Polarsport.pl

Ecommerce optimization

for Core Web Vitals

Polar Sport is one of the largest outdoor retail chains in Poland. The owners come from the environment of mountain guides and they highly understand their customers' needs. They have ten stores in the biggest cities in Poland. However, one of the main points of sales is their ecommerce: polarsport.pl

After a few years of maintenance, all pages slowed down. No optimization had been done before. Inconsistent content management, inadequate image quality, inefficient building of the new pages and subpages generated a lot of bugs and real

problems. The basic indicators got poorer day by day. The necessity of page speed optimization for Core Web Vitals (CWV) arose very fast.

Mobile is first

We started our work with an ecommerce audit. Statistics from CWV show that improving a website's performance on mobile devices proportionally improves the website's performance on other devices. However, CWV statistics do not show how the website should be created but how it should work on devices in the first seconds of the user's visit.

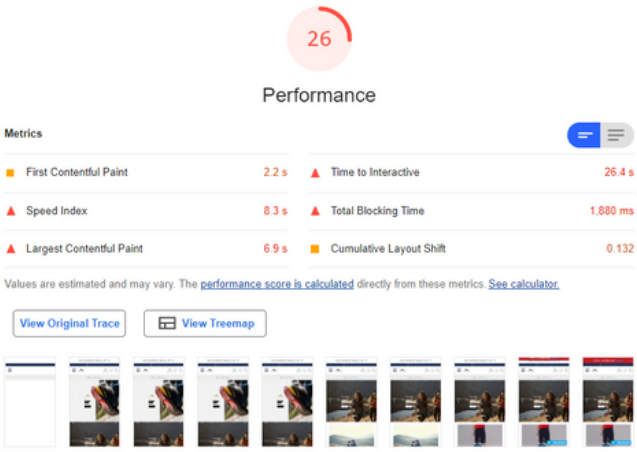
Therefore, the state of the website launched on mobile devices such as smartphones and tablets is considered first - website traffic and its availability must be increased.

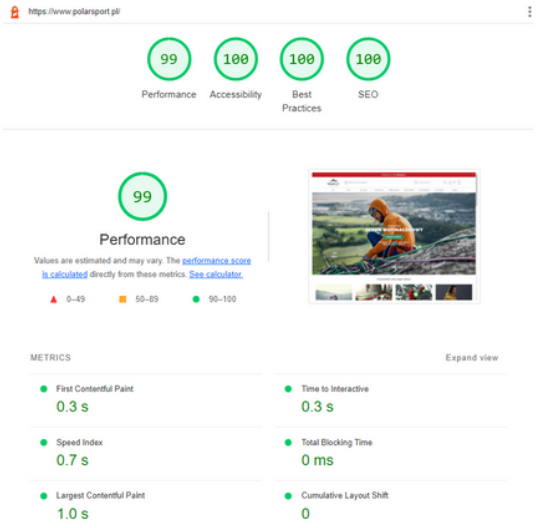
In the first steps, we concentrated on preparing a complex and progressive plan. Each next step gradually increased the result and thus improved the website's response to the customers visiting it.

