2024-01-25 haveaniesday.com

Methodology

I conducted this assessment by inspecting the website on https://haveaniesday.netlify.app/ domain[1] using Chrome DevTools. I was unable to assess the website on its original https://www.haveaniesday.com/ domain because it was inaccessible at the time of this assessment was written.[2]

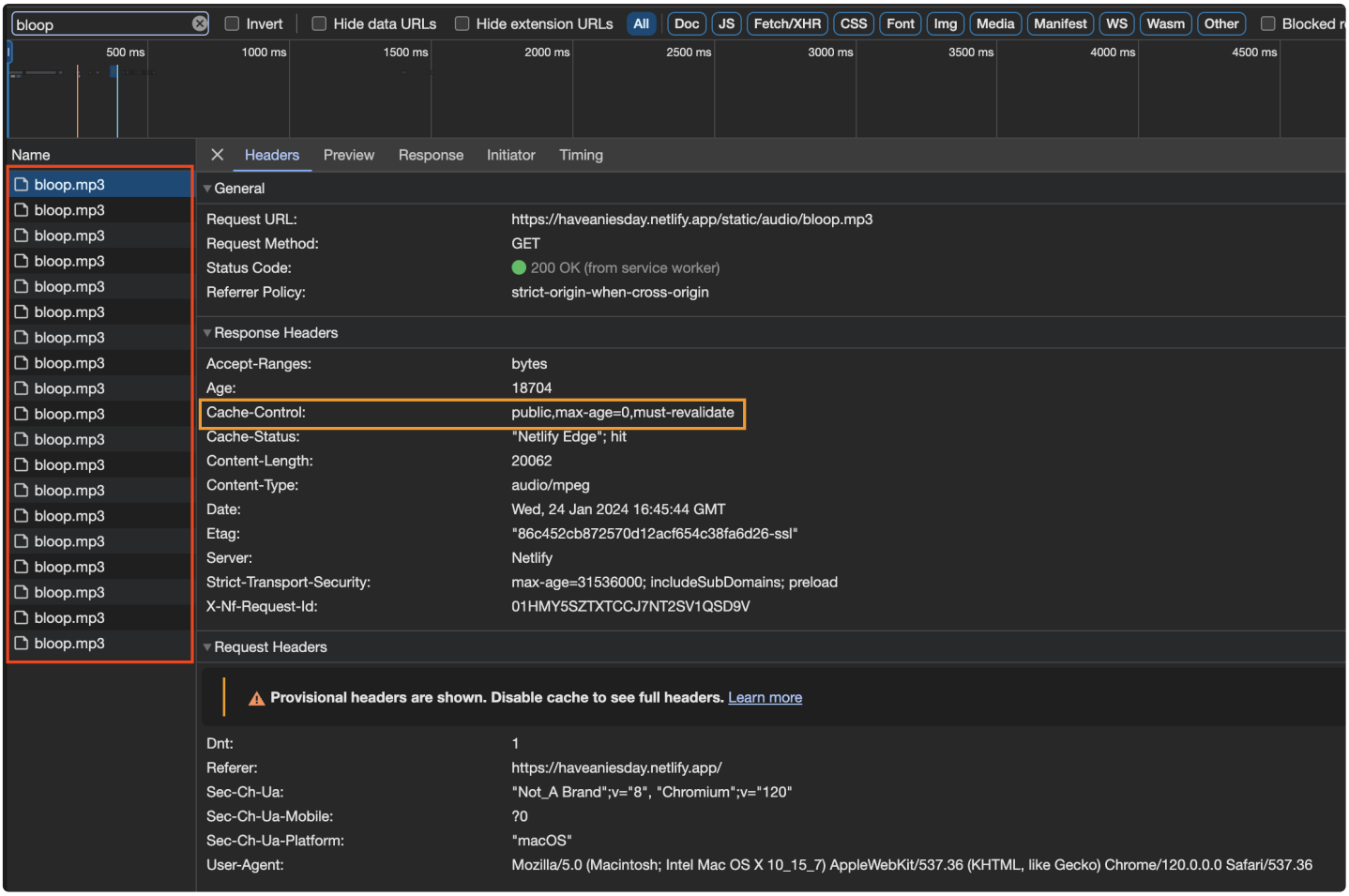
I also run diagnostics from PageSpeed Insights upon that webpage. You can access the diagnostics result here: https://pagespeed.web.dev/analysis/https-haveaniesday-netlify-app/1y21arcxfa?form_factor=mobile.

