Audited Page: https://store.steampowered.com/category/racing

Author: Vladyslav Haranich

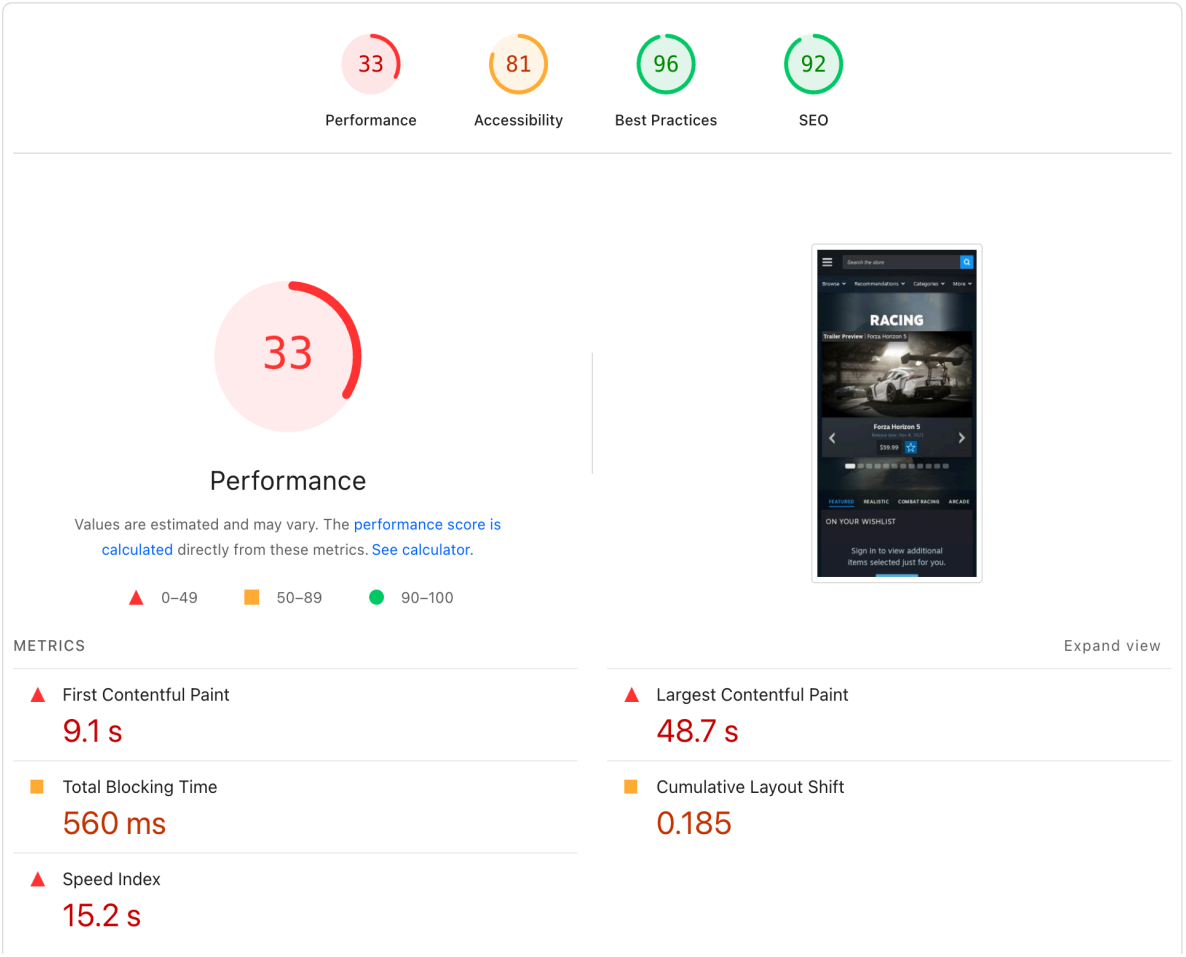
Focus: Performance audit of Steam's Category Page (Racing)

I want to present you this PDF, where I walk you through my performance review for the Steam Games Category page. It has a following structure:

1. A short "diary" to show my thought process while analyzing the performance issues;
2. A list of tasks to address these issues;

I chose Steam for my audit because it has lots of interactive elements, media assets, and I am an active user.

