1a79801dcb0017290e61.mockapi.io/todos/1**

PUT

https://60db1a79801dcb0017290e61.mockapi.io/todos/1

Send

Params | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify

```
1 {
2     "title":"First todo (update)",
3     "description": "First description (update)"
4 }
```

Body | Cookies | Headers (11) | Test Results | Status: 200 OK | Time: 289 ms | Size: 472 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "title": "First todo (update)",
3     "description": "First description (update)",
4     "id": "1"
5 }
```

Working with DELETE Requests

- The DELETE method deletes the specified resource.
- We will use the following URL **https://60db1a79801dcb0017290e61.mockapi.io/todos/1**

DELETE | https://60db1a79801dcb0017290e61.mockapi.io/todos/1 | Send

Params | Authorization | Headers (6) | Body | Pre-request Script | Tests | Settings | Cookies

Query Params

KEY	VALUE	DESCRIPTION	ooo	Bulk Edit
Key	Value	Description		

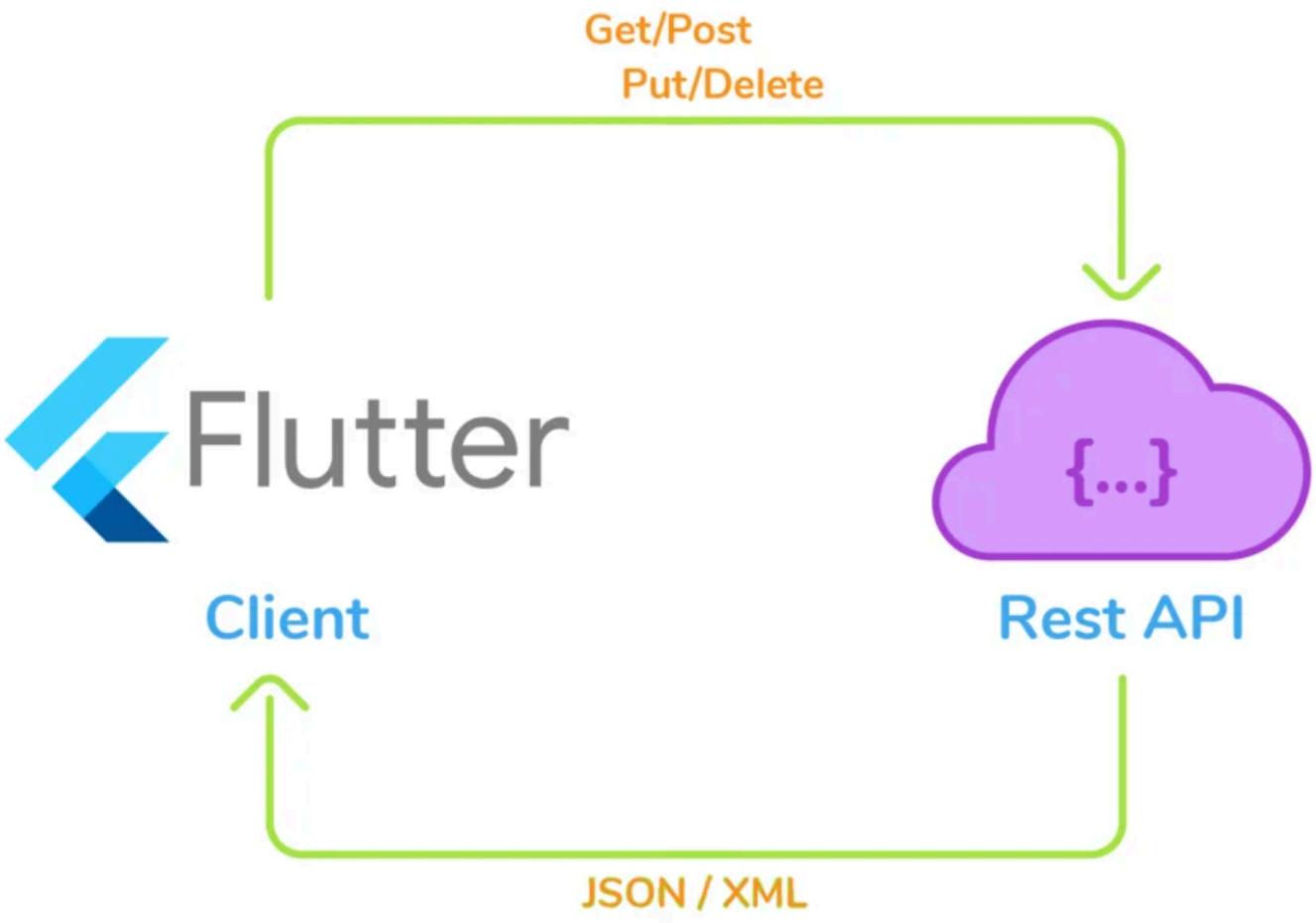
Body | Cookies | Headers (11) | Test Results | Status: 200 OK | Time: 1885 ms | Size: 472 B | Save Response

Pretty | Raw | Preview | Visualize | JSON

