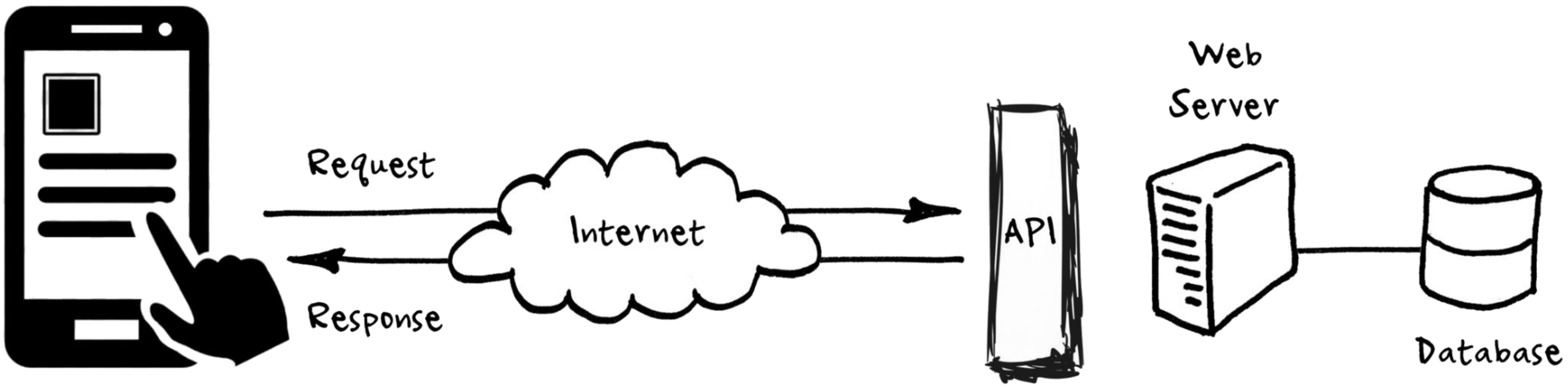
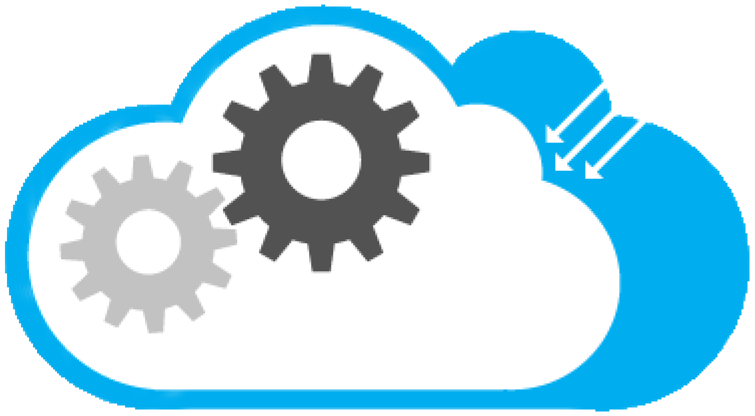
# Calling Web API



# Outline

## Web API
## (aka Web Services / REST API)



< Back

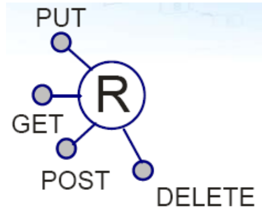
# Working with Web APIs – the Why?

- Phones can not serve as centralized data stores, so we **need servers**
- Even when we can do heavy tasks on-device, we should not
    - Servers are powerful, phones are not
    - Processing a lot of data / complex computation on a phone is a drain on its resources: Battery, CPU, Memory
- As good citizens on an Android phone, our **apps should consume as little resources as possible**
- Calling Web APIs lets the app connect to the outside world

# What is a Web API?

- Web API = Web accessible Application Programming Interface accessible via HTTP to allow programmatic access to applications
  - Also known as Web Services
  - Can be accessed by a broad range of clients including browsers and mobile devices
- Web API is a web service that accepts requests and returns **structured data** (JSON in most cases)
  - Programmatically accessible at a particular URL
  - You can think of it as a Web page returning JSON instead of HTML
- Major goal = **interoperability between heterogeneous systems**

