AUDIT PERFORMANCE REPORT

HTTP://TODOLISTME.NET/

TABLE OF CONTENTS

TODOLISTME.NET/ AUDIT PERFORMANCE REPORT INTRODUCTION 3

PERFORMANCE TEST 4

NETWORK 4

AUDITS 5

GOOGLE PAGESPEED INSIGHTS 6

YSLOW 7

ADDITIONAL TOOLS 8

IMPLEMENTATION 9

TODOLISTME.NET/ AUDIT PERFORMANCE REPORT INTRODUCTION

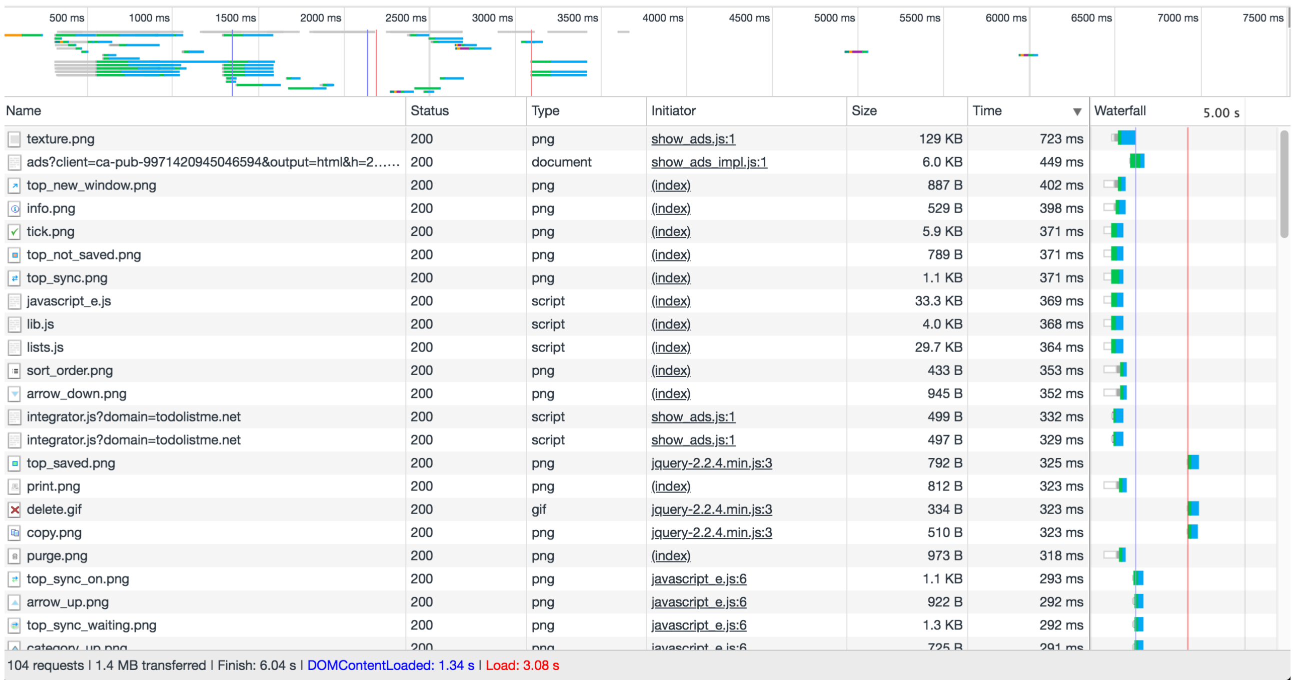
This report analyzes the performance of the ToDoApp's competitor todolistme.net in order to identify what errors can decrease a site's performance. The insights from this report will become helpful when scaling the ToDoApp in the future.

The tools for analyzing the site performance used in this audit are:

- Google Chrome DevTools
- Google PageSpeed Insights
- YSlow

PERFORMANCE TEST

Performance testing is one of the most important steps in software development. It helps to determine the usability and capability of the software, its speed, stability, scalability, reliability and more. For this audit three main tools are used: Google Chrome DevTools, Google PageSpeed Insights and YSlow.



Name	Status	Type	Initiator	Size	Time
texture.png	200	png	show_ads.js:1	129 KB	723 ms
ads?client=ca-pub-9971420945046594&output=html&h=2......	200	document	show_ads_impl.js:1	6.0 KB	449 ms
top_new_window.png	200	png	(index)	887 B	402 ms
info.png	200	png	(index)	529 B	398 ms
tick.png	200	png	(index)	5.9 KB	371 ms
top_not_saved.png	200	png	(index)	789 B	371 ms
top_sync.png	200	png	(index)	1.1 KB	371 ms
javascript_e.js	200	script	(index)	33.3 KB	369 ms
lib.js	200	script	(index)	4.0 KB	368 ms
lists.js	200	script	(index)	29.7 KB	364 ms
sort_order.png	200	png	(index)	433 B	353 ms
arrow_down.png	200	png	(index)	945 B	352 ms
integrator.js?domain=todolistme.net	200	script	show_ads.js:1	499 B	332 ms
integrator.js?domain=todolistme.net	200	script	show_ads.js:1	497 B	329 ms
top_saved.png	200	png	jquery-2.2.4.min.js:3	792 B	325 ms
print.png	200	png	(index)	812 B	323 ms
delete.gif	200	gif	jquery-2.2.4.min.js:3	334 B	323 ms
copy.png	200	png	jquery-2.2.4.min.js:3	510 B	323 ms
purge.png	200	png	(index)	973 B	318 ms
top_sync_on.png	200	png	javascript_e.js:6	1.1 KB	293 ms
arrow_up.png	200	png	javascript_e.js:6	922 B	292 ms
top_sync_waiting.png	200	png	javascript_e.js:6	1.3 KB	292 ms
category_up.png	200	png	javascript_e.js:6	725 B	291 ms

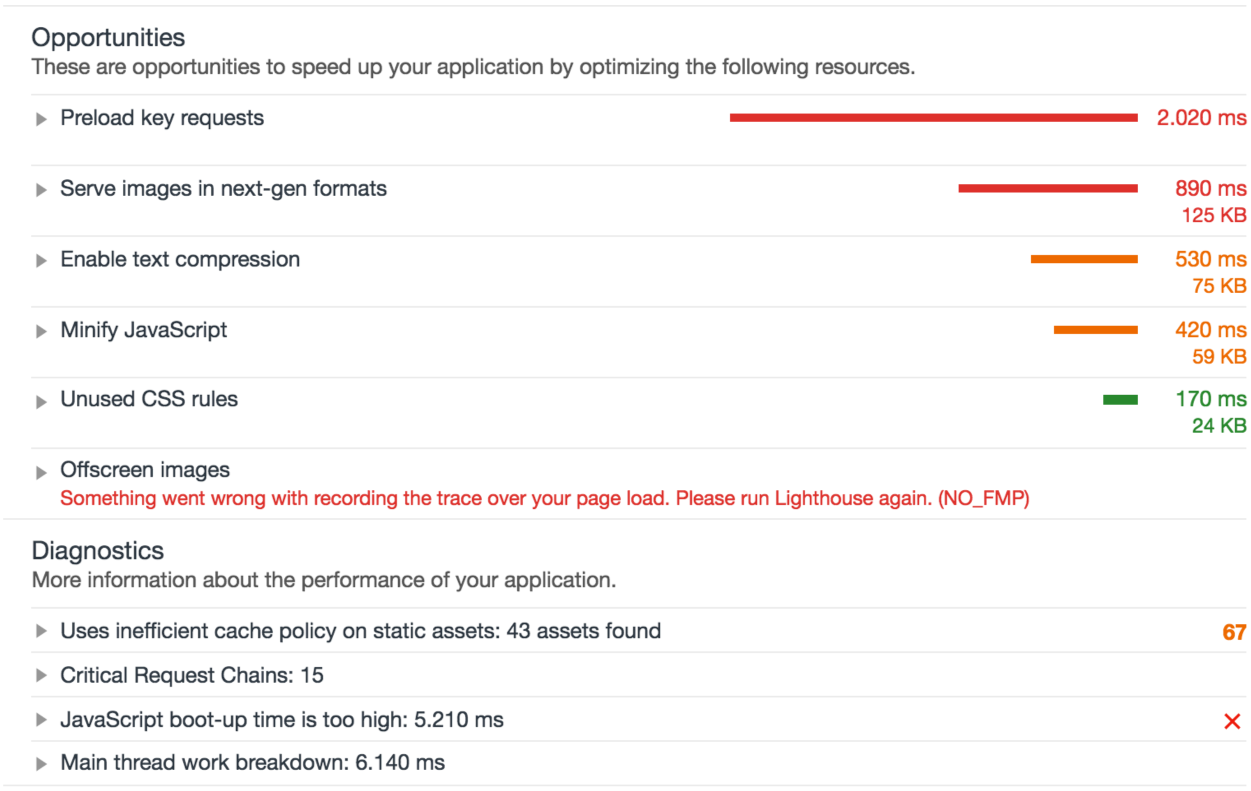
104 requests | 1.4 MB transferred | Finish: 6.04 s | DOMContentLoaded: 1.34 s | Load: 3.08 s

NETWORK

A first look at the Network tab of the DevTools reveals the site's loading time. When these are sorted from longest to shortest, as in the image above, it becomes clear which elements of the site cause performance issues. The longest loading times are due to the image icons in PNG format. The texture.png image has the longest loading time with 731.15 ms. After the request for this image is sent, only about 116 ms are spend waiting, the remaining 452.86 ms are spent downloading the image. This is clearly an indicator for these images – despite their small size – being loaded with unnecessary information which slows down the download.

It is also noteworthy to point out the loading time of the Google Ads document (second item in the list), which takes 449 ms to load. However, in this case only 131.8 ms are spend on the actual content download, whereas more than 300 ms are spend on the browser waiting for the first response from Google Ads.

At the very bottom of the report the number of HTTP requests is shown. In total 104 requests were made and 1.4 MB were transferred. The load time for the DOM is 1.34 seconds, which is not outrageously long, but could be improved by reducing the number of HTTP requests and minimizing the download size for the images.


Opportunities
These are opportunities to speed up your application by optimizing the following resources.

Preload key requests	2.020 ms
Serve images in next-gen formats	890 ms 125 KB
Enable text compression	530 ms 75 KB
Minify JavaScript	420 ms 59 KB
Unused CSS rules	170 ms 24 KB
Offscreen images Something went wrong with recording the trace over your page load. Please run Lighthouse again. (NO_FMP)	

Diagnostics
More information about the performance of your application.

Uses inefficient cache policy on static assets: 43 assets found	67
Critical Request Chains: 15	
JavaScript boot-up time is too high: 5.210 ms	✕
Main thread work breakdown: 6.140 ms	

AUDITS

Google Lighthouse is an open-source, automated tool for improving the quality of web pages. It has audits for performance, accessibility, progressive web apps, and more. In this audit we will mainly be focussing on the website's performance. Unfortunately, when running Lighthouse, the performance score is zero, due to an error when the website's trace over the page load is recorded. This error remains, even when the audit is run again. However, the results are still very helpful in discovering performance improvement opportunities (ass seen in the image above).

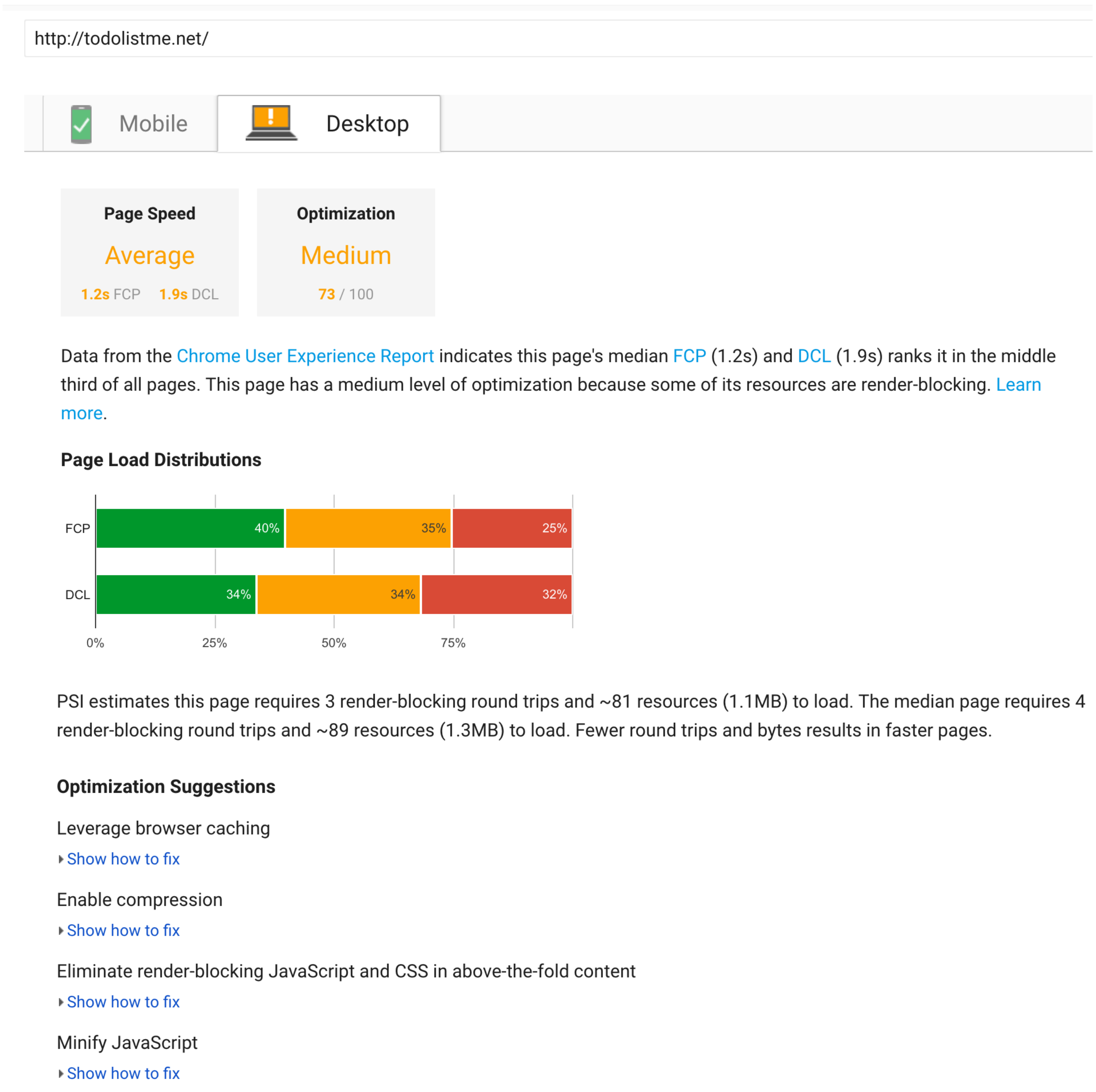
As shown in the opportunities tap, about 2 ms could be saved by preloading key requests. This means, by adding the tag <link rel="preload"> resources are stored locally in the browser, and are effectively inactive until they are referenced in the DOM or elsewhere. For this application, Lighthouse recommends preloading font, Google Ads and Facebook files.

As suspected from the information seen in the Network tab, Lighthouse recognizes the poor performance of the images and recommends next-gen formats such as JPEG. This solution is not ideal for all images in this application, as some of the them do need to be in PNG format to retain their background transparency. Lighthouse predicts a performance improvement of 890 ms by reducing the size of the images.
Another 530 ms could be saved by enabling text compression. Compression is the process of encoding information using fewer bits and eliminating unnecessary data. Files such as lists.js could be reduced by 76 percent, improving performance even further.

Minifying the JavaScript files is also a performance booster: Up tp 420 ms could be saved by removing unnecessary or redundant data - e.g. code comments and formatting, removing unused code, using shorter variable and function names, and so on.

Finally, by removing unused CSS rules from the application's own style sheet, a few more milliseconds can be saved.

Next, we'll be looking at Google PageSpeed Insights to see if this service offers further improvement opportunities.


http://todolistme.net/

Mobile | Desktop

Page Speed
Average
1.2s FCP 1.9s DCL

Optimization
Medium
73 / 100

Data from the Chrome User Experience Report indicates this page's median FCP (1.2s) and DCL (1.9s) ranks it in the middle third of all pages. This page has a medium level of optimization because some of its resources are render-blocking. Learn more.

Page Load Distributions

PSI estimates this page requires 3 render-blocking round trips and ~81 resources (1.1MB) to load. The median page requires 4 render-blocking round trips and ~89 resources (1.3MB) to load. Fewer round trips and bytes results in faster pages.

Optimization Suggestions

Leverage browser caching
▸ Show how to fix

Enable compression
▸ Show how to fix

Eliminate render-blocking JavaScript and CSS in above-the-fold content
▸ Show how to fix

Minify JavaScript
▸ Show how to fix

GOOGLE PAGESPEED INSIGHTS

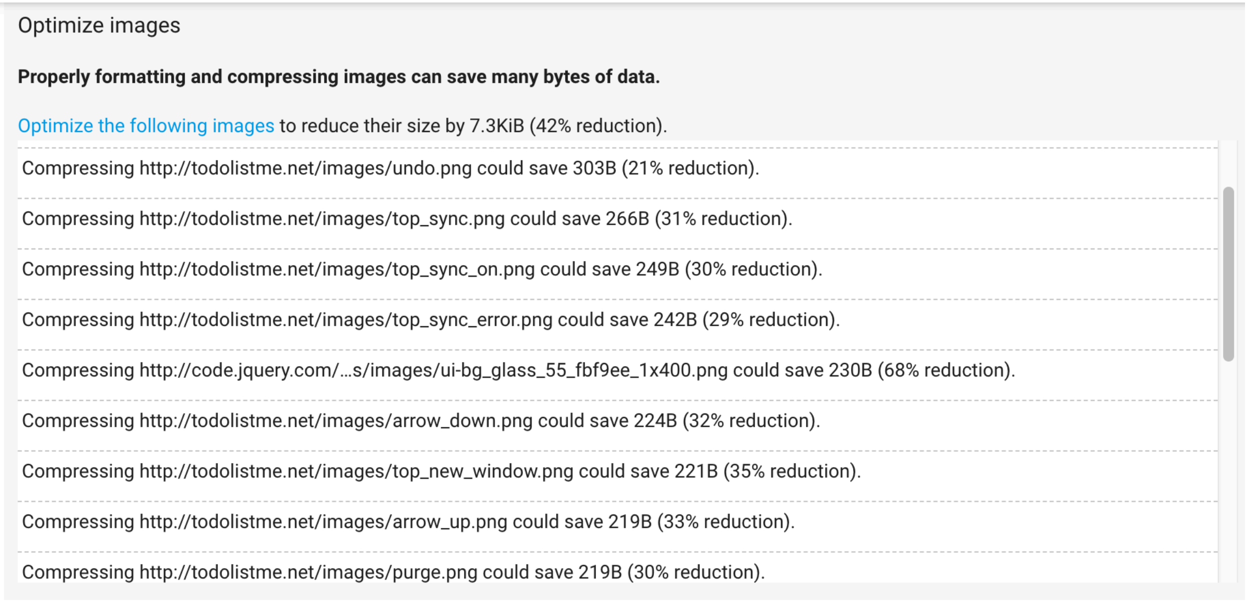
Google PageSpeed Insight analyzes any URL's performance and assigns it a score as well as giving suggestions to make the site faster. todolistme.net receives an average score for page speed. The load time for the DOM according to this tool is 1.9 seconds, with an average of 1.2 seconds, which is similar to the results seen in the DevTools.

The suggestions made by PageSpeed Insights are similar to those of the Lighthouse audit: enabling compression, minifying JavaScript, CSS and HTML, and optimizing image size.

Additionally, it is proposed to leverage browser caching. Setting an expiry date or a maximum age in the HTTP headers for static resources instructs the browser to load previously downloaded resources from the local disk rather than over the network. By using Expires headers these components become cacheable, which avoids unnecessary HTTP requests on subsequent page views.

It is also suggested to eliminate render-blocking JavaScript and CSS files. The application contains three files that cause a delay in rendering the site. These are a font file, a jQuery file and the application's own style sheet.

None of the above-the-fold content on the page can be rendered without waiting for these three files to load. Deferring or asynchronously loading these resources can solve the issue.



Optimize images

Properly formatting and compressing images can save many bytes of data.

Optimize the following images to reduce their size by 7.3KiB (42% reduction).

Compressing http://todolistme.net/images/undo.png could save 303B (21% reduction).

Compressing http://todolistme.net/images/top_sync.png could save 266B (31% reduction).

Compressing http://todolistme.net/images/top_sync_on.png could save 249B (30% reduction).

Compressing http://todolistme.net/images/top_sync_error.png could save 242B (29% reduction).

Compressing http://code.jquery.com/…s/images/ui-bg_glass_55_fbf9ee_1x400.png could save 230B (68% reduction).

Compressing http://todolistme.net/images/arrow_down.png could save 224B (32% reduction).

Compressing http://todolistme.net/images/top_new_window.png could save 221B (35% reduction).

Compressing http://todolistme.net/images/arrow_up.png could save 219B (33% reduction).

Compressing http://todolistme.net/images/purge.png could save 219B (30% reduction).

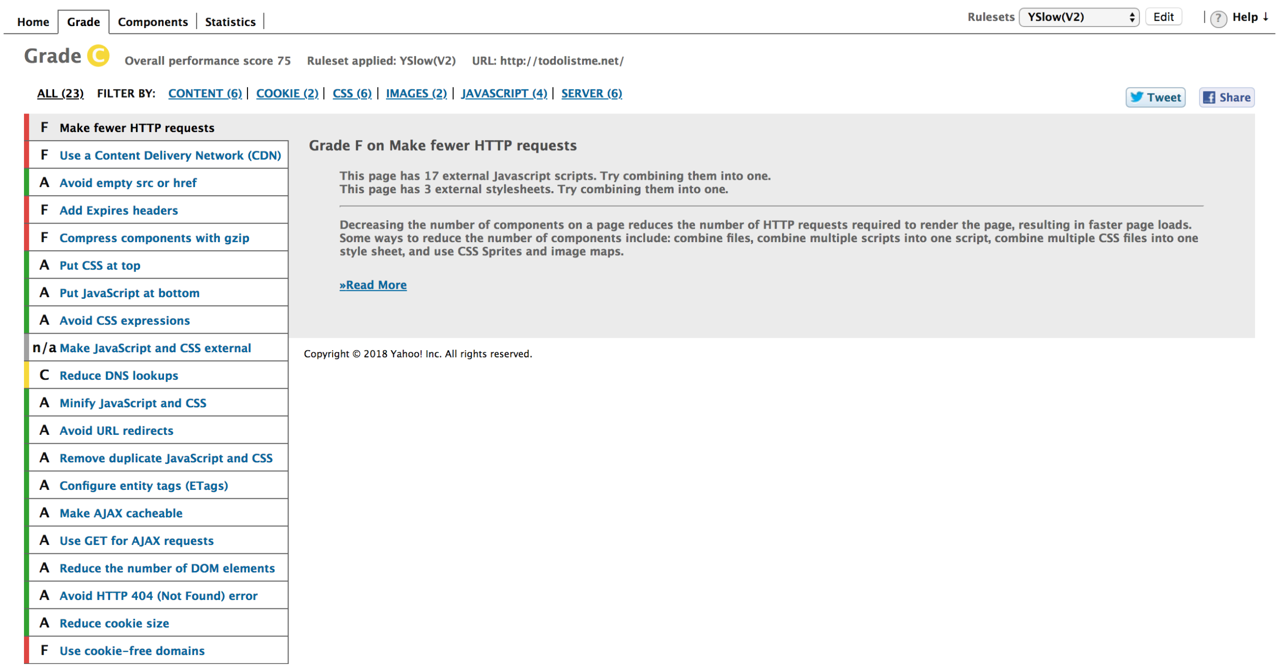
YSLOW

YSlow is a browser plugin that grades website performance and offers suggestions for improvement. The score for todolistme.net is a C with an overall performance score of 75. Many of the tested rules receive an A grade, but a few F grades bring down the score noticeably.

One of the improvement opportunities not mentioned by the previously used tools is making fewer HTTP requests. The application uses 17 external JavaScript files and 3 external CSS stylesheets. It is suggested to try and combine some of these files into one, so less HTTP requests are necessary.

YSlow also recommends using Content Delivery Networks (CDN) for up to 34 components in the application. Components like jQuery, GoogleFonts, GoogleAds or Facebook could be accessed via a CDN. According to YSlow, deploying content across multiple geographically dispersed servers helps users perceive that pages are loading faster.

Finally, it is suggested to use cookie-free domains. In the application are 24 components which are not cookie-free; mostly the PNG images. When a browser requests a static image and sends cookies with the request, the server ignores the cookies. These cookies are unnecessary network traffic. To workaround this problem, requested static components should be cookie-free by creating a subdomain and hosting them there.



Grade C Overall performance score 75 Ruleset applied: YSlow(V2) URL: http://todolistme.net/

ALL (23) FILTER BY: CONTENT (6) | COOKIE (2) | CSS (6) | IMAGES (2) | JAVASCRIPT (4) | SERVER (6)

F	Make fewer HTTP requests
F	Use a Content Delivery Network (CDN)
A	Avoid empty src or href
F	Add Expires headers
F	Compress components with gzip
A	Put CSS at top
A	Put JavaScript at bottom
A	Avoid CSS expressions
n/a	Make JavaScript and CSS external
C	Reduce DNS lookups
A	Minify JavaScript and CSS
A	Avoid URL redirects
A	Remove duplicate JavaScript and CSS
A	Configure entity tags (ETags)
A	Make AJAX cacheable
A	Use GET for AJAX requests
A	Reduce the number of DOM elements
A	Avoid HTTP 404 (Not Found) error
A	Reduce cookie size
F	Use cookie-free domains

Grade F on Make fewer HTTP requests

This page has 17 external Javascript scripts. Try combining them into one.
This page has 3 external stylesheets. Try combining them into one.

Decreasing the number of components on a page reduces the number of HTTP requests required to render the page, resulting in faster page loads. Some ways to reduce the number of components include: combine files, combine multiple scripts into one script, combine multiple CSS files into one style sheet, and use CSS Sprites and image maps.

»Read More

Copyright © 2018 Yahoo! Inc. All rights reserved.

ADDITIONAL TOOLS

This chapter covers additional tools that can solve some of the issues mentioned in the previous chapter.

One major performance issue for todolistme.net is the image size. This can simply be resolved by removing unnecessary information and metadata with free software like gulp-imagemin (https://www.npmjs.com/package/gulp-imagemin) or ImageOptim (https://imageoptim.com).

Since most of the used images are icons in this application, an icon font like Font Awesome (https://fontawesome.com/) should be used. This service can even be accessed via a CDN to further reduce load time.

Alternatively, CSS Sprites are also a solution. Rather than downloading multiple icon images, one image containing all icons can be created. The individual icons are retrieved by using pixel coordinates on the image. This method would drastically reduce the number of HTTP requests for this application. An easy way to generate CSS Sprites is via https://spritegen.website-performance.org.

To remove redundancies from the code, there are many tools to minify JavaScript, HTML and CSS code:

- HTML: HTMLMinifier (https://github.com/kangax/html-minifier)
- CSS: CSSNano (https://github.com/ben-eb/cssnano) and csso (https://github.com/css/csso).
- avaScript UglifyJS (https://github.com/mishoo/UglifyJS2), Closure Compiler (https://developers.google.com/closure/compiler) and JavaScript Minifier (https://javascript-minifier.com/)

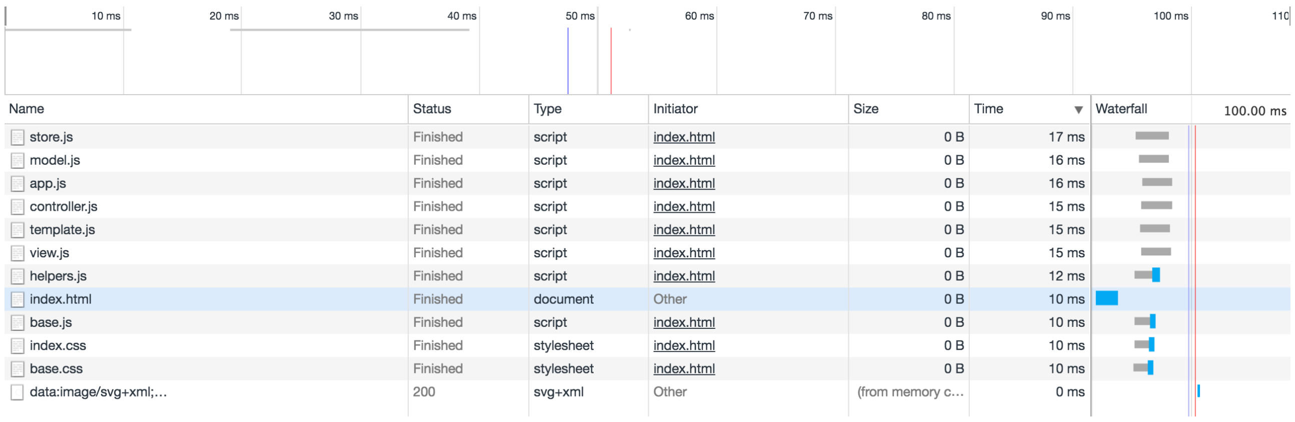
Furthermore: This application uses a jQuery UI version that is not minified. But as mentioned earlier, it is advisable to use a CDN for frameworks to improve performance.

To summarize, this audit has shown that todolistme.net has many opportunities for improving its performance, load times and therefore the user's experience. Now, let's have a look at how these findings can be applied to the ToDoListApp.

IMPLEMENTATION

Compared to the functionality of todolistme.net, the ToDoListApp is still in its early stages. Nonetheless, even here is some room for improvement.

A quick look at the Network tab from the Google DevTools reveals a similar issue compared to its competitor: A multitude of JavaScript and CSS files that could easily be aggregated into one file each, in order to reduce the number of HTTP requests.

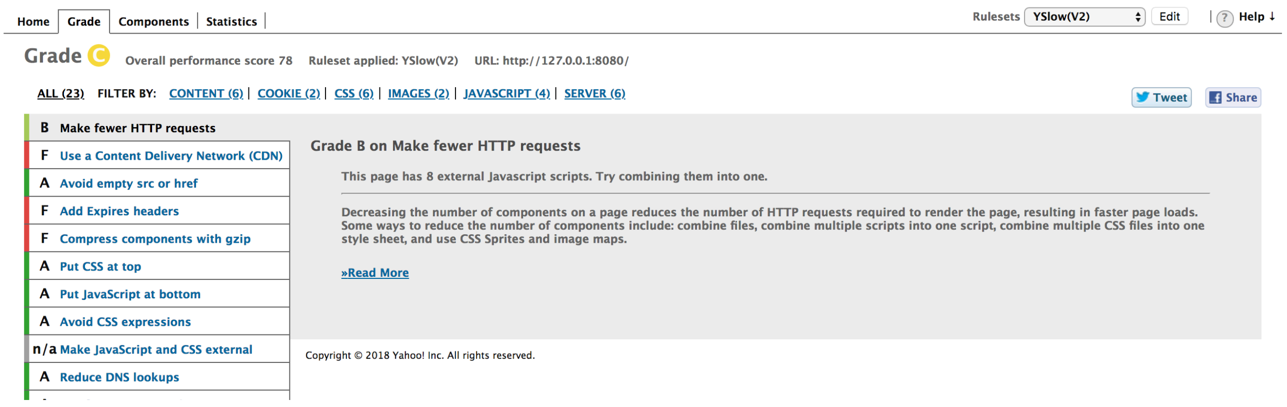


Name	Status	Type	Initiator	Size	Time
store.js	Finished	script	index.html	0 B	17 ms
model.js	Finished	script	index.html	0 B	16 ms
app.js	Finished	script	index.html	0 B	16 ms
controller.js	Finished	script	index.html	0 B	15 ms
template.js	Finished	script	index.html	0 B	15 ms
view.js	Finished	script	index.html	0 B	15 ms
helpers.js	Finished	script	index.html	0 B	12 ms
index.html	Finished	document	Other	0 B	10 ms
base.js	Finished	script	index.html	0 B	10 ms
index.css	Finished	stylesheet	index.html	0 B	10 ms
base.css	Finished	stylesheet	index.html	0 B	10 ms
data:image/svg+xml;...	200	svg+xml	Other	(from memory c...	0 ms

Minifier tools for all the HTML, CSS and JS code would also be of great benefit to further reduce the file size.

By running the application through YSlow, it becomes evident, that these changes are indeed recommended. Furthermore, it is advisable to add Expires headers to reduce HTTP requests.

The overall performance for the ToDoListApp at this stage is similar to that of its competitor: a grade C with an overall performance score of 78.



Grade C Overall performance score 78 Ruleset applied: YSlow(V2) URL: http://127.0.0.1:8080/

ALL (23) FILTER BY: CONTENT (6) | COOKIE (2) | CSS (6) | IMAGES (2) | JAVASCRIPT (4) | SERVER (6)

B	Make fewer HTTP requests
F	Use a Content Delivery Network (CDN)
A	Avoid empty src or href
F	Add Expires headers
F	Compress components with gzip
A	Put CSS at top
A	Put JavaScript at bottom
A	Avoid CSS expressions
n/a	Make JavaScript and CSS external
A	Reduce DNS lookups

Grade B on Make fewer HTTP requests

This page has 8 external Javascript scripts. Try combining them into one.

Decreasing the number of components on a page reduces the number of HTTP requests required to render the page, resulting in faster page loads. Some ways to reduce the number of components include: combine files, combine multiple scripts into one script, combine multiple CSS files into one style sheet, and use CSS Sprites and image maps.

»Read More

Copyright © 2018 Yahoo! Inc. All rights reserved.

These findings emphasize the importance of eliminating performance hurdles and implementing best practices in the early stages. As this report shows, even a seemingly simple application can be riddled with performance-hindering obstacles that might not seem significant at first glance, but when added up, can drastically slow down load times and therefore a user's experience.