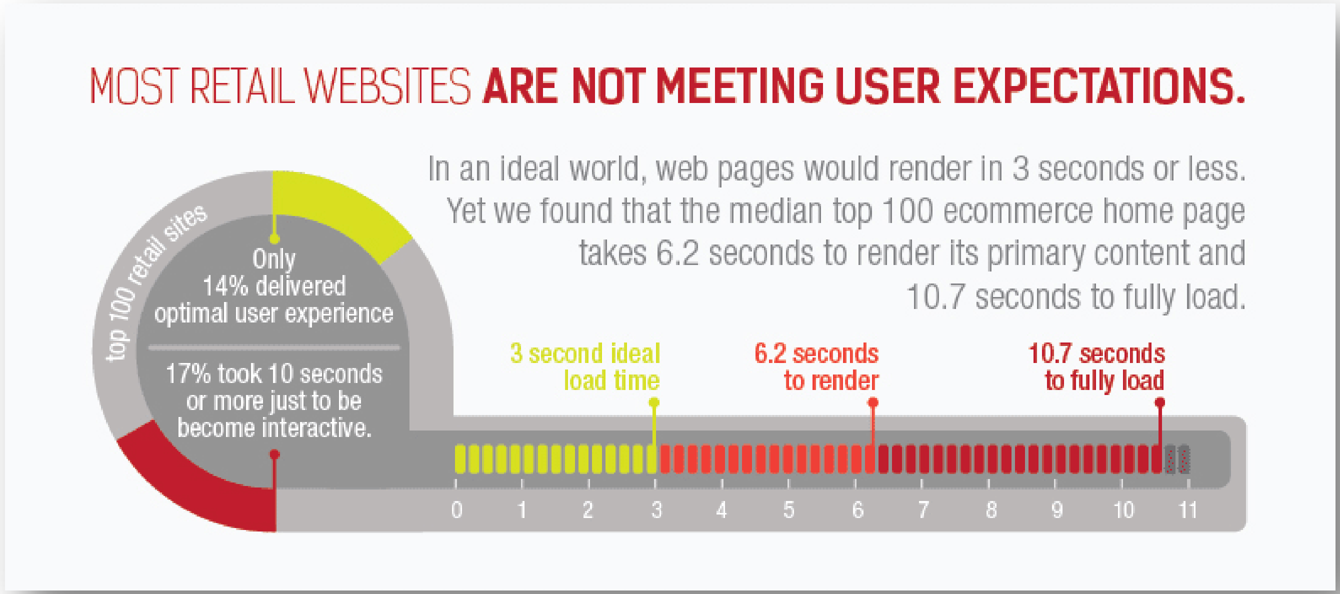
MAVEN eCommerce

Magento Performance Optimization Whitepaper

Websites Decay Fast

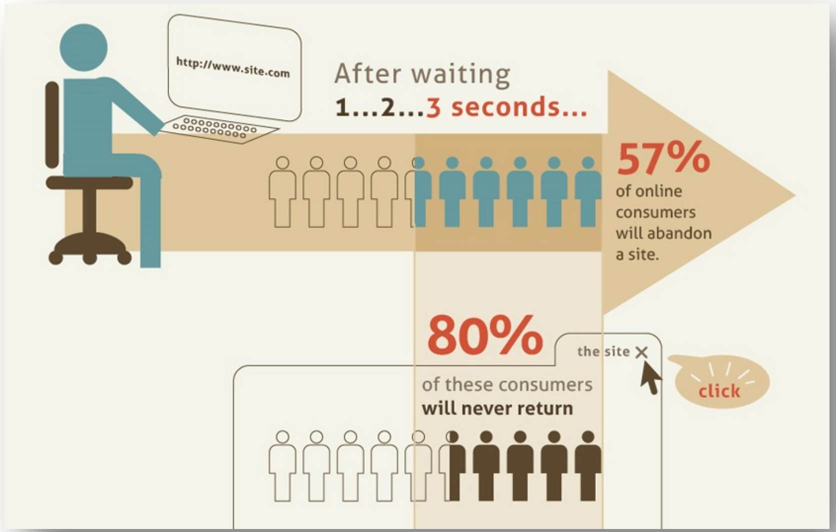
Even one year is a long time on the web, as Magento platform is changing, evolving as it takes on new e-commerce challenges one at a time in steady, ongoing iteration. When you created your store your main goal was most likely to create an easy to use and trustworthy resource that would reflect your focus on customer service and convenience. You probably did that by crafting a delightful user experience, subsequently adding many useful new features over time.

However, this constant addition of ever-new features tends to slow down the websites to the point where they are not meeting even the most basic user expectations in terms of performance:



How old is your Magento store?

