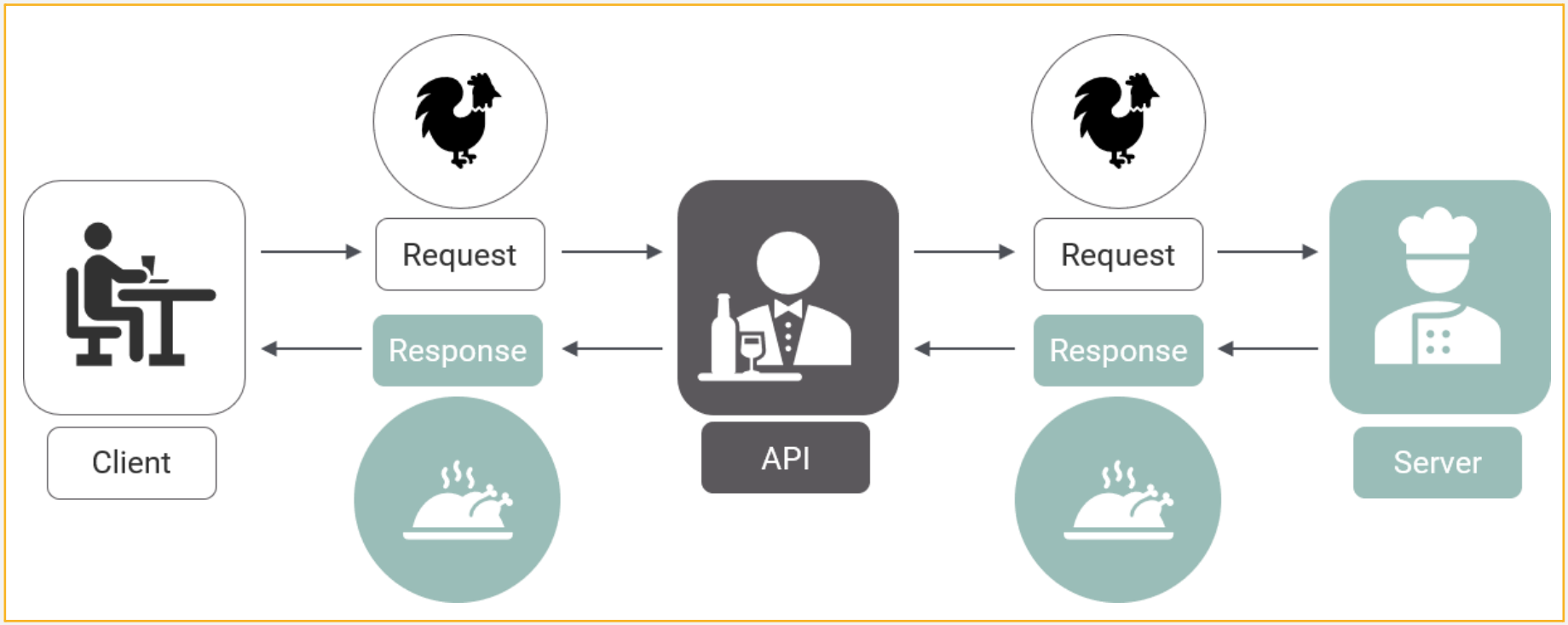
# WEB API Interaction

# Content

1. HTTP Requests & HTTP Responses
2. JSON
3. Postman Application
4. Mock APIs

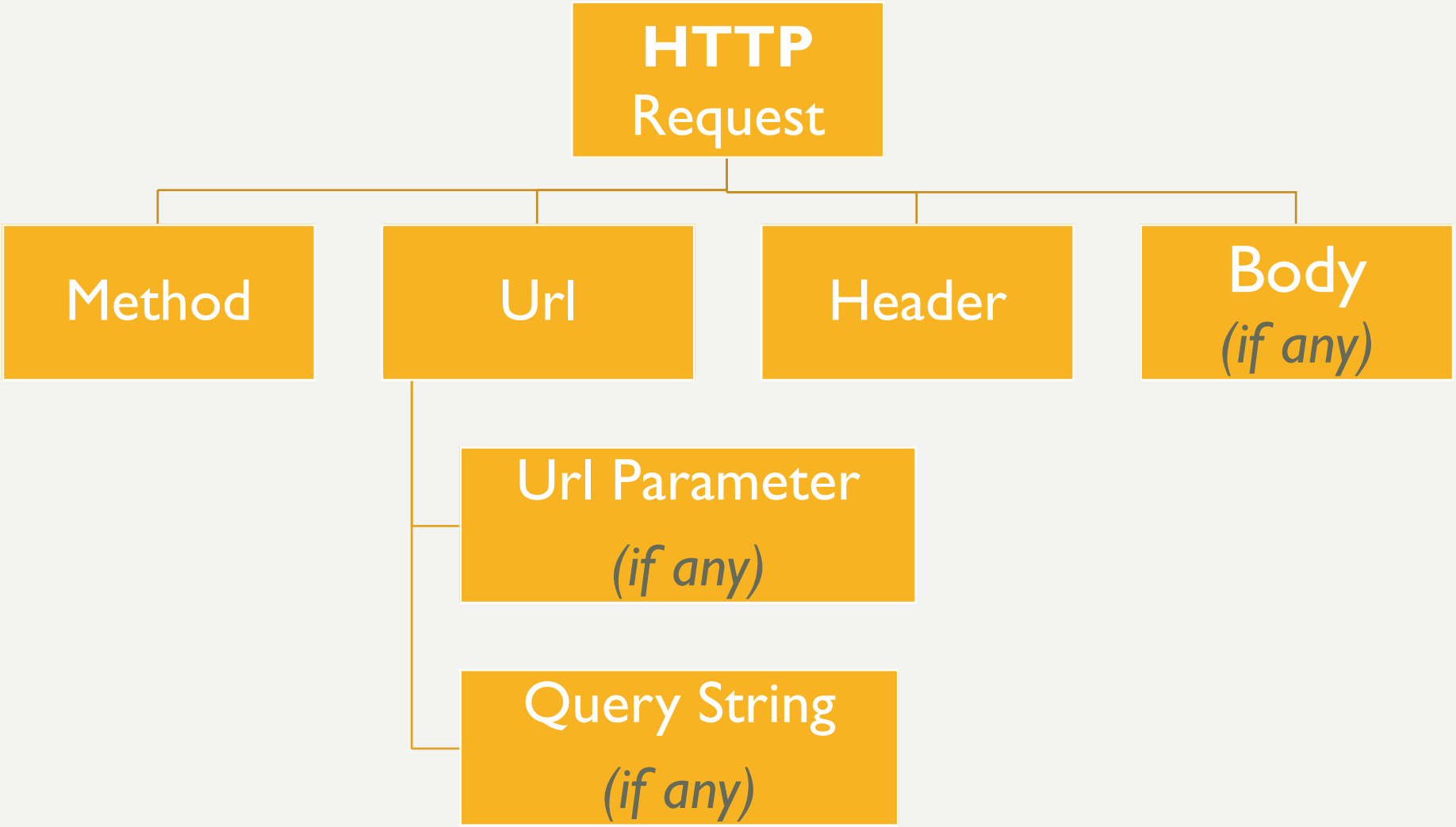
# 1. HTTP Requests & Responses




1.1. HTTP Requests

1.2. HTTP Responses

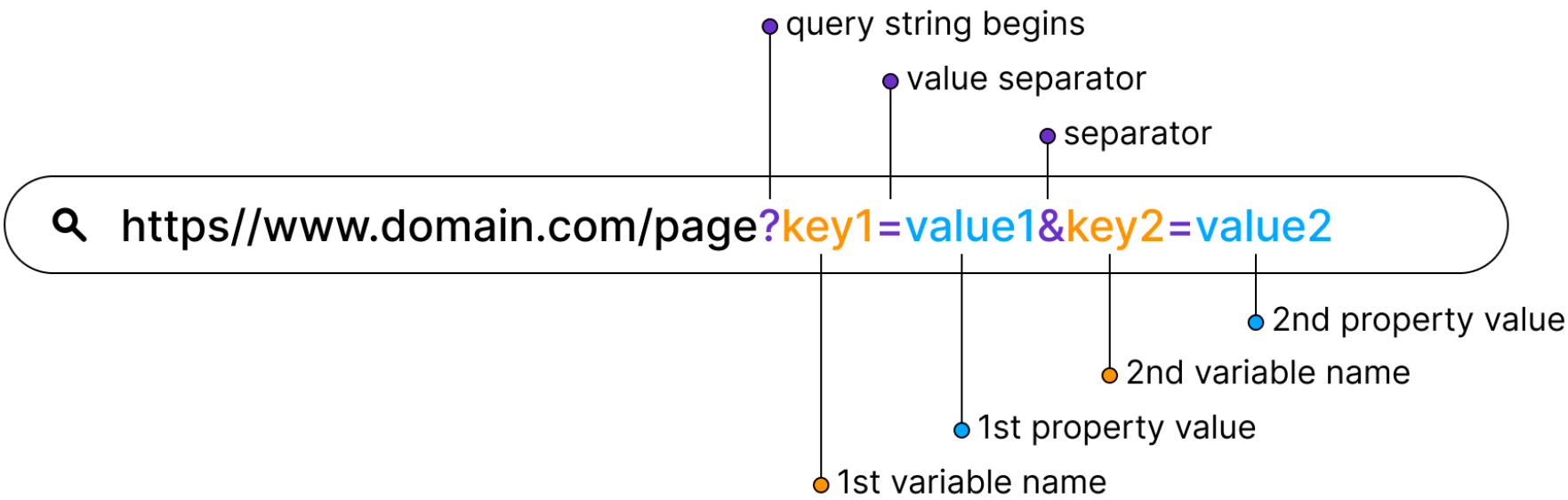
# 1.1. HTTP Requests



# Methods

- **GET**: The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- **POST**: The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- **PUT**: The PUT method replaces all current representations of the target resource with the request payload.
- **DELETE**: The DELETE method deletes the specified resource.
- **PATCH**: The PATCH method is used to apply partial modifications to a resource.

