**Learn REST**

What is REST?
**REST Constraints**
REST Resource Naming Guide

**Guides**

Caching
Compression
Content Negotiation
HATEOAS
Idempotence
Security Essentials
Versioning
Statelessness

**Tech – How To**

REST API Design Tutorial
Create REST APIs with JAX-RS 2.0

**FAQs**

PUT vs POST
N+1 Problem
'q' Parameter

**Resources**

Comparing SOAP vs REST APIs
HTTP Methods
Richardson Maturity Model
HTTP Response Codes
- 200 (OK)
- 201 (Created)
- 202 (Accepted)
- 204 (No Content)

# REST Architectural Constraints

REST stands for **Re**presentational **S**tate **T**ransfer, a term coined by Roy Fielding in 2000. It is an **architecture style** for designing loosely coupled applications over HTTP, that is often used in the development of web services. REST does not enforce any rule regarding how it should be implemented at lower level, it just put high level design guidelines and leave you to think of your own implementation.

In my last employment, I designed RESTful APIs for telecom major company for 2 good years. In this post, I will be sharing my thoughts apart from normal design practices. You may not agree with me on a few points, and that's perfectly OK. I will be happy to discuss anything from you with an open mind.

Let's start with standard design specific stuff to clear what 'Roy Fielding' wants us to build. Then we will discuss my stuff which will be more towards finer points while you design your RESTful APIs.

## Architectural Constraints

REST defines **6 architectural constraints** which make any web service – a true RESTful API.

1. Uniform interface

2. Client–server
3. Stateless
4. Cacheable
5. Layered system
6. Code on demand (optional)

## Uniform interface

As the constraint name itself applies, you MUST decide APIs interface for resources inside the system which are exposed to API consumers and follow religiously. A resource in the system should have only one logical URI and that should provide a way to fetch related or additional data. It's always better to **synonymise a resource with a web page**.

Any single resource should not be too large and contain each and everything in its representation. Whenever relevant, a resource should contain **links (HATEOAS) pointing to relative URIs** to fetch related information.

Also, the resource representations across system should follow certain guidelines such as naming conventions, link formats or data format (xml or/and json).

All resources should be accessible through a common approach such as HTTP GET and similarly modified using a consistent approach.

> *Once a developer becomes familiar with one of your API, he should be able to follow the similar approach for other APIs.*

## Client–server

This essentially means that client application and server application MUST be able to evolve separately without any dependency on each other. A client should know only

resource URIs and that's all. Today, this is normal practice in web development so nothing fancy is required from your side. Keep it simple.

> *Servers and clients may also be replaced and developed independently, as long as the interface between them is not altered.*

## Stateless

Roy fielding got inspiration from HTTP, so it reflects in this constraint. Make all client-server interaction stateless. Server will not store anything about latest HTTP request client made. It will treat each and every request as new. No session, no history.

If client application needs to be a stateful application for the end user, where user logs in once and do other authorized operations thereafter, then each request from the client should contain all the information necessary to service the request – including authentication and authorization details.

> *No client context shall be stored on the server between requests. The client is responsible for managing the state of the application.*

## Cacheable

In today's world, caching of data and responses is of utmost important wherever they are applicable/possible. The webpage you are reading here is also a cached version of the HTML page. Caching brings performance improvement for client side, and better scope for scalability for a server because the load has reduced.

In REST, caching shall be applied to resources when applicable and then these resources MUST declare themselves cacheable. Caching can be implemented on the server or client side.

> *Well-managed caching partially or completely eliminates some client-server interactions, further improving scalability and performance.*

## Layered system

REST allows you to use a layered system architecture where you deploy the APIs on server A, and store data on server B and authenticate requests in Server C, for example. A client

cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way.

### Code on demand (optional)

Well, this constraint is optional. Most of the time you will be sending the static representations of resources in form of XML or JSON. But when you need to, you are free to `return executable code` to support a part of your application e.g. clients may call your API to get a UI widget rendering code. It is permitted.

> *All above constraints help you build a truly RESTful API and you should follow them. Still, at times you may find yourself violating one or two constraints. Do not worry, you are still making a RESTful API – but not "truly RESTful".*

Notice that all the above constraints are most closely related to WWW (the web). Using RESTful APIs, you can do the same thing with your web services what you do to web pages.