When I checked it with Lighthouse, the low performance score confirmed that. The home page was already pretty slow, but when I checked other pages, I found that the Category page was even more interesting to check. So I decided to go with it. I started digging into the reasons for the low performance score myself, and after finding/documenting a few issues, I went through the PageSpeed Insights results and wrote down my thoughts and ideas for how they could be improved.



1. Performance audit

I checked this page https://store.steampowered.com/category/racing.

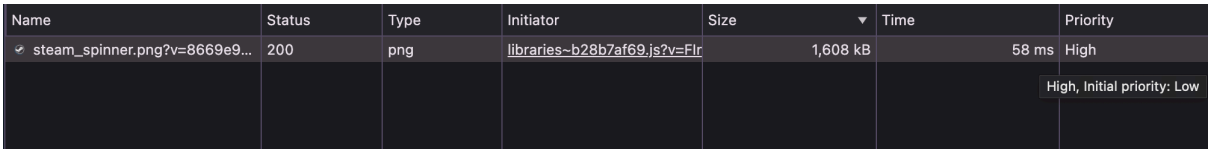
I noticed that it had to download around 12 MB on the initial load just to display the visible area. Other assets are lazy-loaded, but 12 MB upfront is already quite a lot. So, I decided to take a closer look at what exactly gets downloaded and see if there's room for improvement.

Unoptimized Resources

Assets:

The first thing I checked was the assets. Most of them are in JPG/PNG/GIF, which feels a bit outdated, we could easily switch to WEBP or AVIF for better compression and faster downloads.

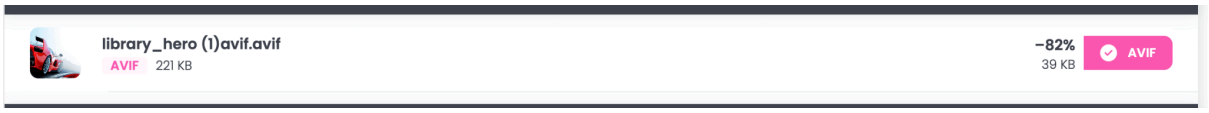
What surprised me most was a 1.6 MB gif of a spinner. I would definitely consider replacing it with a pure CSS spinner or a lightweight SVG/WEBP version if we really need something "fancy."



Next, I noticed that we're loading an image for the hero section as a background, which is used on desktop and mobile. I would recommend switching its format to AVIF/WEBP, with a JPG fallback for full support.

It's also a good idea to load different sizes for different screens using @media.

I checked how much size difference we would get between the original JPG and optimized AVIF, the original JPG was 241 KB, while the optimized AVIF version was only 39 KB. That would make a big difference if we apply the same optimization for all assets on this page.



This hero image is affecting the Largest Contentful Paint (LCP) score a lot, and it's pretty unnecessary.

Also, in the hero section, images are used as a video poster while the trailer preview is loading, which is a good idea overall.

However, the issue is that this image is almost 2 MB in size, even for users on mobile devices.

Name	Status	Type	Initiator	Size	Time	Priority
ss_ae4c258e6f13ddf00ee8e72be8bd0b0fc5e67...	200	jpeg	libraries~b28b7af69.js?v=FlnPE8lrzLqw&l=e	1,900 kB	1.17 s	High
ss_d61184a98c1cf2db2b08b2999c04b0519e361...	200	jpeg	libraries~b28b7af69.js?v=FlnPE8lrzLqw&l=e	1,705 kB	809 ms	High

Next, I checked the fonts. The font format used is a bit outdated, it would be better to switch to WOFF2, which is more suitable for the web. It's compressed and smaller in size, which would also help improve loading time. I also noticed that they didn't include the font-display: swap (or optional) property. Adding this makes readable text to show-up immediately (using a fallback font) instead of waiting for the font to be loaded, which improves UX and performance.

