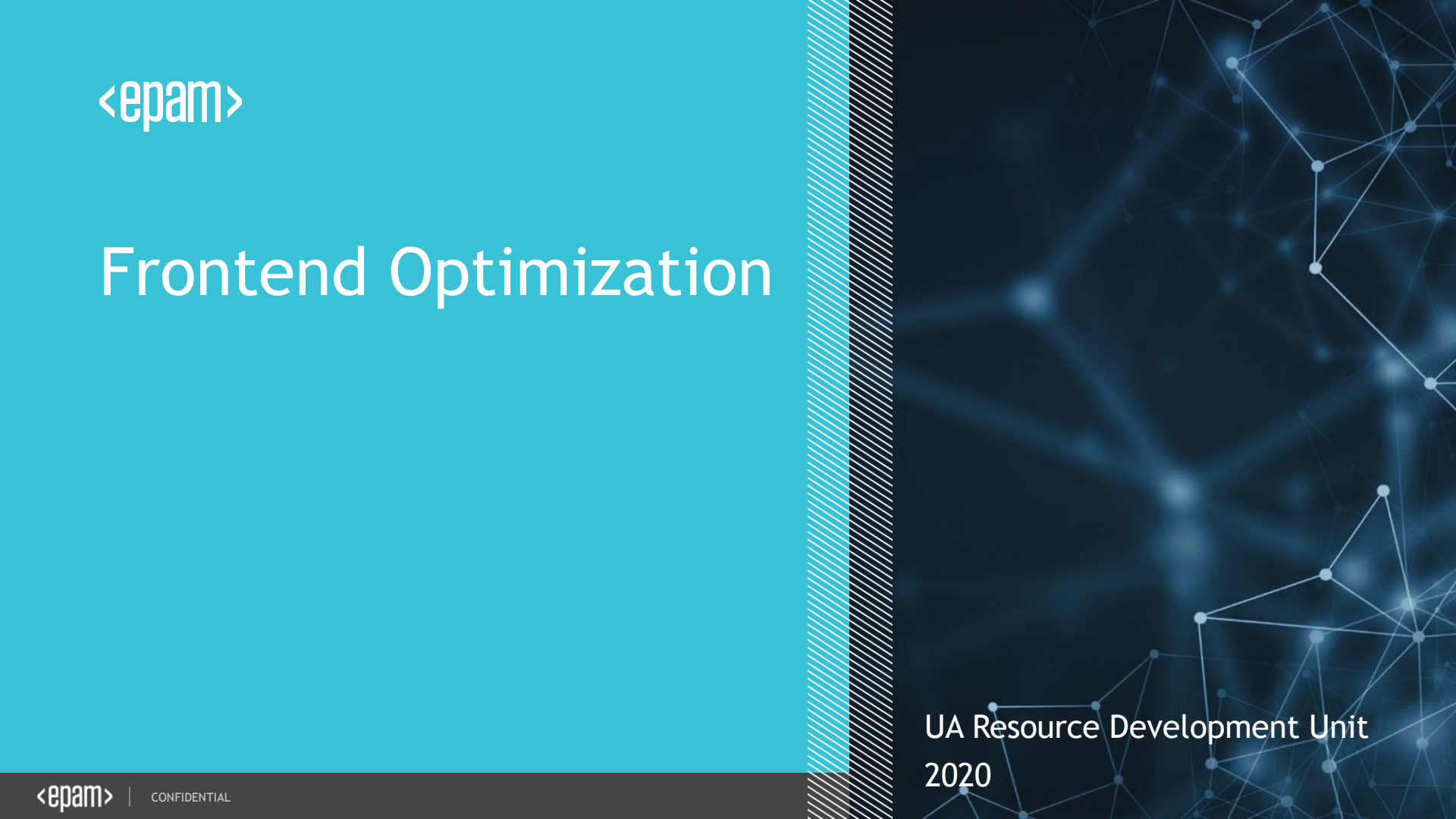

epam
Frontend Optimization
UA Resource Development Unit
2020
epam
CONFIDENTIAL

AGENDA

1. **Intro. What is the purpose?**
2. **Optimization. For load Speed**
3. **Optimization Tools**
4. Image Optimization
5. Conclusion

Frontend optimization

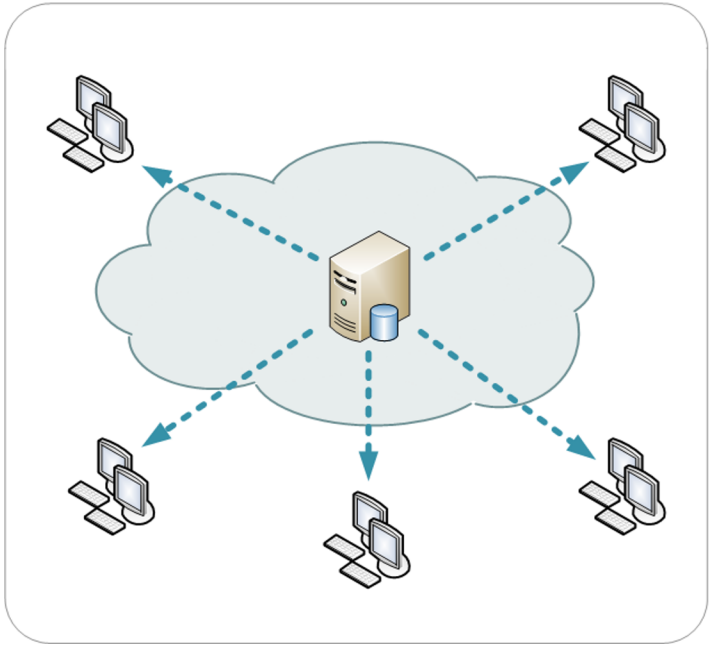
Frontend optimization (FEO), also known as content optimization, is the process of fine-tuning your website to make it more browser-friendly and quicker to load.

