Caching is the ability to **store copies of frequently accessed data** in several places along the request-response path.

When a consumer requests a resource representation, the request goes through a cache or a series of caches (local cache, proxy cache, or reverse proxy) toward the service hosting the resource.

If any of the caches along the **request path** has a fresh copy of the requested representation, it uses that copy to satisfy the request. If none of the caches can satisfy the request, the request travels all the way to the service (or **origin server** as it is formally known).

Using HTTP headers, an origin server indicates whether a response can be cached and, if so, by whom, and for how long. Caches along the **response path** can take a copy of a response, but only if the caching metadata allows them to do so.

Optimizing the network using caching improves the overall quality-of-service in the following ways:

* Reduce bandwidth
* Reduce latency
* Reduce load on servers
* Hide network failures

**2. Caching in REST APIs**

Being [cacheable](https://restfulapi.net/rest-architectural-constraints/#cacheable) is one of the architectural constraints of REST.

* **GET requests** should be cachable by default – until a special condition arises. Usually, browsers treat all GET requests as cacheable.
* **POST requests** are not cacheable by default but can be made cacheable if either an Expires header or a Cache-Control header with a directive, to explicitly allows caching, is added to the response.
* Responses to **PUT** and **DELETE** requests are not cacheable at all.

Please note that [**HTTP dates are always expressed in GMT**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Last-Modified), never in local time.

Below given are main **HTTP response headers** that we can use to control caching behavior:

### **2.1. Expires**

The Expires HTTP header specifies an **absolute expiry time** for a cached representation. Beyond that time, a cached representation is considered **stale** and must be re-validated with the origin server.

To indicate that a representation never expires, a service can include a time up to one year in the future.

**Expires**: Fri, 20 May 2016 19:20:49 GMT

### **2.2. Cache-Control**

The header value comprises one or more comma-separated [directives](https://tools.ietf.org/html/rfc7234#page-24). These directives determine whether a response is cacheable, and if so, by whom, and for how long e.g. max-age or s-maxage directives.

**Cache-Control**: **max-age**=3600

Cacheable responses (whether to a GET or to a POST request) **should also include a validator** — either

an **ETag** or a **Last-Modified** header.

### **2.3. ETag**

An ETag value is an opaque string token that a server associates with a resource to **uniquely identify the state of the resource** over its lifetime.

When the resource changes, the ETag changes accordingly.

**ETag**: "abcd1234567n34jv"

### **2.4. Last-Modified**

Whereas a response’s Date header indicates when the response was generated, the Last-Modified header indicates when the associated resource last changed.

The Last-Modified value cannot be less than Date value.

**Last-Modified**: Fri, 10 May 2016 09:17:49 GMT

Headers `expires` & `last-modified` are a bit buggy if the client and server have **different time zones**.

For example, I request a resource at `/api/v1/articles` and the server responds back with the header `expires: Tue, 30 Nov 2021 00:00:00 GMT.` There would be no problem if the client’s time zone was the same as the server’s but if by any chance they are different the client is going to face an issue. What if the server’s time zone has already passed said time and the client is still using the old response thinking it’s still fresh?

This is the same for `last-modified` header as well.

I believe headers `cache-control` and `e-tag` were introduced to fix this issue since there is no fixed time to cause such problems.

**304 Not Modified**

[304 Not Modified](https://tools.ietf.org/html/rfc7232#section-4.1) is used in response to a conditional GET or HEAD request. A request can be made conditional with one of the following headers:

* If-Match
* If-None-Match
* If-Modified-Since
* If-Unmodified-Since
* If-Range

If-Modified-Since and If-None-Match are used specifically to allow a client to cache results and asks the server to only send a new representation if it has changed.

If-Modified-Since does this based on a Last-Modified header, and If-None-Match with an ETag.

**Example**

A client does an initial request:

GET /foo HTTP/1.1

Accept: text/html

A server responds with an ETag:

HTTP/1.1 200 Ok

Content-Type: text/html

ETag: "some-string"

The next time a client makes a request, it can include the ETag:

GET /foo HTTP/1.1

Accept: text/html

If-None-Match: "some-string"

If the resource didn’t change on the server, it can return a 304.

HTTP/1.1 304 Not Modified

ETag: "some-string"

















