adtrak

Improving Website Speed & Performance

Web Design Training

‹a›

What we’re going to cover...

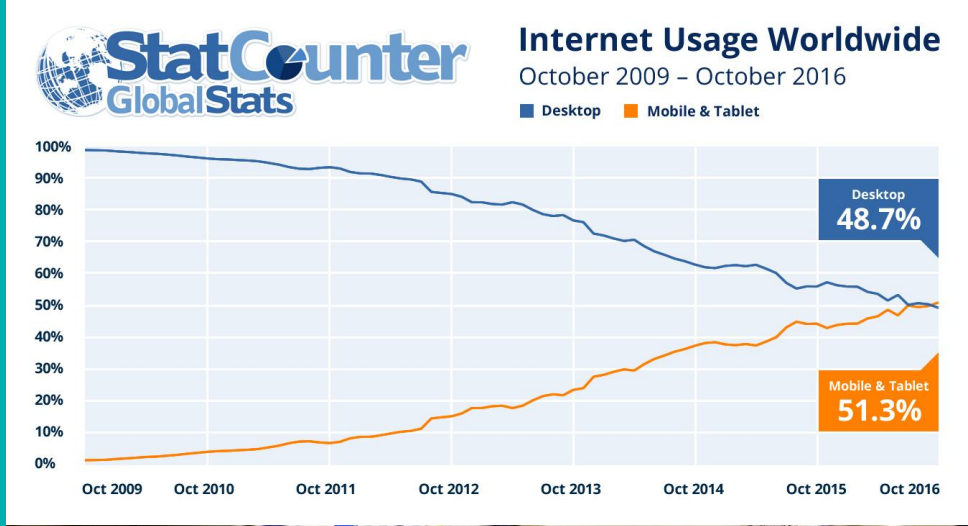
- Why?
- Basics of the page load process
- What determines a website’s performance?
- Each of the PageSpeed Insights rules
- Additional tips and tricks
- Helpful tools
- The future

Why?

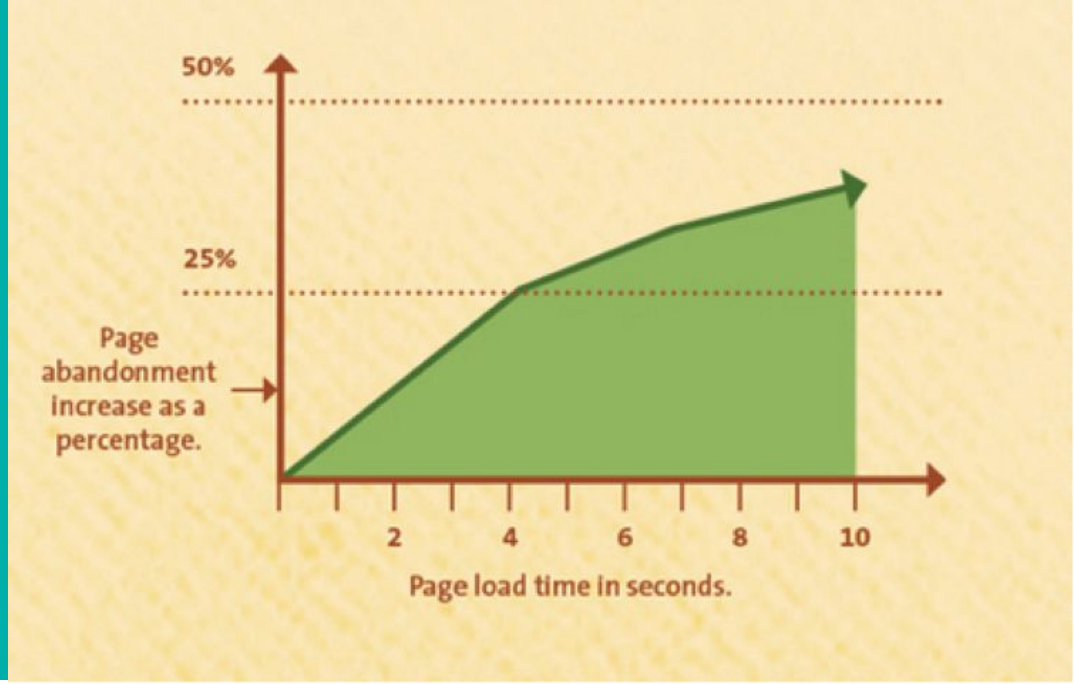
Although Internet speeds are increasing, **the way people access the web is changing**, with mobile traffic increasing day by day

This means we have to cater for users on slow, public connections

...as well as being considerate of data plans



Every Second Counts.

