Reset Api Best Practice

***1. Use JSON as the Format for Sending and Receiving Data***

***Describe:***

In the past, accepting and responding to API requests were done mostly in XML and even HTML. But these days, JSON (JavaScript Object Notation) has largely become the de-facto format for sending and receiving API data.

This is because, with XML for example, it's often a bit of a hassle to decode and encode data – so XML isn’t widely supported by frameworks anymore.

JavaScript, for example, has an inbuilt method to parse JSON data through the fetch API because JSON was primarily made for it. But if you are using any other programming language such as Python or PHP, they now all have methods to parse and manipulate JSON data as well.

For example, Python provides json.loads() and json.dumps() for working with JSON data.

To ensure the client interprets JSON data correctly, you should set the Content-Type type in the response header to application/json while making the request.

For server-side frameworks, on the other hand, many of them set the Content-Type automatically. Express, for example, now has the express.json() middleware for this purpose. The body-parser NPM package still works for the same purpose, too.

***2. Use Nouns Instead of Verbs in Endpoints.***

**Describe:**

When you're designing a REST API, you should not use verbs in the endpoint paths. The endpoints should use nouns, signifying what each of them does.

This is because HTTP methods such as GET, POST, PUT, PATCH, and DELETE are already in verb form for performing basic CRUD (Create, Read, Update, Delete) operations.

GET, POST, PUT, PATCH, and DELETE are the commonest HTTP verbs. There are also others such as COPY, PURGE, LINK, UNLINK, and so on.

So, for example, an endpoint should not look like this:

https://mysite.com/getPosts or https://mysite.com/createPost

Instead, it should be something like this: https://mysite.com/posts

In short, you should let the HTTP verbs handle what the endpoints do. So GET would retrieve data, POST will create data, PUT will update data, and DELETE will get rid of the data.

***3. Name Collections with Plural Nouns***

describe:

You can think of the data of your API as a collection of different resources from your consumers.

If you have an endpoint like https://mysite.com/post/123, it might be okay for deleting a post with a DELETE request or updating a post with PUT or PATCH request, but it doesn’t tell the user that there could be some other posts in the collection. This is why your collections should use plural nouns.

So, instead of https://mysite.com/post/123, it should be <https://mysite.com/posts/123>.

***4. Use Status Codes in Error Handling***

**Describe:**

You should always use regular HTTP status codes in responses to requests made to your API. This will help your users to know what is going on – whether the request is successful, or if it fails, or something else.

Below is a table showing different HTTP Status Code ranges and their meanings:

***5. Use Nesting on Endpoints to Show Relationships***

**Describe:**

Oftentimes, different endpoints can be interlinked, so you should nest them so it's easier to understand them.

For example, in the case of a multi-user blogging platform, different posts could be written by different authors, so an endpoint such as https://mysite.com/posts/author would make a valid nesting in this case.

In the same vein, the posts might have their individual comments, so to retrieve the comments, an endpoint like https://mysite.com/posts/postId/comments would make sense.

You should avoid nesting that is more than 3 levels deep as this can make the API less elegant and readable.

***6. Use Filtering, Sorting, and Pagination to Retrieve the Data Requested***

**Describe:**

Sometimes, an API's database can get incredibly large. If this happens, retrieving data from such a database could be very slow.

Filtering, sorting, and pagination are all actions that can be performed on the collection of a REST API. This lets it only retrieve, sort, and arrange the necessary data into pages so the server doesn’t get too occupied with requests.

An example of a filtered endpoint is the one below:

https://mysite.com/posts?tags=javascript

This endpoint will fetch any post that has a tag of JavaScript.

***7. Use SSL for Security***

**Describe:**

SSL stands for secure socket layer. It is crucial for security in REST API design. This will secure your API and make it less vulnerable to malicious attacks.

Other security measures you should take into consideration include: making the communication between server and client private and ensuring that anyone consuming the API doesn’t get more than what they request.

SSL certificates are not hard to load to a server and are available for free mostly during the first year. They are not expensive to buy in cases where they are not available for free.

The clear difference between the URL of a REST API that runs over SSL and the one which does not is the “s” in HTTP:

https://mysite.com/posts runs on SSL.

http://mysite.com/posts does not run on SSL.

***8. Be Clear with Versioning***

**Describe:**

REST APIs should have different versions, so you don’t force clients (users) to migrate to new versions. This might even break the application if you're not careful.

One of the commonest versioning systems in web development is semantic versioning.

An example of semantic versioning is 1.0.0, 2.1.2, and 3.3.4. The first number represents the major version, the second number represents the minor version, and the third represents the patch version.

Many RESTful APIs from tech giants and individuals usually comes like this:

https://mysite.com/v1/ for version 1

https://mysite.com/v2 for version 2

***9. Provide Accurate API Documentation***

**Describe:**

When you make a REST API, you need to help clients (consumers) learn and figure out how to use it correctly. The best way to do this is by providing good documentation for the API.

The documentation should contain:

relevant endpoints of the API

example requests of the endpoints

implementation in several programming languages

messages listed for different errors with their status codes

One of the most common tools you can use for API documentation is Swagger. And you can also use Postman, one of the most common API testing tools in software development, to document your APIs.