```
1 {
2     "title": "First todo (update)",
3     "description": "First description (update)",
4     "id": "1"
5 }
```

Implementing Rest API in Flutter

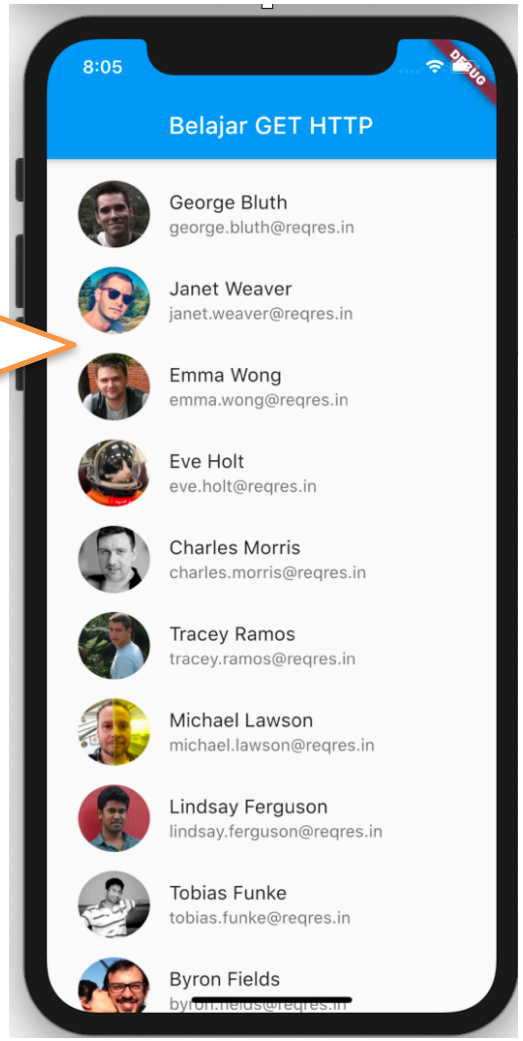
1. **Fetch** data from the internet
2. **Send** data to the internet
3. **Update** data over the internet
4. **Delete** data on the internet



Fetch data from the internet

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 the **data**
5. Display the **data**


```
"data": [
    {
        "id": 1,
        "email": "george.bluth@reqres.in",
        "first_name": "George",
        "last_name": "Bluth",
        "avatar": "https://reqres.in/img/faces/1-image.jpg"
    },
    {
        "id": 2,
        "email": "janet.weaver@reqres.in",
        "first_name": "Janet",
        "last_name": "Weaver",
        "avatar": "https://reqres.in/img/faces/2-image.jpg"
    },
    {
        "id": 3,
        "email": "emma.wong@reqres.in",
        "first_name": "Emma",
        "last_name": "Wong",
        "avatar": "https://reqres.in/img/faces/3-image.jpg"
    },
    {
        "id": 4,
        "email": "eve.holt@reqres.in",
        "first_name": "Eve",
        "last_name": "Holt",
        "avatar": "https://reqres.in/img/faces/4-image.jpg"
    },
```

Fetch data from the internet

- **Add the http package**

1. 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>
```