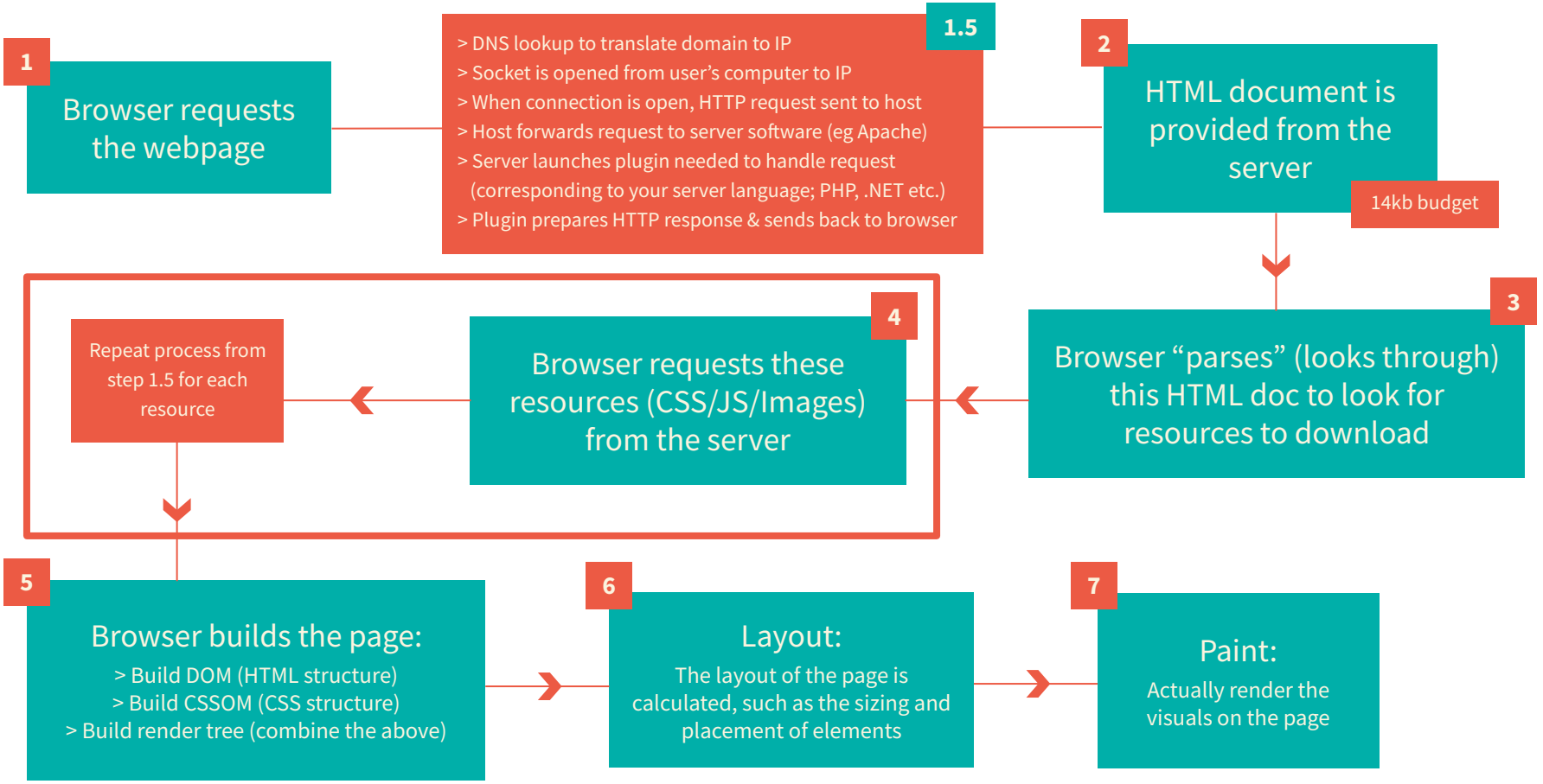


A **1 second delay** can result in a **7% reduction** in conversions

Sources: gomez.com / akamai.com

Subsequently, website performance is now even more important than ever and is something we're really focusing on this year at Adtrak.

The Page Load Process


1
Browser requests the webpage
1.5
> DNS lookup to translate domain to IP
> Socket is opened from user's computer to IP
> When connection is open, HTTP request sent to host
> Host forwards request to server software (eg Apache)
> Server launches plugin needed to handle request (corresponding to your server language; PHP, .NET etc.)
> Plugin prepares HTTP response & sends back to browser
2
HTML document is provided from the server
14kb budget
3
Browser "parses" (looks through) this HTML doc to look for resources to download
4
Browser requests these resources (CSS/JS/Images) from the server
Repeat process from step 1.5 for each resource
5
Browser builds the page:
> Build DOM (HTML structure)
> Build CSSOM (CSS structure)
> Build render tree (combine the above)
6
Layout:
The layout of the page is calculated, such as the sizing and placement of elements
7
Paint:
Actually render the visuals on the page

The most basic rule would be to make as few HTTP requests as possible

- Concatenate **CSS & JS** into single file
- Only use **images** where appropriate
  - Does it need to be an image?
  - Can you use CSS instead?
- Keep **font-face** files to a minimum
  - Don’t go overboard
  - Use as few weights as possible

Questions?

