# Frontend Optimization

November 2022

## Agenda

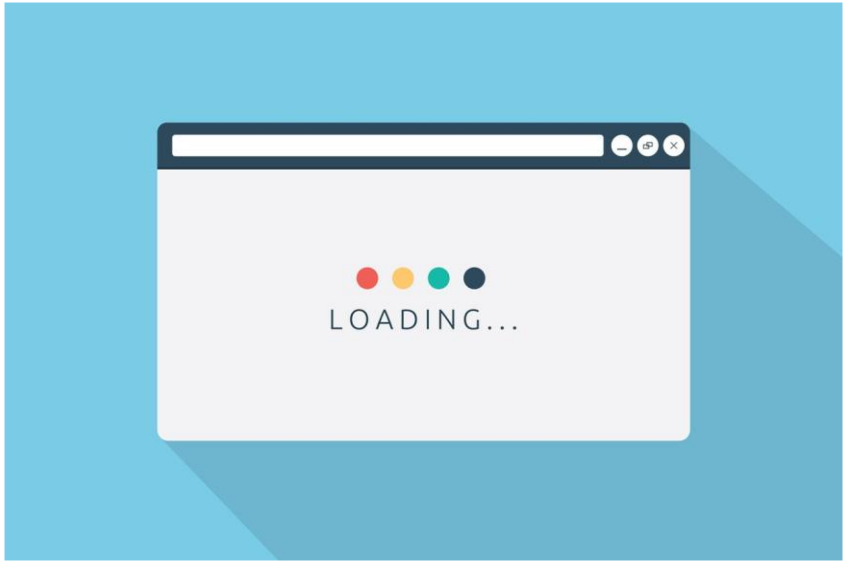
## INTRO

## 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.

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.



# 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%

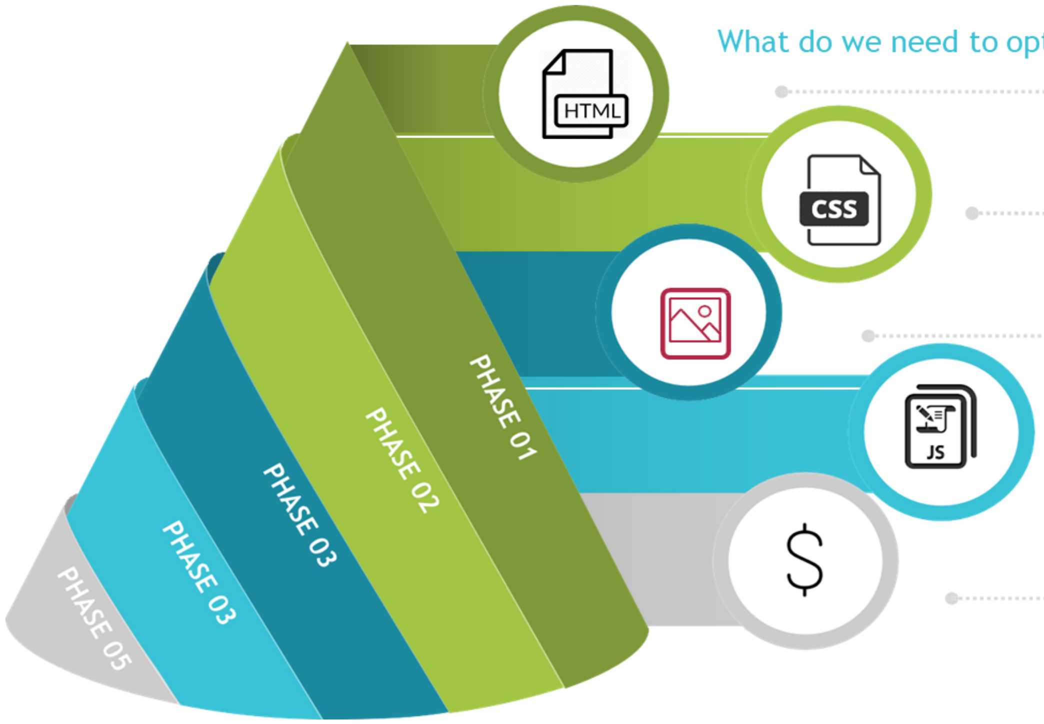
**Tell others** about their negative experience

1%

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

## LOAD SPEED OPTIMIZATION

## OPTIMIZATION



What do we need to optimize ?