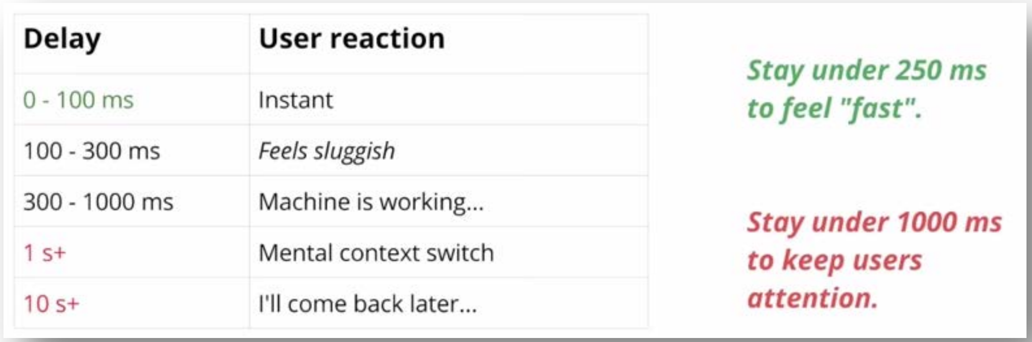
In the world of e-commerce, the average web page has grown 151% over the past three years, with over half that bulk coming from images. No wonder retailers around the world find the industry's 3-second page load benchmark so elusive! Moreover, over 92% of retailers are failing to hit this mark and running as high as eight seconds or more.


http://www.site.com
After waiting
1...2...3 seconds...
57%
of online consumers will abandon a site.
80%
of these consumers will never return
the site ×
click

With the rise of Mobile those very assets that helped you to cater to your users are now likely becoming a major performance bottleneck. Every new feature slowed down your site, with style sheets, markup and JavaScript not making it any leaner. Redesigns to make your store responsive, to embrace flat design and to use retina-grade images might have included quick fixes and compromises. Your site grew, accumulating inconsistencies layout issues. Combined, these decisions have made your website slow, with a loading sequence that didn't prioritize the user experience.

Why Performance Matters

When asked about how fast your website should be, the short answer is: “As fast as possible!”
We tend to assume that users have great Internet connectivity. However, this is not always the case - especially when it comes to mobile device users - as customers may be browsing your website on-the-go over slow 3G Internet connection. How long does it take for the most important page content to load on slow connections? Since no one likes waiting, optimizing the store loading speed makes business sense. Consider this loading speed perception study conducted on the web users:



Delay	User reaction
0 - 100 ms	Instant
100 - 300 ms	*Feels sluggish*
300 - 1000 ms	Machine is working...
1 s+	Mental context switch
10 s+	I'll come back later...

Stay under 250 ms to feel "fast".

Stay under 1000 ms to keep users attention.

Whether your website is unavailable or slow makes no difference for the average user - the effect on the bottom line is devastating in both cases. This is why:

- Websites loading longer than 1.5 seconds are ranked as “slow”, have poor SEO and low AdWords Page Quality (Google)
- Every additional 0.5 second load time decreases traffic by 5-20% (Yahoo and Google)
- Page load time decrease from 7 to 2 seconds alone raises revenue by 10% (Shopzilla)
- Every additional 1/10th of a second page load time decreases sales by 1% (Amazon)
- A page that takes 5 seconds or more to load loses 20% of visitors (Forrester)
- 80% of visitors won’t come back if the site is slow (75% in 2013, 64% in 2006)
- 70% of revenue comes from the fastest visits

Defining Performance

The most important technical performance data metrics for your web page are:

1. **Time to First Byte**
2. **domContentLoaded**
3. **Document Complete**

The first, **Time to First Byte**, is often used as a measure of the web server and network efficiency. If this metric is within 0.2-0.8 seconds for the target user geography and connectivity speed, chances are further server optimization is unlikely to yield noticeable improvements. Time to First Byte metric over 1 second requires investigation of your server or your hosting provider.

The second metric called **domContentLoaded** measures how well your web page components are optimized for displaying in the visitor's browser window. This metric should be lower than 2.5 - 3 seconds, and ideally should be as close as possible to the 0.9 - 1 second range. **This is the single metric that defines whether your visitors will consider your website *fast*.**