2. Import the http package.

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

3. In your **AndroidManifest.xml** file, add the Internet permission.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Fetch data from the internet

- **Make a network request using the http package**
  1. This recipe covers how to fetch a sample album from the JSONPlaceholder using the http.get() method.

```
Future<http.Response> fetchAlbum() async {
  return await http.get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));
}
```

  2. The http.get() method returns a **Future** that contains a **Response**.
     - Future is a class for working with async operations. A Future object represents a potential **value** or **error** in the future.
     - The **http.Response** class contains the data received from a successful http call.

Fetch data from the internet

- Convert the response into a object

```
Future<Album> fetchAlbum() async {
  final response = await http
      .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));

  if (response.statusCode == 200) {
    return Album.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to load album');
  }
}
```

```
class Album {
  final int userId;
  final int id;
  final String title;

  const Album({
    required this.userId,
    required this.id,
    required this.title,
  });

  factory Album.fromJson(Map<String, dynamic> json) {
    return Album(
      userId: json['userId'],
      id: json['id'],
      title: json['title'],
    );
  }
}
```

Fetch data from the internet

- **Fetch the data**

```
class _MyAppState extends State<MyApp> {
  late Future<Album> futureAlbum;

  @override
  void initState() {
    super.initState();
    futureAlbum = fetchAlbum();
  }
  // ...
}
```

```
Future<Album> fetchAlbum() async {
  final response = await http
      .get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));

  if (response.statusCode == 200) {
    // If the server did return a 200 OK response,
    // then parse the JSON.
    return Album.fromJSON(jsonDecode(response.body));
  } else {
    // If the server did not return a 200 OK response,
    // then throw an exception.
    throw Exception('Failed to load album');
  }
}
```

- **Display the data**

```
FutureBuilder<Album>(
  future: futureAlbum,
  builder: (context, snapshot) {
    if (snapshot.hasData) {
      return Text(snapshot.data!.title);
    } else if (snapshot.hasError) {
      return Text('${snapshot.error}');
    }

    // By default, show a loading spinner.
    return const CircularProgressIndicator();
  },
)
```

Fetch data from the internet

- **FutureBuilder** will help you to execute some **asynchronous** function and based on that **function's result** your **UI will update**.
- FutureBuilder - Parameters
  - **future**: assigns the Future **value** that will be delivered asynchronously, such as a list of movies from the Internet
  - **build**: returns a **Widget**
    - **hasData, hasError, loading**
  - **snapshot:** will hold the data once it is delivered

```
FutureBuilder<Album>(
  future: futureAlbum,
  builder: (context, snapshot) {
    if (snapshot.hasData) {
      return Text(snapshot.data!.title);
    } else if (snapshot.hasError) {
      return Text('${snapshot.error}');
    }

    // By default, show a loading spinner.
    return const CircularProgressIndicator();
  },
)
```

Send data to the internet

- This recipe uses the following steps:
  1. Add the http package.
  2. Send data to a server using the http package.
  3. Get a title from user input.

Send data to the internet

- Send data to a server using the http package

```
Future<http.Response> createAlbum(String title) {
  return http.post(
    Uri.parse('https://jsonplaceholder.typicode.com/albums'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{
      'title': title,
    }),
  );
}
```

Send data to the internet

- Get a title from user input

```
Column(
  children: <Widget>[
    TextField(
      controller: _controller,
      decoration: const InputDecoration(hintText: 'Enter Title'),
    ),
    ElevatedButton(
      onPressed: () {
          var result = createAlbum(_controller.text);
      },
      child: const Text('Create Data'),
    ),
  ],
)
```

Update data over the internet

- This recipe uses the following steps:
  1. Add the http package.
  2. Update data over the internet using the http package.
  3. Get a title from user input.
  4. Update the existing title from user input.

Update data over the internet

- Updating data over the internet using the http package

```
Future<http.Response> updateAlbum(String title) {
  return http.put(
    Uri.parse('https://jsonplaceholder.typicode.com/albums/1'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, String>{
      'title': title,
    }),
  );
}
```

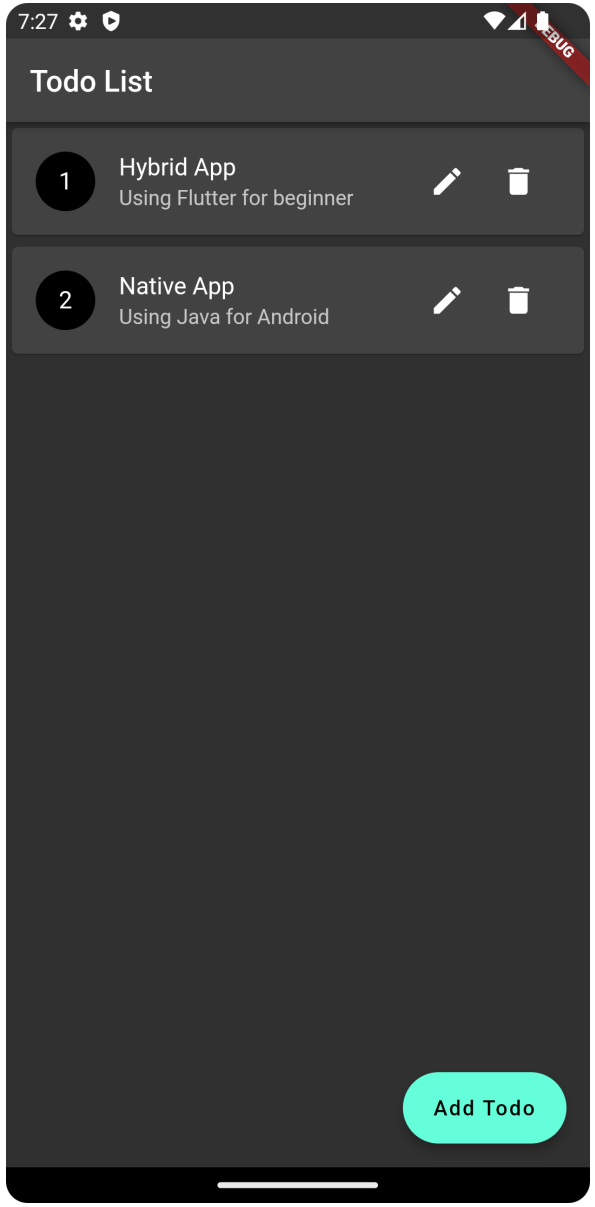
Update data over the internet

- Update the existing title from user input

```
Column(
  children: <Widget>[
    TextField(
      controller: _controller,
      decoration: const InputDecoration(hintText: 'Enter Title'),
    ),
    ElevatedButton(
      onPressed: () {
          var result = updateAlbum(_controller.text);
      },
      child: const Text('Update Data'),
    ),
  ],
)
```

Delete data on the internet

- This recipe uses the following steps:
  1. Add the http package.
  2. Delete data on the server.
  3. Update the screen.



Delete data on the internet

- Using the http.delete() method

```
Future<http.Response> deleteAlbum(String id) async {
  final http.Response response = await http.delete(
    Uri.parse('https://jsonplaceholder.typicode.com/albums/$id'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
  );

  return response;
}
```

Delete data on the internet

- Update the screen

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    ElevatedButton(
      child: const Text('Delete Data'),
      onPressed: () {
        setState(() {
          var result = deleteAlbum(snapshot.data!.id.toString());
        });
      },
    ),
  ],
);
```