Other resources:

Also, I noticed that the main.js file is unusually large (~1.6 MB). As a next step, I'll take a closer look to see if there's a chance to split it for better performance.

Console Coverage ×

Per function | Filter by URL | All | Content scripts

URL	Type	Total Bytes	Unused Bytes	Usage Visualization
https://store.fastly.steamstatic.com/pub... /main.js?v=Lj7LHxJGhpTI&l=english&_cdn=fast	JS (per ...	1,572,670	1,195,692 76%	
https://store.fastly.steam.../libraries~b28b7af69.js?v=FlnPE8lrzLqw&l=english&_cdn=fast	JS (per ...	646,645	292,839 45.3%	
https://store.fastly.steamstat... /chunk~07d43922c.js?contenthash=f90fa773defefa5884	JS (per ...	450,204	283,366 62.9%	
https://store.fastly.steamstatic.com/public.../6597.js?contenthash=15fd615d90973d212f8	JS (per ...	269,214	244,736 90.9%	
https://store.fastly.steamstatic.com/pu... /7048.js?contenthash=64b9c4b5d4b8048ac96	JS (per ...	350,715	216,081 61.6%	
https://store.fastly.steamstatic.com... /main.css?v=JMXx0EuA4DH7&l=english&_cdn=fast	CSS	218,283	206,157 94.4%	
https://store.fastly.steamstatic.co... /game.css?v=PwfpNVm2PMLG&l=english&_cdn=fast	CSS	165,898	165,849 100%	
https://store.fastly.steamstat... /chunk~743f7df70.js?contenthash=35a8af25736bace5c9	JS (per ...	188,793	163,867 86.8%	
https://store.fastly.steamstatic.com/publi... /7436.js?contenthash=f4623e9d25d2b25d10	JS (per ...	177,375	157,270 88.7%	
https://store.fastly.steamstatic.com/... /store.css?v=07RtETKjVCNg&l=english&_cdn=fast	CSS	133,068	130,137 97.8%	
https://store.fastly.steamstati... /shared_global.js?v=7x550RMBik-V&l=english&_cdn=fast	JS (per ...	152,804	127,041 83.1%	
https://store.fastly.steam... /chunk~07d43922c.css?contenthash=6b3886236f1e8cc3b28	CSS	143,951	124,559 86.5%	
https://store.fastly.steamst... /chunk~898bae97c.js?contenthash=ec54bb7f97ae5ca4ba8	JS (per ...	109,764	93,863 85.5%	
https://store.fastly.steams... /libraries~730b00cba.js?contenthash=4a31aff5fbb857527c0	JS (per ...	125,194	93,152 74.4%	
https://store.fastly.steamst... /chunk~45e7b8927.js?contenthash=3642d0186f58bb377d	JS (per ...	94,209	82,911 88%	
https://store.fastly.steamstatic.com/pu... /main.js?v=IBr21mD23BIS&l=english&_cdn=fast	JS (per ...	86,323	80,954 93.8%	
https://store.fastly.steamstatic.com/publi.../4860.js?contenthash=7368aca5877e27296f5	JS (per ...	92,322	80,900 87.6%	
https://store.fastly.steamstatic.com... /saledisplay.js?contenthash=4215359ee75b5931fd	JS (per ...	108,681	77,545 71.4%	
https://store.fastly.steamst.../shared_global.css?v=sKNRqB1SWZVt&l=english&_cdn=fast	CSS	85,534	77,543 90.7%	

Based on the results, I can definitely say that it might be possible, or even necessary to split the bundle into smaller chunks and load only the required data.

Other parts we might not need at all, or we could lazy-load them later.

The main.js file is a blocking request, which makes things worse for UX since we have to wait for it to fully load before anything else happens.