Before Optimization

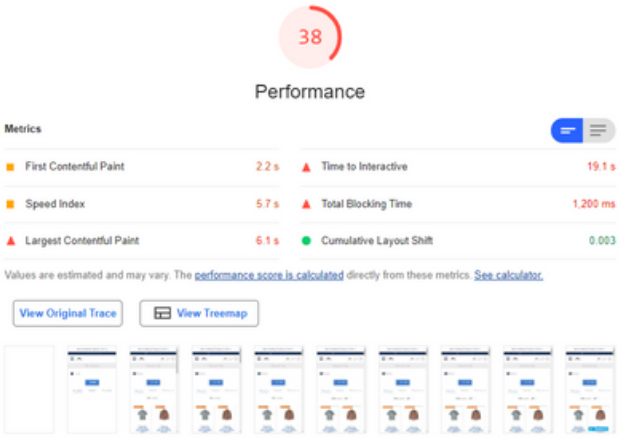
After Optimization

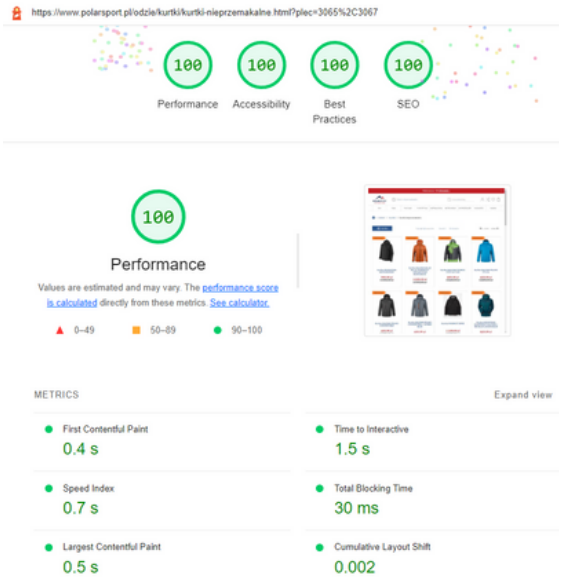
Main Page



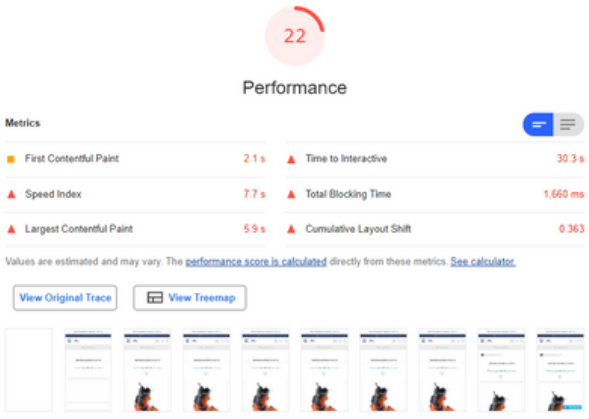


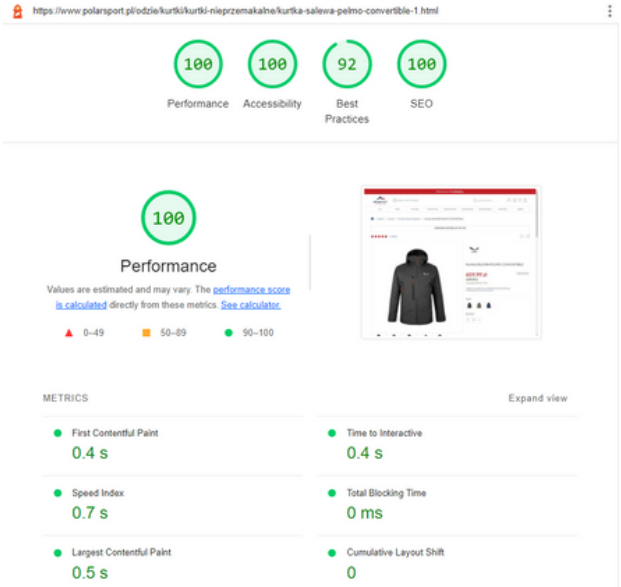
Category Page

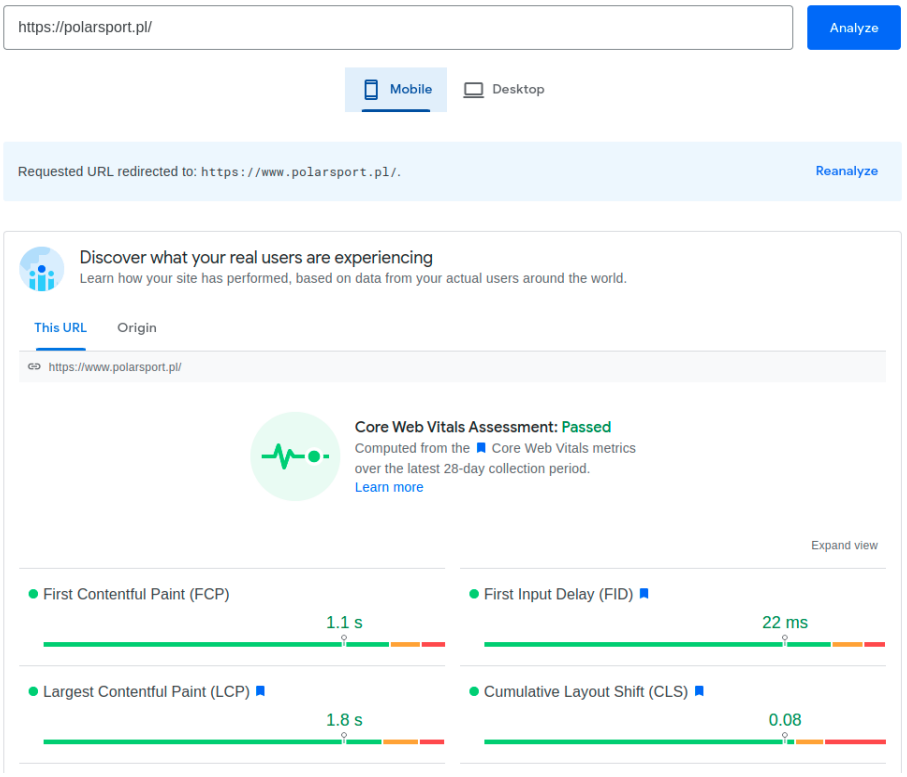




Product Page



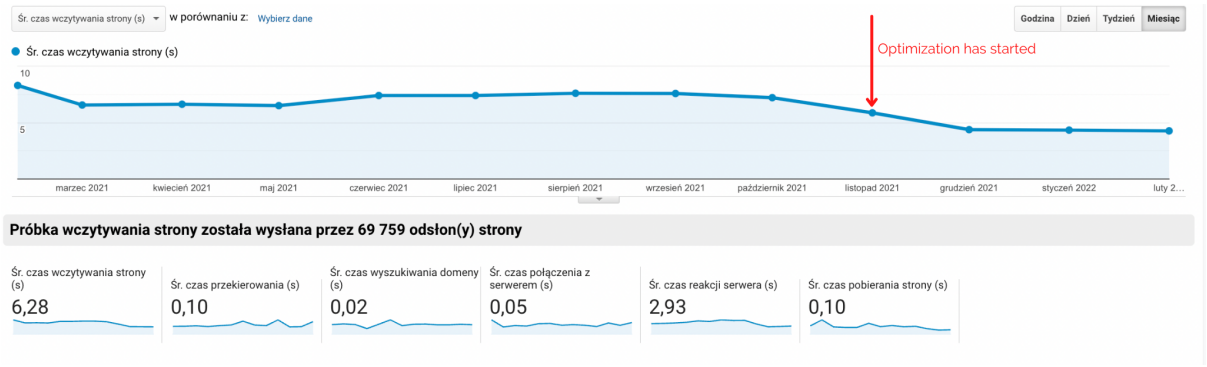




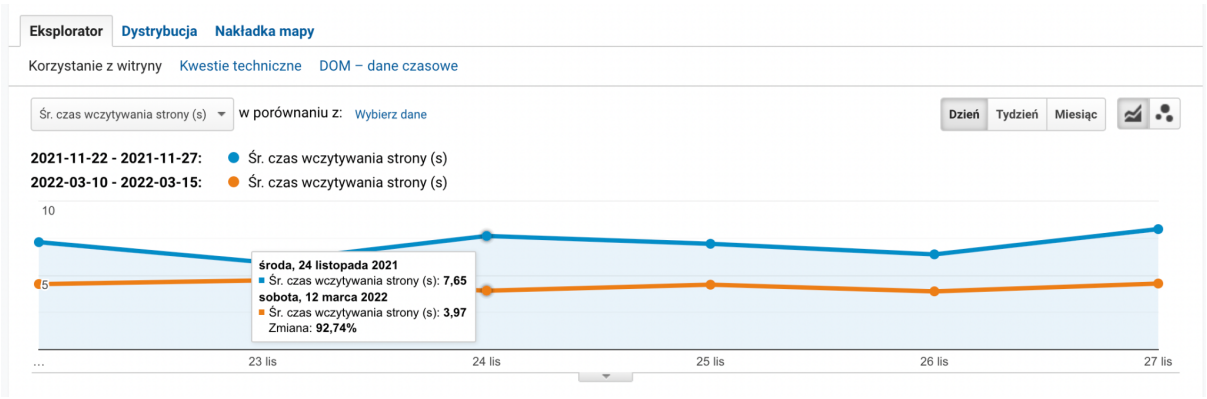
Core Web Vitals Assessment: **Passed.**

Comparison in Google Analytics

Average Page Load Time - general view:

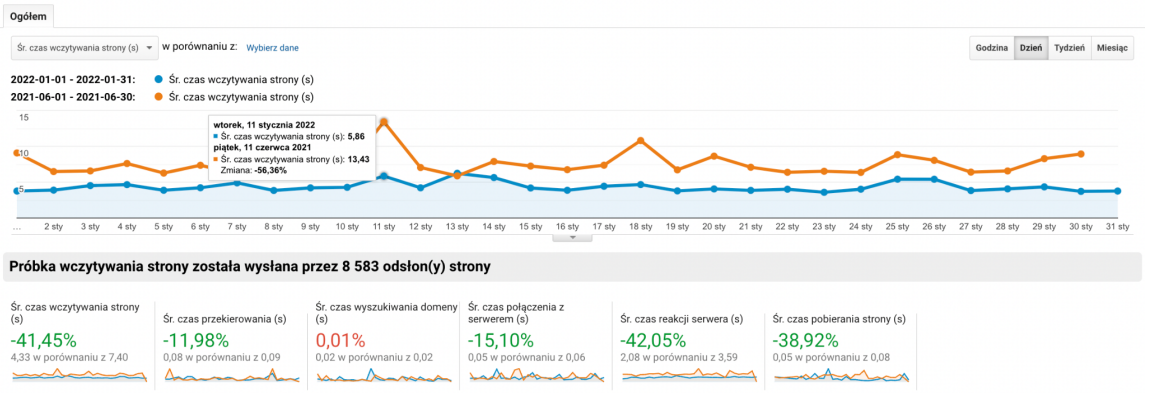


