API

Pre-requisites

- Postman

An **API (Application Programming Interface)** is a way for two systems or applications to communicate and exchange information. APIs act as a bridge, allowing different software components to interact without knowing how each one works internally. For example:

- A weather app uses an API to fetch real-time weather data from a remote server.
- A payment gateway like PayPal provides APIs for online stores to process payments.

APIs simplify integration, enabling developers to build more complex applications by leveraging existing services.

What are the Types of API Protocols?

APIs can use various communication protocols, including:

1. REST (Representational State Transfer):
    - Most common.
    - Stateless, meaning each request from the client contains all the information needed to process it.
    - Uses HTTP for communication.
2. **SOAP (Simple Object Access Protocol)**:
    - A stricter protocol with XML-based messaging.
    - Common in enterprise systems for high-security transactions.
3. **GraphQL**:
    - Flexible query language allowing clients to specify the exact data they need.
    - Commonly used for modern apps.
4. **gRPC**:
    - Efficient binary protocol for high-performance APIs.
    - Often used in microservices and real-time communication.
5. **WebSockets**:

- Enables full-duplex communication for real-time apps like chat or live notifications.

What are HTTP Methods?

HTTP verb	GET	PUT	POST	DELETE	PATCH
What does it do?	Retrieve a resource or resource list	Replace a resource	Create a resource or send data to server	Delete a resource	Partially modify a resource
Example	GET /users/123	PUT /users/123 { ... }	POST /users { ... }	DELETE /user/123	PATCH /user/123 { name: xxx }
Request Has Body?	No	Yes	Yes	Maybe	Yes
Response Has Body?	Yes	No	Yes	Maybe	Yes
Cachable	Yes	No	No	No	No
Idempotent	Yes	Yes	No	Yes	No
Map to Service Method	get(), list()	update()	create()	delete()	update()

from https://blog.bytebytego.com/p/design-effective-and-secure-rest

HTTP methods are verbs used to tell an API what action to perform:

- **GET**: Retrieve data (e.g., `GET /users` for a list of users).
- **POST**: Create new data (e.g., `POST /users` to add a new user).
- **PUT**: Update existing data completely (e.g., `PUT /users/123` to replace user data).
- **PATCH**: Update part of the data (e.g., `PATCH /users/123` to change a user's email).
- **DELETE**: Remove data (e.g., `DELETE /users/123` to delete a user).

Things that are nice to know:

- **Idempotent**: A method is idempotent if making the same request multiple times results in the same state on the server (e.g., GET, PUT, DELETE).
- **Cachable**: If the response can be stored and reused for subsequent requests, it's cachable (e.g., GET responses).

What is JSON in the Context of APIs?

JSON (JavaScript Object Notation) is a lightweight data format used to exchange information between a client and a server in APIs. It's widely used because it's easy for both humans and machines to read and write. In APIs, JSON serves as the "package" that carries the data being sent and received.

JSON in API Requests

When sending data to an API (e.g., with a `POST` or `PUT` request), JSON is often used in the request body. For example:

- **Request Example**:

```
POST /users Content-Type: application/json
{
    "name": "Ash",
    "age": 10,
    "trainer": true
}
```

Here:

- `Content-Type: application/json` tells the server that the data is in JSON format.
- The JSON body contains the user information (`name`, `age`, and `trainer`).

JSON in API Responses

When an API sends back data to the client, it's usually in JSON format. For example:

- **Response Example**:

```
{
    "id": 1,
    "name": "Charmander",
    "type": "Fire",
    "evolution": "Charmeleon"
}
```

This is the server's response to a request for a Pokémon's details. The client (e.g., a web or mobile app) can then use this data.

JSON Features:

1. **Key-Value Pairs**: Data is stored as key-value pairs:

   `{"key": "value"}`

2. **Arrays**: JSON supports arrays for lists of items:

   `{"pokemon": ["Bulbasaur", "Charmander", "Squirtle"] }`

3. **Nested Objects**: JSON can store objects within objects for complex data:

```
{
    "trainer": {
        "name": "Ash",
        "age": 10
    },
    "pokemon": ["Pikachu", "Charizard"]
}
```

4. **Types Supported**: JSON supports strings, numbers, booleans, objects, arrays, and `null`.