HTML

CSS

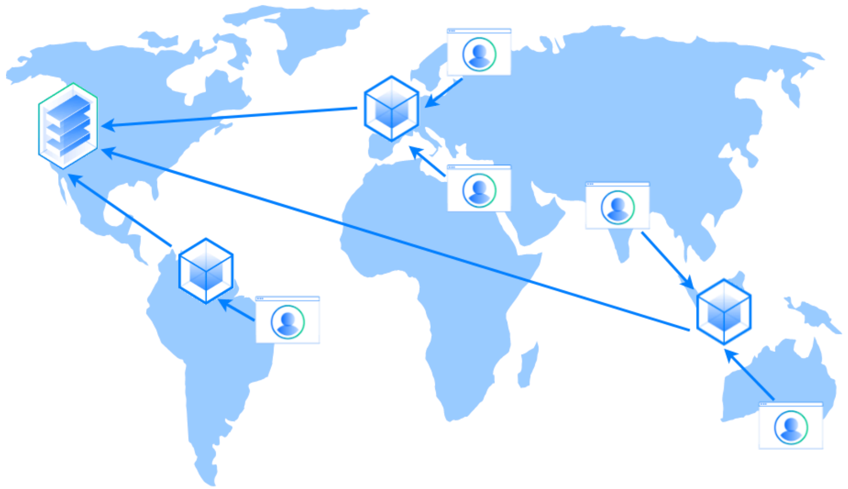
Images

Javascript

jQuery

## CDN

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.

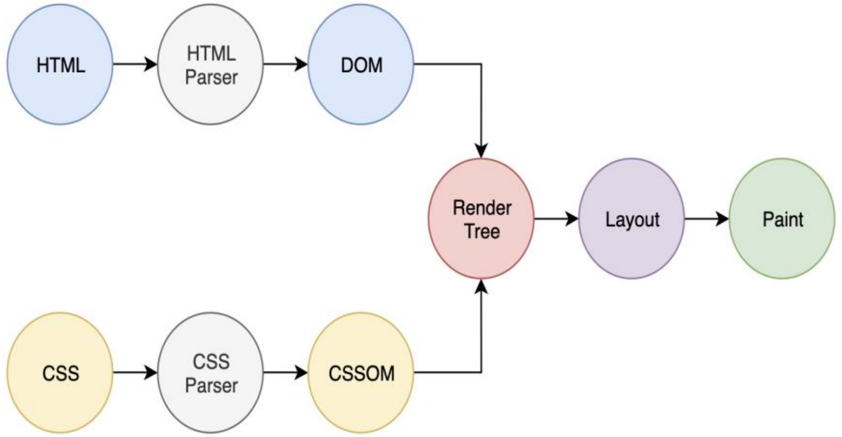


Content Delivery Network (CDN)

## 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.



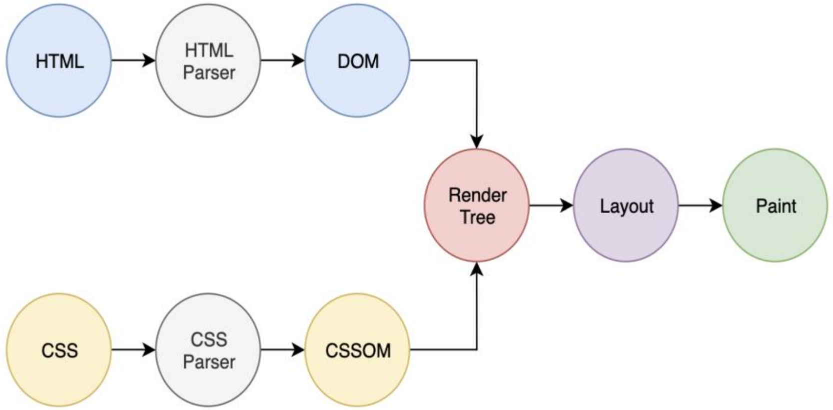
# RENDER, LAYOUT & PAINT

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.