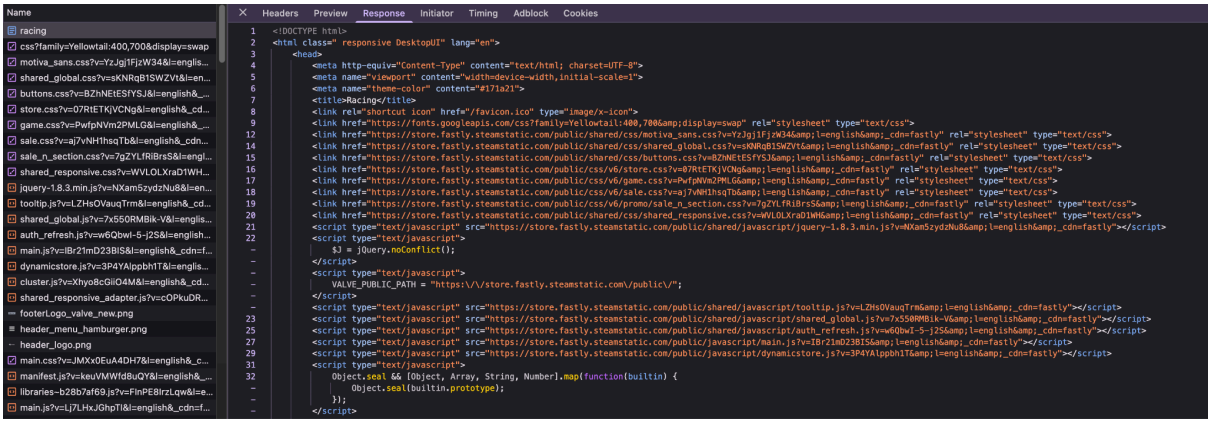
While checking further, I found that there are a lot of files besides main.js that contain a lot of unused code.

For example, I noticed the main.css file, only 5.6% of it is actually used. That makes me think it's just a pile of CSS that covers the entire project, and we really don't need to load it entirely for just this one page.

Interestingly quite a few files have over 60% unused code, and they're loaded right in the head tag, without any attributes to defer or lazy-load them.

That means we're loading non-critical code on the initial page load, which blocks rendering and slows down the overall performance.

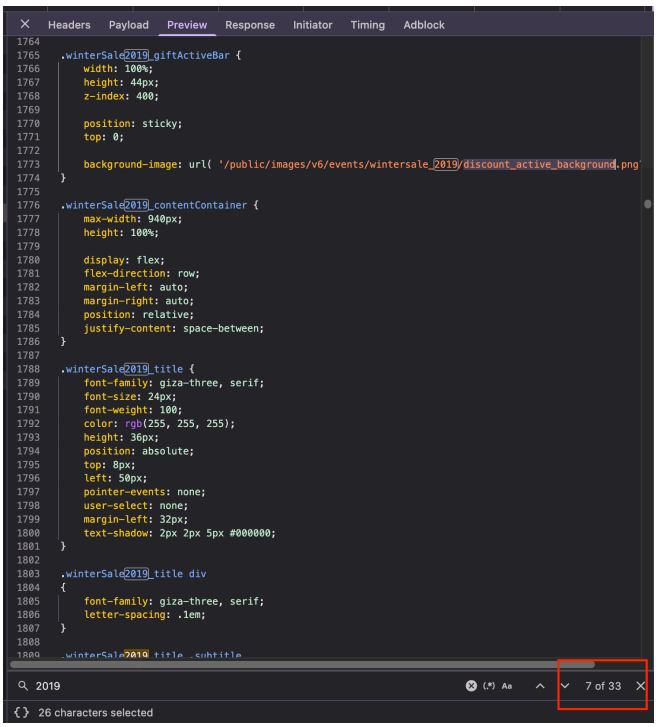
This is a screenshot of a head tag of the page. It looks like we load a lot of files here that block rendering, and I am pretty sure I saw some of them mentioned in the Coverage tab, marked as mostly unused.