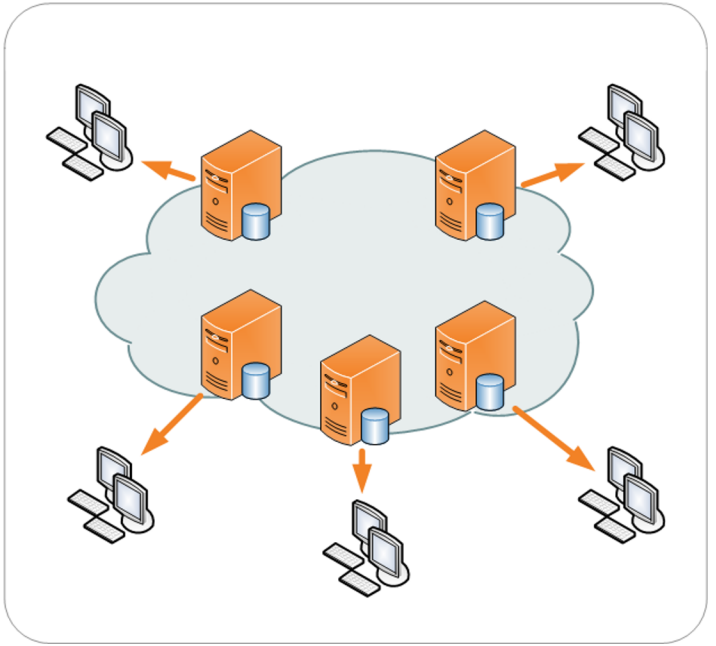
Broadly speaking, FEO focuses on reducing file sizes and minimizing the number of requests needed for a given page to load.

Frontend optimization

During the FEO process, web designers draw a distinction between the perceived and the actual page load time. Perceived load time is considered because of its impact on the overall user experience (UX), while the actual load time is often used as a performance benchmark metric.

Content delivery networks (CDNs) play an important role in the FEO process of front end optimization, as they are commonly used to streamline many of the more time-demanding optimization tasks. For example, a typical CDN offers auto-file compression and auto-minification features, freeing you from having to manually tinker with individual website resources.





PERFORMANCE MATTERS

57%

Abandon a site after waiting 3 seconds for a page to load

80%

Will not return if website loads more than 3 seconds

32%

Tell others about their negative experience

1%

Amazon: 1% Revenue increase for **every 100MS** of improvement

WEBSITE OPTIMIZATION: MAIN OBJECTIVES

Website optimization is the field of knowledge about increasing the speed in which web pages are loaded

Low number of files

Every HTTP request is a gamble, a chance to fail

Small file size

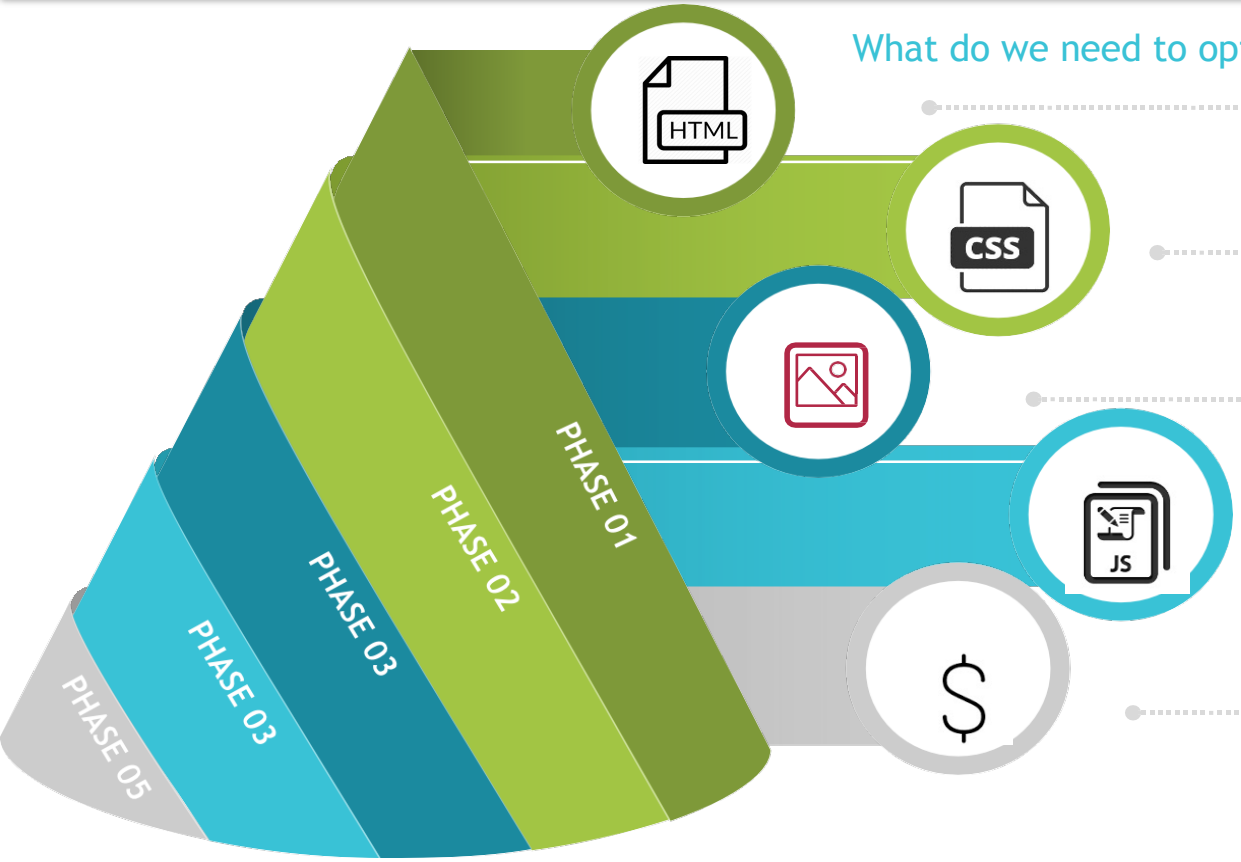
Write semantic HTML avoiding unnecessary elements

Fast rendering

Cache static files. The fastest HTTP request is the one not made

OPTIMIZATION
FOR LOAD SPEED

OPTIMIZATION



What do we need to optimize ?

