Characteristics of REST APIs

Abhishek Sur

December 26, 2019

API is one of the primary requirement for any application. People want to expose data to the external world ensuring all the data to be consumed easily and quickly. With the proliferation of AI and Machine learning, the data created over time becomes more relevant and important to many other vendors. As an integration platform, our main focus is also on how to consume these important data points to get meaningful insights. Just like all other integration platform, our platform also supports a number of standard protocol which can easily connect to various APIs without writing a single line of code. REST is also supported in-built on the platform to ensure any third party applications supporting REST based communication protocol can be easily connected.

To comply with standards for an API, software vendor build REST based APIs to easily connect to a wide variety of software tools available online. Some of the free tools like Swagger, PostMan, Rest-Assured, RESTClient etc. allows developers or API consumers to quickly and easily test a REST API without writing a single line of code. To comply the usage of these APIs, one must ensure the standards are followed and Request / Response structure provides meaningful insights.

Introduction

REST stands for Representational State Transfer. Web application generating data over time requires an endpoint which could easily consumed in real time by different set of applications such that the data generated by these application can be made meaningful to other applications. Integration allows the users to generate services which allows seamless interaction between different set of applications to and fro. REST ensures a standard mechanism to ensure smooth transfer of data without affecting or minimally affecting the applications.

Why REST?

REST provides standard protocol to make data transfer between applications easy and effective. Think of traditional web services, where you expose interfaces as methods. Clients who are consuming them, requires to know the method names which you intent to provide. For this reason, documentation of an APIs were utmost important. SOAP APIs are unnecessarily complex and requires indepth understanding of WSDL (a completely new language) to understand the request and response structure. These APIs are not readable and function oriented. But as per present age, where objects and classes are in place, why do we still go on use traditional approach. REST exposes objects as per their data. For instance, to get Customers, Action GET is used as method and the object Customer is used in URL. This makes the APIs so much simple to understand and uses all the features which HTTP 1.1 protocol provides. REST is very easily understandable and does not require any special knowledge to understand a language.

In addition to that, REST is also preferred because of large number of tooling available on the market which easily consumes a REST API. As REST being one of the standard way of communicating data, any application can easily consume it.

Various Elements of REST API

As REST is a standard, there are already standard protocol defined for it to execute. There are various components of a REST API which one needs to follow while building an API. Let us discuss some of the existing elements of a REST API which one needs to follow while developing an API.

Navigation

Documentation of a REST API is the core of any API. It is important to understand that every data exposed to a REST API should be self explanatory and any developer who is consuming it can use one similar knowledgebase to understand and update API. Let us take an example of how APIs are designed.

https://restclient.appseconnectapi.com/Customers
https://restclient.appseconnectapi.com/Customers?filter=department[eq]sales&limit=20&offset=40
https://restclient.appseconnectapi.com/customer/10/details

If you look at the first 3 urls, you can easily understand how the different types of REST navigation url exposes. The first URL provides a list of Customers. You can use the similar knowledge to list products, organizations, comments or any data. You just need to replace the noun in the url here, which is Customer.

The second URL provides additional information like filter structure, paging etc. Paging is maintained using limit and offset query string properties. The filter is also maintained using standard filter query string key. The same structure works just similarly to the other objects.

The third URL retrieves detailed information of a particular customer. Check the url is not pluralized here and the id is replaced as a part of the URL. The value 10 represents the id of the customer for which all the details are returned.

Embracing statelessness in the core of an API is utmost important. Web is stateless as we all know, but how does API provide the same feeling to the consumer ? With REST, navigation provides complete flow of data and every url that exposes a set of information does not provide any state information.

Methods

While navigation represents the endpoints, the methods represent the actions. For each URL, it is important to understand what kind of action the user wants to perform. For instance, in all the above URLs, the user only want to get data from the server. In such cases, the method will always be GET. Here is the list of all methods supported for any REST API.

