**Web Concepts**

**HTTP Methods**

**Request Methods**: HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred to as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them: e.g. a request method can be safe, idempotent, or cacheable.

**Safe**: an HTTP method is safe if it doesn’t alter the state of the system. In other words, a method is safe if it leads to a read-only operation. Several common HTTP methods are safe: GET, HEAD, or OPTIONS. All safe methods are also idempotent, but not all idempotent methods are safe. For example, PUT and DELETE are both idempotent but unsafe.

Even if safe methods have a read-only semantic, servers can alter their state: e.g. they can log or keep statistics. What is important here is that by calling a safe method, the client doesn't request any server change itself, and therefore won't create an unnecessary load or burden for the server. Browsers can call safe methods without fearing to cause any harm to the server; this allows them to perform activities like pre-fetching without risk. Web crawlers also rely on calling safe methods.

Safe methods don't need to serve static files only; a server can generate an answer to a safe method on-the-fly, as long as the generating script guarantees safety: it should not trigger external effects, like triggering an order in an e-commerce website.

It is the responsibility of the application on the server to implement the safe semantic correctly, the web server itself, being Apache, Nginx or IIS, can't enforce it by itself. In particular, an application should not allow GET requests to alter its state.

**Idempotent:** An HTTP method is idempotent if the intended effect on the server of making a single request is the same as the effect of making several identical requests.

This does not necessarily mean that the request does not have any unique side effects: for example, the server may log every request with the time it was received. Idempotency only applies to effects intended by the client: for example, a POST request intends to send data to the server, or a DELETE request intends to delete a resource on the server.

All safe methods are idempotent, as well as PUT and DELETE. The POST method is not idempotent.

To be idempotent, only the state of the server is considered. The response returned by each request may differ: for example, the first call of a DELETE will likely return a 200, while successive ones will likely return a 404. Another implication of DELETE being idempotent is that developers should not implement RESTful APIs with a delete last entry functionality using the DELETE method.

Note that the idempotence of a method is not guaranteed by the server and some applications may incorrectly break the idempotence constraint.

GET /pageX HTTP/1.1 is idempotent, because it is a safe (read-only) method. Successive calls may return different data to the client, if the data on the server was updated in the meantime.

POST /add\_row HTTP/1.1 is not idempotent; if it is called several times, it adds several rows

**Cacheable:** A cacheable response is an HTTP response that can be cached, that is stored to be retrieved and used later, saving a new request to the server. Not all HTTP responses can be cached; these are the constraints for an HTTP response to be cacheable:

The method used in the request is cacheable, that is either a GET or a HEAD method. A response to a POST or PATCH request can also be cached if freshness is indicated and the Content-Location header is set, but this is rarely implemented. For example, Firefox does not support it (Firefox bug 109553). Other methods, like PUT or DELETE are not cacheable and their result cannot be cached.

The status code of the response is known by the application caching, and is cacheable. The following status codes are cacheable: 200, 203, 204, 206, 300, 301, 404, 405, 410, 414, and 501.

There are no specific headers in the response, like Cache-Control, with values that would prohibit caching.

CORS, Preflight

Local and Session Storage