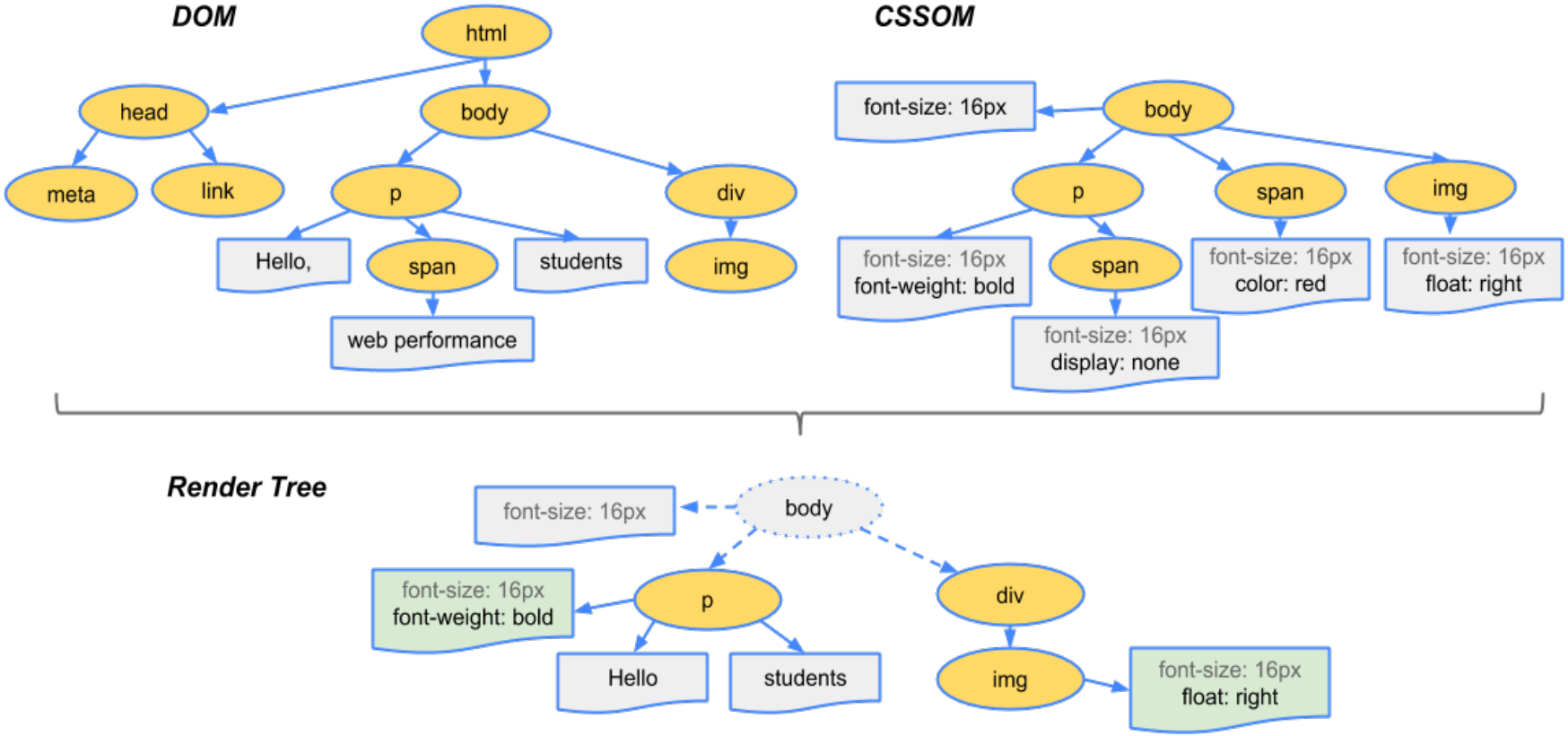
- HTML
- CSS
- Images
- Javascript
- jQuery

RENDER, LAYOUT & PAINT

All the steps the browser went through:

- Process HTML markup and build the DOM tree.
- Process CSS markup and build the CSSOM tree.
- Combine the DOM and CSSOM into a render tree.
- Run layout on the render tree to compute geometry of each node.
- Paint the individual nodes to the screen.

RENDER, LAYOUT & PAINT



REFLOW & REPAINT

The render tree is the visual part of the DOM tree. So if you're hiding a div with display: none, it won't be represented in the render tree. Changing a property of a Render Tree node could trigger:

Reflow (or layout) - parts of the render tree will need to be revalidated and the node dimensions recalculated (resize window, font changes, height, scrollTop, etc).

Repaint (or redraw) - Once the render tree is constructed, the browser can paint (draw) the render tree nodes on the screen (changes in geometric properties of a node or stylistic change: color, visibility, outline).

Repaints and reflows can be expensive, they can hurt the user experience, and make the UI appear sluggish.

WHAT TRIGGERS REFLOW & REPAINT?

- Adding, removing, updating DOM nodes
- Hiding a DOM node with display: none; (reflow and repaint) or visibility: hidden; (repaint only, because no geometry changes)
- Moving, animating a DOM node on the page
- Adding a stylesheet, tweaking style properties
- User action such as resizing the window, changing the font size, or scrolling

MINIMIZING REFLOWS & REPAINTS

Batch DOM changes and perform them not in the live DOM tree. You can:

- use a documentFragment to hold temp changes;
- clone the node you're about to update, work on the copy, then swap the original with the updated clone;
- hide the element with display: none (1 reflow, repaint), add 100 changes, restore the display (another reflow, repaint). This way you trade 2 reflows for potentially a hundred.

Don't ask for computed styles excessively. If you need to work with a computed value, take it once, cache to a local var and work with the local copy.

Using absolute positioning makes element a child of the body in the render tree, so it won't affect too many other nodes when you change it's styles.

HTML OPTIMIZATION

The reason to keep markup clean is not so much about faster load times, as it is about **having a solid and robust foundation** to build upon.