# Web Services Principles



- **Resources have unique address (nouns) i.e., a URI**

Any **information that can be named can be a resource**: a document or image, a dynamic service to get weather or news, a collection of books and their authors, and so on

e.g., http://example.com/customers/123

- **Can use a Uniform Interface (verbs) to access them:**

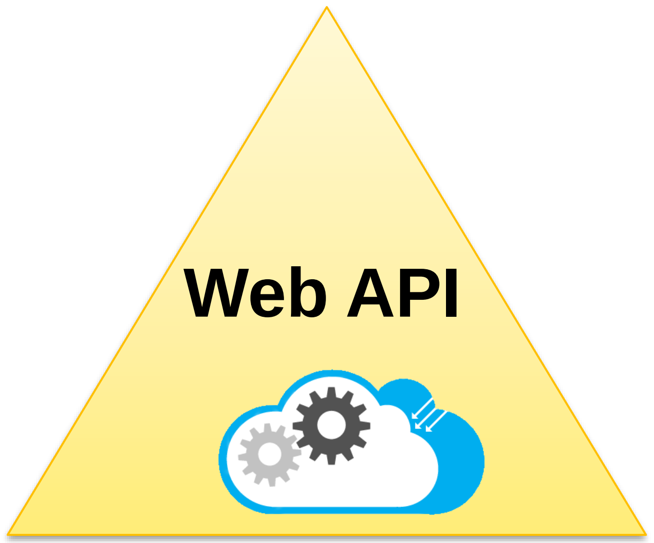
  - HTTP verbs: GET, POST, PUT, and DELETE

- **Resource has representation(s) (data format)**

  - A resource can be in a variety of data formats such as JSON and XML

# Web API Main Concepts

**Nouns (Resources)**
e.g., http://example.com/employees/12345



**Verbs**
e.g., GET, POST

**Representations**
e.g., XML, JSON

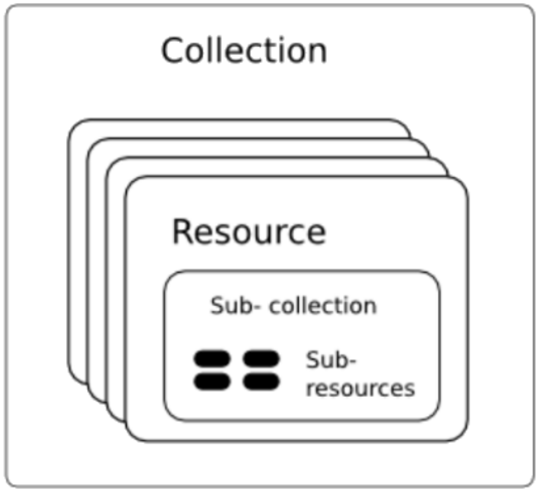
# Naming Resources

- Web API uses URL to identify resources

```
Often **api** path is used
for better organization
```

  - http://localhost/**api**/books/
  - http://localhost/api/books/ISBN-0011
  - http://localhost/api/books/ISBN-0011/authors