GET – This is an HTTP method used to get data from server.
POST – This is used to send or push data to the server. Hence, if the user need to update data on server, it is POST which is required.
PUT – This is to update an existing information in the server.
DELETE – As the action suggests, this method will allow deleting an existing data from the server.
These are the main methods that a REST API use for CRUD operations. There are few other methods too, which one can use to infer special operations. Some of the methods like :

HEAD – Returns the header information about a resource. It is used to identify what formats are accepted for a particular API resource.
PATCH – This is used to partially update data. In case of PUT, the whole data is updated whereas in PATCH, only the properties that are passed were updated.
OPTIONS – They do not have any side effect, and this will return which HTTP methods are available for the resource. For instance, if only GET and POST method can be applied, the Allow header will return these.
TRACE – TRACE is used for diagnostics purpose.
The methods GET, HEAD, PUT And delete is regarded as idempotent methods as these operations will produce same result if they are called multiple times.

Headers

Just like methods, REST also uses headers extensively to ensure all the information about a resource is returned as response. For instance, the Content-length returns the number of characters that are returned by the API to the user such that they can have a check on the actual length of the body before downloading the content. The headers of a REST API represents the metadata information associated with an API or a resource.
The headers of a REST resource is divided into Request and Response headers, where Request is the metadata which is sent to the server for a resource and response is the data generated by the server as response.

Request Headers

Accept : Specifies the content types that are valid as response to the requester. If the server cannot respond, it will return 406 Not acceptable status code.
Content-Type : Specifies the content type or the format of the body element.
If-Match : This header will indicate the entity version before updating it to the server. The entity version is sent as ETag.

If-None-Match : This header is used to check whether the server have modified a particular resource. If ETag is not matched, the server responds with the new update or it returns 304 – Not modified.

Response Headers

Allow : Lists the allowed Http Methods for a particular request.
Content-Type : Represents the type of the content returned as response.
ETag : An alphanumeric string representing the version of a resource.
Status : Http status code.

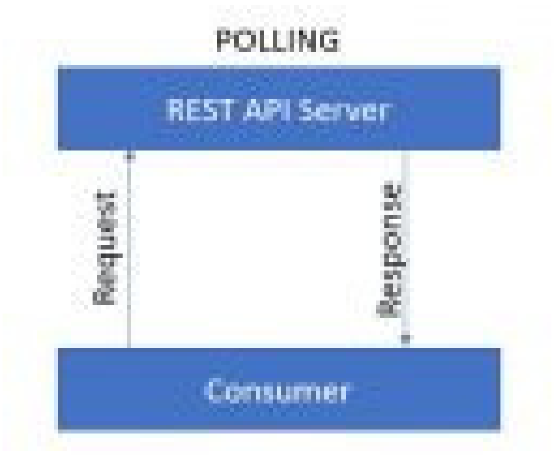
In addition to them, there are a number of other important headers, which are used in special scenarios.

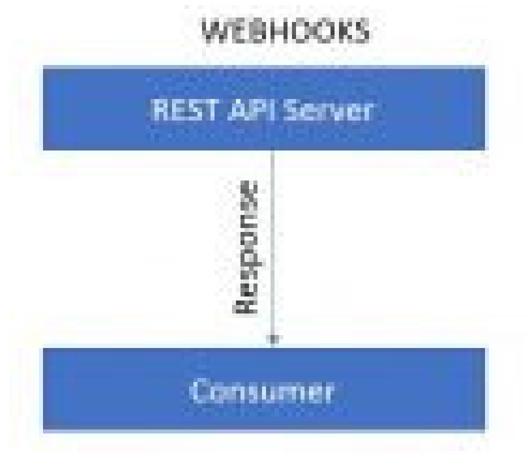
Eventing

There are two types of APIs available.

1. Polling
2. Webhooks

Generally API means the Polling API where the updates are only retrieved when the consumer initiates a new request. All the APIs discussed here are polling APIs. But in some scenarios, when we truly require API to be real-time, we take a different approach where the API is configured with a callback url, which is automatically triggered when an update event occurs on the server.





In this diagram, it is depicted that the polling requires request to be triggered to return a response. In case of Webhooks the callback url is already configured and the consumer receives the callback when update occurs on the server from time to time.

Error Codes

As any standard API requires active solution to errors, REST standard also provides such support. Here is the list of all error codes that are more prominent and usually returned when there is an exception on the server.