# URL (Uniform Resource Locators)

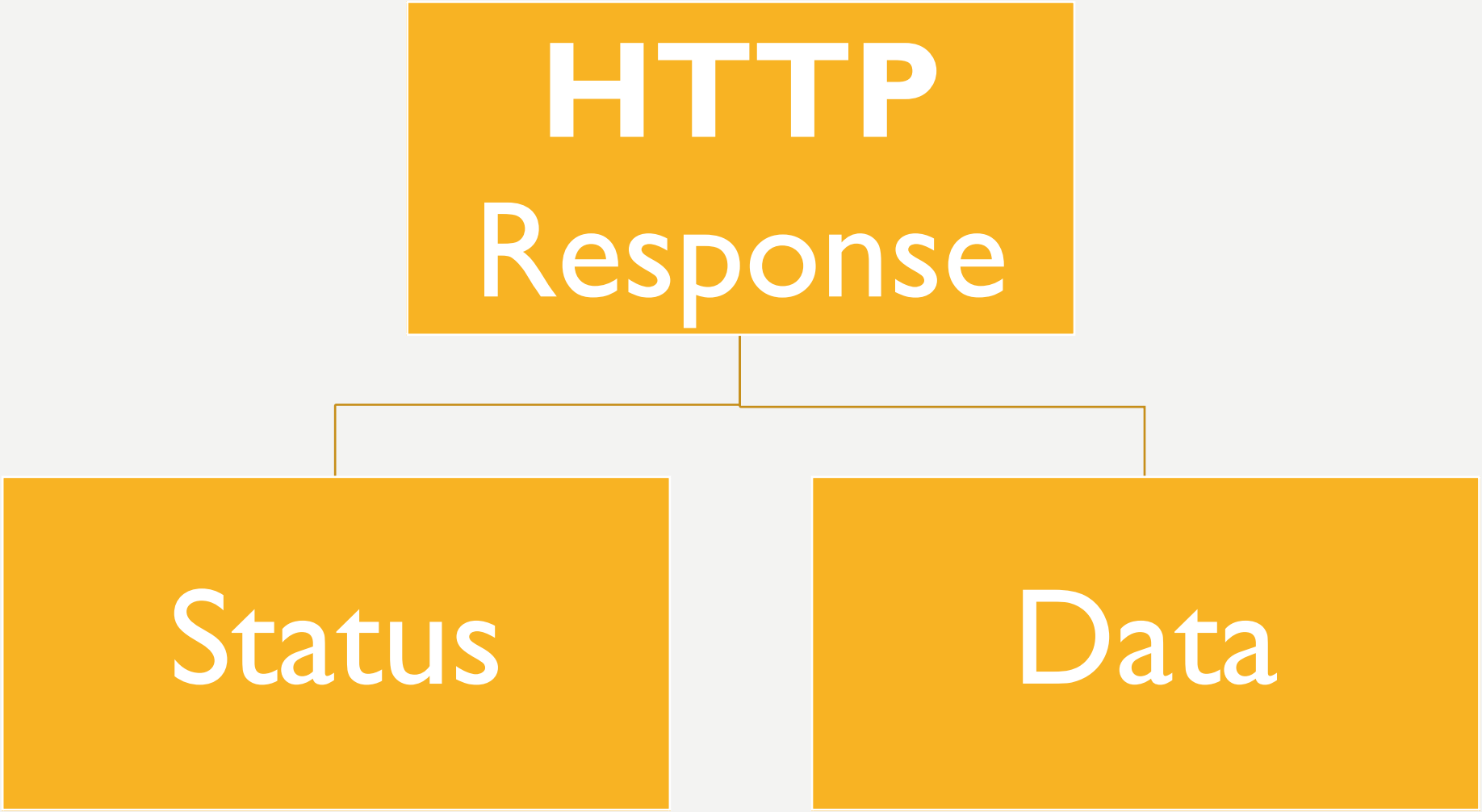
# Header

- Content-Type
- Accept
- Authorization
- …

# Body

- Used in: POST, PUT, PATCH
- Data:
  - x-www-form-urlencoded
  - form-data
  - raw
    - text
    - **json**
    - xml
    - html
  - binary
  - GraphQL

# 1.2. HTTP Responses



# HTTP response status codes

Responses are grouped in five classes

- Informational responses (100–199)
- Successful responses (200–299)
- Redirects (300–399)
- Client errors (400–499)
- Server errors (500–599)

# Successful responses (200–299)

- 200 OK

  The request has succeeded.

- 201 Created

  The request has succeeded and a new resource has been created as a result.

- 204 No Content

  There is no content to send for this request, but the headers may be useful.

# Client errors (400–499)

- 400 Bad Request

  The server could not understand the request due to invalid syntax.

- 401 Unauthorized

  Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated".

- 403 Forbidden

  The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource.

- 404 Not Found

  The server can not find the requested resource.

- 405 Method Not Allowed

  The request method is known by the server but has been disabled and cannot be used.

- 409 Conflict

  This response is sent when a request conflicts with the current state of the server.

- 415 Unsupported Media Type

  The media format of the requested data is not supported by the server, so the server is rejecting the request.

# Server errors (500–599)

- 500 Internal Server Error

  The server has encountered a situation it doesn't know how to handle.

# Data

- **json**
- xml
- plain-text
- html
- stream
- …

# 2. JSON

2.1. What is JSON?

2.2. Why Use JSON?

2.3. JSON vs XML

2.4. Syntax

2.5. JSON Data Types

2.6. JSON Uses JavaScript Syntax