> **Disclaimer**
>
> It also means that some of the recommendations below may be no longer valid due to a different setup on the to-be-recovered https://www.haveaniesday.com/ domain. Despite of that, I will keep mentioning them below so we all can learn together from this incident. 😉

Recommendations

Based on my assessments as an outsider, I found a few low-hanging fruits to optimise the website both in the server bandwidth consumption.

1. Inefficient cache policies



Problem

The website requested for the same static resources multiple times in a single page. It may be caused by several elements referring to the same static assets multiple times.

However, if the static assets are cached properly on the browser, there's no need to perform additional roundtrip requests to the servers for exactly the same static assets.

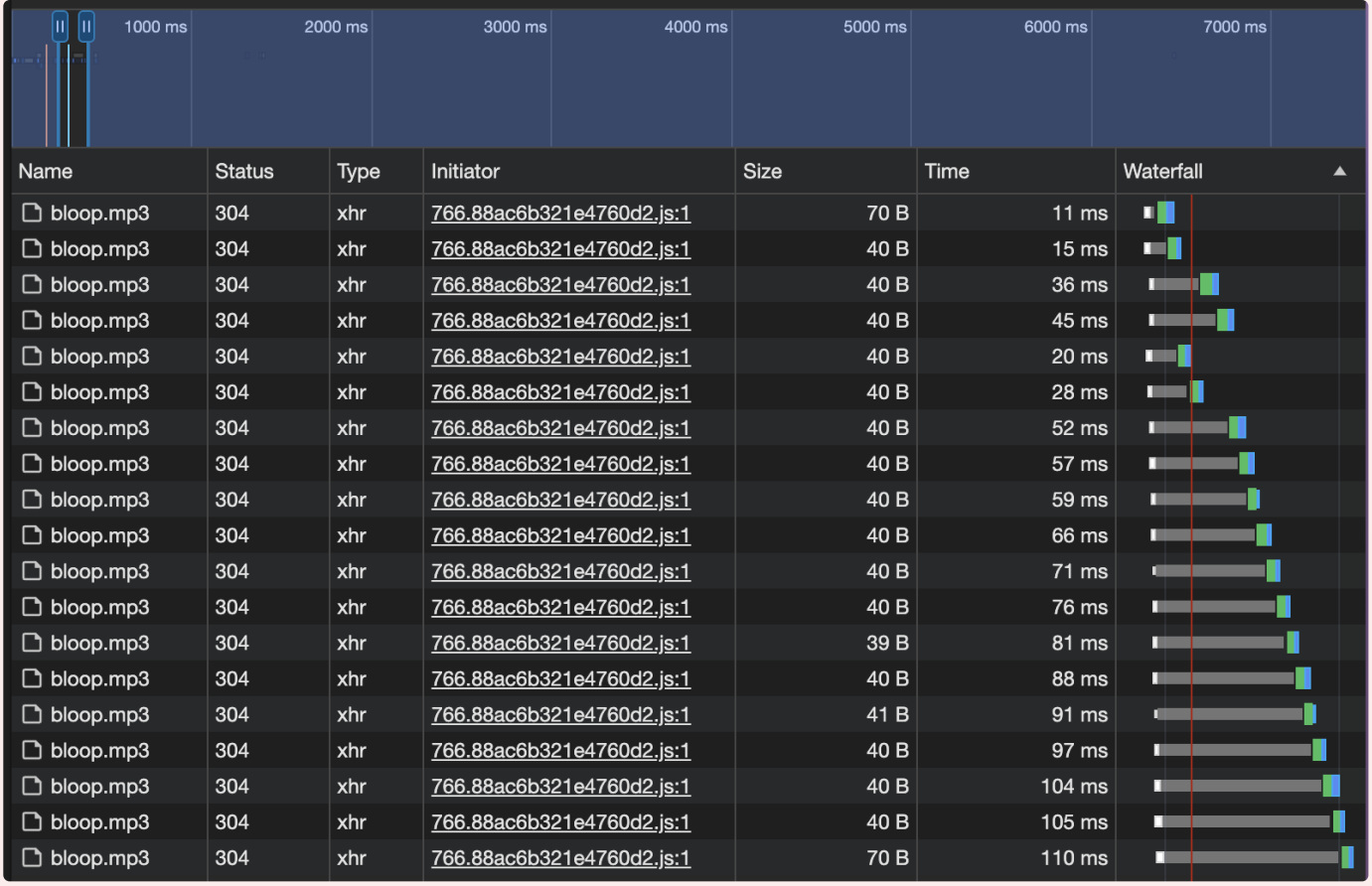
Tip

I noticed that this website installed a service worker on the initial page load. So the asset requests for subsequent page loads will be intercepted by the service worker instead.