So I used ChatGPT to compare the last 2 screenshots and check if any of the unused files are blocking the render, here's the result:

File	Type	Total	Unused	% Unused	Notes
`main.js`	JS	1.57 MB	1.19 MB	**76%**	Heavy, render-blocking, mostly unused on load
`shared_global.js`	JS	153 KB	127 KB	**83%**	Blocking, but little used immediately
`dynamicstore.js`	JS	~94 KB	~83 KB	**88%**	Blocking, barely used
`tooltip.js`	JS	~188 KB	~164 KB	**87%**	Blocking, minor feature
`main.css`	CSS	218 KB	206 KB	**94%**	Render-blocking, almost totally unused
`game.css`	CSS	166 KB	166 KB	**100%**	Render-blocking, 0% used initially
`store.css`	CSS	133 KB	130 KB	**98%**	Blocking, 98% unused
`shared_global.css`	CSS	86 KB	78 KB	**91%**	Blocking, almost entirely unused

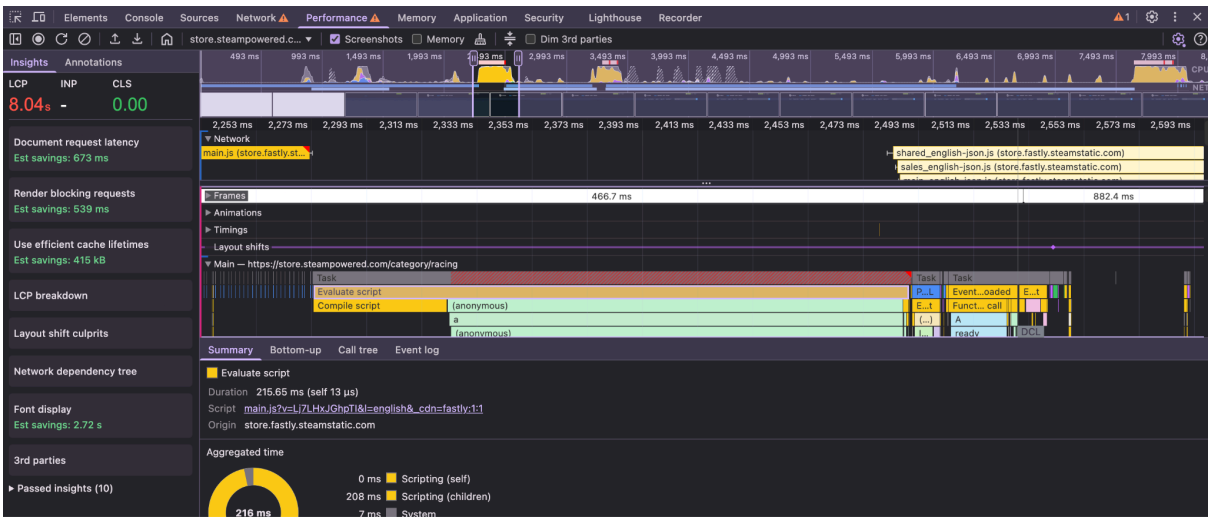
That tells us that we totally unnecessary block rendering with a code that is barely used. One funny thing I noticed during investigation, they still keep some css from summer and winter 2019 sale :D



Main-thread work

Based on PageSpeed Insights and performance checks in DevTools, it looks like there are quite a lot of calculations happening on the frontend.

Getting rid of unnecessary code and resources would definitely help to reduce the load.



After a quick look at what's being executed during these expensive operations, I found a few files that store quite large amounts of data, mostly arrays and objects, which the scripts are likely searching or filtering through.

If it’s not possible to move this logic to the backend, we could consider using Web Workers. This way, the main thread won’t get blocked while all this is being processed, which would make the page feel much smoother.

Third party is blocking the main thread

Based on PageSpeed Insights the page has significant issues with third party code which is blocking the main thread.