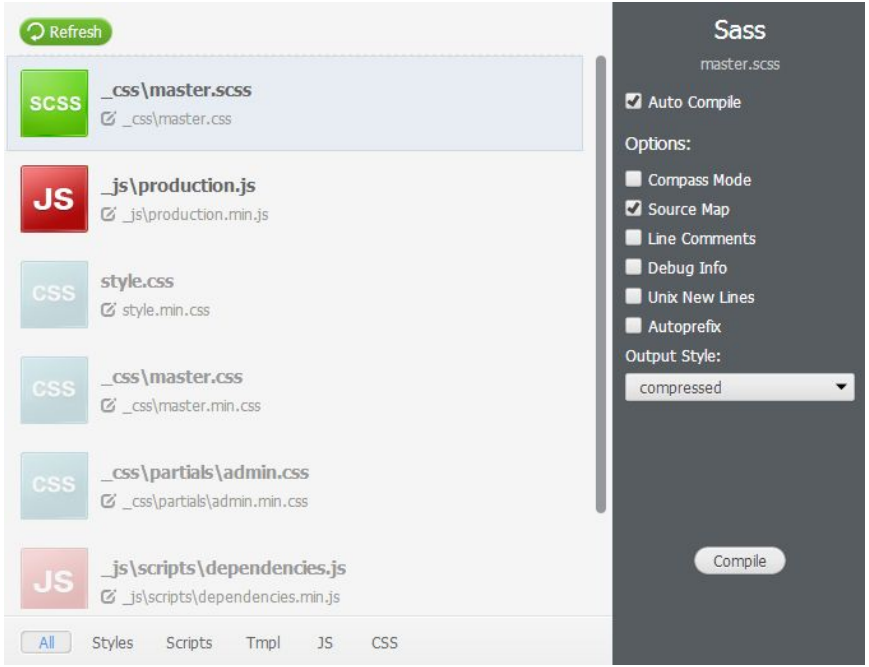
PageSpeed Insights Rule 1

Minify CSS

Minify CSS

- Concatenate all of your site's stylesheets into a single CSS file and **minify** it to ensure the file size is as small as possible

- Should be done automatically by your compiler tool (Koala, Prepros etc.) each time you save a Sass file



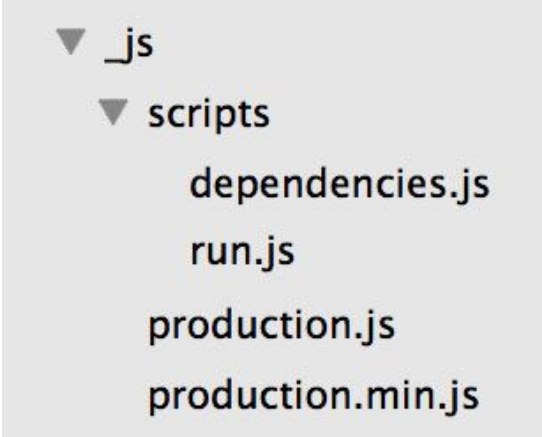
PageSpeed Insights Rule 2

Minify JavaScript

Minify JS

- Again, concatenate all of your site's scripts into a single JS file and **minify** it to ensure the file size is as small as possible

- Should also be done automatically by your compiler tool (Koala, Prepros etc.) each time you save a JS file



List your scripts in your **production.js** file...

```
// @prepros-prepend "scripts/dependencies.js"
// @prepros-prepend "scripts/run.js"
```

In Prepros/Koala select your production.js file and enable **auto compile** and **compress**

production.min.js should ultimately be the only script your site uses

Keep large scripts that are only needed on one page separate

if you have a fairly large script that is only used on one page, it makes more sense to keep that separate and include it only on that page

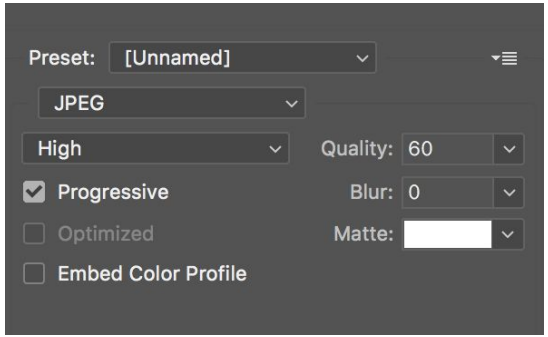
PageSpeed Insights Rule 3

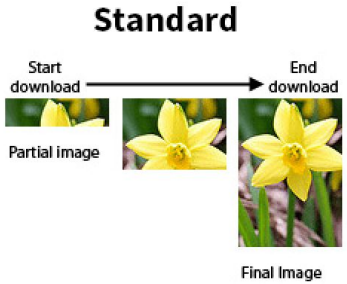
Optimise Images

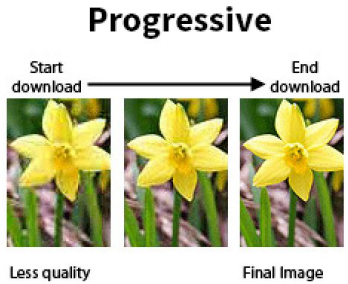
Images are often one of the biggest performance hits, so there are plenty of gains to be made here

Saving images

- Does it really *need* to be an image? Could you just use CSS instead?
- Can you reduce the complexity of the image itself, such as the number of colours it uses?
- If saving your images from Photoshop, be sure to "**Save for Web**" and consider the quality setting if you're saving as JPG
  - "High" is usually fine, and "Medium" can also work for smaller images
- Also for JPGs, you should select "**Progressive**"