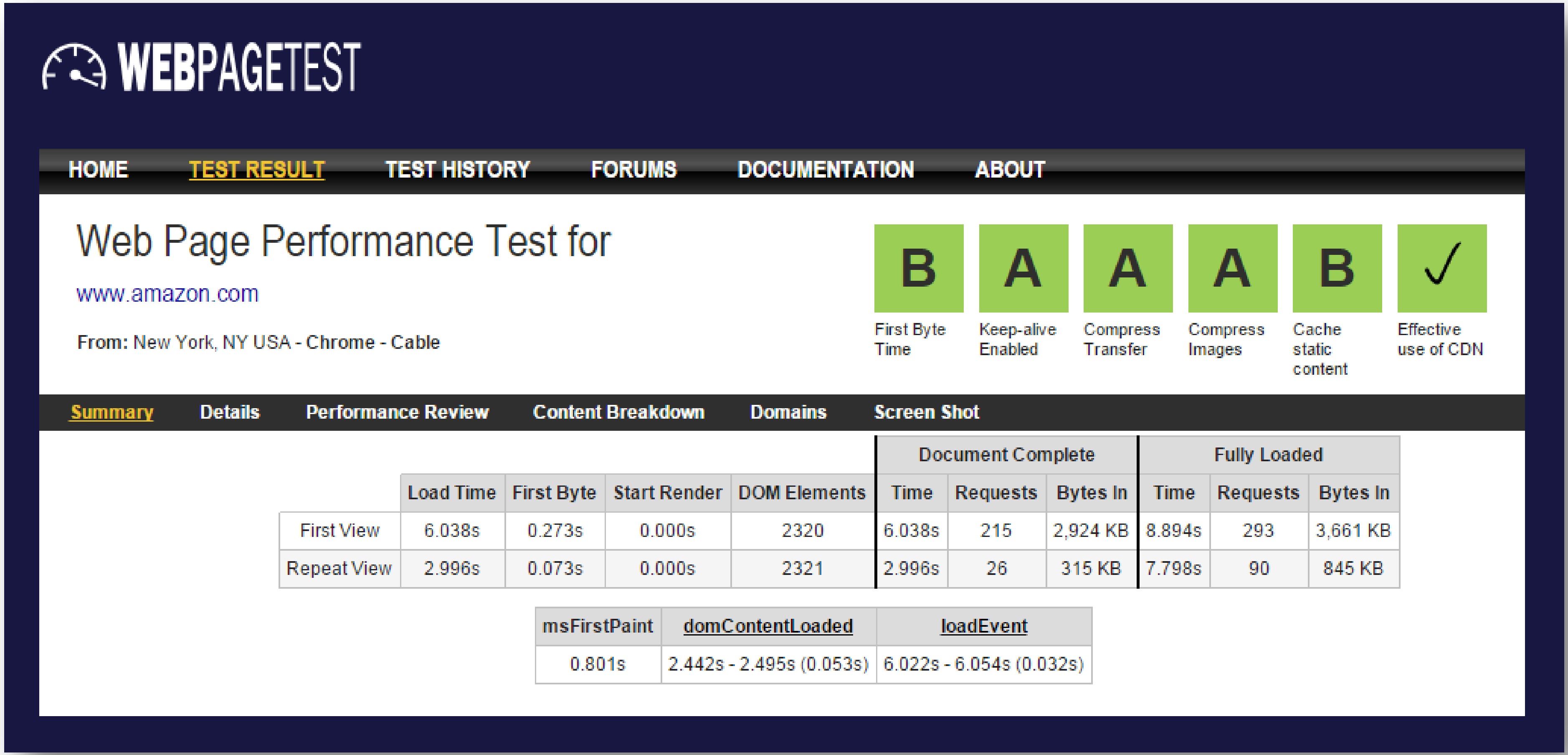
Why the 0.9 - 1 second range? As mentioned on the user perception table on the previous page, web page visitors must see *something* appear in less than one second for the experience to *feel* instant. When something takes more than one second, most people notice they are waiting. After 2.5 seconds, attention switches to a different task and websites lose users. This second metric, **domContentLoaded**, determines when user's browser responds and begins showing content that is visible to the user. No matter the web page size, domContentLoaded time will determine the user's perception of how fast the web page is loading!

Last metric called **Document Complete** marks the point at which all text and images appear and the page is considered to be loaded.

Until recently, the most commonly used performance metric used to be the *average page loading (Document Complete) time*. But given that the average time to fully load a page doesn't measure when a user can *start seeing usable content* makes that measurement not very reliable.

Good Performance Example

Let's examine performance metrics using Amazon as a case study:



Web Page Performance Test for

www.amazon.com

From: New York, NY USA - **Chrome** - **Cable**

B	A	A	A	B	✓
First Byte Time	Keep-alive Enabled	Compress Transfer	Compress Images	Cache static content	Effective use of CDN

					Document Complete			Fully Loaded		
	Load Time	First Byte	Start Render	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In
First View	6.038s	0.273s	0.000s	2320	6.038s	215	2,924 KB	8.894s	293	3,661 KB
Repeat View	2.996s	0.073s	0.000s	2321	2.996s	26	315 KB	7.798s	90	845 KB

msFirstPaint	domContentLoaded	loadEvent
0.801s	2.442s - 2.495s (0.053s)	6.022s - 6.054s (0.032s)

Here are the metrics for Amazon, based on WebPageTest results:

1. Time to First Byte: 0.27 seconds - excellent
2. domContentLoaded: 2.44 seconds - good
3. Document Complete: 6.04 seconds – average

Showing Visible Content Fast

The metric that shows optimized performance is not how long it takes to show content of entire page faster (the Document Complete metric), but ***to load visible page content as soon as possible***. The main metric for this event is called **domContentLoaded**.

WebPageTest estimates that on both First and Repeat view Amazon barely manages to load the visible part of the page (domContentLoaded) under 2.5 seconds. Compare that with almost 8.8 seconds on the First View and 7.9 seconds on the Repeat View for the Full Load time of the page and it will become evident that ***the average page load speed is not that useful as a metric in measuring performance***. Considering that Amazon is a heavily trafficked marketplace with lots of features (hence the hefty 3.66 Mb web page size), the three performance metrics are quite impressive.

