Italo Baeza Cabrera Follow

Apr 8, 2019 · 4 min read · ✨ · Listen

Laravel: The hidden SetCacheHeaders Middleware

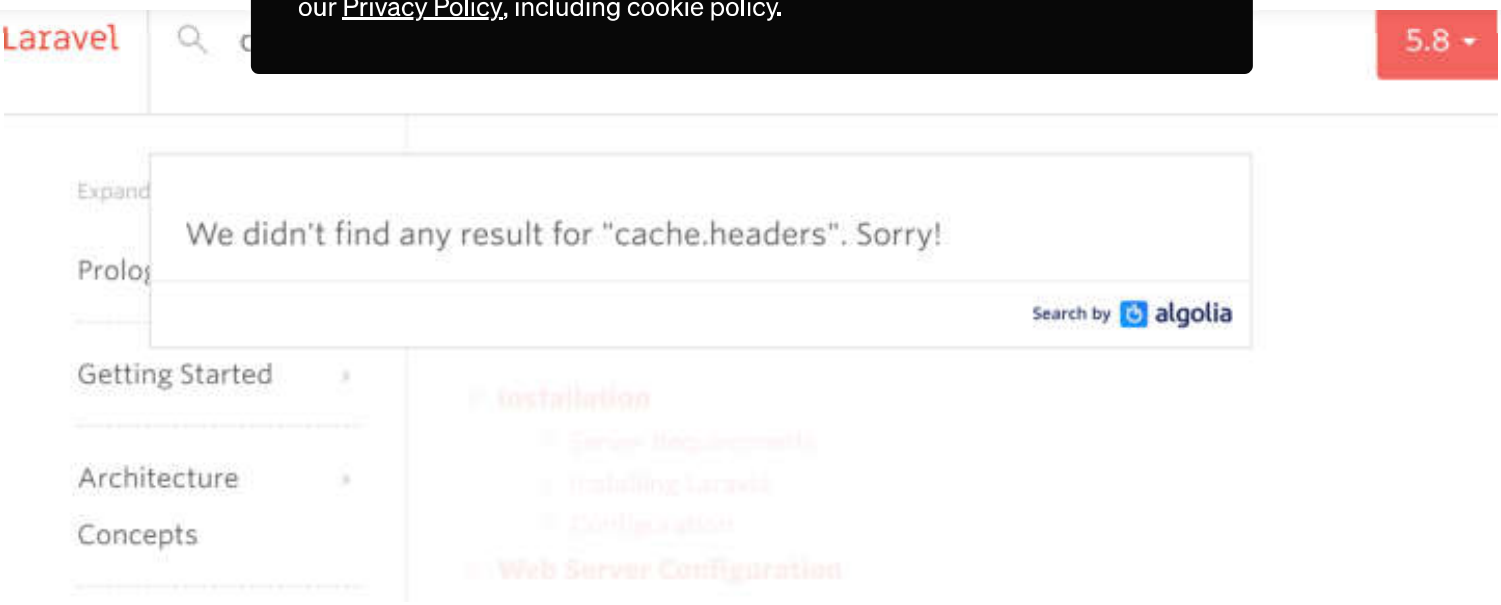
Never try to reinvent the wheel.

```
53      protected $routeMiddleware = [
54          'auth' => \App\Http\Middleware\Authenticate::class,
55          'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
56          'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
57          'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
58          'can' => \Illuminate\Auth\Middleware\Authorize::class,
59          'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
60          'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
61          'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
62          'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
63      ];
```

FOUND YOU!

There are a lot of useful middlewares already registered inside Laravel, like the authentication mechanisms, authorization, a throttler, and even the one responsible to make route model binding possible. Practically, your middleware needs are covered. Except for one.

There is one middleware that doesn't get the spotlight in Laravel. It's called `SetCacheHeaders`, which is aliased as `cache.headers`. And, there is no mention of it in the documentation. I'm not joking, guys:



Who, when, how?

A quick look around in the api documentation and the source repository says this middleware was added added before christmas of 2017 after a lengthy discussion, but it's a mystery why nobody has talked about it since it's seems very... *convenient*. In any case, after making this article, my PR to the documentation was accepted, so you should find it now in the documentation.

A simple glance on the middleware tell us that we can use it to **add cache headers into the Response**, saving us the work to create our own middleware.

Cache in the Response? What do you mean?

If you are asking that, it means you don't know what is the HTTP Cache Control directive.

> *In a nutshell, the Cache Control is a way to tell the browser if the response your application sent should be cached, and under what conditions, so it can decide if it should receive again a full response or not.*

This is a very browser-centric mechanism, you will see later why. There is a very good article in Google Developers that it will teach how it works that you definitely should read. I will wait.