Average Page Load Time - comparison before (blue line) and after (orange line) optimization. **Change 92,74%.**

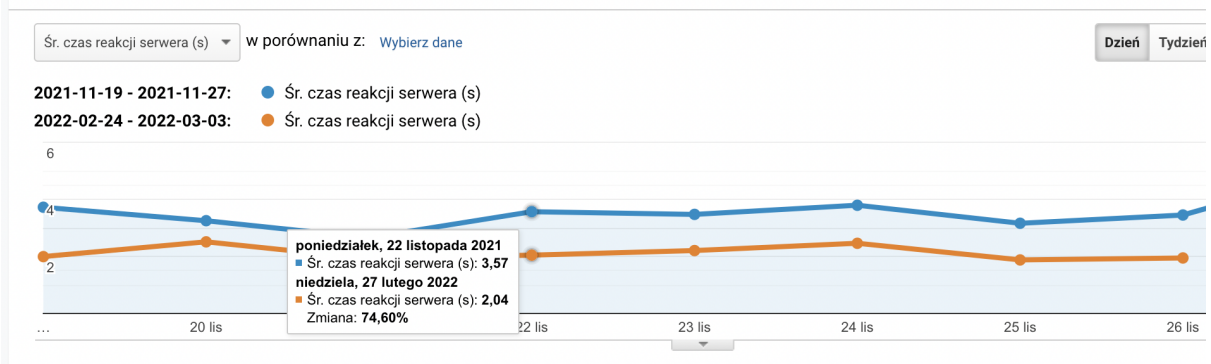


Average Page Load Time - comparison before (orange line) and after (blue line) optimization. Key metrics are much better and in green.

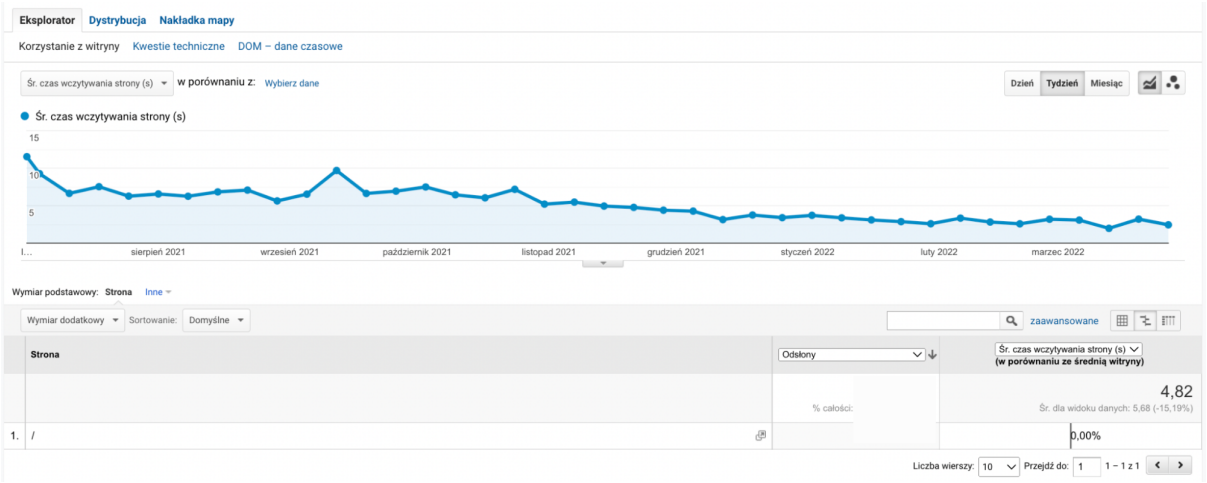
- Avg. Page Load Time: **-41,45%**
- Avg. Redirect Time: **-11,98%**
- Avg. Domain Search Time: **0,01%**
- Avg. Time of Connection to the Server: **-15,10%**
- Avg. Server Response Time: **-42,05%**
- Avg. Page Download Time: **-38,92%**



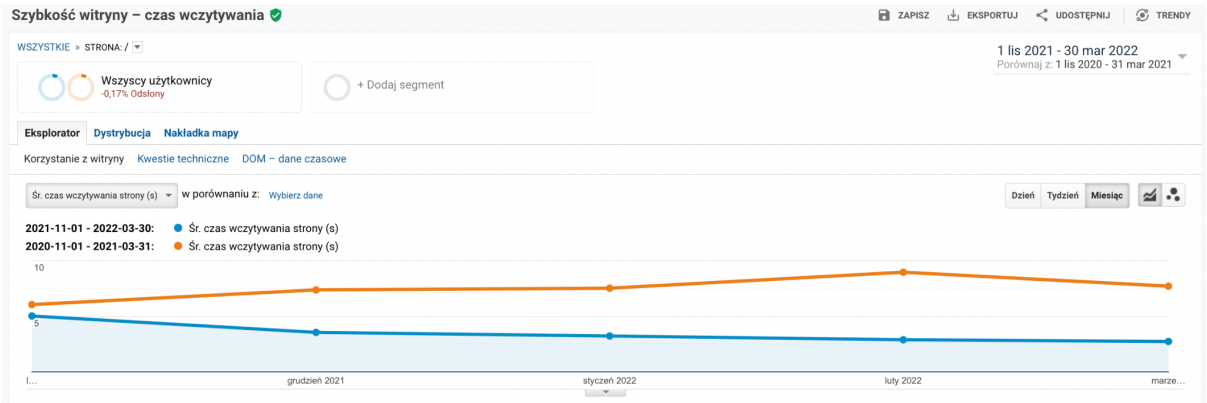
Average Server Response Time - comparison before (blue line) and after (orange line) optimization. **Change 74,60%.**



Page Optimization - results in Avg. Page Load Time. Significant improvement in page speed.



Page Optimization - comparison Avg. Page Load Time in the same period in the previous year - before (orange line) and after optimization (blue line).
Significant improvement in page speed.



Improving the performance

Currently, the Performance statistics (a set of metrics that represent the page's performance numerically, which translates into its loading time in the browser and the convenience of use) for CWV consist of the following components, which needed to be improved:

- **FCP (First Contentful Paint)** - measures the loading time of the first element on the page after the user launches it.
- **LCP (Largest Contentful Paint)** - measures the loading time of the largest element on the page visible to the user.
- **TBT (Total Blocking Time)** - measures the time needed for the website to react after the user interacts with it.
- **CLS (Cumulative Layout Shift)** - measures the extent to which the page template has shifted on the screen, not due to user actions, but due to programming problems.

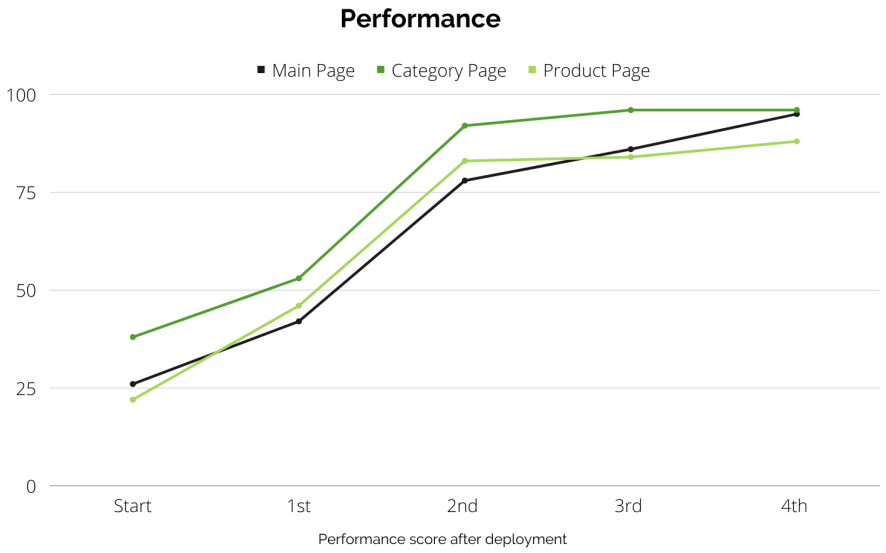
To deliver the best improvement we added two more indicators that had to be included in the optimization:

- **SI (Speed Index)** - measures how quickly the scheduled content fills the page, i.e. how quickly the user sees the page content. The smaller the value the better.
- **TTI (Time to Interactive)** - measures the time it takes for a page to load so that the user can use it and interact with it.

We planned backend optimization too. Essentially, the server points came down to minimizing the TTFB value, which affected the TBT and FCP ratios of Core Web Vitals.

TTFB (Time to First Byte) is the time from when a request is sent by the client to when the first response is received. Before the optimization, TTFB on category pages was in the range from a few to even a dozen or so seconds.

Here's how the progress of performance optimization looks:



What have we done?

Optimization for CWV is a long process and it requires systematic and exact work. This work is endless - as long as an ecommerce exists, it requires daily maintenance. However, all neglect can be removed and permanent good practices can be implemented.

We have worked for a few months in several interactions to achieve the expected results. What's important is that we have worked on the native Magento Theme.

Frontend Optimization

We can divide our optimization into four areas:

1. Limit unused JavaScript
2. Eliminate render-blocking resources
3. Postpone loading images off-screen
4. Limit unused CSS

This is in accordance with Google's best practices for developers for Google Lighthouse. In practice there were many minor tasks including logo compression, image compression and the change to the webp format, optimization of sliders according to Google's recommendations, clean colors in CSS, the elimination of render-blocking resources and the deferring of offscreen images. We verified that all HTML <a> and <img> tags had the title="particular images titles" and alt="Alternative content to display" attributes, we got rid of double IDs, validated CSS and HTML, upgraded the software version on the server and removed unused external modules.

Each task improved page speed and saved requests for the server.

We started our work from SEO optimization and accessibility and we implemented good practices. At the same time, we conducted work to improve the key indicators.

FCP improvement

To improve FCP (most of the work coincided with the optimization activities for LPC, where there was a reduction or postponement of the JavaScript and CSS code used) we preloaded resources from external sites such as googlefont.com or fontawesome.com. We also analyzed the method for loading Google fonts, we limited their number by loading them via files located directly as static font files on the server instead of using an external url such as fonts.googleapis.com and we implemented the display = swap option.

LCP reduction

To reduce the value of LCP, we reviewed and eliminated excessively occurring JS files, eliminated excessively occurring queries to external websites (e.g. Facebook, Edrone, Sugodeku, Anilima, Retagro, Citydsp and others) and checked and configured the existing CDN. The next step was the analysis of the resulting merged CSS file, the elimination of duplicate entries and unused CSS class entries (such as bootstrap.css or datepicker libraries), the review of other libraries and modules that add their own CSS files to individual pages and the minification of CSS and JS files.

TBT improvement

The improvement of the TBT index was influenced by the verification of the installed additional modules, the slimming down of the DOM structure by modifying individual template files (PHTML), the verification and deferring of external websites

- e.g .: ZenDesk, Facebook, Google Analytics, Google CDN, Bootstrap CDN, FontsAwesome CDN, Google Fonts, Google Tag Manager, New Relic, Other Google APIs / SDKs, Google / Doubleclick.

There were also common operations that improved both TBT and LCP indicators like lazy-loading for photos, images, pop-ups, videos and more complicated scripts.

CLS improvement

The CLS indicator was improved by verifying which img elements do not have the width and height attributes - limiting them on the page allowed to avoid shifts in the rendering time of the website. We verified additional elements and banners that pop up or scroll immediately after loading the page, preventing the proper interaction of users with the elements displayed in the “above the fold” view. We verified and improved animations on the website.

We also took into account activities dedicated to specific subpages - different for the main, category and product pages.

Backend optimization

To properly implement optimization there is a necessity for backend work. Our goals were:

1. Speeding up the online store so that the front loading for the x-cache: MISS header was not longer than 1 second,
2. Correcting the setting of Varnish, Redis cache, correct headers,
3. Reducing the TTFP time to below 400ms and what affects the long TTFB in Magento 2:
    - No source code optimization,
    - Wrong server settings,
    - No backend caching,
    - No database optimization.
4. Eliminating the mySQL queries burdening the database
5. Improving the stability of www.polarsport.pl

We planned our work along the following steps:

- Analysis and selection of key KP
- Prioritizing tasks and scheduling work
- Programming work, tests and implementation of planned tasks

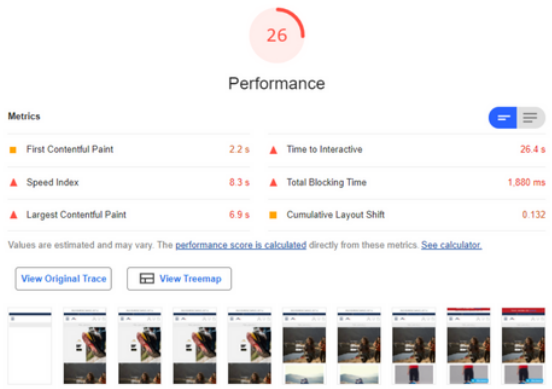
Our tasks included:

1. Improving Headers headers (cache-control, expires)
2. Gzip compression optimization
3. Analysis of views with a profiler
4. Optimizing the Varnish configuration
5. MySQL database optimization
6. Redis optimization
7. Cron Job Optimization
8. Dropping cookies
9. Structured data improvement (json-ld)
10. Improving xml-i and correct cache configuration (sitemap.xml, product feeds)
11. Analysis of unused add-ons / modules
12. Magento system logs
13. Debian upgrade to version 10 / PHP upgrade
14. Cache Warming mechanism (prototype tests)
15. Others, e.g.:
    - Server settings, cache e.g. well-known
    - Access robots to files from the admin!
    - Disable Ajax indexing, e.g. accessibility
    - Magento blocks cacheable = "false"

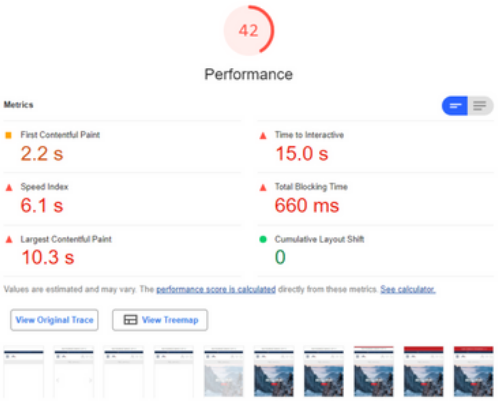
We updated all software: Magento engine, Varnish, Redis, Database, Elasticsearch, nginx, server system.

Main Page

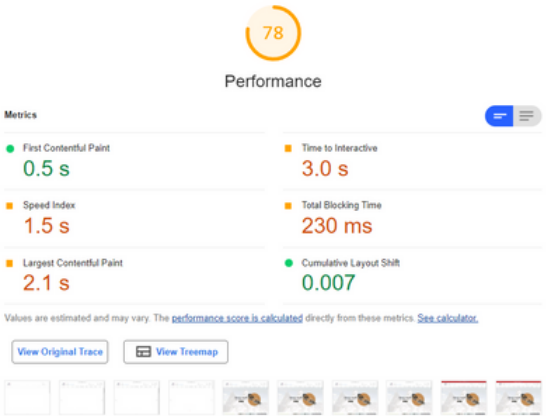
Start



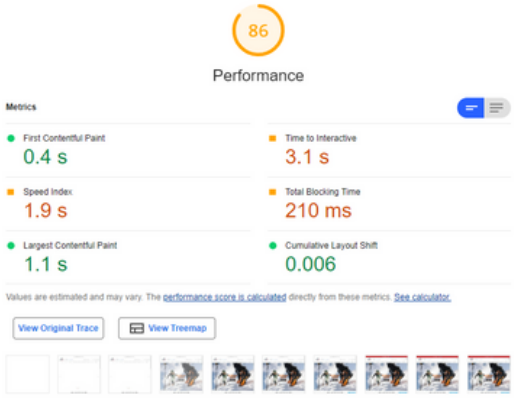
After 1st deployment



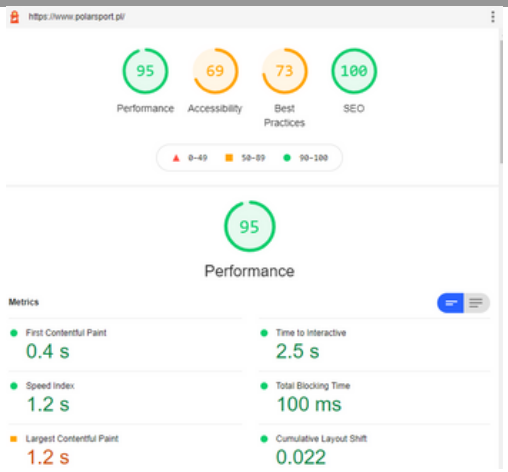
After 2nd deployment



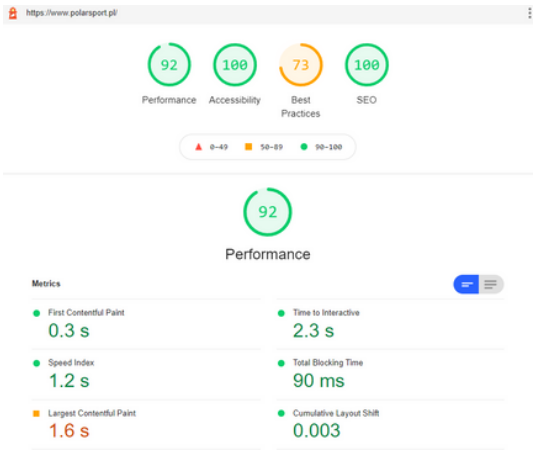
After 3rd deployment



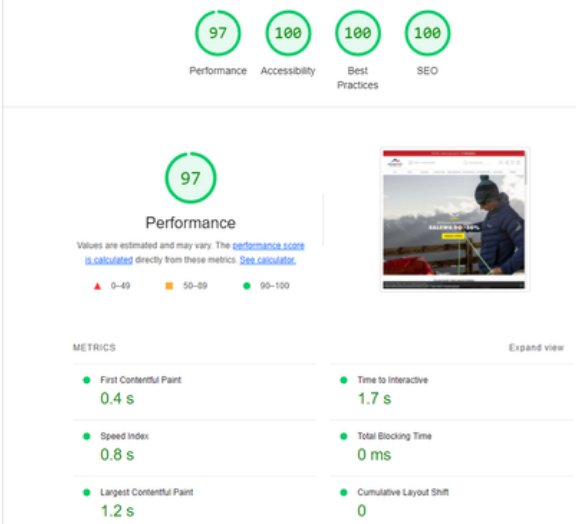
After 4th deployment



After 5th deployment

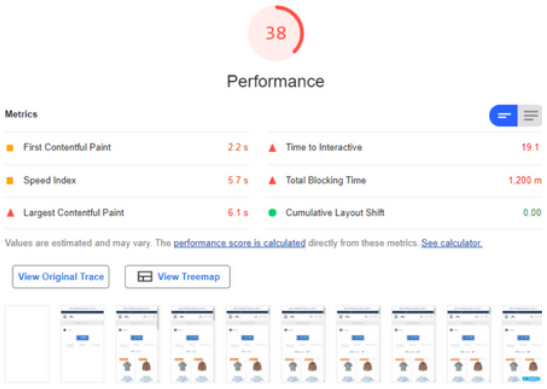


After 6th deployment

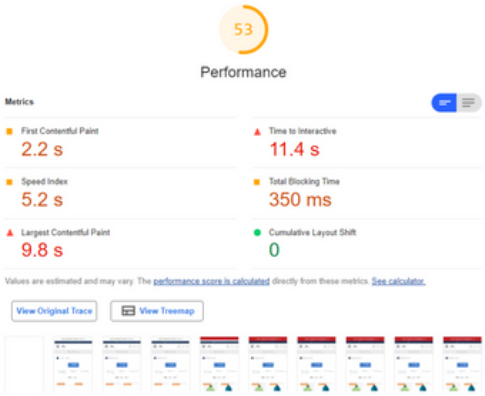


Category Page

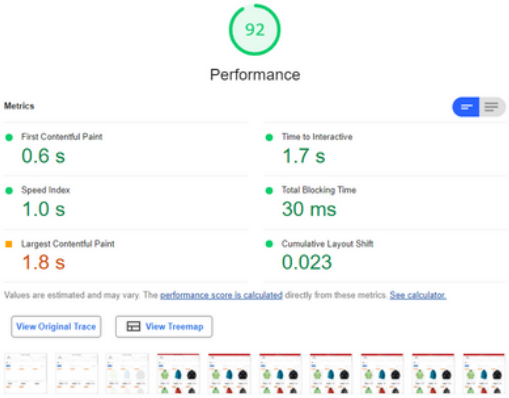
Start



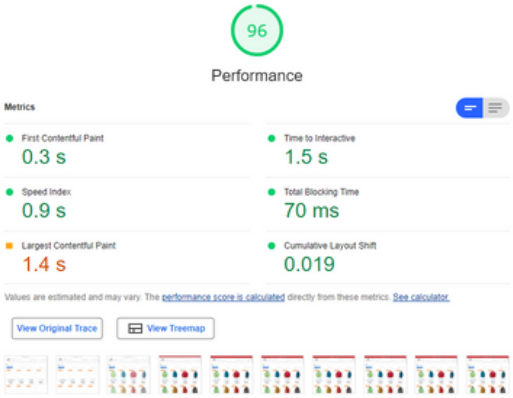
After 1st deployment



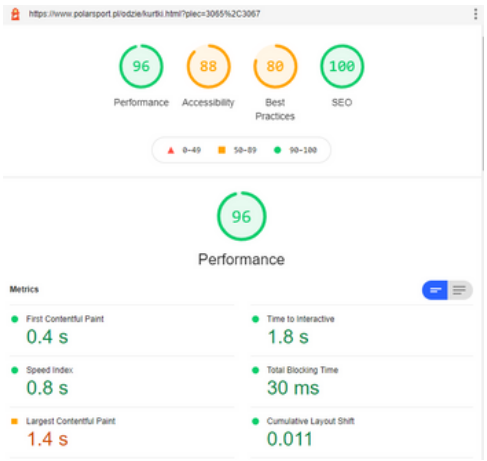
After 2nd deployment



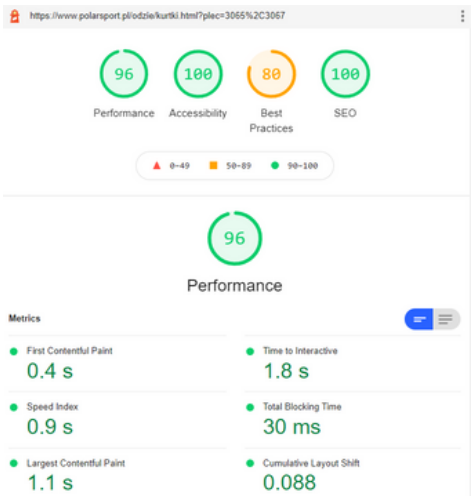
After 3rd deployment



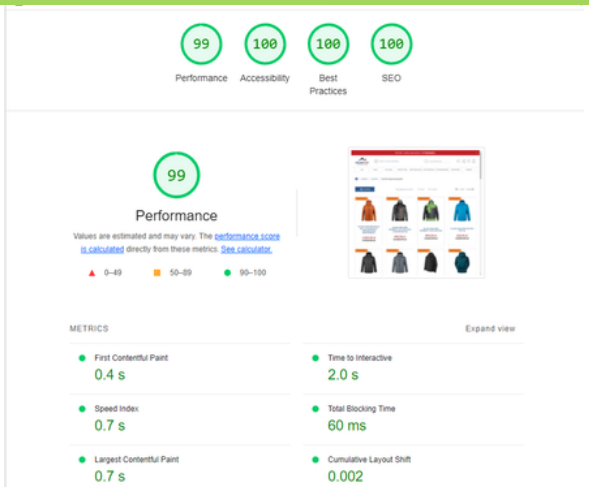
After 4th deployment



After 5th deployment

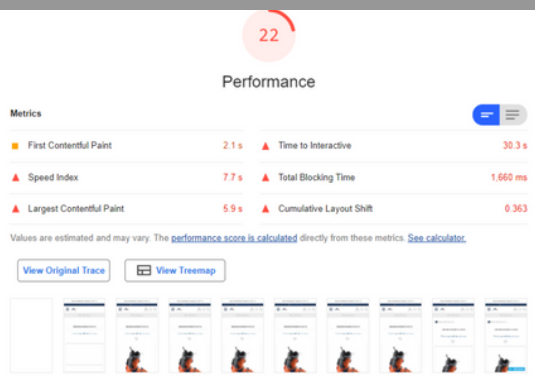


After 6th deployment

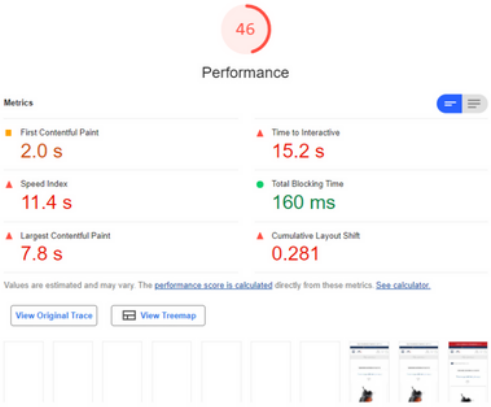


Product Page

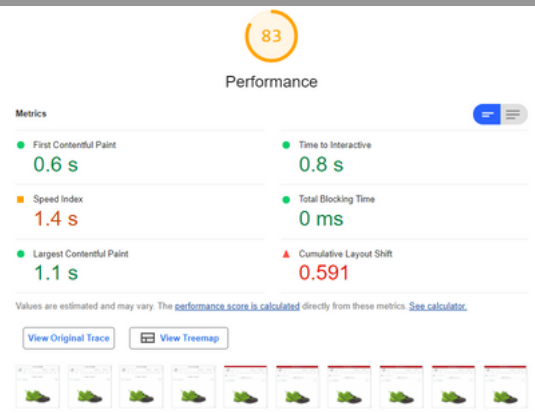
Start



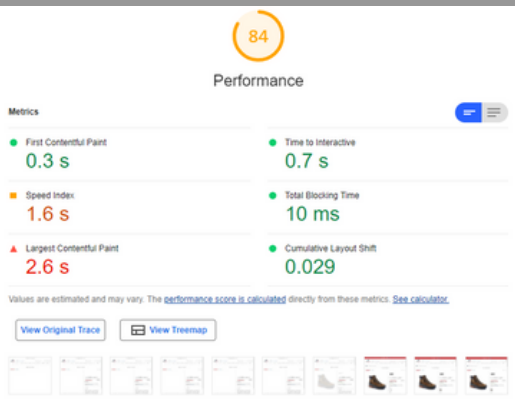
After 1st deployment



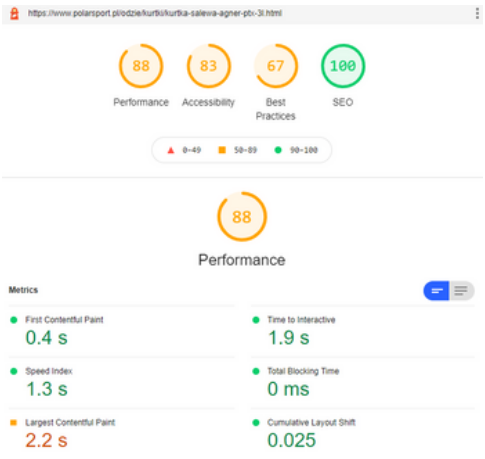
After 2nd deployment



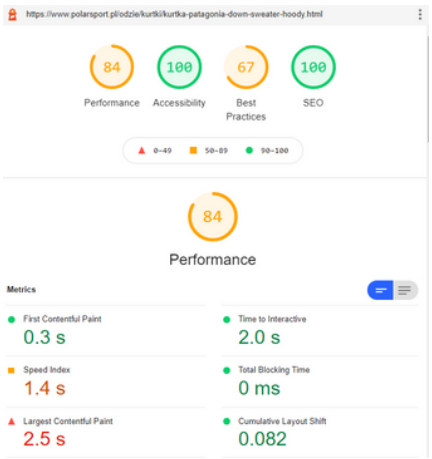
After 3rd deployment



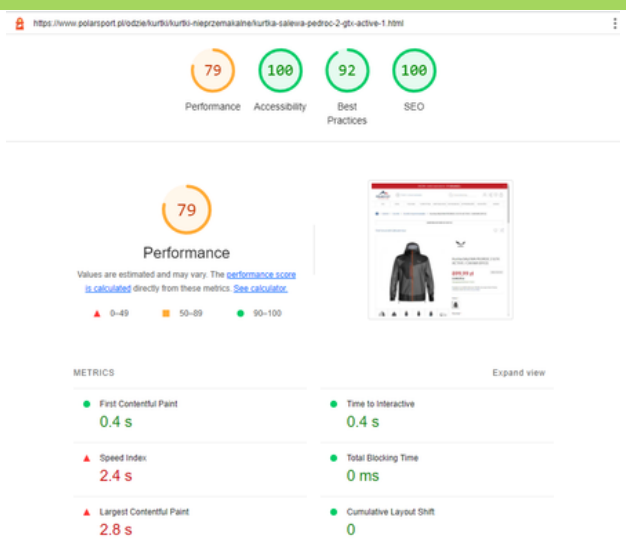
After 4th deployment



After 5th deployment



After 6th deployment



Core Web Vitals Improvements - our suggestions

Here's a generalized Magento website optimization process for Core Web Vitals improvements:

- Google fonts should be transformed into local files instead of requests from the google-fonts API. This would avoid a request to an external url that needs some additional time to establish communication.

- Add width and height attributes to all images across the website. By doing this, we reduce the time a browser needs to recalculate a layout. The image needs to at least have sizes that represent an image ratio.

- Use optimized images (jpg, png, gif). This can be achieved by using an online tool.

- Verify and improve lazy-load images. Take care of things that are currently not covered by Lazy-load script.

- Check, clean and sort the order of preloading assets (fonts, CSS, LCP elements like the first biggest image on the page).

- For some particular assets, mobile versions recognize which images can be provided in smaller and more responsive sizes. For example a homepage banner - see what we can do here by using the installed slider module.

- Create preconnect links for external services according to the order in the network tab - this step would let the browser know to start connections before receiving assets and unblock the main task thread

- Move all JS <script> tags to the end of the document, just before the </body> tag.

- Eliminate all CLS during a page load. Layout shifts would generate a layout reflow event, which is responsible for page repaint and makes the main thread task work longer than needed

- Recognize and eliminate all assets (by checking the network tab) which can be replaced by icon fonts or anything else. The main point is to reduce server requests for necessary assets.

- Create and implement a Critical.css

- Order all elements related to 3rd-party requests, like google-analytics, google-api, external services APIs. Reduce any which are not currently in use or can be avoided.

- Use an intersection event and polyfill script for IE to execute the necessary scripts only when they are visible to the end-user. This technique can be used to run sliders only when they are visible to the end-user, which would give the ability to save in the main thread work task.

- Reorder a JS execution by delaying some of the functionalities which block the initial main thread task. Execute the external script by some kind of particular event or after the window load event.

- Reduce the number of initial animations (css, js)

- Clean up CSS in sources to remove any old and unused code.

- Clean up JS in sources to remove any old and unused code.

- Use the performance tab to recognize long tasks. It would be helpful to find out the next steps to take.

To see real score progress use the dev-tools lighthouse tab, this is a laboratory diagnosis to see increases. Google insights are based on real customers' devices (CrUX). After any production deployment, the score should increase in a 28 day time period.