## RENDER, LAYOUT & PAINT

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:
  - 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 its styles.

## HTML OPTIMIZATION

# HTML OPTIMIZATION

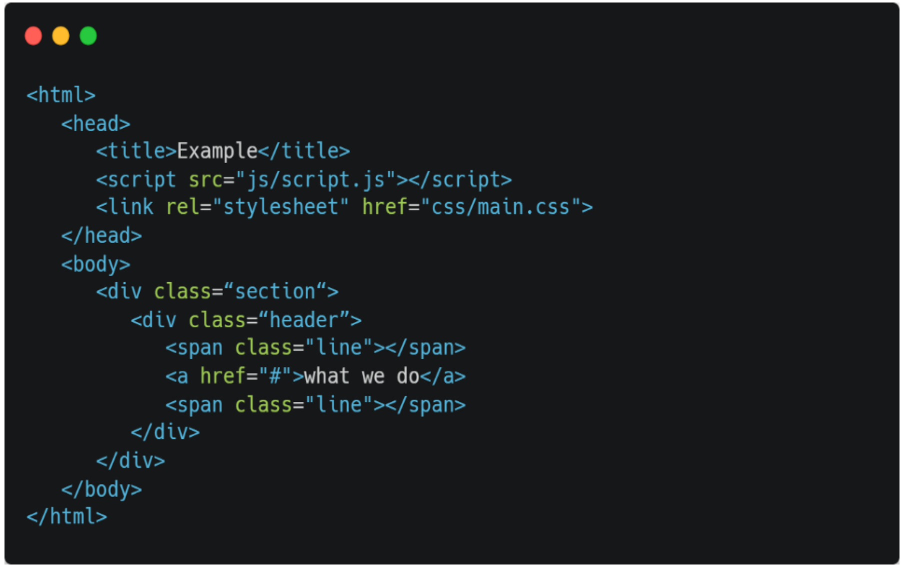
The reason to keep markup clean is not so much about faster load times, as it is about having a solid and robust foundation to build upon.

**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

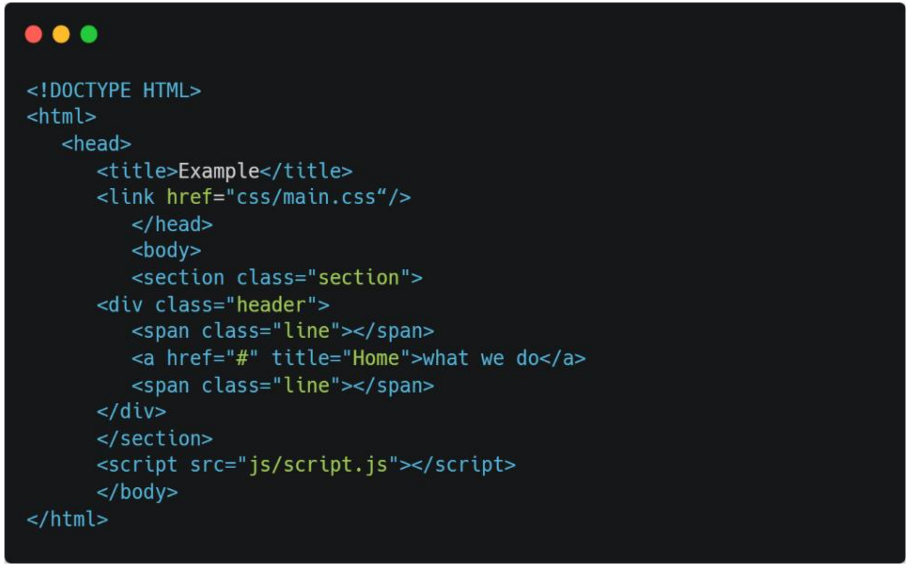
**Use semantic tags**
Semantic code tends to improve your placement on search engines, and make code more readable