Third-Party	Transfer Size	Main-Thread Blocking Time
steamstatic.com	**14,407 KiB**	**1,048 ms**
...store/libraries~b28b7af69.js?v=... (store.fastly.steamstatic.com)	165 KiB	473 ms
...store/main.js?v=Lj7LHxJGhpTI&l=english&_cdn=fastly (store.fastly.steamstatic.com)	359 KiB	311 ms
...store/chunk~07d43922c.js?contenthash=f90fa77... (store.fastly.steamstatic.com)	440 KiB	251 ms

After checking the source code, I noticed that the scripts I mentioned earlier are loaded without the defer attribute.


ids":[],"build_id":0,"build_branch":"","unlisted":0,"votes_up":243,"votes_down":120,"comment_type":"ForumTopic","gidfeature":"3826411316950185693","gidfeature2":"6007915741258776 88","clan_steamid
">
…
<script>window.g_wapit="";</script>
<link href="https://store.fastly.steamstatic.com/public/css/applications/store/main.css?v=1MXv0EuA4DH7&l=english&_cdn=fastly" rel="stylesheet" type="text/css">
<script type="text/javascript" src="https://store.fastly.steamstatic.com/public/javascript/applications/store/manifest.js?v=keuVMWfd8uQY&l=english&_cdn=fastly"></script> == $0
<script type="text/javascript" src="https://store.fastly.steamstatic.com/public/javascript/applications/store/libraries~b28b7af69.js?v=FInPE8IrzLqw&l=english&_cdn=fastly"></script>
<script type="text/javascript" src="https://store.fastly.steamstatic.com/public/javascript/applications/store/main.js?v=Lj7LHxJGhpTI&l=english&_cdn=fastly"></script>
<script type="text/javascript">…</script>
<style type="text/css">…</style>

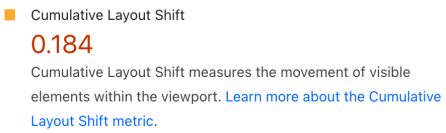
…
<!-- End Main Background -->

<!-- responsive_page_legacy_content -->
<div id="footer_spacer" style class>
<div id="footer" role="contentinfo" class>…

<!-- responsive_page_content -->

Defer lets the browser download scripts while parsing the HTML and only run them after the page is ready. Without it, these scripts block rendering and slow down the initial load.

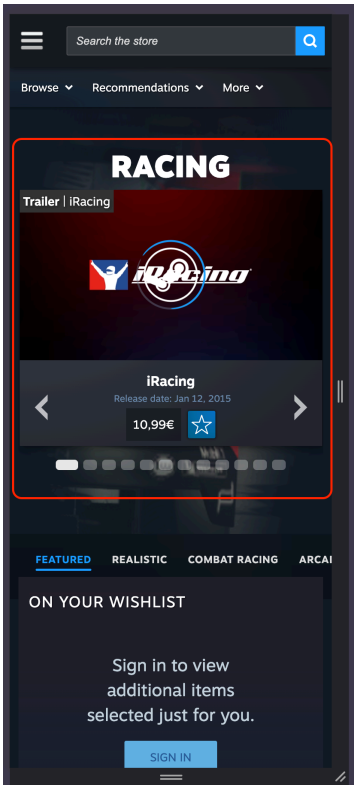
Large layout shifts



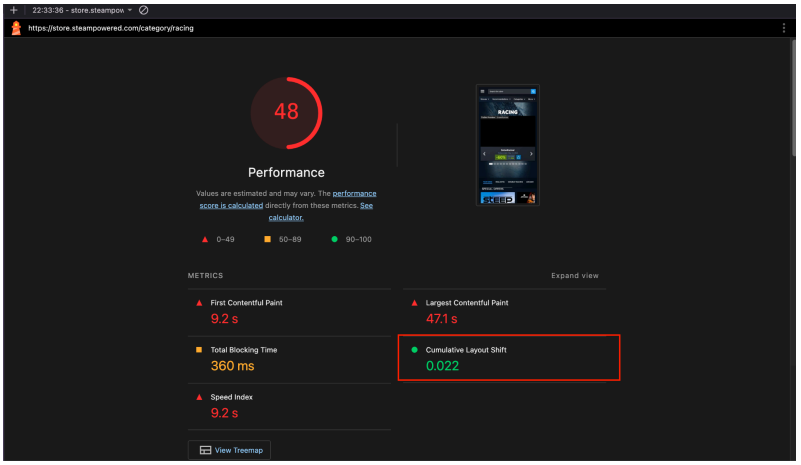
That’s the current CLS score.