## Comments

Paul says

July 30, 2019 at 6:16 am

"All above constraints help you build a truly RESTful API and you should follow them. Still, at times you may find yourself violating one or two constraints. Do not worry, you are still making a RESTful API – but not "truly RESTful"

This is completely not true. The only constraint which is optional is code on demand all the other constraints

are required otherwise it is not REST.

"Notice that all the above constraints are most closely related to WWW (the web)."
This is misleading. REST is describing the web. Before REST the architecture of the web was more or less non-existent. REST was designed to map closely to what was already there BUT enable future improvement as long as it didn't break any of the constraints of REST.

Reply

> Admin says
>
> July 31, 2019 at 5:20 am
>
> Thanks for the comment. I prefer to call them "not true REST API". And most APIs built in public/private spaces are built like this. How term you suggest for them?
>
> Roy was one of the principal authors of the HTTP specification (current web standard). Fielding's doctoral dissertation, describes Representational State Transfer (REST) as a key architectural principle of the World Wide Web. [Link]
>
> Reply

Volpe Gentile says

August 20, 2019 at 3:52 pm

Sorry, but imho Paul is right. How about "not truly (or 'not true') (a) car"? It's close to a car, though it has two wheels? Or it is a REST or it is not REST. "Not truly" does not tell us anything useful – we can't even quantify how close to REST.
Which term? The only term, when talking about REST compliance, can be "not RESTful".

As for the Web vs Http vs Rest, I am going to take a rest from totalizing REST.

As for the rest, I enjoyed learning from your article. Thanks.

Reply

Admin says

August 20, 2019 at 5:03 pm

I like the car example. ☺ I cannot disagree because there is no official term for APIs – not following all 6 constraints.

Reply

buddhika says

June 5, 2019 at 7:05 am

This made a very useful foundation for me to start exploring web services. Thank you so much.

Reply

Abdul says

November 23, 2018 at 9:20 am

Its good article on REST, could you explain all phases in detail one client request to REST endpoint.

Thanks in advance

Reply

Sivan says

October 9, 2018 at 9:24 am

Thanks for a such precise explanation. A quick read with a a very quick win. A must read for Api developers.

Reply

Stuart Wilson says

September 24, 2018 at 5:34 am

As I'm new to API's this really gave me the explanation I needed to the meaning of REST. Thank you.

Reply

Ravi says

July 31, 2018 at 3:58 pm

RESTApi is over http, it could accessed by browser. How to restrict access to users? How authentication happens and how authenticated information is passed over API for subsequent calls.

Reply

Eric Johnson says

February 22, 2019 at 7:02 pm

@Ravi I have found JSON Web Tokens (JWTs) great for authentication/authorization when implementing RESTful API. https://jwt.io/

Reply

Akber Ali says

May 29, 2019 at 2:39 pm

you can use OAuth, API Key (public /private) or other authentication system.

Reply

Pradeep Godse says

July 14, 2018 at 4:47 am

Do we have any restrictions on the size of data posting to api. Is there any limit. Can you please let me know.

Reply

arun says

February 20, 2019 at 12:35 pm

As already mentioned, HTTP itself doesn't impose any hard-coded limit on request length; but browsers have

limits ranging on the 2kb – 8kb (255 bytes if we count very old browsers).

Is there a response error defined that the server can/should return if it receives a GET request exceeds this length?

That's the one nobody has answered.

HTTP 1.1 defines Status Code 414 Request-URI Too Long for the cases where a server-defined limit is reached. You can see further details on RFC 2616.

For the case of client-defined limits, there is no sense on the server returning something, because the server won't receive the request at all.

Browser Address bar document.location
or anchor tag
———————————————

Chrome 32779 >64k
Android 8192 >64k
Firefox >64k >64k
Safari >64k >64k
IE11 2047 5120
Edge 16 2047 10240

Hope this helps.

Reply

SwethaRani Kobbanna says

June 6, 2018 at 1:47 am

Excellent, Excellent Tutorial. Cannot get a better view of RESTful than this. It is simple and easily understandable. Good to start for beginners to make a hit in the RESTful world.