1XX – Informational

100 – Continue

101 – Switching protocol

102 – Processing

103 – Early hints

2XX – Success

200 – OK

201 – Created (For Post operation)

202 – Accepted

204 – No Content

206 – Partial Content

3XX – Redirection

301 – Moved Permanently

302 – Found

304 – Not modified

308 – Permanent Redirect

4XX – Client Error

400 – Bad request

401 – Unauthorized

403 – Forbidden

404 – Not Found

405 – Method not allowed

406 – Not Acceptable

429 – Too many request (Used to indicate server throttling)

5XX – Server Error

500 – Server error

501 – Not implemented

502 – Bad gateway

503 – Service unavailable

Authentication

Similar to every area of REST, Authentication is also standardized. Authentication represents the accessibility for a particular resource such that only right person having right permission can access a resource. Authentication at its core has the ability to identify the caller such that it can log a request client id from the server for auditing. There are a number of authentication mechanism, some being very simple, while some are extensive with multiple request response structure involved. Some of the widely known authentication mechanisms are :

Basic

HTTP basic authentication is most straight forward method and the easiest. With this method the sender places username:password in the request header. The basic authentication directly parses the base64 string based userid and password combination to authenticate every request. This method does not rely on any security headers like cookies, session ids etc, and also does not maintain any state for the customer login.
Example, Authorization: Basic bfYl39syZQGVOn==

Bearer

Bearer type of authentication uses bearer token which is passed to a customer to call a resource. The bearer token is a security token generated by the user and used for every call he make on the server. Bearer token is generated behind a valid login by the user.

OAuth

Open authentication is currently most secure authentication mechanism that requires the user to generate a token by logging in using user id and password, and then reusing the refresh token for every session failure. This type of authentication allows establishing scopes which provide different parts of an API server and hence can easily revoked permission to a resource after a certain timespan.

Bulk APIs

In case of applications requiring bulk operation or send out bulk data to and from a source, bulk API comes very handy. REST supports bulk APIs which can be used to programmatically send out large number of transaction requests to the server. As web is stateless, it has latency when calling the same API multiple times. Think about a scenario where you want to update 100 records at a time. In case of normal API, you might need to do 100 number of connects, then call the API and then disconnect. This additional connecting and disconnecting requires lot of time for the consumer.

Using the Bulk APIs, one can easily send all the data at a time and server returns 202 – Accepted status. The server in turn uses batch processing job to do the operation on each of the data passed.

Best practices of developing REST APIs

Now as you already know the basics of REST API, let us consider developing some of the best practices that one should follow while developing a REST API. Here are some of the points :

1. **Avoid using Verbs in URLs.** – When defining a restful service, avoid using verb for the resource inside the URL such that we dont confuse the operation with a resource.
   Generally the Http method is sufficient to do such a thing.
   # Don't
   POST: /articles/createNewBlog/
   # Do
   POST: /articles/
2. **Always use plural form of noun** such that all URLs are consistent. You could use singular form to get a particular resource, but in such a case the base url will be invalid.
   GET: /blogs/2/
   POST: /blogs/
3. **Avoid returning plain text.** Even though REST supports returning plain text, but it is not recommended approach. One should always return data using valid format such that tools can easily parse the texts. JSON or XML is the standard way of defining a response.
4. **Return error details in Body.** Even though in most of the cases, error can be identified by the status code, but the details are sometimes necessary. To define error messages, you must return the data in body such that any structure of data could be managed.
5. **Use Status codes consistently.** Status code are something which returns the response status of a resource. If you are using 201-created for one POST operation, you should always use the same for every resource creation.
6. **Make use of query strings for Pagination and Filtering.** All the filtering criteria for REST should use filtering through query string such that one can bookmark the link on the web browser. RESTful APIs are stateless, and hence it is easy to identify the state information if the query string of the url refers to it.

Conclusion

As you understood all the basic information about REST, it is important to consider how these are consumed. I see a large number of badly behaving REST APIs in the market which still works but not much representable. Designing a good REST API requires lot of architectural work to plan and execute a plan. Even though writing a REST API from scratch is sometimes useful, it is also good idea to take help of some of the popular platforms present in the market which could help in developing rock solid REST APIs. I hope this topic becomes helpful to all of you. Thank you for reading.