Styles up top, scripts down bottom:

When we put stylesheets in the <head> it gives the impression that the page is loading quickly.
Place scripts at the bottom to not block rendering

Use semantic tags

Semantic code tends to improve your placement on search engines, and make code more readable

Get rid of redundancies, and inefficient or archaic structures:

Collapse Whitespace
Remove HTML comments
Elide attributes

EXAMPLE

HTML

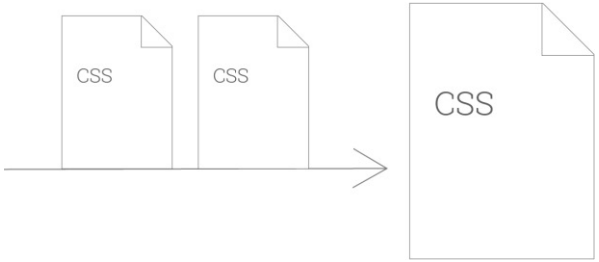
```
<html>
<head>
	<title>Example</title>
	<script src="js/script.js"></script>
	<link rel="stylesheet" href="css/main.css">
</head>
	<body>
		<div class=“section“>
			<div class=“header”>
				<span class = "line"></span>
				<a href = "#" >what we do</a>
				<span class = "line"></span>
			</div>
		</div>
	</body>
</html>
```

HTML

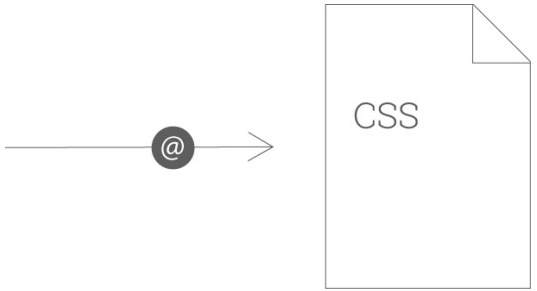
```
<!DOCTYPE HTML>
<html>
<head>
	<title>Example</title>
	<link href="css/main.css“/>
</head>
	<body>
<section class="section">
	<div class="header">
				<span class="line"></span>
				<a href="#" title="Home">what we do</a>
				<span class="line"></span>
			</div>
		</section>
	<script src="js/script.js"></script>
</body>
</html>
```

CSS OPTIMIZATION STEPS: 8 STEPS TO OPTIMIZE YOUR CSS

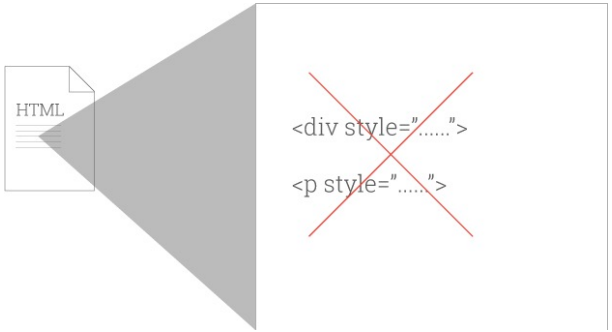
Combine multiple css files



Prefer <link> over @import



Don’t use inline styles



Use Shorthand

CSS

```
h1 {
  margin:1em 0 2em 0.5em;
}
```

Avoid the child selector

CSS

```
/* BAD */
.treehead > .treerow > .treecell {...}

/* GOOD */
.treecell-header {...}
```

Rely on inheritance

CSS

```
/* BAD */
#bookmarkMenuItem > .menu-left {
  list-style-image: url(url)
}

/* GOOD */
#bookmarkMenuItem {
list-style-image: url(url)
}
```

CSS sprites

Minify your stylesheets

```
body{font-family:‘Lato Light',Arial,
Helvetica,sans-serif;color:#3c3c3c}
.clearfix:after{content:".";display:block;
height:0;clear:both;
visibility:hidden}.section{padding:10em 0}
```

EFFICIENTLY RENDERING CSS

Browsers read your CSS selectors from **right to left**. ID's are the most efficient, Universal are the least

```
#main-navigation { }                    /* ID (Fastest)*/
body.home #page-wrap { }                /* ID */
.main-navigation { }                    /* Class */
ul li a.current { }                     /* Class *
ul { }                                  /* Tag */
ul li a { }                             /* Tag */
* { }                                   /* Universal(Slowest) */
#content [title='home'] { }             /* Universal */
```

EXAMPLE

CSS

```
div > .item {
   margin-right: 9px;
   margin-left: 9px;
   margin-bottom: 15px;
   margin-top: 15px;
   color: #ffffff;
   font-size: 0.8em;
}

.item .text {
   font-size: 0.8em;
}
```

CSS

```
.item {
   margin: 15px 9px;
   color: #fff;
   font-size: 0.8em;
}
```

IMAGE OPTIMIZATION

Optimizing your images for the web means saving or compiling your images in a web friendly format **depending on what the image contains.**

Save files in appropriate format depending on usage

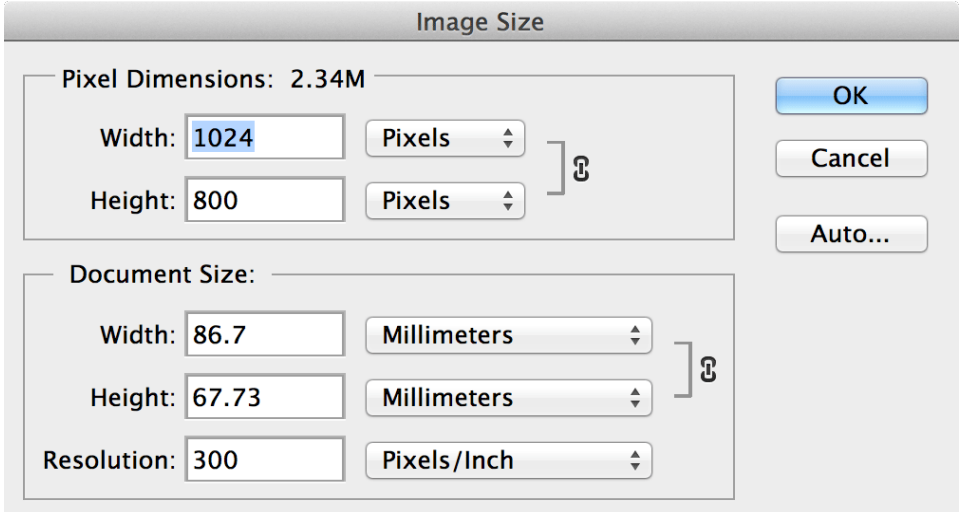
Resize image files themselves instead of via CSS

Use CSS & HTML instead of images if it's possible

OPTIMIZE IMAGE SIZE

Save your images in the proper dimensions. If you are having to use HTML or CSS to resize your images, stop right there. Save the image in the desired size to reduce the file size.

```
.img {
    width: 1024px;
    height: 800px;
}
```



PNG VS JPEG. SIZE

PNG-24:

1.28MB

JPEG @ 75 Quality:

288KB

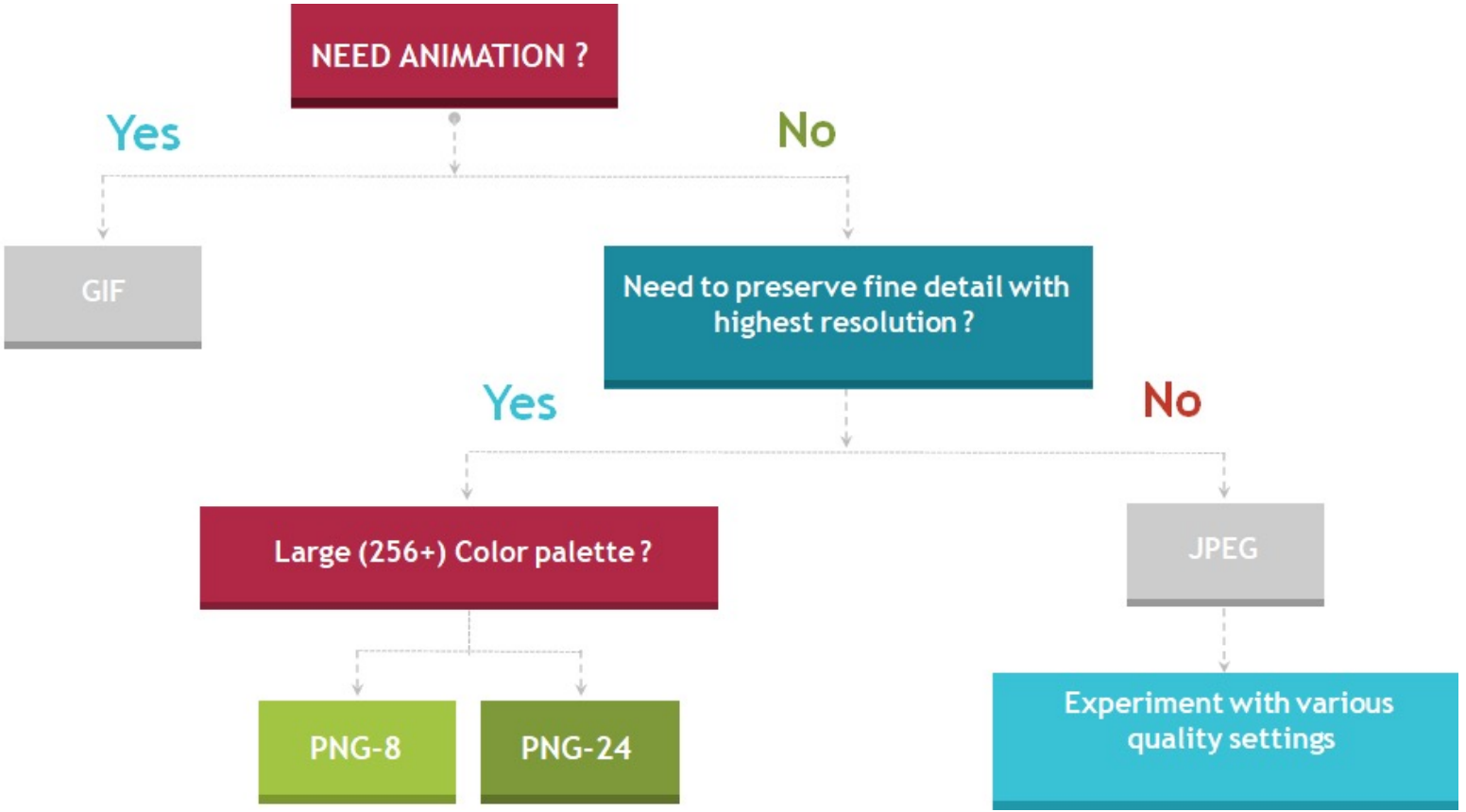
RASTER VS VECTOR. SCALING

PNG, 1.92 KB

SVG, 1.76 KB

(Scaled to 200% height & width)

WHICH FORMAT SHOULD YOU USE?