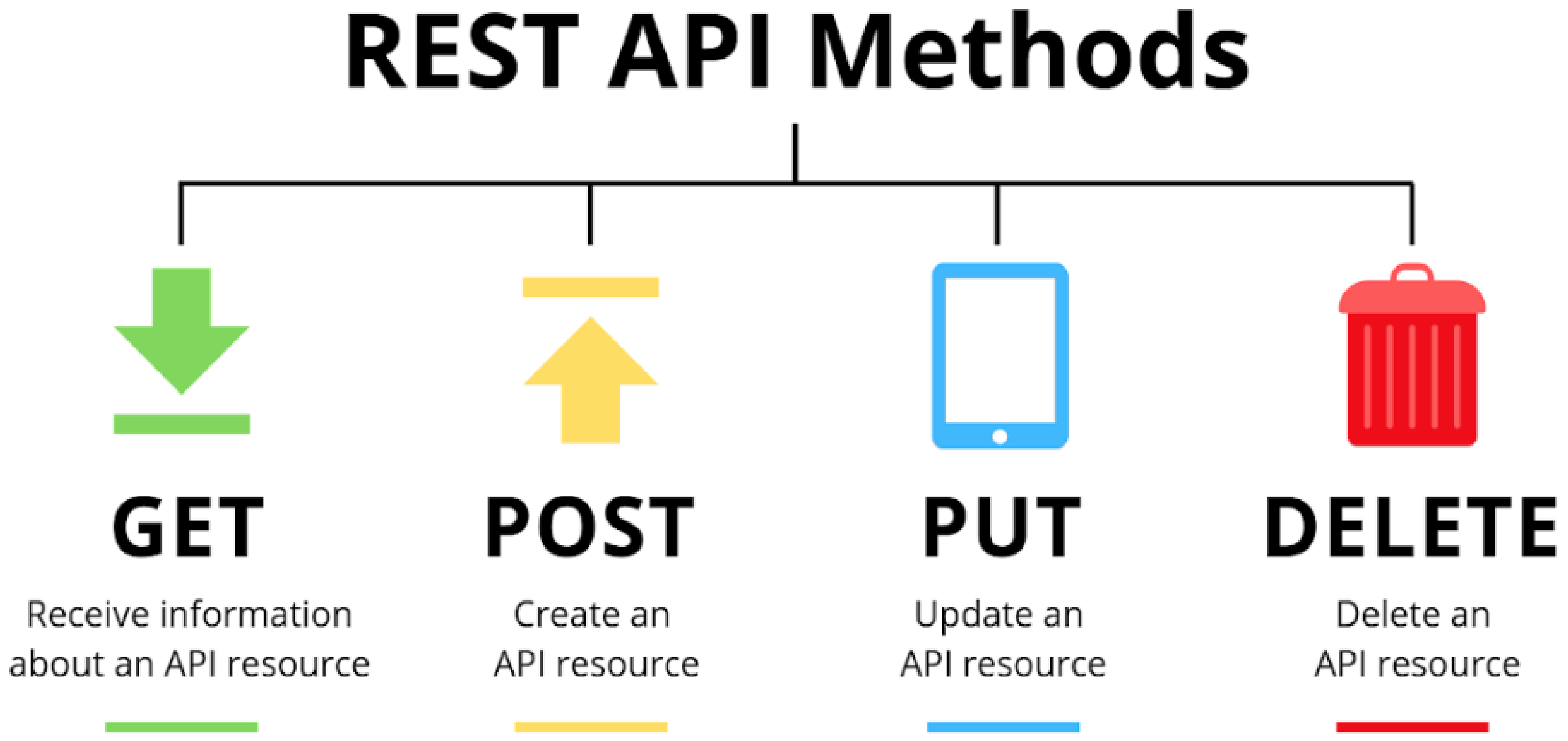
  - http://localhost/api/classes
  - http://localhost/api/classes/cmps356
  - http://localhost/api/classes/cs356/students



- As you traverse the **path** from more generic to more specific, you are navigating the data