First View versus Repeat View

There are two metrics that measure how efficient is the website in terms of speeding up the page on the second view and after called **Repeat View**. The first time a visitor enters a web address the browser has to load every component from the page - a First View. If the user closes the browser session and then enters the web address, some of the components would be already available in the browser memory (cached) and that user’s webpage will load faster - this is called a Repeat View.

The Performance Budget Approach to Magento Optimization

How can you achieve performance improvements? Want to figure out how fast your website could be? One way to do so is to implement some concepts from the performance budget approach for your Magento store.

Optimizing website loading speed does not involve magic, but does require a specific approach to web page components. First, audit your website for legacy components that are installed, but are no longer in use - and remove them. Second, re-structure content loading to deliver usable content as fast as possible.

Performance budget concepts focus on rewriting the HTTP loading sequence, pushing back all content that isn't required to start rendering the page. The result is a site that loads as fast as possible in all settings and on all connections, both unreliable and stable, slow and fast. The entire philosophy can be described in the following three steps:

1. Visible content is loaded first

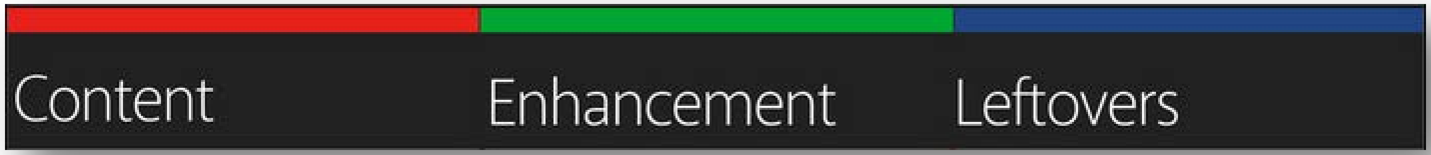
Page speed can be improved by delaying loading and rendering any content that appears below the area that is visible to the user. Placeholder tags can also inserted to specify the correct height and width for components that are still loading to avoid the page content from jumping (reflowing) as the remainder of the page is being loaded.

2. Interactive features are deferred

Carousels (or sliders) are often not optimized and are not among the first elements to load. When these elements cannot be optimized a better idea might be to defer or to remove them to improve the user experience.

3. Non-essential scripts loaded last

Many script libraries aren't needed until after a page has finished rendering. Downloading and parsing these scripts can safely be deferred until after the onload event. For example, scripts that support interactivity, such as drag and drop, can't even be used before the user has even seen the page. Deferring supporting files helps the users to see the usable content quicker.



The approach is best performed by assigning every piece of content to one of three groups based on their importance to the user:

- Core content: essential HTML, critical CSS, usable non-JS enhanced experience
- Enhancement (after all core is loaded): JS, geolocation, touch, enhanced CSS, Web fonts, widgets
- Leftovers (everything not sought by users): analytics, advertising, third-party content

Optimized content categories strictly separate the loading process throughout these three phases, so that the loading of the Core content is never blocked by any resources grouped in Enhancement or Leftovers. In other words, the critical rendering path required for the content to start displaying is shortened by razor-sharp focus on the content seen by the user and by deferring supporting and meta files to the back of the loading queue.

What is Actually Optimized When Improving Performance?

A Magento performance optimization roadmap involves making performance improvements in the following areas:

- Content / Images
- CSS
- JavaScript (JS)
- Server
- Cache

The initial process should optimize the critical rendering path for content delivery, and defers all components not needed for the user. Most important information should be placed in the first 14 KB. Content should be loaded by modules, core JavaScript can be loaded asynchronously (while JS is being loaded, browser still can parse the page and show content), and server-side device detection can render content based on browser classification. CSS, HTML and JavaScript components should be inspected: unused ones should be removed, the remaining ones combined. Here are detailed guidelines of the optimization process:

Content / Images

Optimization requires taking a closer look at each page component and asking what's absolutely required in order to render the page initially. Those components need to be re-loaded, and rest slated for post-loading. Candidates for post-loading include hidden content (content that appears after a user action) and images below the part of the page initially seen by the user.

Implement a lossless image compression solution

Most images can be reduced in size without compromising on quality. Compress the images and implement a solution that automatically decreases page size without affecting user experience. Also, use PNG or GIF format rather than Jpeg and don’t use transparency.

Combine images with CSS image sprites

An image sprite is a collection of images combined into a single file. A web page with many images can take a long time to load and generates multiple server requests. Using image sprites will reduce the number of server requests and save bandwidth.

Serve scaled images

Resizing images to exact standardized dimensions decreases file size while making responsive web design possible.

Use data URIs

Using data URIs for smaller images cuts the number of requests by inlining images into HTML or CSS files, making the images immediately available as the document is being downloaded.

Specify image dimensions

Specifying image dimensions in the HTML or CSS prevents the browser reflows by proactively providing the image dimensions on the fly.

Use a Content Delivery Network (CDN)

Leveraging a CDN to cache static assets in data centers around the world deliver a website faster.

Specify a character set

Making the fonts and characters used by the website explicit in the HTML response headers allow browsers to begin parsing HTML more efficiently and to execute scripts immediately.