Name	Status	Type	Initiator	Size	Time	Waterfall
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	1 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	
bloop.mp3	200	xhr	766.88ac6b321e4760d2.js?__...	(ServiceWorker)	2 ms	

Bug

That is a good strategy, but it doesn't help with the initial page load (notice the `Size` column that doesn't mention `(ServiceWorker)`). We can see that the `Time` spent and the `Waterfall` diagram is significantly longer than the ones from the screenshot above. Therefore, we still need to make the cache policies more efficient.



Name	Status	Type	Initiator	Size	Time	Waterfall
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	70 B	11 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	15 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	36 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	45 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	20 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	28 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	52 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	57 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	59 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	66 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	71 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	76 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	39 B	81 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	88 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	41 B	91 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	97 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	104 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	40 B	105 ms	
bloop.mp3	304	xhr	766.88ac6b321e4760d2.js:1	70 B	110 ms	

Solution

Implement a more efficient cache policy by customising the Cache-Control header based on the types of assets. The lowest hanging fruits in this aspect are usually media files like image or audio.

Configure the server to return this response header:

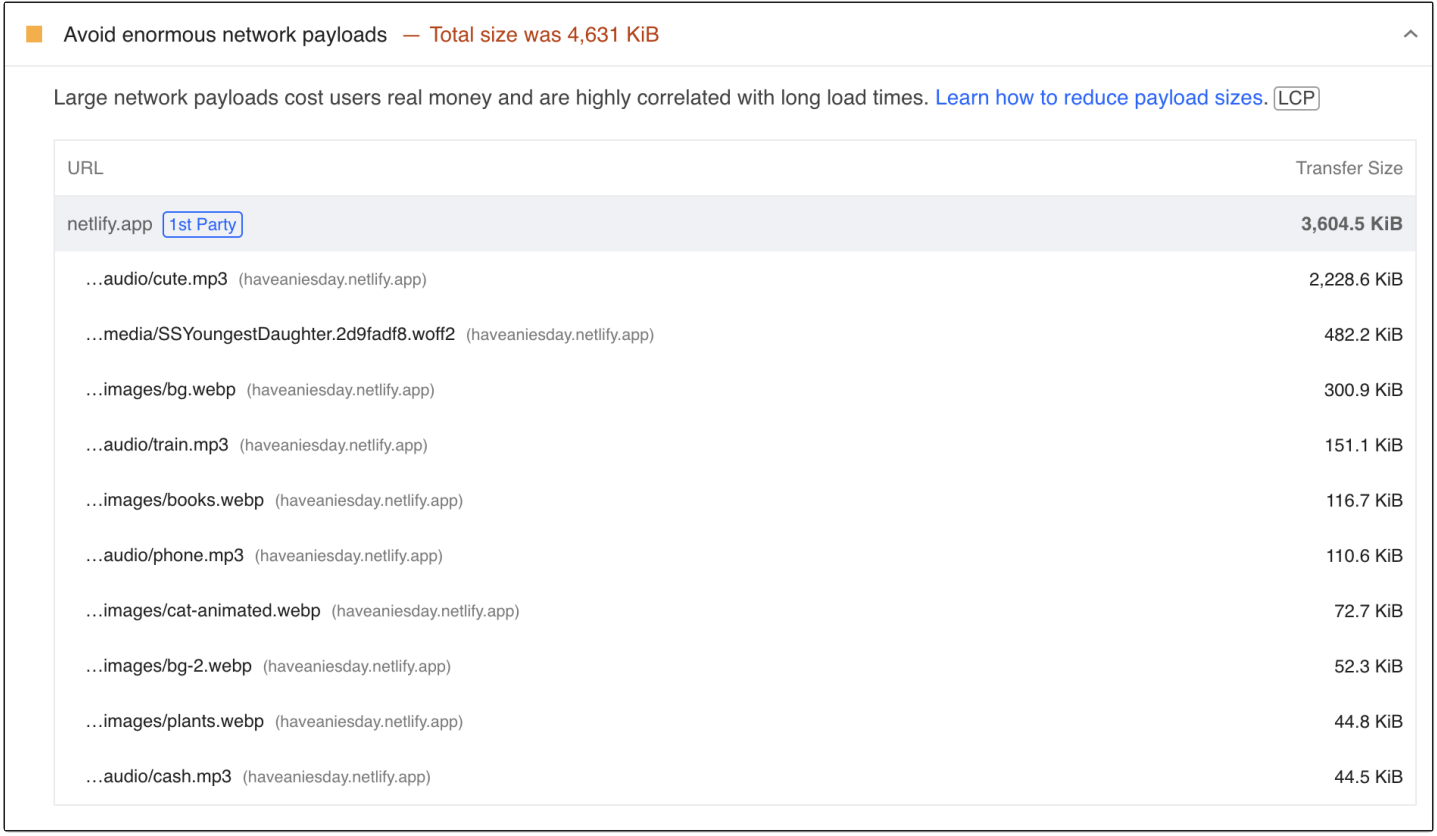
```
Cache-Control: public, max-age=604800, stale-while-revalidate=86400 immutable
```