Ready? Now that we know how the *HTTP Cache Control* works, we can understand what this middleware tries to achieve. Or skip the next section for the *useful* part.

```
namespace Il

use Closure;

class SetCacheHeaders
{
    /**
     * Add cache related HTTP headers.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @param  string|array  $options
     * @return \Symfony\Component\HttpFoundation\Response
     *
     * @throws \InvalidArgumentException
     */
    public function handle($request, Closure $next, $options = [])
    {
        $response = $next($request);

        if (! $request->isMethodCacheable() || ! $response->getContent()) {
            return $response;
        }

        if (is_string($options)) {
            $options = $this->parseOptions($options);
        }

        if (isset($options['etag']) && $options['etag'] === true) {
            $options['etag'] = md5($response->getContent());
        }

        $response->setCache($options);
        $response->isNotModified($request);

        return $response;
    }

    /**
     * Parse the given header options.
     *
     * @param  string  $options
     * @return array
     */
    protected function parseOptions($options)
```

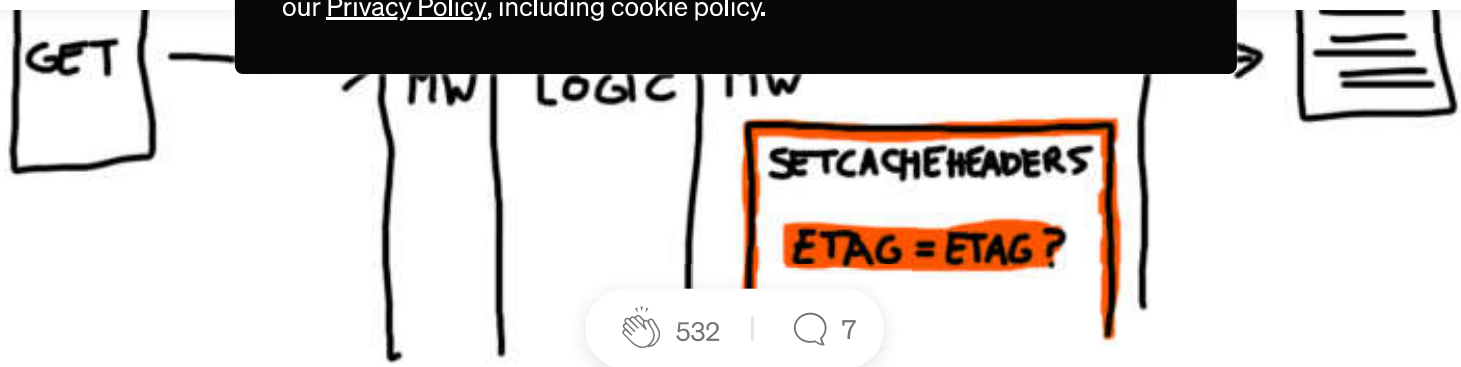

```
53         }
54     }
```

4cd594ba462f-SetCacheHeaders.php hosted with ❤ by GitHub view raw

As you can see, there are five main statements inside the `handle()` method in this middleware. Trying to verbalize what each does:

1. It starts with checking if the request is `GET` or `HEAD`. You can't cache a `POST` response, since this method means *changed* things.
2. Checks if you passed any parameter to the middleware, and parses them into the Response headers using the handy `parseOptions()` method.
3. If the `etag` option has been set, it will automatically hash the response content so it can be quickly compared against the `etag` the Request sent.
4. It will set the `Cache-Control` options inside the Response headers.
5. And finally, it will check if the response has not been modified. If it wasn't, it will remove the content and only return the `etag`, saving precious kilobytes.

The focus is on the third and fifth points. The browser will make a Request to the application with the `etag` of the cached response, if the original response came with that. Your application will receive the `etag`, **process the whole request**, and in the end, hash the content again in the `etag` of the response. By comparing both `etag` the application can know if it should send the whole response again, or only the same `etag` which tells the browser the content hasn't been changed.



Artistic representation of how the etag works in this application.

That emphasis on "process the whole request" is the key part. The application or the browser won't know if the content has changed unless the whole response is ready, meaning: **all of your application logic will run from start to finish anyways**. If you need a more aggressive caching, you could set your eyes on the awesome Spatie's Response Cache.