# HTTP Verbs

HTTP Verbs represent the **actions** to be performed on resources

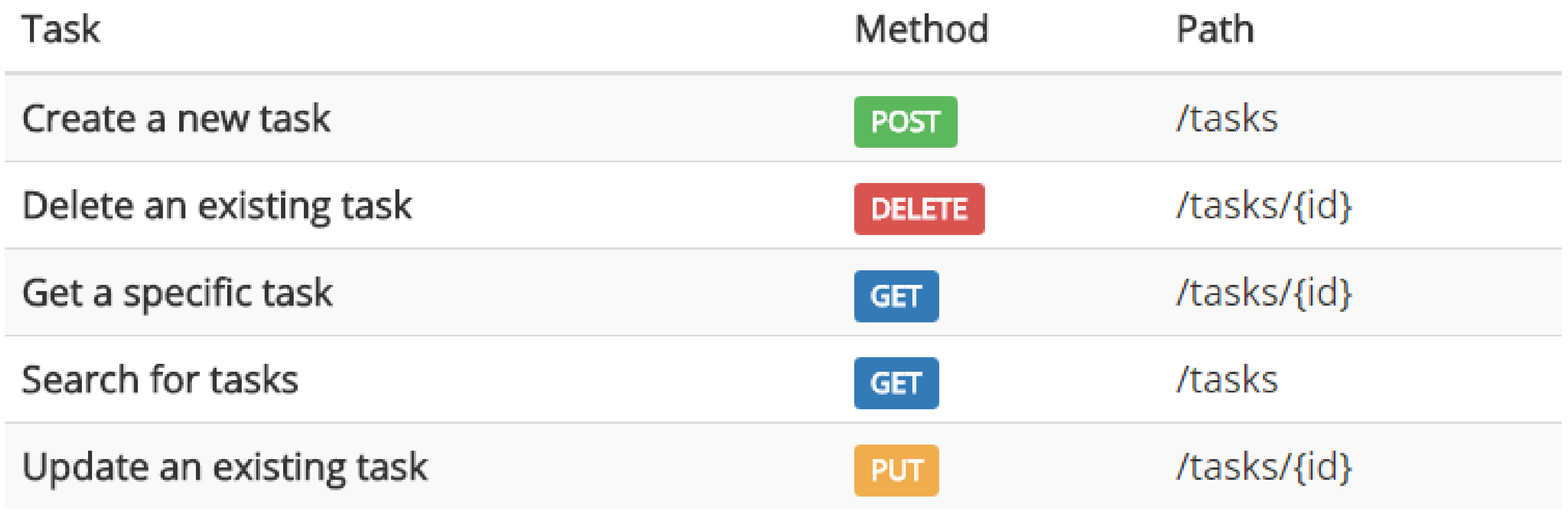


## CRUD (Create, Read, Update and Delete) Operations and their Mapping to HTTP Verbs

- **GET** - Read a resource
  - **GET** /books - Retrieve all books
  - **GET** /books/:id - Retrieve a particular book
- **POST** - Create a new resource
  - **POST** /books - Create a new book
- **PUT** - Update a resource
  - **PUT** /books/:id - Update a book
- **Delete** – Delete a resource
  - **DELETE** /books/:id - Delete a book

$\}$ The resource data (e.g., book details) are placed in the **body** of the request

# Example 2 - Task Service API



| 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} |

## Representations

- In all requests and responses, it is important to share data in a **format which both the client and server can understand**
- Two main formats are commonly used:
- **JSON**

```
{
     code: 'cmp312',
     name: 'Mobile App Development'
}
```

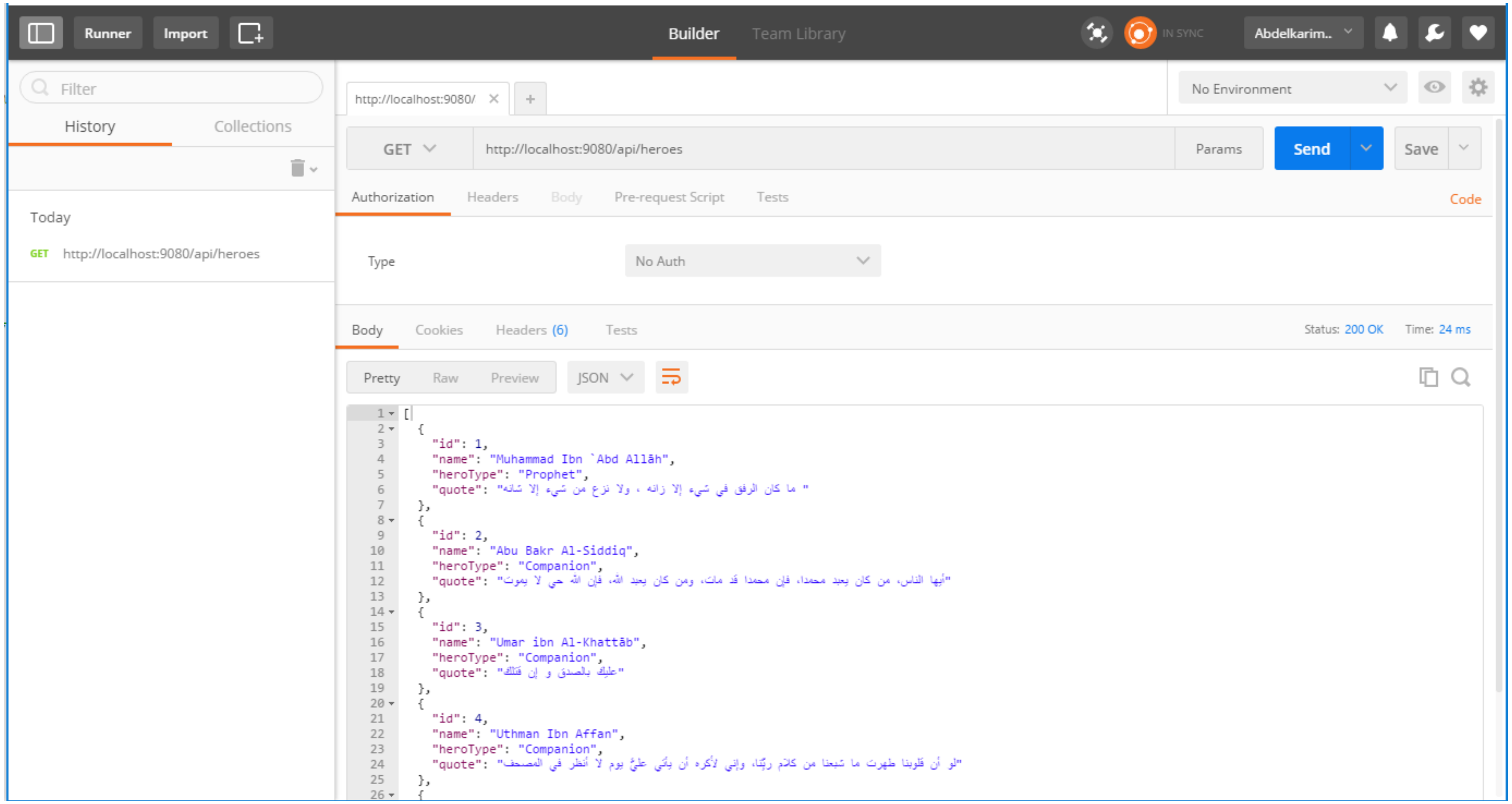
- **XML**

```
<course>
  <code>cmps312</code>
  <name>Mobile App Development</name>
</course>
```