BASE 64

<img
src="[illegible]
mAPgTEMImIN9gUFCEm/gDALULDN8PAD6atYdCTX9gUNKlj8wZAKUsAOzZz+UMAOsJAP/Z2ccMDA8PD/95eX5NWvsJCOVNQPtfX/8zM8+QePLl38MGBr8JCP+zs9myn/8GB
qwpAP/GxgwJCPny78lzYLgjAJ8vAP9fX/+MjMUcAN8zM/9wcM8ZGcATEL+QePdZWf/29uc/P9cmJu9MTDImIN+/r7+/vz8/P8VNQGNugV8AAF9fX8swMNgTAFlDOICAgPN
SUnNWSMQ5MBAQEJE3QPIGAM9AQMqGcG9vb6MhJsEdGM8vLx8fH98AANIWAMuQeL8fABkTEPPQ0OM5OSYdGFl5jo+Pj/+pqcsTE78wMFNGQLYmID4dGPvd3UBAQJmTkP+8v
H9QUK+vr8ZWSHpzcJMmILdwcLOGcHRQUHxwcK9PT9DQ0O/v70w5MLypoG8wKOuwsP/g4P/Q0IcwKEswKMl8aJ9fX2xjdOtGRs/Pz+Dg4GImIP8gIH0sKEAwKKmTiKZ8aB/
f39Wsl+LFt8dgUE9PT5x5aHBwcP+AgP+WltdgYMyZfyywz78AAAAAAD///8AAP9mZv///wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAACH5BAEAAKgALAAAAAA9AEQAAAj/AFEJHEiwoMGDCBMqXMiwocAbBww4nEhxoYkUpzJGrMixogkfGUNqlNixJEIDB0SqHGmyJSojM1b
KZOmyop0gM3Oe2liTISKMOoPy7GnwY9CjIYcSRYm0aVKSLmE6nfq05QycVLPuhDrxBlCtYJUqNAq2bNWEBj6ZXRuyxZyDRtqwnXvkhACDV+euTeJm1Ki7A73qNWtFiF+/g
A95Gly2CJLDhwEHMOUAAuOpLYDEgBxZ4GRTlC1fDnpkM+fOqD6DDj1aZpITp0dtGCDhr+fVuCu3zlg49ijaokTZTo27uG7Gjn2P+hI8+PDPERoUB318bWbfAJ5sUNFcuGR
TYUqV/3ogfXp1rWlMc6awJjiAAd2fm4ogXjz56aypOoIde4OE5u/F9x199dlXnnGiHZWEYbGpsAEA3QXYnHwEFliKAgswgJ8LPeiUXGwedCAKABACCN+EA1pYIIYaFlcDh
ytd51sGAJbo3onOpajiihlO92KHGaUXGwWjUBChjSPiWJuOO/LYIm4v1tXfE6J4gCSJEZ7YgRYUNrkji9P55sF/ogxw5ZkSqIDaZBV6aSGYq/lGZplndkckZ98xoICbTcI
JGQAZcNmdmUc210hs35nCyJ58fgmIKX5RQGOZowxaZwYA+JaoKQwswGijBV4C6SiTUmpphMspJx9unX4KaimjDv9aaXOEBteBqmuuxgEHoLX6Kqx+yXqqBANsgCtit4FWQ
AEkrNbpq7HSOmtwag5w57GrmlJBASEU18ADjUYb3ADTinIttsgSB1oJFfA63bduimuqKB1keqwUhoCSK374wbujvOSu4QG6UvxBRydcpKsav++Ca6G8A6Pr1x2kVMyHwsV
xUALDq/krnrhPSOzXG1lUTIoffqGR7Goi2MAxbv6O2kEG56I7CSlRsEFKFVyovDJoIRTg7sugNRDGqCJzJgcKE0ywc0ELm6KBCCJo8DIPFeCWNGcyqNFE06ToAfV0HBRgx
svLThHn1oddQMrXj5DyAQgjEHSAJMWZwS3HPxT/QMbabI/iBCliMLEJKX2EEkomBAUCxRi42VDADxyTYDVogV+wSChqmKxEKCDAYFDFj4OmwbY7bDGdBhtrnTQYOigeChU
mc1K3QTnAUfEgGFgAWt88hKA6aCRIXhxnQ1yg3BCayK44EWdkUQcBByEQChFXfCB776aQsG0BIlQgQgE8qO26X1h8cEUep8ngRBnOy74E9QgRgEAC8SvOfQkh7FDBDmS43
PmGoIiKUUEGkMEC/PJHgxw0xH74yx/3XnaYRJgMB8obxQW6kL9QYEJ0FIFgByfIL7/IQAlvQwEpnAC7DtLNJCKUoO/w45c44GwCXiAFB/OXAATQryUxdN4LfFiwgjCNYg+
kYMIEFkCKDs6PKAIJouyGWMS1FSKJOMRB/BoIxYJIUXFUxNwoIkEKPAgCBZSQHQ1A2EWDfDEUVLyADj5AChSIQW6gu10bE/JG2VnCZGfo4R4d0sdQoBAHhPjhIB94v/wRo
RKQWGRHgrhGSQJxCS+0pCZbEhAAOw==">

BASE 64 ENCODING

How to

With this technique you can prevent multiple HTTP requests, reducing the loading time of a web page. You can force the e-mail client to display your images sent via e-mail. This is very helpful in newsletters.

HTML

```
<img width="16" height="16" alt="star"
src="
Op3WOyDZu6QdvCchPGolfO0o/XBs/fNwfjZ0frl3/zy7////wAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AACH5BAkAABAALAAAAAAQABAAAAVVICSOZGlCQAosJ6mu7fiyZeKqNK
ToQGDsM8hBADgUXoGAiqhSvp5QAnQKGIgUhwFUYLCVDFCrKUE1lBavA
ViFIDlTImbKC5Gm2hB0SlBCBMQiB0UjIQA7" />
```

CSS

```
 li {
background:url(
dfOulrSOp3WOyDZu6QdvCchPGolfO0o/XBs/fNwfjZ0frl3/zy7////wAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAACH5BAkAABAALAAAAAAQABAAAAVVICSOZGlCQAosJ6mu7fiyZeKqNKTo
QGDsM8hBADgUXoGAiqhSvp5QAnQKGIgUhwFUYLCVDFCrKUE1lBavAViFIDl
TImbKC5Gm2hB0SlBCBMQiB0UjIQA7) no-repeat left center;
    padding: 5px 0 5px 25px;
}
```

Why should you do this?

BASE 64

Pros:

- **It saves HTTP Requests.**
  You should only use this in documents that are heavily cached
  Having a CSS file that is 300k instead of 50k is fine if it saves 6 HTTP requests
- **Easy to convert online**

Cons:

- **Size**
  Size of embedded code is somewhat larger than size of resource by itself. GZip compression will help.
- **Maintenance**
  It's hard to maintain site with embedded data URIs for everything.

AVOID IMAGES

The best image isn't actually an image at all. Whenever possible, use the native capabilities of the browser to provide the same or similar functionality.

Wherever possible, **text should be text**, not embedded into images:

- never use images for headlines;
- avoid placing contact information like phone numbers or addresses directly into images.