Reduce and front-load fonts

Removing unused custom fonts and styles decreases the initial load time. In any case fonts can be further condensed and front-loaded with AJAX.

Remove bad requests

Removing broken links, missing pages or other asset requests that result in 404 errors reduces the number of requests for non-existent or disabled resources, speeding up the page load time.

Use domain sharding for parallel processing

Serve resources from two different hostnames to increase parallel processing, facilitating more simultaneous downloads that the browser would previously allow.

Serve assets from a single URL

Serving all assets and files from a single URL eliminates any extra overhead with duplicate downloads and extra round trips by the server.

Serve static content from a cookie-free domain

Serving static resources from a cookieless domain reduces the total size of requests made for a page.

Minimize DNS lookups

Reducing the number of unique hostnames from which assets are served cuts down on the number of DNS resolutions and round trips by the browser.

Minify HTML

Compress the HTML to remove comments, empty space between tags, unnecessary closing tags to reduce the overall HTML file size and speed up the parsing and execution of a webpage

CSS

Dramatic performance improvement may require a departure from the default HTTP loading path to deliver content quickly. Loading critical CSS (defined as the top portion of the page) within one single HTTP request, followed by the rest of the CSS once the page has rendered makes content appear to render more quickly.

Combine CSS Files

Combining (or "concatenating") the CSS files together decreases the number of round trips the browser has to make to the server, decreasing the page load time. Custom functionality plugins in the <head> part need to be manually reviewed and removed/turned-off before files are combined.

Minify CSS

Compressing, or "minifying" the CSS files for best practices, such as removing all the unused spaces and superfluous punctuation reduces the overall file size, decreasing page load times

Put “critical” CSS in <head>

Putting the CSS in the <head> of an HTML document while removing inline style blocks and using <link> CSS files in the <head> section improves browser execution and display load times

Load CSS before JavaScript

Loading external CSS files first (before external and inline JavaScript files) in the <head> enables browser execution without delays and makes pages appear to be loading faster by loading the page progressively; that is, the browser will display content as soon as possible - an especially important change for pages with a lot of content and for users on slower Internet connections

Remove unused CSS

Cleaning code and removing any unused CSS to decrease the overall file size improves browser execution and page load times

Enable gzip compression

Compress CSS files with gzip to reduce asset count and file size

Avoid CSS @import

Avoiding the use of CSS @import decreases the number of requests and allows the browser to download CSS files in parallel. Use of the traditional <link> tag ensures parallelized downloads

Avoid CSS expressions

CSS expressions slow down browser execution. The use of standard CSS properties or core JavaScript improves rendering for the non-modern browser and IE users.

JavaScript

The problem caused by scripts is that they block parallel downloads. If a script can be deferred, it can also be moved to the bottom of the page. Because the optimization process requires the removal of all unnecessary assets from the critical rendering path, JavaScript optimization often involves replacing jQuery with lightweight modular JavaScript components. We investigate the prospect of removing jQuery altogether by decoupling the jQuery dependencies from the library, the possibility of replacing it with a PHP solution and look into the impact of writing custom scripts. We defer the loading of all JavaScripts identified and clear a path in the header for HTML and CSS.

Combine JavaScript Files

Combining (or "concatenating") JavaScript files together decreases the number of requests and round trips between the browser and the server, lowering the page load time

Minify JavaScript

Compressing, or "minifying" the JavaScript files is done by eliminating unnecessary breaks, extra spaces and indentation. This reduces the overall size of the JavaScript files and increase page load speed.

Load 3-rd Part assets asynchronously

Loading third party assets asynchronously will not block important resources from loading. These are mainly social sharing widgets (Facebook, Twitter, etc.) and analytics tracking (Google Analytics, etc.)

Load JavaScript after CSS

Loading external and inline JavaScript after the CSS files in the <head> will not block browser download and execution of critical assets.

Defer loading of JavaScript

Deferring loading JavaScript reduces the initial download size and allows downloading of other resources, speeding up execution and rendering.

Enable gzip compression

Compress JavaScript file with gzip to reduce size.

Use intelligent script loading for parallel processing

Using intelligent script loaders downloads page assets asynchronously, bypassing the problem of blocked scripts common with parallel processing.

Avoid using document.write()

Avoiding the document.write() to load external resources (especially early in the document) prevents slow display and long load times

Include <noscript>

Faster page rendering is possible when pages that do not work with JavaScript are specifically marked.

Remove unused and duplicate components

Removing legacy jQuery files that no longer serve a useful purpose (such as Prototype-JS files like scriptaculos) declutters the page. Scripts and components that are native to the theme, but are not in use (like sliders that are not used) need to be turned off. It hurts performance to include the same JavaScript file twice in one page, duplicates need to be removed.

Update jQuery plugins

If the use of jQuery cannot be avoided or replaced, it needs to be updated along with all the plugins using it.

Server

Flush the Buffer Early

Consider flushing right after the HEAD because the HTML for the head is usually easier to produce and it allows to include any CSS and JavaScript files for the browser to start fetching in parallel while the backend is still processing.

Use efficient web servers

NGINX is a high performance, high concurrency edge web server with the key features to build modern, efficient, accelerated web infrastructure.