# Testing Web API

- Using Postman to test Web API

https://www.postman.com/downloads/



Today

GET http://localhost:9080/api/heroes

No Environment

GET http://localhost:9080/api/heroes

Authorization Headers Body Pre-request Script Tests

Type: No Auth

Body — Status: 200 OK Time: 24 ms

JSON

```
[
  {
    "id": 1,
    "name": "Muhammad Ibn `Abd Allāh",
    "heroType": "Prophet",
    "quote": " ما كان الرفق في شيء إلا زانه ، ولا نزع من شيء إلا شانه"
  },
  {
    "id": 2,
    "name": "Abu Bakr Al-Siddiq",
    "heroType": "Companion",
    "quote": "أيها الناس، من كان يعبد محمدا، فإن محمدا قد مات، ومن كان يعبد الله، فإن الله حي لا يموت"
  },
  {
    "id": 3,
    "name": "Umar ibn Al-Khattāb",
    "heroType": "Companion",
    "quote": "عليك بالصدق و إن قتلك"
  },
  {
    "id": 4,
    "name": "Uthman Ibn Affan",
    "heroType": "Companion",
    "quote": "لو أن قلوبنا طهرت ما شبعنا من كلام ربّنا، وإني لأكره أن يأتي عليَّ يوم لا أنظر في المصحف"
  },
  {
```

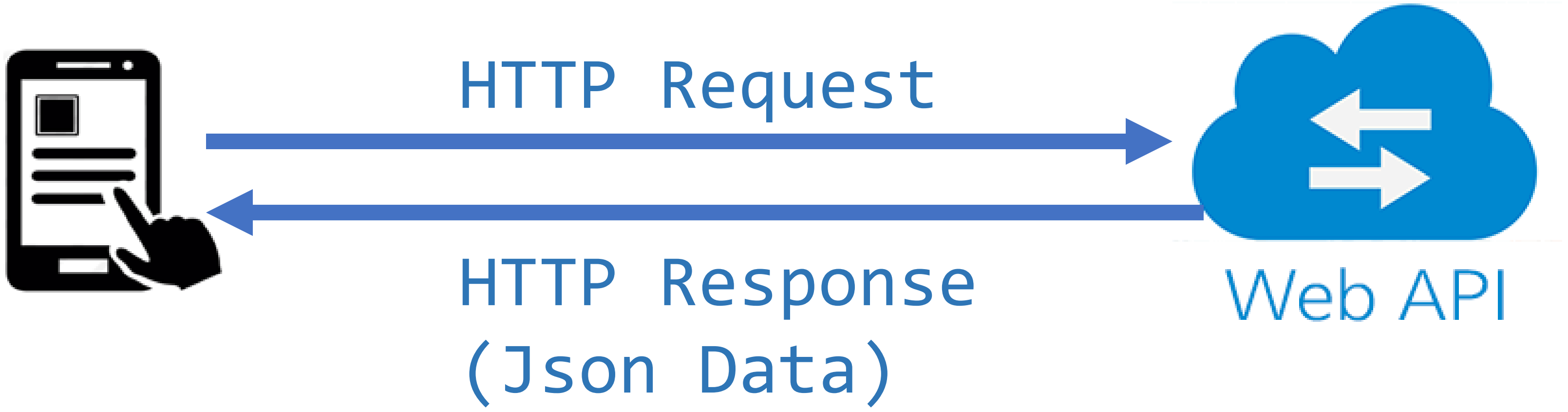
2

# Dio

< Back

## dio

- **dio** is HTTP networking package for Dart/Flutter, for a mobile app to call a remote Web API
  - Make HTTP requests and handle responses