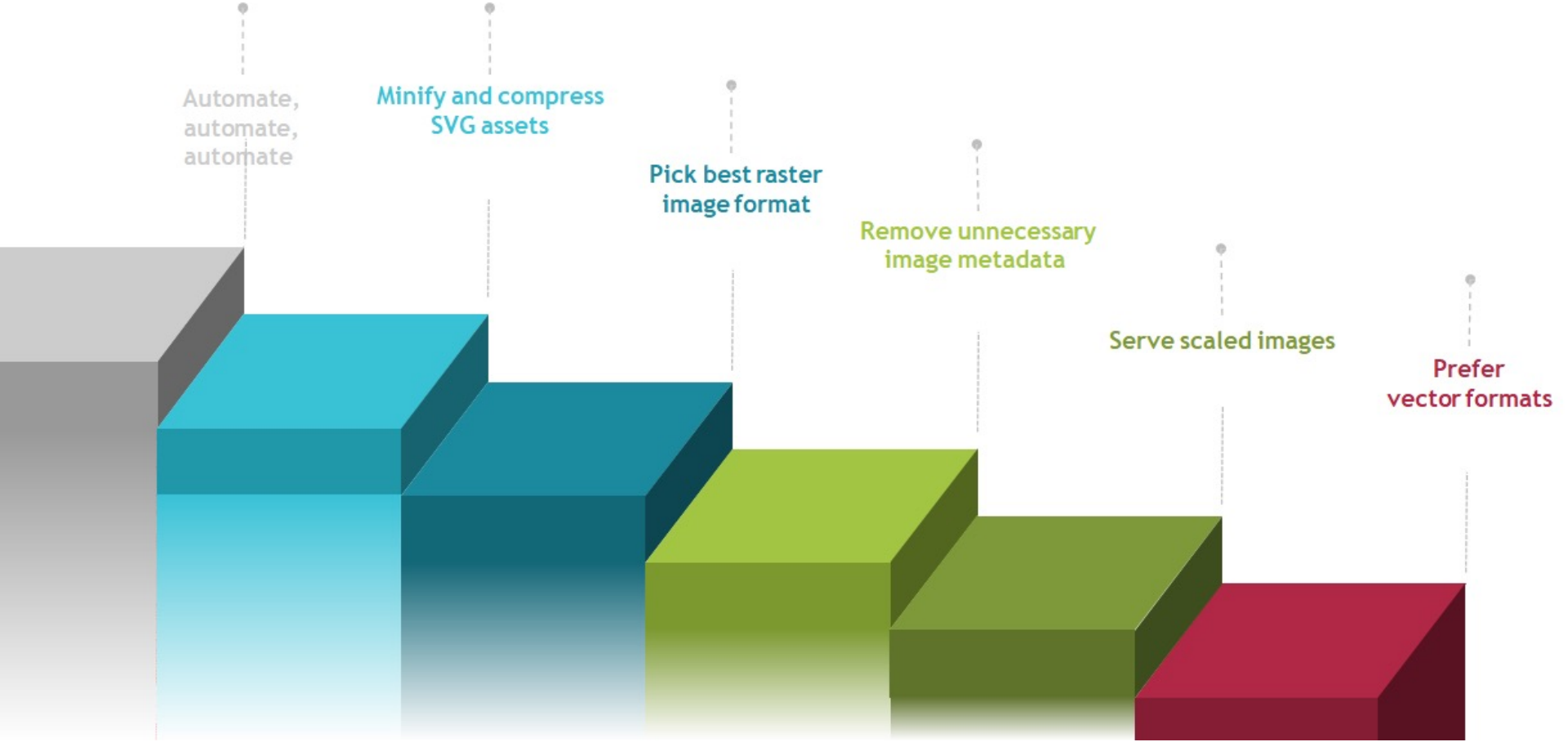
Place text in markup, instead of embedded in images

Use **CSS features** to create styles that previously required images:

- create complex gradients with background property;
- use box-shadow for shadows;
- border-radius for rounded corners.

Use CSS to replace images

IMAGE OPTIMIZATION: CHECKLIST



Performance

Performance is a huge subject, but it’s not always a “back-end” or an “admin” subject: it’s a **Front-End responsibility too**. The Front-End Performance Checklist is an exhausted list of elements you should check or at least be aware of, as a Front-End developer and apply to your projects (personal and professional).

Performance Tools

List of the tools you can use to test or monitor your website or application:

🛠️ WebPagetest — Website Performance and Optimization Test

🛠️ ☆ Dareboost: Website Speed Test and Website Analysis

🛠️ GTmetrix | Website Speed and Performance Optimization

🛠️ PageSpeed Insights

📖 Pagespeed — The tool and optimization guide

📖 Make the Web Faster | Google Developers

📖 Sitespeed.io — Welcome to the wonderful world of Web Performance

HTML

Minified HTML:

The HTML code is minified, comments, white spaces and new lines are removed from production files.

Why:

- Removing all unnecessary spaces, comments and break will reduce the size of your HTML and speed up your site’s page load times and obviously lighten the download for your user.

HTML

Remove unnecessary attributes:

Type attributes like type="text/javascript"' or type="text/css anymore and should be removed.

```
<!-- Before HTML5 -->
<script type="text/javascript">
    // Javascript code
</script>


<!-- Today -->
<script>
    // Javascript code
</script>
```

HTML

Place CSS tags always before JavaScript tags:

Ensure that your CSS is always loaded before having JavaScript code.

```
<!-- Not recommended -->
<script src="jquery.js"></script>
<script src="foo.js"></script>
<link rel="stylesheet" href="foo.css"/>


<!-- Recommended -->
<link rel="stylesheet" href="foo.css"/>
<script src="jquery.js"></script>
<script src="foo.js"></script>
```

CSS

Minification:

All CSS files are minified, comments, white spaces and new lines are removed from production files.

Why:

- When CSS files are minified, the content is loaded faster and less data are send to the client. It's important to always minified CSS files in production. It is beneficial for the user as it is for any business who wants to lower bandwidth costs and lower resource usage.

CSS

Concatenation:

CSS files are concatenated in a single file *(Not always valid for HTTP/2)*.

```
<!-- Not recommended -->
<link rel="stylesheet" href="="fonts.css"></link>
<link rel="stylesheet" href="components.js"></link>

<!-- Recommended -->
<link src="combined.js"></link>
```

CSS

Non-blocking:

CSS files need to be non-blocking to prevent the DOM from taking time to load.

```
<link rel="preload" href="global.min.css" as="style"
onload="this.rel='stylesheet'">
<noscript><link rel="stylesheet" href="global.min.css"></noscript>
```

CSS

Length of CSS classes:

The length of your classes can have an (slight) impact on your HTML and CSS files (eventually).

Why:

- Even performance impacts can be disputable, taking a decision on a naming strategy regarding your project can have a substantial impact on the maintainability of your stylesheets. If you are using BEM, in some cases, you can ended up with classes having more characters than need. It’s always important to choose wisely your names and namespaces.

CSS

Unused CSS:

Remove unused CSS selectors.

Why:

- Removing unused CSS selectors can reduce the size of your files and then speed up the load of your assets.

CSS

Embedded or inline CSS:

Avoid using embed or inline CSS inside your <body> *(Not valid for HTTP/2)*

Why:

- One of the first reason it's because it's a good practice to **separate content from design**. It also help you have a more maintainable code and keep your site accessible. But regarding performance, it's simply because it decrease the file-size of your HTML pages and the load time.

Fonts

Webfont formats:

You are using WOFF2 on your web project or application.

Why:

- According to Google, the WOFF 2.0 Web Font compression format offers 30% average gain over WOFF 1.0. It’s then good to use WOFF 2.0, WOFF 1.0 as a fallback and TTF.

How:

- **-** Check before buying your new font that the provider gives you the WOFF2 format. If you are using a free font, you can always use Font Squirrel to generate all the formats you need.