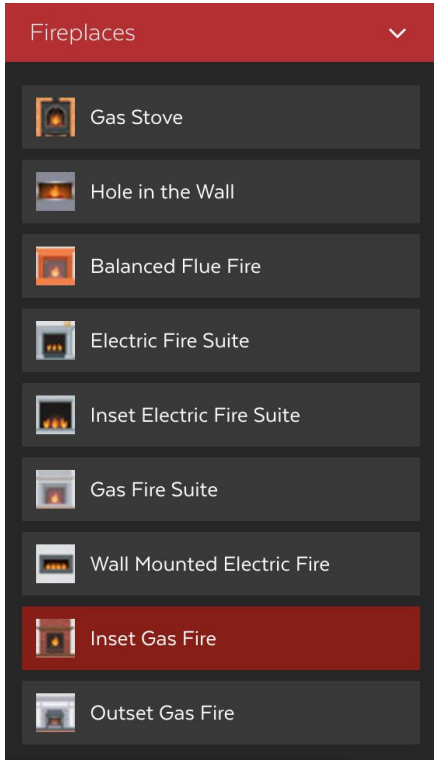




Saving images

- Can the image be saved as an SVG instead? These are usually much smaller in file size
- Make sure the image you deliver to the user isn't much bigger than it needs to be in terms of its physical dimensions
    - i.e. don't use a 3000px wide image as your hero image!
    - Make use of media queries and Picturefill to help with this
- If you have several images to be used as non-fluid backgrounds, then save these as a single **sprite image** in order to reduce HTTP requests

Image Sprites



```
.icons {
    background-image: url(icons.jpg);
    width: 25px;
    height: 25px;
}

.icon1 { background-position: 0 0; }
.icon2 { background-position: 0 -25px; }
.icon3 { background-position: 0 -50px; }
```

The icon essentially becomes a window through which you show a certain portion of the image

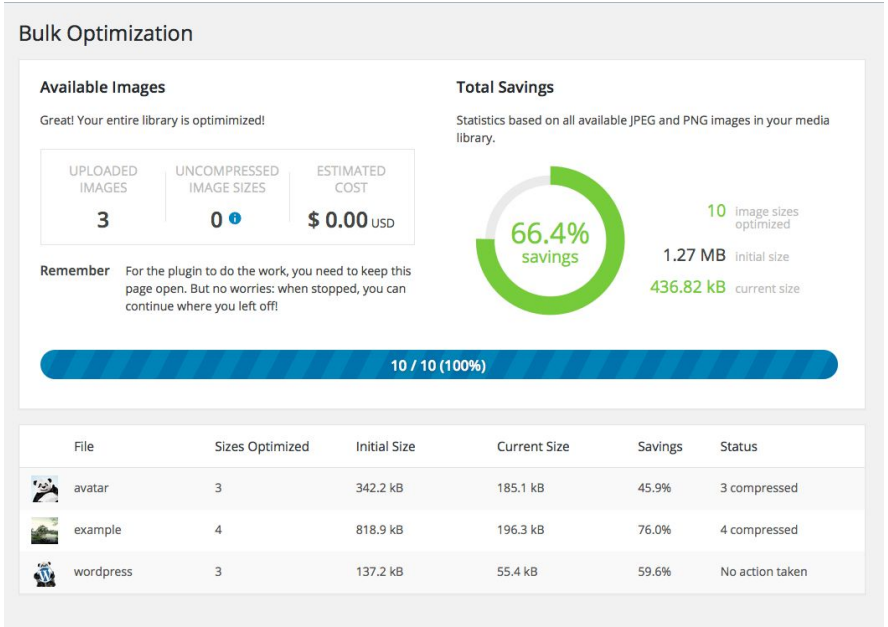
Optimising Images

Tiny PNG

Optimise small batches of images by simply dragging them into the TinyPNG site and downloading the minified images

Optimising Images - Tiny PNG

- However when you have potentially hundreds of images in several folders (such as in the WordPress uploads folder), this isn't practical
- TinyPNG WordPress plugin solves this issue
- Once installed, all future image uploads (up to 500 per month) will be automatically optimised on upload
- You can also bulk optimise all previously uploaded images (up to 500)
- Additional images can be optimised at a small cost - for CM/MP clients, you could work this into their budgets



Optimising Images

PageSpeed Insights

Download optimized image, JavaScript, and CSS resources for this page.

- Usually saves considerably more bytes than the TinyPNG method
- Need to use with caution as the quality reduction can occasionally be too noticeable
- Only downloads optimised images for page currently being analysed

Questions?

PageSpeed Insights Rule 4

Leverage Browser Caching

What is browser caching?

One of the most fundamental and effective methods of improving a website's performance

Caching is when we tell the user's browser to save some of the website's resources (images, CSS, JS etc.) on the user's computer

This means on subsequent page views, they don't have to download these files from the server each time, because they're now saved locally

Caching is enabled by default on our sites

Caching is often set in a website's htaccess file, but can also be enabled in the actual server settings

Cloud Unboxed (our server providers) have enabled caching on our server so this isn't something you'll need to worry about!

PageSpeed Insights Rule 5

Enable Compression

Gzip Compression

Significantly compresses the size of the resources that the website has to download from the server

Another very quick and easy performance win

Can also be set in a website's htaccess file, but again, this has been **enabled in the default server settings** by Cloud Unboxed

PageSpeed Insights Rule 6

Eliminate render-blocking JavaScript and CSS in above-the-fold content

What does this mean?

