What is REST — A Simple Explanation for Beginners, Part 2: REST Constraints

Shif Ben Avraham Follow
Sep 5, 2017 · 4 min read

This is part 2 of 2 articles explaining the basic concepts of REST. This part explains the 6 REST constraints. Read part 1 here.

In order for an API to be RESTful, it has to adhere to 6 constraints:

- Uniform interface
- Client — server separation
- Stateless
- Layered system
- Cacheable
- Code-on-demand

Uniform interface

This constraint has 4 parts:

1. The request to the server has to include a resource identifier
2. The response the server returns include enough information so the client can modify the resource
3. Each request to the API contains all the information the server needs to perform the request, and each response the server returns contain all the information the client

needs in order to understand the response.

4. Hypermedia as the engine of application state — this may sound a bit cryptic, so let's break it down: by application we mean the web application that the server is running. By hypermedia we refer to the hyperlinks, or simply links, that the server can include in the response. The whole sentence means that the server can inform the client , in a response, of the ways to change the state of the web application. If the client asked for a specific user, the server can provide not only the state of that user but also information about how to change the state of the user, for example how to update the user's name or how to delete the user. It is easy to think about the way it's done by thinking about a server returning a response in HTML format to a browser (which is the client). The HTML will include tags with links (this is the hypermedia part) to another web page where the user can be updated (for example a link to a 'profile settings' page). To put all of this in perspective, most web **pages** do implement hypermedia as the engine of application state, but the most common web **APIs** do not adhere to this constraint. To further understand this concept, I highly recommend watching this 30 minutes YouTube video.

The result of the uniform interface is that requests from different clients look the same, whether the client is a chrome browser, a linux server, a python script, an android app or anything else.

Client — server separation

The client and the server act independently, each on its own, and the interaction between them is only in the form of requests, initiated by the client only, and responses, which the server send to the client only as a reaction to a request. The server just sits there waiting for requests from the client to come. The server doesn't start sending away information about the state of some resources on its own.

Stateless

Stateless means the server does not remember anything about the user who uses the API. It doesn't remember if the user of the API already sent a GET request for the same resource in the past, it doesn't remember which resources the user of the API requested before, and so on.

Each individual request contains all the information the server needs to perform the request and return a response, regardless of other requests made by the same API user.

Layered system

Between the client who requests a representation of a resource's state, and the server who sends the response back, there might be a number of servers in the middle. These servers might provide a security layer, a caching layer, a load-balancing layer, or other functionality. Those layers should not affect the request or the response. The client is agnostic as to how many layers, if any, there are between the client and the actual server responding to the request.

Cacheable

This means that the data the server sends contain information about whether or not the data is cacheable. If the data is cacheable, it might contain some sort of a version number. The version number is what makes caching possible: since the client knows which version of the data it already has (from a previous response), the client can avoid requesting the same data again and again. The client should also know if the current version of the data is expired, in which case the client will know it should send another request to the server to get the most updated data about the state of a resource.

Code-on-demand

This constraint is optional — an API can be RESTful even without providing code on demand.

The client can request code from the server, and then the response from the server will contain some code, usually in the form of a script, when the response is in HTML format. The client then can execute that code.

API Rest Rest Api Tech Web Development


Download on the
App Store
GET IT ON
Google Play