# dio – 3 Programming Steps

1. Define **Classes** for input/output objects used when interacting with the Web API

2. Create and configure an instance of dio

3. Use its **`.get`**, **`.post`**, **`.put`**, **`.delete`** methods to interact with the remote Web API

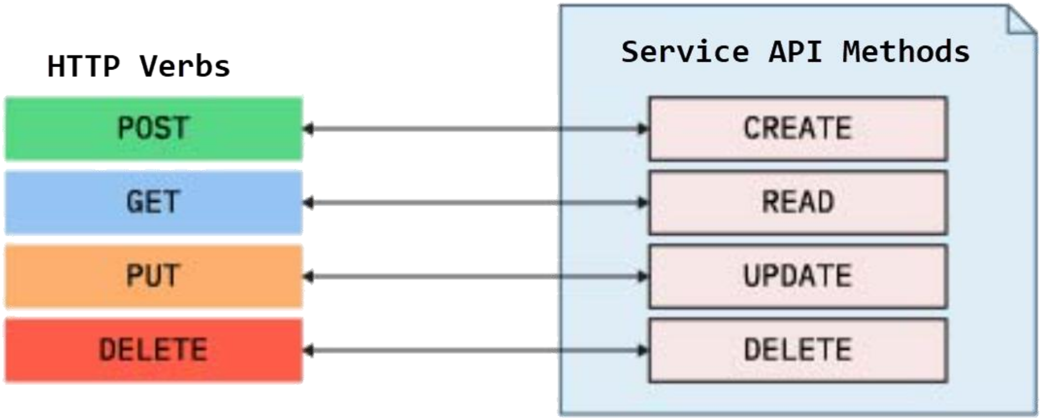
HTTP GET HTTP POST HTTP PUT HTTP DELETE

## Define Serializable Classes for input/output objects used when interacting with the Web API

```dart
class ToDo {
  final int id;
  final String title;
  final int userId;
  final bool completed;

  ToDo({
    this.id = 0,
    required this.title,
    this.userId = 1,
    this.completed = false,
  });

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

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'title': title,
      'userId': userId,
      'completed': completed,
    };
  }
```

# Use Get/Post/Put/Delete to interact with the Web API

- **dio** provides specific functions for basic HTTP methods: get, post, put, and delete.



```
const BASE_URL = "https://api.polygon.io/v1/open-close"
// Create and configure an instance of dio
final _dio = Dio();
_dio.options.baseUrl = BASE_URL;
final symbol = "Tesla"
final response = _dio.get("/$symbol");
final MarketStockQuote.fromJson(response.data);
```

## Another get example aexample

```
import 'package:dio/dio.dart’;

// Create and configure an instance of dio
final _dio = Dio();
...
const String BASE_URL = 'http://api.example.net/todos';
_dio.options.baseUrl = BASE_URL;

Future<ToDo> getToDo(int toDoId) async {
  final response = await _dio.get('/$toDoId');
  return ToDo.fromJson(response.data);
}
```

## Path Parameters vs. Query Parameters

- Required parameters can be passed using **path parameters** appended to the URL path
  - E.g., **/students/1234** this will return the details of the student with the id 1234
- Named **query parameters** can be added to the URL path after a **?** E.g., **/posts?sortBy=createdOnDate**
- Query parameters are often used for **optional** parameters (e.g., optionally specifying the property to be used to sort of results)

# Post / Put Request

- Assign the json data to be sent in the body of the request using data property
    - Use post for add and put for update

```
Future<ToDo> addToDo(ToDo toDo) async {
  final response = await _dio.post("/", data: toDo.toJson());
  return ToDo.fromJson(response.data);
}

Future<void> updateToDo(ToDo toDo) async {
  final response = await _dio.put('/${toDo.id}', data: toDo.toJson());
  ToDo.fromJson(response.data);
}
```

## Delete Request

- Use the dio.**`delete`** method to delete a resource
  - Specify the resource id to be deleted in the request url

```
Future<bool> deleteToDo(int toDoId) async {
  final response = await _dio.delete('/$toDoId');
  return response.statusCode == 200;
}
```