The browser can’t render the “above-the-fold” portion of your website without having to download external JavaScript or CSS files first

Eliminate render-blocking JavaScript and CSS in above-the-fold content

Move all scripts to the footer

```php
/* ========================================================================================================================
Enqueue scripts and stylesheets
======================================================================================================================== */
function site_script_loader() {
    // Style sheets
    wp_enqueue_style( 'master', get_stylesheet_directory_uri() . '/_css/master.css', '', null );

    // Scripts
    wp_enqueue_script( 'production', get_template_directory_uri() .'/_js/production.min.js','','', true  );

    // Add variables to WP_Localize so we can use them in ajax-form
    $translation_array = array( 'templateUrl' => get_stylesheet_directory_uri() );
    wp_localize_script( 'production', 'object_name', $translation_array );
}

add_action( 'wp_enqueue_scripts', 'site_script_loader' );
```

That last `true` parameter where the production.min.js file is enqueued states that the script should be included in the footer

If it was set to false, it would be included in the header

Eliminate render-blocking JavaScript and CSS in above-the-fold content

Move jQuery to the footer

```php
/* ========================================================================
Move jQuery to the footer
======================================================================== */
function wpse_enqueue_scripts() {
    wp_scripts()->add_data( 'jquery', 'group', 1 );
    wp_scripts()->add_data( 'jquery-core', 'group', 1 );
}
add_action( 'wp_enqueue_scripts', 'wpse_enqueue_scripts' );
```

As long as you don't have any jQuery dependent JavaScript in the `<head>` or inline in the `<body>` of your page, you should move jQuery to the footer

What if you do have jQuery dependent JavaScript in the <head> or <body>?

If you are using a plugin which outputs jQuery dependent code in the <body> of your page (such as the branch finder plugin), you will need to include jQuery in the <head>

If this plugin is just being used on a single page, you could use an if statement to only include jQuery in the <head> on that specific page, and include it in the footer on all other pages

```
01  /* ========================================================================
02  Move jQuery to the footer
03  ========================================================================
04  function wpse_enqueue_scripts() {
05    // Only move jQuery to the footer if we're NOT on the branch finder page
06    if(!is_page('branch-finder')) {
07      wp_scripts()->add_data( 'jquery', 'group', 1 );
08      wp_scripts()->add_data( 'jquery-core', 'group', 1 );
09    }
10  }
11  add_action( 'wp_enqueue_scripts', 'wpse_enqueue_scripts' );
```

If you are using a plugin which includes a jQuery dependent external script in the `<head>` of your page, you can use a function to remove that script include

You can then manually include that script in the footer yourself, or add it to your minified production file

```
function my_deregister_javascript() {
  wp_deregister_script( 'contact-form-7' );
}
add_action( 'wp_print_scripts', 'my_deregister_javascript', 100 );
```

This example will remove the contact form 7 script from the `<head>`

The **`'contact-form-7'`** bit is the script's handle

To find this, search the plugin folder for the phrase "wp_enqueue_script" – this will show you where the script is originally enqueued, and the handle it uses.

Best to try and avoid using plugins for front-end functionality

Instead, bake them into your code, so you can include the scripts in your minified production files

Eliminate render-blocking JavaScript and CSS in above-the-fold content

Defer your JavaScript files

Even after moving all scripts to the footer, PageSpeed Insights will usually tell you that they are *still* render-blocking

To get around this, we need to defer the scripts, which in theory is as simple as adding the `defer` attribute to the script include:

```
1 | <script defer type='text/javascript' src='[your-js-file]'></script>
```

Because we enqueue our scripts in the WordPress functions file, we need to use a PHP function to add this defer attribute to our JavaScript includes:

```php
function add_defer_attribute($tag, $handle) {
   // add script handles to the array below
   $scripts_to_defer = array('jquery', 'production');

   foreach($scripts_to_defer as $defer_script) {
      if ($defer_script === $handle) {
         return str_replace(' src', ' defer src', $tag);
      }
   }
   return $tag;
}
add_filter('script_loader_tag', 'add_defer_attribute', 10, 2);
```

`'jquery'` and `'production'` are the handles for your JavaScript files

You can include as many handles as you need to here, separating them by commas

Eliminate render-blocking JavaScript and CSS in above-the-fold content

Move non-critical CSS to the footer

As well as scripts, if you have any plugins that are including additional CSS files in the `<head>` of your page, these should also be moved to the footer

```
1  function my_deregister_styles() {
2    wp_deregister_style('wp-paginate');
3  }
4  add_action('wp_print_styles', 'my_deregister_styles', 100);
```

This will remove the plugin's stylesheet from the <head> - you can then either include the CSS in your master.css file or manually add the include in your footer

The **`'wp-paginate'`** bit is the stylesheet's handle

To find this, search the plugin folder for the phrase "wp_enqueue_style" – this will show you where the stylesheet is originally enqueued, and the handle it uses

One step further - conditional loading

With this particular example, the wp-paginate plugin is only used on pages with pagination (the news section), so when manually including the stylesheet in your footer, you could also use an if statement to only include the stylesheet on news pages

```
<?php if(is_home() || is_archive() || is_category() || is_tag()): ?>
  <link rel='stylesheet' href='<?php echo get_site_url(); ?>/wp-content/plugins/wp-paginate/wp-paginate.css'
<?php endif; ?>
```

Oh for fu… what do you want from me, Google?