Enable Load Balancing

Setting up load balancing when using multiple servers offers better user experience for all users and faster performance.

Configure the PHP Server

Maintained PHP accelerators (Zend Opcache, XCache, Nusphere PhpExpress) decrease page load time by reducing parsing, especially after the first page load and will optimize the number of concurrent users by using the same server capacity more efficiently.

Split the database and the webserver (if possible)

A webserver and a database server have different requirements. A database server needs fast hard disks (e.g. SSD), much memory and not that much CPU. A webserver needs more CPU and less memory.

Uninstall xdebug or zend debugger on production server

Great tools for a development or testing environment are not a good fit with the live, production server. In some projects xdebug can have a performance impact of 10-15%, especially with high traffic load.

Cache

Caching refers to a component that transparently stores data so that when users visit the same page again it loads faster. The greater the number of requests that can be served from the cache, the faster the overall system performance becomes.

Use browser caching

Setting an expiry date or a maximum age in the HTTP headers for static resources directs the browser to load previously downloaded resources from the local storage rather than over the network, increasing loading speed.

Make redirects cacheable

Caching redirects by a user's browser when possible decreases page load times for repeat visitors.

Use proxy caching

Enabling public caching in the HTTP headers for static assets allows the browser to download resources from a nearby proxy server rather than from a remote origin server, speeding up load times.

Slider content caching

Identifying HTML-inline sliders that do not use cache and replacing them with a modern slider not only improves makes pages more responsive, but also improves loading performance.

Set up a specialized cache solution

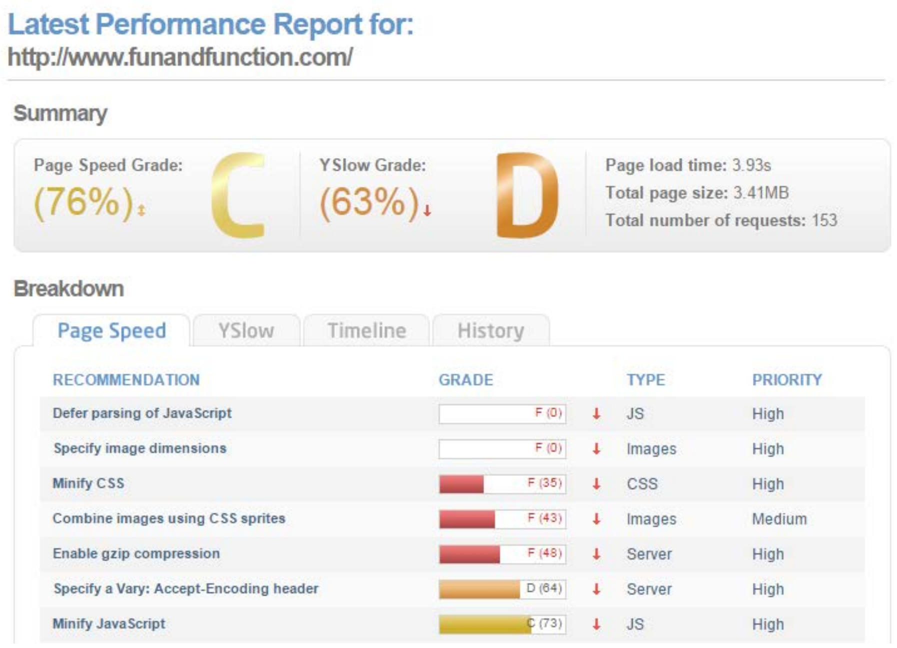
Setting up Varnish (an HTTP accelerator), Memecached (general-purpose distributed memory caching system) can reduce the number of times an external data source (such as a database or API) must be read.

A Real Life Example

Fun and Function, a Developmental Toys E-Commerce Magento Store

Prior to optimization in the early part of 2015, the Fun and Function website suffered from several deficiencies, some of which were of high priority: JavaScript, CSS, Images and Server areas all needed work.

Before Optimization:



Latest Performance Report for:
http://www.funandfunction.com/

Summary

Page Speed Grade: (76%)↕ C

YSlow Grade: (63%)↓ D

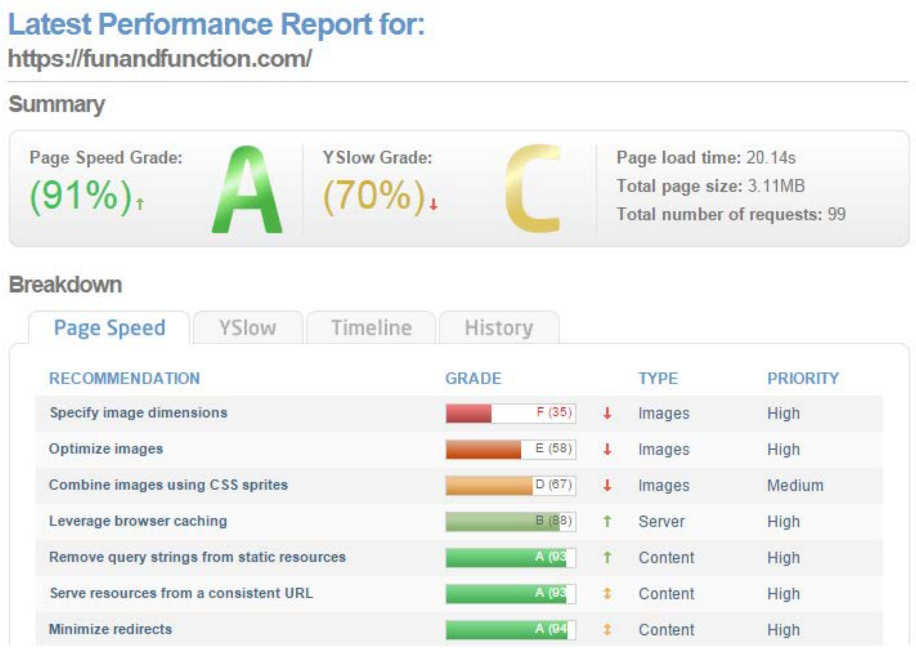
Page load time: 3.93s
Total page size: 3.41MB
Total number of requests: 153

Breakdown

Page Speed | YSlow | Timeline | History

RECOMMENDATION	GRADE		TYPE	PRIORITY
Defer parsing of JavaScript	F (0)	↓	JS	High
Specify image dimensions	F (0)	↓	Images	High
Minify CSS	F (35)	↓	CSS	High
Combine images using CSS sprites	F (43)	↓	Images	Medium
Enable gzip compression	F (48)	↓	Server	High
Specify a Vary: Accept-Encoding header	D (64)	↓	Server	High
Minify JavaScript	C (73)	↓	JS	High

After Optimization:



Latest Performance Report for:
https://funandfunction.com/

Summary

Page Speed Grade: (91%)↑ A

YSlow Grade: (70%)↓ C

Page load time: 20.14s
Total page size: 3.11MB
Total number of requests: 99

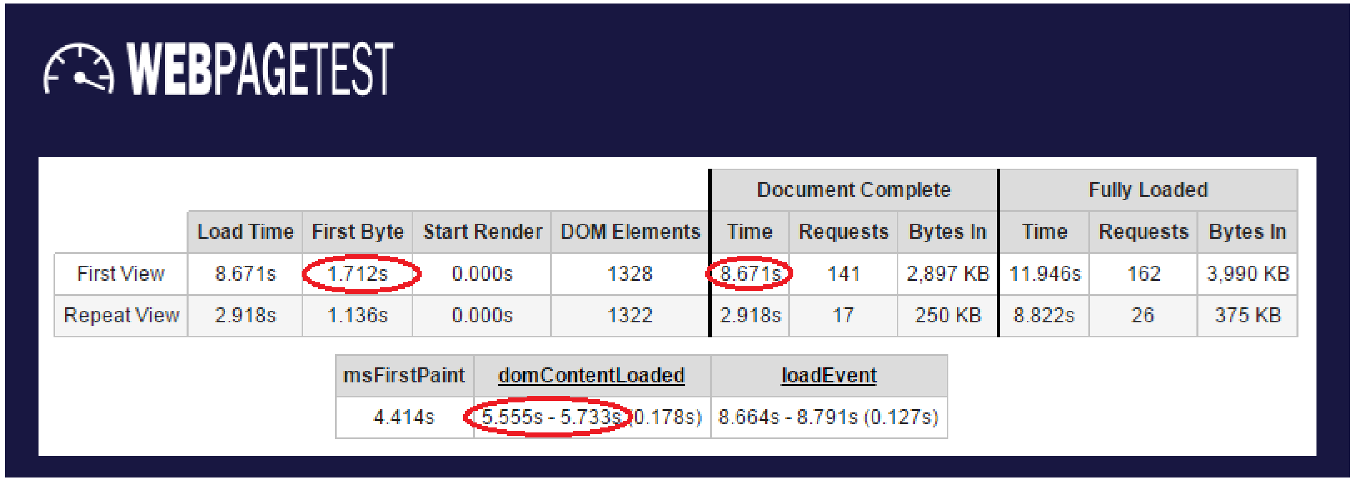
Breakdown

Page Speed | YSlow | Timeline | History

RECOMMENDATION	GRADE		TYPE	PRIORITY
Specify image dimensions	F (35)	↓	Images	High
Optimize images	E (58)	↓	Images	High
Combine images using CSS sprites	D (67)	↓	Images	Medium
Leverage browser caching	B (88)	↑	Server	High
Remove query strings from static resources	A (93	↑	Content	High
Serve resources from a consistent URL	A (93	↕	Content	High
Minimize redirects	A (94	↕	Content	High

In terms of performance that matters to the user, defined by ***loading the visible part of page content*** (the **domContentLoaded** metric), it took **over 5.5 seconds** - far over the 2.5 second limit when most people notice inactivity and start to lose interest.