# 2.1. What is JSON?

- JSON: **J**ava**S**cript **O**bject **N**otation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.
- JSON is a lightweight data-interchange format.
- JSON is "self-describing" and easy to understand.

# 2.2. Why Use JSON?

2.2.1. Exchanging Data

2.2.2. Sending Data

2.2.3. Receiving Data

2.2.4. Storing Data

# 2.2.1. Exchanging Data

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

# 2.2.2. Sending Data

- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server

```
var myObj = {name: "John", age: 31, city: "New York"};
var myJSON = JSON.stringify(myObj);
window.location = "demo_json.php?x=" + myJSON;
```

# 2.2.3. Receiving Data

- If you receive data in JSON format, you can convert it into a JavaScript object

```
var myJSON = '{"name":"John", "age":31, "city":"New
York"}';
var myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.name;
```

# 2.2.4. Storing Data

- When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.
- JSON makes it possible to store JavaScript objects as text.

```
// Storing data:
myObj = {name: "John", age: 31, city: "New York"};
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

# 2.3. JSON vs XML

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

```json
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

# 2.4. Syntax

- The JSON syntax is a subset of the JavaScript syntax.
- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects "{..}"
  - Square brackets hold arrays "[..]"
- JSON data is written as name/value pairs.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
"name":"John"
```

*JSON names require double quotes. JavaScript names don't.*

# 2.5. JSON Data Types

- In JSON, values must be one of the following data types:
  - a string
  - a number
  - an object (JSON object)
  - an array
  - a boolean
  - *null*

- **JSON Strings:** Strings in JSON must be written in double quotes. In JavaScript can use single quotes.