Why am I still getting this warning?

Viewing the source code of your page is the best way to see what's being included where

```
01  <head>
02    <link rel='stylesheet' href='master.css' />
03  </head>
04
05  <body>
06
07    <script type='text/javascript' defer src='jquery.min.js'></script>
08    <script type='text/javascript' defer src='production.min.js'></script>
09  </body>
```

Any external resources in the `<head>` are still render-blocking resources

The page can't start to render until it's downloaded these external resources

Can’t I just move the CSS include to the footer?

This will result in the dreaded
“Flash of Unstyled Content” (FOUC)

Your page will display the content without any styling applied, until it has downloaded the CSS file

Eliminate render-blocking JavaScript and CSS in above-the-fold content

Inline Critical CSS

(or probably don’t)

Google wants you to extract the CSS that's needed to style the top of your page, and inline it in `<style>` tags directly in the **`<head>`** of your HTML document

```
<head>
  <style>/* CSS needed to render above-the-fold portion of page */</style>
</head>

<body>

  <link rel='stylesheet' href='master.css' />
  <script type='text/javascript' defer src='jquery.min.js'></script>
  <script type='text/javascript' defer src='production.min.js'></script>
</body>
```

This allows your page to appear visually complete to the user without having to download any external resources

You can then defer the loading of your main stylesheet by moving it to the footer

Questions?

Additional Tips & Tricks

Dealing with third party resources

Dealing with third party resources

- Even if you've done everything up to this point, you might still be seeing warnings such as: **Minify CSS/JS**, **Leverage Browser Caching** and **Enable Compression**
- This is due to third party resources such as Google Maps/Analytics, Wistia, Response Tap etc.
- We can't do much about resources on third party servers, but we can limit the damage
- When your page calls resources from an external location, it has to do a DNS lookup before it can actually download the resource
- You can make your page perform the DNS lookup before it's actually needed using `dns-prefetch`
- By the time the browser gets to that resource, it can download it instantly because the DNS lookup has already been done

```
1 | <link rel="dns-prefetch" href="http://www.google-analytics.com">
```

Removing Unnecessary Scripts and HTTP Requests from the WordPress <head>

- WordPress includes a number of scripts by default to enable certain features, such as **emojis** and **automatic embedding** when you post URLs in the backend
- We don’t want this to be the default behaviour

```php
/* ========================================================================================================
Remove Unwanted Scripts & HTTP Requests from the <head>
======================================================================================================== */

remove_action( 'wp_head', 'print_emoji_detection_script', 7 ); // Remove emojis script
remove_action( 'wp_print_styles', 'print_emoji_styles' ); // Remove emojis styles
add_filter( 'emoji_svg_url', '__return_false' ); // Remove DNS prefetch for '//s.w.org' - only required for emojis

// Functions to remove auto embed functionality
function disable_embeds_code_init() {
 // Remove the REST API endpoint.
 remove_action( 'rest_api_init', 'wp_oembed_register_route' );

 // Turn off oEmbed auto discovery.
 add_filter( 'embed_oembed_discover', '__return_false' );

 // Don't filter oEmbed results.
 remove_filter( 'oembed_dataparse', 'wp_filter_oembed_result', 10 );

 // Remove oEmbed discovery links.
 remove_action( 'wp_head', 'wp_oembed_add_discovery_links' );

 // Remove oEmbed-specific JavaScript from the front-end and back-end.
 remove_action( 'wp_head', 'wp_oembed_add_host_js' );
 add_filter( 'tiny_mce_plugins', 'disable_embeds_tiny_mce_plugin' );

 // Remove all embeds rewrite rules.
 add_filter( 'rewrite_rules_array', 'disable_embeds_rewrites' );

 // Remove filter of the oEmbed result before any HTTP requests are made.
 remove_filter( 'pre_oembed_result', 'wp_filter_pre_oembed_result', 10 );
}

add_action( 'init', 'disable_embeds_code_init', 9999 );
```

```
<link rel='dns-prefetch' href='//s.w.org' />
<script type="text/javascript">
 window._wpemojiSettings = {"baseUrl":"https:\/\/s.w.org\/images\/core\/emoji\/2.2.1\/72x72\/","ext":".png","svgUr
{"concatemoji":"http:\/\/dev-server\/e\/elite-fire\/wp-includes\/js\/wp-emoji-release.min.js?ver=4.7.4"}};
 !function(a,b,c){function d(a){var b,c,d,e,f=String.fromCharCode;if(!k||!k.fillText)return!1;switch(k.clearRect(0
k.fillText(f(55356,56826,55356,56819),0,0),!(j.toDataURL().length<3e3)&&
(k.clearRect(0,0,j.width,j.height),k.fillText(f(55356,57331,65039,8205,55356,57096),0,0),b=j.toDataURL(),k.clearRe
k.fillText(f(55357,56425,55356,57341,8205,55357,56507),0,0),d=j.toDataURL(),k.clearRect(0,0,j.width,j.height),k.fi
c=b.createElement("script");c.src=a,c.defer=c.type="text/javascript",b.getElementsByTagName("head")[0].appendChild
f,g,h,i,j=b.createElement("canvas"),k=j.getContext&&j.getContext("2d");for(i=Array("flag","emoji4"),c.supports=
{everything:!0,everythingExceptFlag:!0},h=0;h<i.length;h++)c.supports[i[h]]=d(i[h]),c.supports.everything=c.suppor
(c.supports.everythingExceptFlag=c.supports.everythingExceptFlag&&c.supports[i[h]]);c.supports.everythingExceptFla
{c.DOMReady=!0},c.supports.everything||(g=function(){c.readyCallback()},b.addEventListener?(b.addEventListener("DO
(a.attachEvent("onload",g),b.attachEvent("onreadystatechange",function(){"complete"===b.readyState&&c.readyCallbac
(window,document,window._wpemojiSettings);
</script>
<style type="text/css">
img.wp-smiley,
img.emoji {
 display: inline !important;
 border: none !important;
 box-shadow: none !important;
 height: 1em !important;
 width: 1em !important;
 margin: 0 .07em !important;
 vertical-align: -0.1em !important;
 background: none !important;
 padding: 0 !important;
}
</style>
<link rel="alternate" type="application/json+oembed" href="http://dev-server/e/elite-fire/wp-json/oembed/1.0/embed
<link rel="alternate" type="text/xml+oembed" href="http://dev-server/e/elite-fire/wp-json/oembed/1.0/embed?url=htt
<script type='text/javascript' src='http://dev-server/e/elite-fire/wp-includes/js/wp-embed.min.js?ver=4.7.4'></scr
```

