Your Simple Guide to RESTFul APIs

Introduction

A

mong the various types of APIs out there, REST APIs are the most common and have become indispensable part of WEB Development to manage client and server interactions. Don’t worry if you find these technical terms overwhelming. In this Blog we will be going deep into RESTFul APIs and understand how they became a standard.

Understanding the Term ‘REST’

REST stands for **RE**presentational **S**tate **T**ransfer. It is an architectural style which provide a framework for the interactions between client-side and server-side applications.

***“Whenever a REST API will be called, a representation of the state of the resource will be transferred by the Server to the Client.”***

Understanding Client and Resource

Before going further, let’s understand what is client and resource.

* **Client:** It is any person or software which is using API. It is anyone who is trying to retrieve information.
* **Resource:** It is any object about which API provide the information. Resources are identified by a Uniform Resource Identifier.

For example, let’s take a scenario when you are checking your profile on Codeforces, you are hitting a GET Request or in layman terms you are retrieving information using their API.

(You can have a look later at: <https://codeforces.com/apiHelp>)



You can see above how I fetch user details. So here I am Client who is using Codeforces API.

Now, once you know who is a client, it is very obvious for you to identify the resource. Ya, you got it right. Here resource is the user detail which is being identified using a username.

The representation of the state can be in a **JSON**, XML or HTML format. For most APIs JSON is used as you can also see in the above case.

In case you don’t know what JSON is, no need to worry. JSON stands for **J**ava**S**cript **O**bject **N**otation. It is a human readable data interchange format for defining and transmitting objects.

Basic syntax for JSON is **name(key)-value pairs** and arrays separated by commas with names following Camel Case convention. For example,

{

"status": "OK",

"result": [

{

"contribution": 0,

"lastOnlineTimeSeconds": 1712374124,

"rating": 1107,

"friendOfCount": 3,

"titlePhoto": "https://userpic.codeforces.org/no-title.jpg",

"rank": "newbie",

"handle": "aryam\_agarwal",

"maxRating": 1158,

"avatar": "https://userpic.codeforces.org/no-avatar.jpg",

"registrationTimeSeconds": 1634376349,

"maxRank": "newbie"

}

]

}

Now let’s have look at the URI (Uniform Resource Identifier) used in the previous example.

<https://codeforces.com/api/user.info?handles=aryam_agarwal>

A URI looks very similar to a website address. The basic format is

scheme :// host:port / path to resource ? query parameter

You are thinking right :) . ***When you type a URL into a browser, an HTTP GET request is sent to the website and the response data is used to render the page.*** Now it is time to get into HTTP Methods.

Understanding REST HTTP Methods

The most commonly used HTTP methods in REST carry out CRUD operations (Create, Read, Update, Delete).

* **POST** – Create a new resource
* **GET** – Read access to a resource
* **PUT** – Update or create a resource
* **DELETE** – Delete a resource

Before taking a deeper dive into HTTP Methods let’s have a closer look at request and response components.

REST Request Components

There are mainly four components: -

* **Method** – It refers to the type of operation (CRUD).
* **URI** – Sounds Familiar ;) It is used to identify the resource.
* **Headers** – It is optional and is used to provide additional information about the resource. For example, CONTENT-TYPE header refers to the type of the resource.
* **Body** – It basically refers to the object being passed in POST and PUT methods. There is not body in GET and DELETE methods.

It will get clearer once we get into detail of each method :)

**REST Response Components**

* **Status Code** – It is used to represent the result of the operation we requested to be performed.
* **Header** – It provides additional information about the response, just like the header component of the Request.
* **Body** – It is the response we receive. It Usually in JSON format as we have already seen. You can have a look at that response again,

{

"status": "OK",

"result": [

{

"contribution": 0,

"lastOnlineTimeSeconds": 1712374124,

"rating": 1107,

"friendOfCount": 3,

"titlePhoto": "https://userpic.codeforces.org/no-title.jpg",

"rank": "newbie",

"handle": "aryam\_agarwal",

"maxRating": 1158,

"avatar": "https://userpic.codeforces.org/no-avatar.jpg",

"registrationTimeSeconds": 1634376349,

"maxRank": "newbie"

}

]

}

Let’s add some more HTTP Status Codes to our database,

* **200/OK** – Request was completed successfully
* **201/Created** – Resource was created successfully
* **400/Bad Request** – Body of request was invalid
* **403/Forbidden** – Caller does not have permissions for the requested resource
* **404/Not Found** – Requested resource could not be found
* **500/Internal Server Error** – Server cannot fulfill request and does not want to expose specifics to client
* **501/Not Implemented** – Requested method is not currently supported

Kudos for making till here :) Now **rest** is a cakewalk. Sit tight as we take deep dive into the HTTP Methods.

REST HTTP Methods – POST

As we already know it will create a new resource. Let’s take example where we create a new user in our database.

Take a look at the **Request -**

• **URI** specifies the resource to be created

http://localhost:8080/user

• **Header** tells the REST API the format of the Body

Content-Type: application/json

• **Body** is a representation of the user object

{

"firstName": "Aryam",

"lastName": "Agarwal",

"position": "coordinator",

"club": "Axios"

}

***You can see that body contains no field to uniquely identify the user in the database. The unique identifier of a resource should be created and managed by the REST API service.*** If you thought this, great! Give a pat on your back.

**Response** –

* **Status code** – 201 (created)
* **Body** – information received in JSON Format

{

"id": "101",

"firstName": "Aryam",

"lastName": "Agarwal",

"position": "coordinator",

"club": "Axios"

}

REST HTTP Methods – GET

Used to receive an information or resource.

Take a look at the **Request -**

• **URI** helps to identify the resource

http://localhost:8080/user/101

Generally, GET Requests do not have a body and header.

**Response** –

* **Status code** – 200 (OK)
* **Body** – information received in JSON Format

{

"id": "101",

"firstName": "Aryam",

"lastName": "Agarwal",

"position": "coordinator",

"club": "Axios"

}

REST HTTP Methods – POST

Used to update a resource or create the resource if it does not exist.

Take a look at the **Request -**

• **URI** helps to identify the resource

http://localhost:8080/user/update

• **Header** tells the REST API the format of the Body

Content-Type: application/json

• **Body** is a representation of the user object

{

"id": "101",

"firstName": "Aryam",

"lastName": "Agarwal",

"position": "future-coordinator",

"club": "Axios"

}

**Response** –

* **Status code** – 200 (OK)
* **Body** – information received in JSON Format

{

"id": "101",

"firstName": "Aryam",

"lastName": "Agarwal",

"position": "future-coordinator",

"club": "Axios"

}

REST HTTP Methods - DELETE

Used to delete an existing resource through unique id of the resource.

Take a look at the **Request -**

• **URI** helps to identify the resource

http://localhost:8080/user/100

Header and Body are not applicable for DELETE Requests.

**Response** –

* **Status code** – 404 (NOT FOUND)

Body is not generally applicable for DELETE Response.

**Conclusion**

This blog demonstrates and explains the fundamental operations and patterns that underpin most RESTful services. In real-life scenarios you need to consider many more things. Finally, you are ready to embark on your journey of creating RESTFul Applications :)