JSON Best Practices in APIs:

1. **Always Use Proper Formatting**: Valid JSON syntax is critical (`{}` for objects, `[]` for arrays).

2. **Include Descriptive Keys**: Keys should clearly define what the value represents (e.g., use `userName` instead of `u`).

3. **Error Handling**: When errors occur, APIs should return clear JSON responses:

   `{"error": "Resource not found", "code": 404 }`

4. **Use JSON Validation**: Validate JSON payloads to ensure they match expected schemas.

What is the Structure of a Typical API Endpoint?

A typical API endpoint consists of:

1. **Domain**: The base address of the API (e.g., `https://api.example.com`).
2. **Path**: Specifies the resource (e.g., `/users` or `/users/123`).
3. **Query Parameters**: Key-value pairs to filter or customize the request (e.g., `/users?age=25`).
4. **Headers**: Metadata about the request, such as `Authorization`, `Content-Type`, or `Accept`.
5. **Body**: Used with methods like `POST` or `PUT` to send data in JSON or other formats.

Example:

```shell
curl -X GET "https://pokeapi.co/api/v2/pokemon?limit=2&offset=0"
```

Result:


```
{
  "count": 1302,
  "next": "https://pokeapi.co/api/v2/pokemon?offset=2&limit=2",
  "previous": null,
  "results": [
    {
      "name": "bulbasaur",
      "url": "https://pokeapi.co/api/v2/pokemon/1/"
    },
    {
      "name": "ivysaur",
      "url": "https://pokeapi.co/api/v2/pokemon/2/"
    }
  ]
}
```


- **Domain**: `https://pokeapi.co`
- **Path**: `/api/v2/pokemon`

- **Path Parameter**: `v2`
- **Query Parameter**: `limit=2&offset=0`
- **HTTP Method**: `GET Method`
  - `-X GET`: Specifies the HTTP method (GET is the default, so this part is optional)

What are Status Codes?

Status codes indicate the result of an API request. Common codes include:

- **2xx**: Success
  - `200 OK`: Request succeeded.
  - `201 Created`: Resource created.
- **4xx**: Client Errors
  - `400 Bad Request`: Invalid request.
  - `401 Unauthorized`: Authentication required.
  - `404 Not Found`: Resource not found.
- **5xx**: Server Errors
  - `500 Internal Server Error`: Something went wrong on the server.

```shell
curl -X GET "https://http.cat/102.jpg" --output 102.jpg
```

Basic REST API Calling

JavaScript

1. **Using Fetch API**:

```js
fetch('https://pokeapi.co/api/v2/pokemon?limit=2&offset=0')
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

2. **Using Axios**:

```JS
import axios from 'axios';

axios.get('https://pokeapi.co/api/v2/pokemon?limit=2&offset=0')
.then(response => console.log(response.data))
.catch(error => console.error('Error:', error));
```

Python

1. **Using `requests`**:

```PYTHON
import requests
response = requests.get('https://pokeapi.co/api/v2/pokemon?
limit=2&offset=0')
if response.status_code == 200:
    print(response.json())
else:
    print(f"Error: {response.status_code}")
```

2. **Using `http.client`**:

```PYTHON
from http.client
import HTTPSConnection
conn = HTTPSConnection("pokeapi.co")
conn.request("GET", "/api/v2/pokemon?limit=2&offset=0")

response = conn.getresponse()

if response.status == 200:
    print(response.read().decode())
else:
    print(f"Error: {response.status}")
```

cURL

`cURL` (short for **Client URL**) is a command-line tool used to transfer data to and from servers using various protocols, including HTTP, HTTPS, FTP, SMTP, and more. It's commonly used for testing APIs, downloading files, or automating web interactions.

SHELL

```
curl -X GET "https://pokeapi.co/api/v2/pokemon?
limit=2&offset=0"
```

or you can add `| jq` to pretty print the output

SHELL

```
curl -X GET "https://pokeapi.co/api/v2/pokemon?
limit=2&offset=0" | jq
```

Using API Documentation

You can use tools like:

- **Swagger**: If available, Swagger UI for PokeAPI can help you test the endpoint interactively.
- **Postman**: Import the endpoint `https://pokeapi.co/api/v2/pokemon?limit=2&offset=0`, set the method to `GET`, and send the request to view the response.