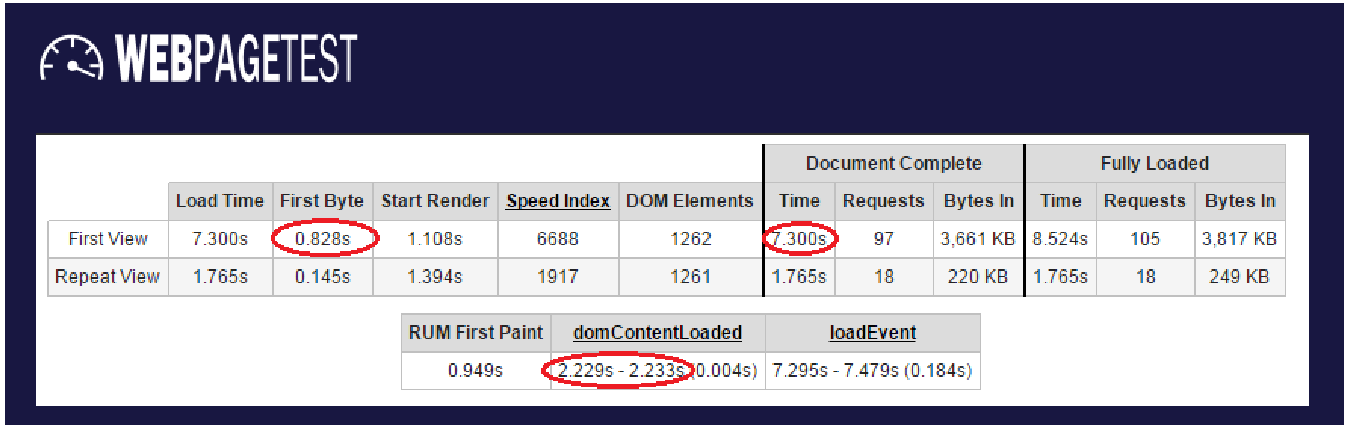
Before Optimization:



					Document Complete			Fully Loaded		
	Load Time	First Byte	Start Render	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In
First View	8.671s	1.712s	0.000s	1328	8.671s	141	2,897 KB	11.946s	162	3,990 KB
Repeat View	2.918s	1.136s	0.000s	1322	2.918s	17	250 KB	8.822s	26	375 KB

msFirstPaint	domContentLoaded	loadEvent
4.414s	5.555s - 5.733s (0.178s)	8.664s - 8.791s (0.127s)

After Optimization:



						Document Complete			Fully Loaded		
	Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In
First View	7.300s	0.828s	1.108s	6688	1262	7.300s	97	3,661 KB	8.524s	105	3,817 KB
Repeat View	1.765s	0.145s	1.394s	1917	1261	1.765s	18	220 KB	1.765s	18	249 KB

RUM First Paint	domContentLoaded	loadEvent
0.949s	2.229s - 2.233s (0.004s)	7.295s - 7.479s (0.184s)

What Was Optimized

MavenEcommerce had audited the HTTP loading sequence, identified and optimized the following areas for improvement:

1. Content: font library was cleaned of unused fonts and loading speed improved with Ajax; image loading was optimized with image sprites; unused image slider was turned off.
2. CSS: several user interface components were previously being loaded, but were not used – they were turned off.
3. JavaScript (JS): the 63 original JavaScript HTTP requests were concatenated and several were merged.
4. Server: database was converted into flat tables, which sped up loading; database requests were optimized, too.
5. Cache: the caching approach was simple – to cache everything that was cacheable! Results are clearly visible in the repeat view statistics.

One way to optimize a website is by minimizing the number of components. But even when simplifying the page's design is not an option, the number of HTTP requests can be drastically reduced by combining scripts, keeping richer content while also achieving fast response times. We were able to reduce the number of HTTP requests from 153 to 97 and halved the time it took for the user to see useful page content: the domContentLoaded decreased from 5.56 seconds to 2.23 seconds!

For the users, this translated into much less waiting.

The results were achieved without removing any functionality or content form the optimized website. Beyond displaying useful visible content twice as fast (under 2.5 seconds on the first visit to the site, under 1.8 seconds thereafter) the following improvements were achieved:

- reduction of waiting time before website content begins to load (First Byte Time) to 0.83 seconds from 1.71 seconds before
- page load gain on the first visit of almost one and a half seconds: 7.30 seconds compared with 8.67 seconds previously
- halved the time to load the page on the second visit: 1.77 seconds to load the page again versus 2.92 seconds before

MavenEcommerce Can Help

Digging into HTTP request sequence and cleaning up web page file components isn't the first idea of a good time, for most. But not for us!

In fact we pride ourselves on our ability to delight our customers' visitors with online stores that are both functional and fast.

Whether you need help estimating how much faster your Magento store could be without cutting down on functionality, or even if you're more ambitious and want to make your store radically fast – we'll find a cost-efficient way to help you out.

For additional information contact:

Andrey Korolyov, CEO
MavenEcommerce Inc.
20 West 47th St.
Suite 402
New York, NY 10036
(302) 409-0927
contacts@mavenecommerce.com
www.mavenecommerce.com

MavenEcommerce is a team of Certified Magento Software Developers who are passionate about creating amazing web storefronts (and optimizing them, too!) for customers of all sizes everywhere.