Reply

Prem Kumar says

April 19, 2018 at 6:40 am

Hello Team,

I have a question regarding limitation of Parameters processing between Request and Response.

Can we send 100 or more as request parameters. If yes how does the system perform and what challenges we will be facing in terms of resource consumption. If no, how does the caching help in this regard

Reply

Admin says

April 19, 2018 at 8:19 am

HTTP itself doesn't impose any hard-coded limit on request length; but browsers have limits ranging on the 2kb – 8kb. Along with browsers, there may be limits in proxies and server at backend. So can you send 100 or more as request parameters? Answer is Yes if infrasturture support it.

I don't think there will be any system performance downgrade due to number of paratemeters considering huge processing power of todays hardware. Similarily, there will not be any impact on caching except you may end up having very large amount of cache. Remember, for any parameter change in URL, there will be a new entry in cache. If cache storage fill up, cache will be invalidated more often and there will be more server hits than expected.

Finally, it will be a really bad design. Cheers !!

Reply

Raju says

April 6, 2018 at 2:23 am

In the contraint cachaeble it is written "Caching can be implemented on server or client side". If chaching is done on Server side, does it not break the Stateless contraint?

What I understand from stateless is the server should forget what request came after it sends the response. So if server caches, then it would remember the previous requests.. so it is no more stateless..

Please help me understand. I am trying to understand the concept. I am an SAP ABAP developer who is trying to learn OData which uses RESTful architecture.

Regards,
Raju.
http://www.sapyard.com

Reply

Admin says

April 9, 2018 at 6:29 am

Stateless and chaching are two different concepts. Please do not mix them. Stateless is about maintaining resource's state-transition information on server side and caching is about just return the cached copy (of one particular state of resource) until it expires or resource state is updated making this cache invalid.

Reply

Ankur says

March 15, 2018 at 5:18 am

What exactly is a resource? Can the endpoint url considered as a resource ?

Reply

Admin says

March 18, 2018 at 2:47 pm

Endpoint URLs are referred as URIs. They are not resources. They are links from where latest state of resources can be pulled.

Reply

John says

January 28, 2018 at 8:21 pm

Very nice explanation.
I just have one question about Stateless. The second paragraph of “Stateless” section which is “If client application need to be a stateful application …”

My question, each time the client sends a request, the server will go to the database and check if this user authenticated and authorized?

Thanks in advance.

Reply

Admin says

February 2, 2018 at 11:03 am

Yes. Each request must have authentication information in headers.

Reply

Suresh says

January 10, 2018 at 11:31 am

Hi, a question about ‘cacheable’, is it just about declaring whether is it cacheable or not or supporting the cache on both ends ? This point is little fuzzy in the article apparently. Thanks in advance!

Suresh

Reply

Admin says

January 15, 2018 at 9:28 am

Good question. As API developer, you are responsible for server side caching (if caching is needed). If API response changes very rare (say yearly) e.g. list of holidays in calendar year then you can mark the API as cacheable so client applications can create their own cached version of response and avoid one additional network call, completely.

Reply

Rajan says

September 8, 2017 at 1:11 pm

Nice explanation.

Reply

Shronika Brihan says

August 20, 2017 at 8:04 am

You are doing a Wonderful Job Lokesh !! 😀

Reply

Admin says

August 22, 2017 at 9:08 am

Thanks for the motivation.

Reply

Sundar Patil says

July 7, 2017 at 4:12 pm

Great explanation….
I really understood ground up…

Reply

## Ask Questions & Share Feedback

Please do not submit a comment only to say "Thank you"

Comment

*Want to Post Code Snippets or XML content? Please use [java] ... [/java] tags otherwise code may not appear partially or even fully. e.g.

```
[java]
public static void main (String[] args) {
...
}
[/java]
```

Name *

Email *

Website

POST COMMENT

References

The dissertation by Roy Thomas Fielding
Uniform Resource Identifier (URI, URL, URN) [RFC 3986]
Internet MediaTypes
Web Application Description Language (WADL)

Meta Links

About
Contact Us
Privacy Policy

Blogs

How To Do In Java

This work by RESTfulAPI.net is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.