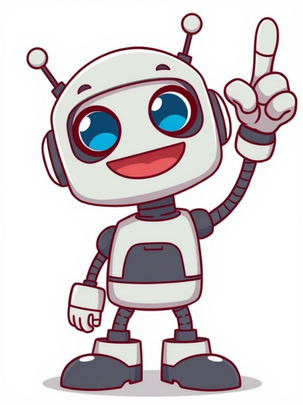
I'm not a bot



By the way, it's worth noting that some mobile devices completely ignore headers like no-cache, no-store, Expires: 0, or whatever else you may try to force them to not re-use expired form pages. This has caused us a lot of headaches as we try to get the issue of a user's iPad say, being left asleep on a page they have reached through a form process, say step 2 of 3, and then the device totally ignores the store/cache directives, and as far as I can tell, simply takes what is a virtual snapshot of the page from its last state, that is, ignoring what it was told explicitly, and, not only that, taking a page that should not be stored, and storing it without actually checking it again, which leads to all kinds of strange Session issues, among other things. I'm just adding this in case someone comes along and can't figure out why they are getting session errors with particularly iPhones and iPads, which seem by far to be the worst offenders in this area. I've done fairly extensive debugger testing with this issue, and this is my conclusion, the devices ignore these directives completely. Even in regular use, I've found that some mobiles also totally fail to check for new versions via say, Expires: 0 then checking last modified dates to determine if it should get a new one. It simply doesn't happen, so what I was forced to do was add query strings to the css/js files I needed to force updates on, which tricks the stupid mobile devices into thinking it's a file it does not have, like: my.css?v=1, then v=2 for a css/js update. User browsers also, by the way, if left to their defaults, as of 2016, as I continuously discover (we do a LOT of changes and updates to our site) also fail to check for last modified dates on such files, but the query string method fixes that issue. This is something I've noticed with clients and office people who tend to use basic normal user defaults on their browsers, and have no awareness of caching issues with css/js etc, almost invariably fail to get the new css/js on change, which means the defaults for their browsers, mostly MSIE / Firefox, are not doing what they are told to do, they ignore changes and ignore last modified dates and do not validate, even with Expires: 0 set explicitly. This was a good thread with a lot of good technical information, but it's also important to note how bad the support for this stuff is in particularly mobile devices. Every few months I have to add more layers of protection against their failure to follow the header commands they receive, or to properly interpret those commands. Most of the information here are correct. Here is a compilation of them and my way of using them. The recommended approach : 1) Force the execution of each step/instruction in the Dockerfile , docker build --no-cache or with docker-compose build : docker-compose build --no-cache . We could also combine that to the up sub-command that recreate all containers: docker-compose build --no-cache && docker-compose up -d --force-recreate Wipe the docker builder cache (if we use Buildkit we very probably need that) , docker builder prune -af If we don't want to use the cache of the parent images, we may try to delete them such as : docker image rm -f fooParentImage . In most of cases, these 3 things are perfectly enough to allow a clean build of our image. So we should try to stick to that. More radical approach : In corner cases where it seems that some objects in the docker cache are still used during the build and that looks repeatable, we should try to understand the cause to be able to wipe the missing part very specifically. If we really don't find a way to rebuild from scratch, there are other ways but it is important to remember that these generally delete much more than it is required. So we should use them with cautious overall when we are not in a local/dev environment. Fetch can take an init object containing many custom settings that you might want to apply to the request, this includes an option called "headers". The "headers" option takes a Header object. This object allows you to configure the headers you want to add to your request. By adding pragma: no-cache and a cache-control: no-cache to your header you will force the browser to check the server to see if the file is different from the file it already has in the cache. You could also use cache-control: no-store as it simply disallows the browser and all intermediate caches to store any version of the returned response. Here is a sample code: var myImage = document.querySelector('img'); var myHeaders = new Headers(); myHeaders.append('pragma', 'no-cache'); myHeaders.append('cache-control', 'no-cache'); var myInit = { method: 'GET', headers: myHeaders, }; var myRequest = new Request('myImage.jpg'); fetch(myRequest, myInit) .then(function(response) { return response.blob(); }) .then(function(response) { var objectURL = URL.createObjectURL(response); myImage.src = objectURL; }); I had problem with caching my css files. Setting headers in PHP didn't help me (perhaps because the headers would need to be set in the stylesheet file instead of the page linking to it?). I found the solution on this page: The solution: Append timestamp as the query part of the URI for the linked file. (Can be used for css, js, images etc.) For development: