API DEVELOPMENT < HTTPS://WWW.KENNETHLANGE.COM/CATEGORY/API-DEVELOPMENT/>

Boost Your REST API with HTTP Caching

By Kenneth Lange < https://www.kennethlange.com/author/kenneth-lange/>

February 13, 2016 < https://www.kennethlange.com/boost-your-rest-api-with-http-caching/>

It's a core part of the REST architectural style to use caching!

That's nice, you might think, but why should I use it?

Because it will allow you to show off against other API Designers by claiming that your REST services are twice as RESTful as theirs 😉

But more seriously, Roy Fielding < http://roy.gbiv.com/> , who invented the REST architectural style, didn't add caching as a requirement just for the fun it! He added it because it can seriously boost performance, which is also shown in the numbers in Tom Christie's great post < http://www.dabapps.com/blog/api-performance-profiling-django-rest-framework/> on performance tuning of Django REST services.

So, how do you get started with HTTP caching?

It's only for HTTP GET requests!

At first it may seem like an overwhelming task to implement HTTP Caching; especially if you have already developed a huge number of services.

The good news is that it's only **GET** methods where you need to think about caching as it doesn't really make much sense to cache **POST**, **PUT** or **DELETE** responses.

The even better news is that if you simply specify the right HTTP header then the browser will do all the heavy lifting for you!

Code, please!

That's all very nice! But can you please show us the code?

Definitely, but the only code you need is the **Cache-Control** header in your HTTP response. There are a number of directives in this header you can use to control the caching:

Directive	Description
max-age	The maximum time that the cached response should be used (in seconds). The maximum value is 1 year. Example: `Cache-Control: max-age=3600` Kyle Young < http://www.mobify.com/blog/beginners-guide-to-http-cache-headers/> writes that a rule of thumb is to use between 60 seconds and 1 hour for most content, but for pseudo dynamic content, use less than 60 seconds (or don't cache it at all).

Directive	Description
s-max-age	This directive overrides **max-age** for shared cache, such as proxy servers. You usually have more control over the proxy cache than the client's local cache, so you can add longer values here. Example: `Cache-Control: max-age=0, s-max-age=3600` Thuva Tharma < http://techblog.thescore.com/2014/11/19/are-your-cache-control-directives-doing-what-they-are-supposed-to-do/> has some interesting thoughts on why **s-max-age** may be better than **max-age**.

Directive	Description

public
private
	Is the response specific to the client, so it cannot be used for other clients? For example, **/tasks/myTasks** is client-specific.
	
	If the response is client-specific, use **private**. Otherwise, use **public**, which is also the default.
	
	Examples:
	
	`Cache-Control: private, max-age=3600`
	
	`Cache-Control: public, max-age=3600`

Directive	Description
no-store	This is used for sensitive data (like credit card details) that must not be stored in caches or proxies under any circumstances. Example: `Cache-Control: no-store`
no-cache	The client must not use cached responses. Unless, it first sends a conditional **GET** (with an **ETag**) to the server to check if the data has been updated in the meantime. Example: `Cache-Control: no-cache`

Directive	Description
must-revalidate	If the cached response has expired, it must be revalidated at the server. HTTP might under some circumstances serve cached responses that have expired (for instance, under poor network connectivity), but using this directive ensures that this won't happen. Example: `Cache-Control: max-age=3600, must-revalidate`
proxy-revalidate	Same as **must-revalidate**, but for proxy servers. Example: `Cache-Control: s-max-age=3600, proxy-revalidate`

So let's say that the client sends a request for some metadata, and we want the client to cache it for 1 hour:

```
GET /customers/metadata HTTP/1.1
Host: api.example.com
Accept: application/json
Accept-Language: en
```

To do this, we just add the **Cache-Control** header to our response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: max-age=3600
Content-Length: 88
Etag: "6d82cbb050ddc7fa9cbb659014546e59"

{
  "languageCodes": [
    {"da":"Danish"},
    {"no":"Norwegian"},
    {"en":"English"}
  ]
}
```

As you can see it's pretty easy to add caching to your RESTful services…

So if you have performance issues then HTTP caching could be the power tool you are looking for to seriously reduce your response times!

//

← Avoid Data Corruption in Your REST API with ETags < https://www.kennethlange.com/avoid-data-corruption-in-your-rest-api-with-etags/>

→ What are RESTful Web Services? < https://www.kennethlange.com/what-are-restful-web-services/>

//

/ /

Up ↑