The browsers are supposed to interpret it as:

1. `public` directive indicates that the response can be stored in a shared cache. This includes server-side caches, such as CDNs.
2. `max-age=604800` directive indicates the freshness of the response. In this case, the response will be considered fresh for 7 days (604800 seconds). This is assuming that those static assets are unlikely to change within the span of 7 days.
3. `stale-while-revalidate=86400` directive indicates that the cache could use a stale response while it revalidates. If you are familiar with SWR library, it actually attempts to replicate this HTTP cache invalidation strategy on the JavaScript layer.[3]
4. `immutable` directive indicates that the response will not be updated while it's fresh[4].

> **Note**
>
> You may need to understand Netlify's default caching behaviour before implementing this to ensure that the rules are not conflicting.

2. Lazy load unused assets



Problem

One prominent unused asset I noticed early is this 2MB cute.mp3 audio file. With the music muted by default on the first page load, assuming that most visitors won't turn it on (I myself only realised it after my subsequent visits to the website), loading the file eagerly will potentially waste the bandwidth unnecessarily.

Solution

Only load the audio file when the user indicates that they are about to use the asset. In this cute.mp3 case, you can load it when the user **hovers** the **unmute** button.

To do so, you may listen to the `MouseEvent` and/or `TouchEvent` handlers in React to detect whether the cursor/pointer is over the **unmute** button or not. This will limit the 2MB cute.mp3 audio file download only for the users who intend to use it.

> **Tip**
>
> This action item is applicable to any other deferrable assets. Once you are done with this 2MB cute.mp3 audio file lazy loading, you can continue the same step for any other potentially unused assets. I suggest starting from the bigger files so that you gain the most of the benefits with as little effort as possible. e.g.,
>
> 1. train.mp3
> 2. phone.mp3
> 3. cash.mp3
> 4. etc.

Mid-term improvements

Some improvements require more significant efforts that might not be addressable in the short terms, but they are worth mentioning.

1. Host the website on the Cloudflare Pages

I understand that hosting Next.js app in non-Vercel platforms is quite challenging nowadays, especially after they introduced the new App Router.

However, with all the bandwidth optimisation steps above, if the website traffic still hits the bandwidth of Netlify that we can afford to pay, we may want to move to another static hosting alternative that provides unlimited bandwidth. And as far as I know, Cloudflare Pages is the only hosting provider that provides an unlimited bandwidth on its Free tier. CMIIW.

If the this is something worth looking into, we can explore this solution by either:

1. Find a way to host Next.js 14 apps on Cloudflare Pages reliably, or
2. Migrate the implementation from Next.js to a simpler framework that does the job well with Cloudflare Pages, e.g. Vite 3.

Conclusions

The initial issue that triggered me to write this report is the inaccessible https://www.haveaniesday.com/ domain which is likely because the website hits the bandwidth quota on Vercel.[2-1]

Therefore, I heavily focused this analysis on the bandwidth-saving efforts. Once we solve this problem, and we want to improve the website further in the other aspects, such as web performance and rendering optimisations.

Hopefully, this document is beneficial for all the readers. Even if we don't have the time or capacity to execute all the recommendations above, at least we learn something to keep in mind in our upcoming projects.

Best regards, Zain Fathoni

---

1. https://x.com/aniesbubble/status/1750119440564543930?s=20↩
2. https://x.com/zainfathoni/status/1750158587203383597?s=20↩↩
3. Please note that the `stale-while-revalidate` directive is still considered experimental. But providing it won't hurt, anyway.↩
4. Please note that the `immutable` directive is incompatible in some major browsers, including Chrome. But providing it won't hurt, anyway.↩