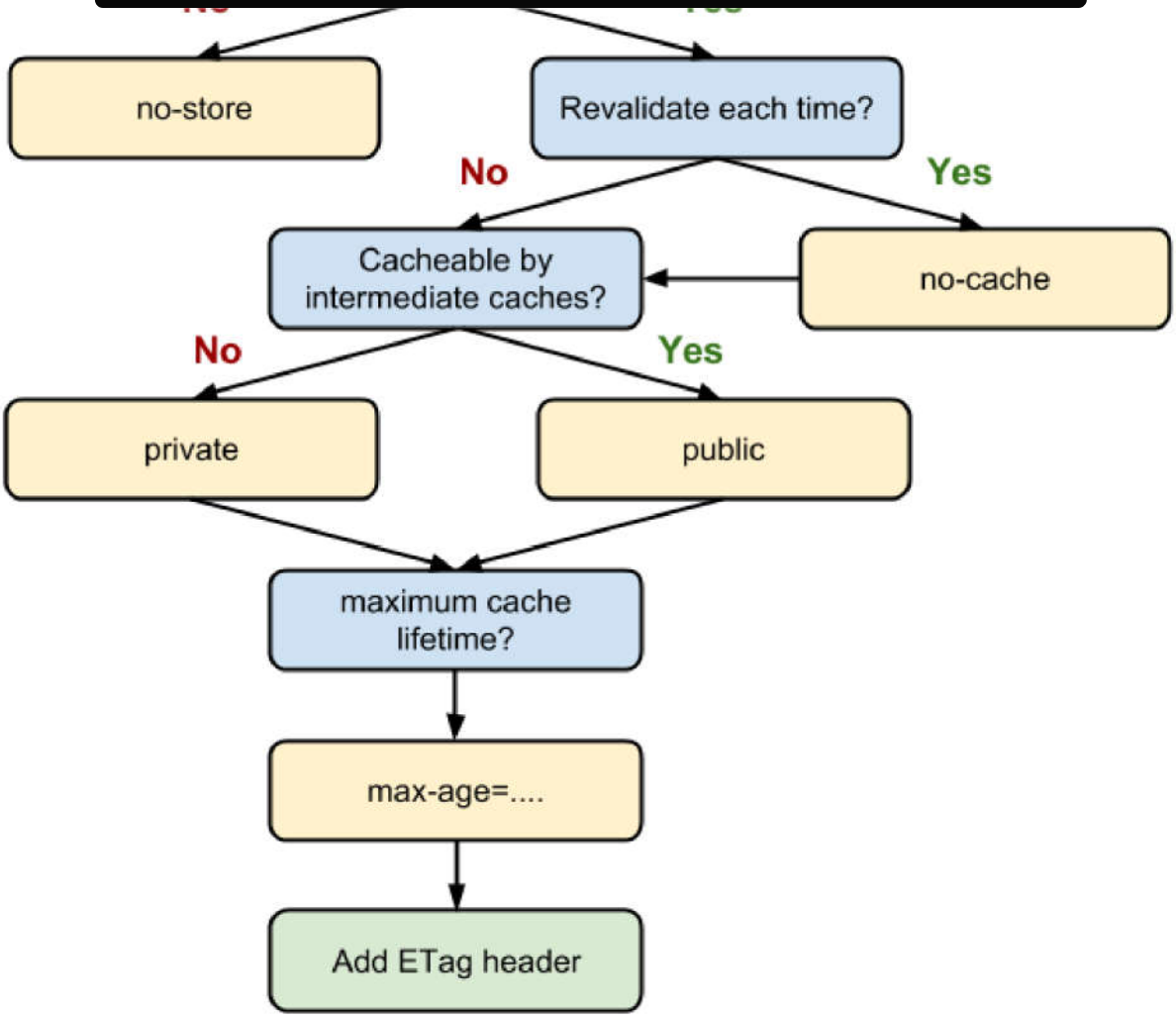
Personally, I consider the Cache Control as a *niche* header. If I would want to implement this technique, it's better to cache the response in-server and then add the `etag` to save time on both ends.

Using the SetCacheHeaders middleware

Now that we know what this middleware does, it's easy to just use it. Let's use a simple real-world example, like my eternal Podcast application.

> *I'm my Podcast application, the Home page has a large list of latest published podcasts, which varies depending on the user authenticated and subscribed podcasts. My metrics says that there is, at least, 5 minutes between each podcast publication.*

Let's see Google's flow chart about what cache-control policy to use:



Google Developers — Google (2019)

Okay, we get the idea:

- `no-cache` and `etag` will allow the user to refresh the page (even before the expiration time) without downloading the whole page if the latest published podcasts are still the same.
- `private` says the homepage is per-user, and should be cached only in the user device rather than, for example, a proxy.
- `max-age=300` will set an expiration time of 5 minutes, since after that time surely there is a new Podcast published.

```
Route::get('/', 'PodcastController@index')
    ->middleware('cache.headers:no-cache,private,max-age=300;etag');
```

Enjoy the read? Reward the writer.[Beta]

Your tip will go to Italo Baeza Cabrera through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Get an email whenever Italo posts something interesting

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Subscribe