```
{ "name":"John" }
```

- **JSON Numbers:** Numbers in JSON must be an integer or a floating point.

```
{ "age":30 }
```

- **JSON Objects:** Object in JSON must be enclosed with curly braces.

```
{
"employee":{ "name":"John", "age":30, "city":"New York" }
}
```

- **JSON Arrays:** Object in JSON must be enclosed with square brackets.

```
{
"employees":[ "John", "Anna", "Peter" ]
}
```

- **JSON Booleans:** Values in JSON can be true/false.

```
{ "sale":true }
```

- **JSON null:** Values in JSON can be null.

```
{ "middlename":null }
```

# 2.6. JSON Uses JavaScript Syntax

```
var person = { name: "John", age: 31, city: "New York" };
```

- Access

```
// returns John
person.name;
```

```
// returns John
person["name"];
```

- Modify

```
person.name = "Gilbert";
```

```
person["name"] = "Gilbert";
```

# 3. Postman Application

- **Postman** is a Google Chrome app for interacting with HTTP APIs. It presents you with a friendly GUI for constructing requests and reading responses.
- The people behind **Postman** also offer an add-on package called Jetpacks, which includes some automation **tools** and, most crucially, a JavaScript testing library.

# Downloading Postman

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

## The Postman app

The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

Download the App

By downloading and using Postman, I agree to the Privacy Policy and Terms.

Version 8.12.4 · Release Notes · Product Roadmap

Not your OS? Download for Mac (macOS) or Linux (x64)

# GET Method

https://restfulapi.dnd-group.net/api/majors

Save

GET https://restfulapi.dnd-group.net/api/majors Send

Params

Query Params

| KEY | VALUE | DESCRIPTION | Bulk Edit |
|---|---|---|---|
| Key | Value | Description | |

Body 200 OK 4.44 s 710 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "errorCode": 0,
    "message": "",
    "data": [
        {
            "id": 909,
            "name": "AngularJS"
        },
        {
            "id": 107,
            "name": "Business Administration"
        },
```

# POST Method

https://restfulapi.dnd-group.net/api/majors

Save

POST https://restfulapi.dnd-group.net/api/majors Send

Body raw JSON Beautify

```
1 {
2     "name": "New Major"
3 }
```

Body 201 Created 784 ms 346 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2     "errorCode": 0,
3     "message": "",
4     "data": {
5         "name": "New Major",
6         "id": 928
7     }
8 }
```

# PUT Method

https://restfulapi.dnd-group.net/api/majors/928

Save

PUT https://restfulapi.dnd-group.net/api/majors/928 Send

Body raw JSON Beautify

```
{
    "name": "Update a Major"
}
```

Body 200 OK 331 ms 346 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "errorCode": 0,
    "message": "",
    "data": {
        "id": 928,
        "name": "Update a Major"
    }
}
```

# DELETE Method with Token

https://restfulapi.dnd-group.net/api/majors/928

Save

DELETE https://restfulapi.dnd-group.net/api/majors/928 Send

Headers

Headers 9 hidden

| | KEY | VALUE |
|---|---|---|
| ☑ | Content-Type | application/json |
| ☑ | Authorization | Bearer f7b5c076b7ffe5fb3ff6242fd5b2fcd1989f70d74d0e |
| ☐ | Accept | application/json |
| | Key | Value |

Body 200 OK 347 ms 346 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "errorCode": 0,
    "message": "",
    "data": {
        "id": 928,
        "name": "Update a Major"
    }
}
```

# 4. Mock APIs

- mockAPI: https://mockapi.io
- testAPI: https://testapi.io
- fakejson: https://fakejson.com

# THE END