Fonts

Use preconnect to load your fonts faster:

```
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
```

Why:

- When you arrived on a website, your device needs to find out where you site live and which server it needs to connect with. Your browser had to contact a DNS server and wait for the lookup complete before fetching the ressource (fonts, CSS files…). Prefetches and preconnects allow the browser

Fonts

Webfont size:

Webfont sizes don’t exceed 300kb (all variants included)

Images

Your images are optimized, compressed without direct impact to the end user.

Why:

- Optimized images load faster in your browser and consume less data.

How:

- - Try using CSS3 effects when it’s possible (instead of a small image)
  - When it’s possible, use fonts instead of text encoded in your images
  - Use SVG
  - Use a tool and specify a level compression under 85.

Images

Lazy loading:

Images are lazyloaded.

Why:

- It will improve the response time of the current page and then avoid loading unnecessary images that the user may not need.

Images

Responsive images:

Ensure to serve images that are close to your display size.

Why:

- Small devices don’t need images bigger than their viewport. It’s recommended to have multiple versions of one image on different sizes.

How:

- **-** Create different image sizes for the devices you want to target.
  - Use srcset and picture to deliver multiple variants of each image.

JS Minification

All JavaScript files are minified, comments, white spaces and new lines are removed from production files *(still valid if using HTTP/2)*.

Why:

- Removing all unnecessary spaces, comments and break will reduce the size of your JavaScript files and speed up your site's page load times and obviously lighten the download for your user.

How:

- **-** Use the tools suggested below to minify your files automatically before or during your build or your deployment.

  🛠️ uglify-js - npm
  🛠️ Online JavaScript Compressor
  📖 Short read: How is HTTP/2 different? Should we still minify and concatenate?

No JavaScript inside:

(Only valid for website) Avoid having multiple JavaScript codes embedded in the middle of your body. Regroup your JavaScript code inside external files or eventually in the <head> or at the end of your page (before </body>).

Why:

- Placing JavaScript embedded code directly in your <body> can slow down your page because it loads while the DOM is being built. The best option is to use external files with async or defer to avoid blocking the DOM. Another option is to place some scripts inside your <head>. Most of the time analytics code or small script that need to load before the DOM gets to main processing.

How:

- Ensure that all your files are loaded using async or defer and decide wisely the code that you will need to inject in your <head>.

📖 11 Tips to Optimize JavaScript and Improve Website Loading Speeds

Non-blocking JavaScript:

JavaScript files are loaded asynchronously using async or deferred using defer attribute.

```
<!-- Defer Attribute -->
<script defer src="foo.js"></script>

<!-- Async Attribute -->
<script async src="foo.js"></script>
```

Why:

- JavaScript blocks the normal parsing of the HTML document, so when the parser reaches a <script>tag (particularly is inside the <head>), it stops to fetch and run it.
  Adding async or defer are highly recommended if your scripts are placed in the top of your page but less valuable if just before your </body> tag. But it's a good practice to always use these attributes to avoid any performance issue.

How:

- - Add async (if the script don't rely on other scripts) or defer (if the script relies upon or relied upon by an async script) as an attribute to your script tag.
  - If you have small scripts, maybe use inline script place above async scripts.

  📖 Remove Render-Blocking JavaScript
  📖 Defer loading JavaScript

Optimized and updated JS libraries:

All JavaScript libraries used in your project are necessary (prefer Vanilla JavaScript for simple functionalities), updated to their latest version and don't overwhelm your JavaScript with unnecessary methods.

Why:

- Most of the time, new versions come with optimization and security fix. You should use the most optimized code to speed up your project and ensure that you'll not slow down your website or app without outdated plugin.

How:

- If your project use NPM packages, npm-check is a pretty interesting library to upgrade / update your libraries. Greenkeeper can automatically look for your dependencies and suggest an update every time a new version is out.
  - 📖 You may not need jQuery
  - 📖 Vanilla JavaScript for building powerful web applications

Check dependencies size limit:

Ensure to use wisely external libraries, most of the time, you can use a lighter library for a same functionality.

Why:

- You may be tempted to use one of the 745 000 packages you can find on npm, but you need to choose the best package for your needs. For example, MomentJS is an awesome library but with a lot of methods you may never use, that's why Day.js was created. It's just 2kB vs 16.4kB gz for Moment.

How:

- Always compare and choose the best and lighter library for your needs. You can also use tools like npm trends to compare NPM package downloads counts or Bundlephobia to know the size of your dependencies.

  🛠️ ai/size-limit: Prevent JS libraries bloat. If you accidentally add a massive dependency, Size Limit will throw an error.
  🛠️ webpack-bundle-analyzer - npm
  🛠️ js-dependency-viewer - npm
  📖 Size Limit: Make the Web lighter — Martian Chronicles, Evil Martians' team blog

JavaScript Profiling:

Check for performance problems in your JavaScript files (and CSS too).

Why:

- JavaScript complexity can slow down runtime performance. Identifying these possible issues are essential to offer the smoothest user experience.

How:

- Use the Timeline tool in the Chrome Developer Tool to evaluate scripts events and found the one that may take too much time.

Server (Webpage size < 1500 KB)

Reduce the size of your page + resources as much as you can.

Why:

- Ideally you should try to target < 500 KB but the state of web shows that the median of Kilobytes is around 1500 KB (even on mobile). Depending on your target users, network connection, devices, it's important to reduce as much as possible your total Kilobytes to have the best user experience possible.

How:

- **-** All the rules inside the Front-End Performance Checklist will help you to reduce as much as possible your resources and your code.

📖 Page Weight
🛠️ What Does My Site Cost?

Server (Page load times < 3 seconds)

Reduce as much as possible your page load times to quickly deliver your content to your users.

Why:

- Faster your website or app is, less you have probability of bounce increases, in other terms you have less chances to lose your user or future client. Enough researches on the subject prove that point.

How:

- Use online tools like Page Speed Insight or WebPageTest to analyze what could be slowing you down and use the Front-End Performance Checklist to improve your load times.

🛠️ Compare your mobile site speed
🛠️ Test Your Mobile Website Speed and Performance - Think With Google
📖 Average Page Load Times for 2018 - How does yours compare? - MachMetrics Speed Blog

FE Online UA Training Course Feedback

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

serhii_shcherbak@epam.com

With the note [FE Online UA Training Course Feedback]

Thank you.

Q&A

UA Frontend Online LAB