- Previous snippet removes:
  - A DNS prefetch, inline JS and inline CSS required for emojis
  - 3 HTTP requests for the auto embed functionality

Deferring images and iframes

- Deferring the loading of your images (often called lazy loading) can also significantly help your load times
- This is done by using your already deferred JavaScript to load your images

```
<!-- CHANGE FROM THIS -->
<img src="images/your-img.jpg" alt="Lorem ipsum">

<!-- TO THIS... -->
<img src="" data-src="images/your-img.jpg" alt="Lorem ipsum">
```

- Then we can use JS to get the `data-src` attribute and move it to the `src` attribute to load the image

```
// Defer loading of iframes and images
function init() {
    var imgDefer = document.querySelectorAll('iframe, img');
    for (var i=0; i<imgDefer.length; i++) {
        if(imgDefer[i].getAttribute('data-src')) {
            imgDefer[i].setAttribute('src',imgDefer[i].getAttribute('data-src'));
        }
    }
}
window.onload = init;
```

- This only happens after your JS has loaded, **allowing your page to load first**

- Your images will now be deferred until after your page loads as desired, but during the page load you will see the image's alt tags making things look untidy
- Can use CSS to hide any elements that have a **`data-src`** attribute

```
[data-src] {
    opacity: 0;
}
```

- And then amend the JS to restore the image's opacity once it has loaded

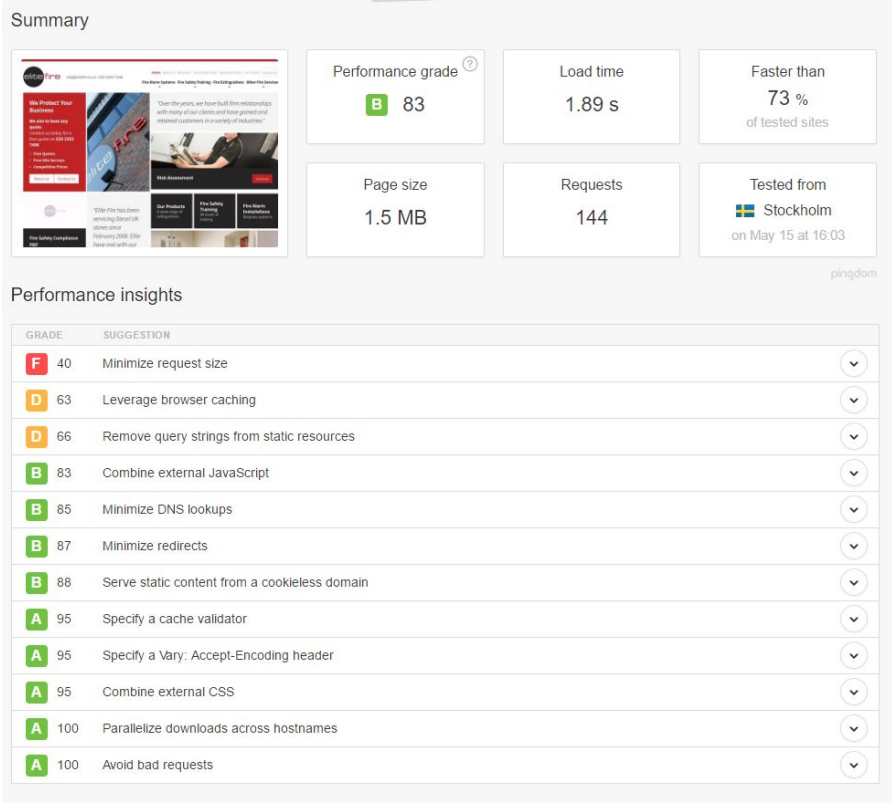
```
// Defer loading of iframes and images
function init() {
    var imgDefer = document.querySelectorAll('iframe, img');
    for (var i=0; i<imgDefer.length; i++) {
        if(imgDefer[i].getAttribute('data-src')) {
            imgDefer[i].setAttribute('src',imgDefer[i].getAttribute('data-src'));
            imgDefer[i].setAttribute('style','opacity:1;');
        }
    }
}
window.onload = init;
```

Questions?