After checking what’s causing the layout shift, I found that the main banner collapses when it has no content because there’s no fixed height set.

Then, when the content loads, it suddenly expands and pushes everything below it down, which causes the layout to shift and the page to re-render.



I’ve applied override to css (using DevTools overrides feature), set aspect ratio to the banner and checked the current score with lighthouse. The score has significantly improved:



2. Addressing performance issues

Task 1: Optimize Hero Background Image

The hero images on the Category page are currently around 250 KB each, they are used as a background for both desktop and mobile.

That's one of the first things the browser loads, heavily affecting LCP.

I tested converting it to AVIF - it dropped to 39 KB, which is quite an improvement.

Fix:

- convert images used for hero section to AVIF, keep JPG fallback for older browsers
- load smaller variants for mobile via media queries
- verify improved LCP in Lighthouse

Code example:

```
.div1 {
  background-image: image-set(
    "kitten.avif" type("image/avif"),
    "kitten.jpg" type("image/jpeg")
  );
}
```

(image-set() is already quite well supported https://caniuse.com/?search=image-set)

Task 2: Compress Trailer Poster Images (~2 MB)

Before the trailer video loads, the page shows a large static image as a placeholder, it's around 2 MB, which is crazy for a background.

This image is loaded even on mobile, delaying FCP and LCP.

Based on my knowledge, we can't set an adaptive image for the poster attribute of the <video> tag. So if we need to support older devices, I would keep it in JPG format but reduce the resolution and optimize it properly. Even reducing the file size from 2 MB to around 150 KB would already be a significant improvement.

Implementation:

- compress and resize the images used for trailer poster
- verify improvement with Lighthouse

Task 3: Optimize game info card images

There are a lot of game thumbnails on this page. All of them could be optimized and reformatted to modern formats for better loading speed.

Fix:

- convert images to AVIF, keep JPG fallback
- load smaller variants for mobile via media attribute
- verify improvement in Lighthouse

```
<picture>
  <source src="images/eve.avif" media="(max-width: 800px)" />
  <source src="images/eve-xl.avif" />
  <img src="images/eve.png" alt="Eve" />
</picture>
```

Task 4: Replace the 1.6 MB GIF spinner

There's a 1.6 MB animated GIF used just for a loading spinner.

That's more than the total size of many websites 😅 we can replace it with a CSS spinner or a small SVG animation.

Implementation:

- remove the GIF file completely
- implement a CSS or SVG spinner with similar appearance
- in case of SVG, please don't forget to optimize it (for example, use SVGO)
- ensure it animates smoothly

Task 5: Use WOFF2 fonts with font-display: swap

Fonts are currently loaded in outdated formats and block rendering since there's no font-display property set.

Users see blank text until fonts finish loading.

Implementation:

- convert all web fonts to WOFF2
- add font-display: swap to all @font-face rule
- check if we load some fonts, that not used on this page or at all

Task 6: Split and optimize main.js (1.6 MB)

main.js is around 1.6 MB, fully blocking render until it finishes downloading. PageSpeed and DevTools both show significant main-thread activity. We can split the bundle into smaller chunks and lazy-load non-critical logic.

Implementation:

- analyse the bundle through vite-bundle-analyzer
- figure out how we can split main.js into smaller, feature-based chunks
- create sub-tasks for the splits (doing it all at once is too risky)

For subtasks mention these action items:

- lazy-load this part
- make sure everything still works after the split
- recheck performance, especially Time to Interactive (TTI) score

Task 7: Remove unused CSS

Most of the loaded CSS files are huge, but only a small part of it is actually used on the category page. The rest is just some legacy code or global, which is not used on this page.

Right now, most of those huge files are loaded in the <head>, which blocks the render.

Fix:

- audit main.css using DevTools Coverage (or PurgeCSS) to identify unused selectors
- split critical and non-critical CSS
- remove outdated and unused rules (especially legacy CSS)
- validate that layout and styling are not broken
- re-test render speed and LCP via Lighthouse

That’s pretty much it! Thank you for your attention!