**REST API — the modern way of web-based architecture and design -** [Gaveen Nayanajith](https://medium.com/@gaveen0513?source=post_page-----de4335062e61--------------------------------) Jan 18, 2022

First of all, we can take a look at what does the acronym “REST” is used for. The REST stands for “REpresentational State Transfer” which is an architecting and designer method for the development of web-based technologies.

The acronym “API” stands for “Application Programming Interface” which is used to manage services for a certain software with other software as well as the hardware components.

So the “REST API” is an Application Programming Interface that will adhere to the REpresentational State Transfer mechanism and Constraints.

**Six Constraints of the REST Architecture**

1. Client-Server Architecture — This constraint divide up the responsibilities for the program into two. The client manages the responsibilities related to the user interface/frontend side while the Server manages the responsibilities related to the backend/database side for the certain software. So with this, we can build a centralized REST server which we can use to employ the same logic and databases among a few various interfaces/frontends which will be built upon a few various technologies and frameworks. So we can completely separate the content from how they are going to be presented.
2. Statelessness — So when we produce a REST request no client context/information will not be stored in the Server. Instead of that, the request for the Server with relevance to the state will be handled by the Client all by itself. But sometimes some of the “states” are used in the authentication requests and responses, so for that, the server stores the state for a short period of time and after the token mediation with the database, it clears the state. (When we log out we clear out the state of the authentication REST request)
3. Cacheability — With the cacheability the software allows the REST server to issue a response to the Client that can be held for a short period of time. With that, all the web-app processes involving recurring REST requests will be optimized performance-wise. So for this, all the REST requests must be clearly marked as cacheable or not. The cacheability must be blocked for the highly dynamic, constantly changing responses.
4. Layered System — With this constraint, the REST cut off the Client from the knowledge of the type of connection. So that the Client is not aware of whether it is connected to the server via a direct mode or through an intermediary like CDN(Content Delivery System). With this, the security aspects for the web app climbs up.
5. Code on demand — With this REST allows the servers to transfer executable code snippets as JS scripts and compiled components for the REST clients to execute. These codes will customize and extend some functionalities of the software.
6. Uniform Interface — This constraint has 4 different sub constraints. The first one is, when a URI request is made, it must specify what resource it is requesting along with the required format. The second one specifies that when a client has a copy of the resources that have been sent by the server, the client can modify or delete the resource. The third one is all the requests and responses must have a descriptive message on them. The last sub constraint specifies that once the client has access to the REST service it access all the resources and methods through the hyperlinks provided.

With all these constraints fulfilled within an API which sends the data through an HTTP service, it is known as a RESTful API.

**REST Requests and REST Responses**

**Request**

There is a sort of uniform and universal way for writing the REST Request.

***Method + URI***

The method is one of the REST methods that are being used, and the URI means the URI of the resources we want to access.

The methods we used in the REST request are,

* GET — Get the resource content
* POST — Add new data/content to the resource
* PUT — Replace current data/content in the resource
* PATCH — Edit the current data/content in the resource
* DELETE — Delete the data/content in the resource
* OPTIONS — GET all the information/options on all the resources
* HEAD — GET just the response headers from the resource

Along with these main 2 components, we can send metadata such as Authorization details, content type, cache control etc. And when we are using the methods that we need to provide any data for the request, all that data is also passed along with the request. If you are unable to find documentation for a REST API, you can always ask from REST API itself. By the “OPTION” method we can discover the methods that we can use with their corresponding resources and their specific metadata.

**REST Response**

For every REST request that is been made, a specific REST Response is created and sent, most of the time it consists of a header along with the requested content. A response always carries a header with it. This header specifies the result and content for the given request and response pair.

Usually, for the REST response headers they give an HTTP status message, the codes types for the HTTP messages are specified as below.

* 1xx — Information
* 2xx — Success
* 3xx — Redirection
* 4xx — Client error
* 5xx — Server error

**How to use the REST API and Who uses it?**

In a real-world scenario, we are not actually using the REST APIs. The REST API is accessed through a web app or an application that uses that certain software. So this component is the “CLIENT” we are referring to in the REST architecture. So as humans we just control those clients rather than being the clients. Hence we are not the clients we don’t get responses in a readable manner to us rather than a response we have to illustrate through a human friendly UI. So to monitor the way a REST API is working we have some tools such as POSTMAN, with these tools we get a great insight on how these APIs are working in relation to the requests that are being sent the responses that is been generated. The next chapter provide a high level peek onto a REST API demo.

To give you an idea on how the RESTful services work I am going to use this sandbox [API service](https://reqres.in/). Here we can see the full documentation on how to use this particular service.

**Retrieve the users in the browser from reqres.in**

As this is a RESTful API, we can call the get HTTP method to get the API response:

[*https://reqres.in/api/users*](https://reqres.in/api/users)

Type this on the browser and we can have an unparsed body for the content.

As mentioned above, to get a comprehensive look at this API responses we have to parse it through a human friendly presentation layer application such as [POSTMAN](https://www.postman.com/downloads/) or any other REST client or we could use REST client extension for the code editors as well.

In POSTMAN we can send out both GET and POST requests and observe the responses from each.

Graphical user interface, text, application, email

Description automatically generated

GET Method

Graphical user interface, text, application, email

Description automatically generated

POST Method

So with this information, you can get a comprehensive idea of how the REST APIs work. If we take a closer look we can see that the WWW is also a sort of giant RESTful application. So let’s touch on those little alleys along with the huge web and get to know the directions for the world of REST.