Helpful Tools

Helpful Tools

Pingdom



Helpful Tools

Varvy

Summary

Server:

- ✓ Quick server response time
- ✓ Compression enabled
- ✓ Browser caching enabled
- ✓ Keep-alive enabled
- ✓ Minimal redirects

Page:

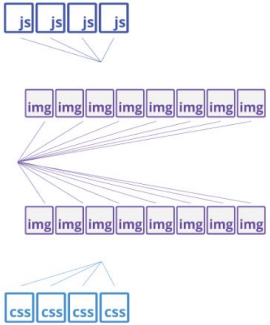
- ✓ No bad requests
- ✓ HTML minified
- ✓ Request size is fine
- ✓ Visible content prioritized
- ✓ No render blocking CSS / JS

Resources:

- ✓ Images optimized
- ✓ Javascripts seem async
- ✓ CSS minified
- ✓ No @import CSS
- ✓ JS minified

Note: These values may change often if you display third party content (ads, widgets, etc.).

Resources (4 js files / 16 images / 4 css files)



CSS delivery

CSS size: 188 k

External: 187414 / Internal: 0 / In attribute: 142

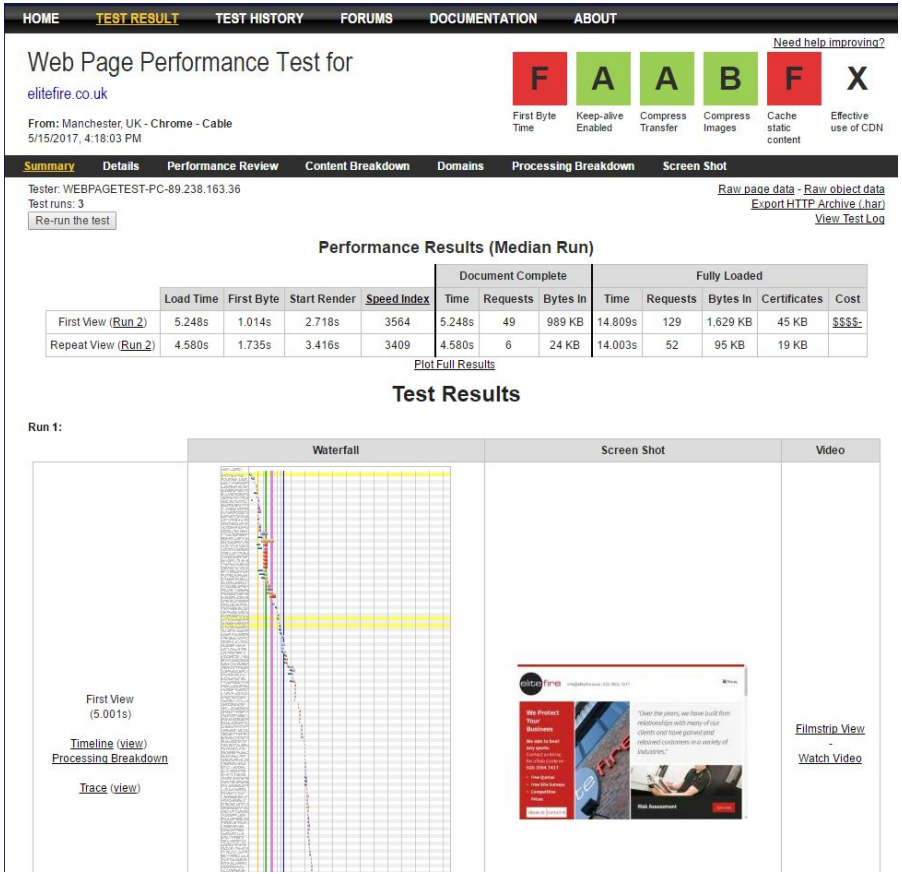
Javascript usage

Total Javascript size: 1427 k

External: 1425005 / Internal: 2436

Helpful Tools

WebPageTest



The Future

HTTP2

The Future - HTTP2

- HTTP2 will bring with it a whole host of **new best practices**, which will effectively make our lives easier
- Basically; increased, smaller HTTP requests will be better than fewer, larger downloads with HTTP2 due to "**multiplexing**" (concurrent requests)
  - No need to concatenate all of your CSS and JS into a single file
  - Better to just serve assets to pages they will be used on
  - Sprites will no longer be more efficient
- Requires a secure connection (HTTPS)
- Needs to be supported by the server *and* the browser
  - Only supported on Windows 10+

Questions?