**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>
```

## HTML OPTIMIZATION. USE CASES

Remove unnecessary attributes:

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

Minify HTML:

- 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.

Place CSS tags always before JavaScript tags:

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

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

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

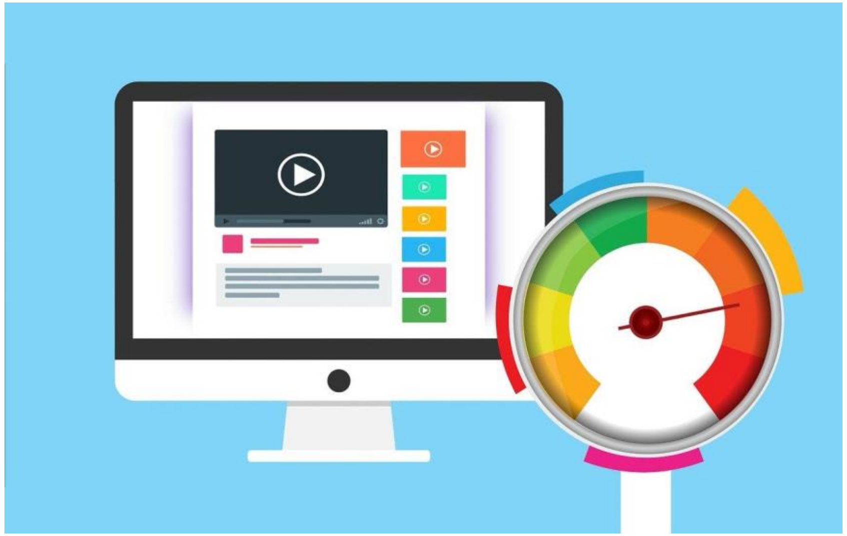
```
<!-- Not recommended -->
<script src="jquery.js"></script>
<script src="foo.js"></script>
<link rel="stylesheet" href="foo.css"/>

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

## CSS OPTIMIZATION

## CSS OPTIMIZATION

- Combine multiple CSS files
- Don’t use inline styles
- Prefer <link> over @import
- Minify your stylesheets
- Rely on inheritance
- Simplify Selectors
- Use CSS sprites
- Use animation on the GPU
- Avoid Base64 Bitmap Images
- Check Google Page Speed recommendations



## CSS OPTIMIZATION. USE CASES

**Minification:**

- When CSS files are minified, the content is loaded faster, and less data are sent 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.

**Concatenation:**

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

**Non-blocking CSS:**

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

**Length of CSS classes:**

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

**Unused CSS:**

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

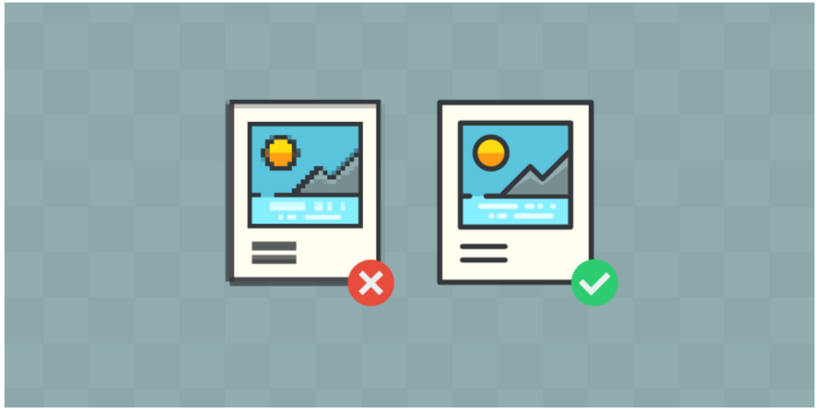
**Embedded or inline CSS:**

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

## IMAGE OPTIMIZATION

## 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.



The benefits of image optimization include:

- **Reduced bounce rates** – users are more likely to stay on the page if it loads quickly
- **Improved customer engagement** – images are a central part of the user experience in a modern website. If images load slowly (even if the rest of the content is visible), web visitors are less likely to engage with the website and eventually convert.
- **Cost savings** – bandwidth costs money, and by optimizing images, websites use less of it and save on hosting and content delivery costs.
- **Improved search rankings** – because page load time and performance is a ranking factor on Google and other search engines. Optimizing images can directly lead to higher rankings and more traffic.

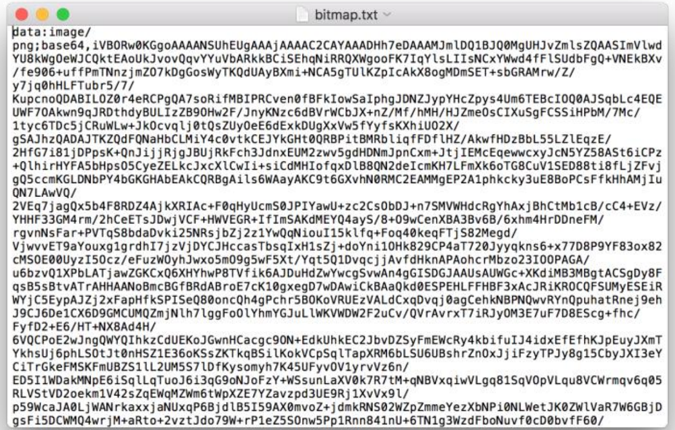
# IMAGE USAGE

| | BMP | JPG | PNG | GIF |
|---|---|---|---|---|
| **Compressed** | FALSE | TRUE | TRUE | TRUE |
| **Lossless** | TRUE | FALSE | TRUE | TRUE |
| **Transparency** | FALSE | FALSE | TRUE | TRUE |
| **Translucency** | FALSE | FALSE | TRUE | FALSE |
| **Recommended for photographs** | FALSE | TRUE | FALSE | FALSE |
| **Recommended for static graphics/icons** | FALSE | FALSE | TRUE | FALSE |
| **Recommended for animated graphics/icons** | FALSE | FALSE | FALSE | TRUE |

| RASTER | Image types | Where to use |
|---|---|---|
| **JPEG** | Photographs, illustrations, web and digital graphics | Website, PowerPoint and Keynote presentation. Ideal for when small files are required. |
| **TIFF** | Photographs and illustrations | Print. Ideal for when high resolution is required. |
| **GIF** | Web and digital graphics | Web applications, PowerPoint and Keynote presentations. Ideal for when you want a simple animation. |
| **PNG** | Web and digital graphics | Web applications, PowerPoint and Keynote presentations, and digital printed materials. Ideal for when you need an image with a transparent background. |
| **VECTOR** | **Image types** | **Where to use** |
| **EPS** | Logos, diagrams, illustrations, figures, charts | Print materials produced using professional print design programs like Adobe InDesign or Illustrator. |

# Base64 Image

Base64 is an encoding algorithm that converts any characters, binary data, and even images or sound files into a readable string, which can be saved or transported over the network without data loss. ... Base64 images are primarily used to embed image data within other formats like HTML, CSS, or JSON.



```
data:image/
png;base64,iVBORw0KGgoAAAANSUhEUgAAAjAAAAC2CAYAAADHh7eDAAAMJmlDQ1BJQ0MgUHJvZmlsZQAASImVlwd
YU8kWg0eWJCQktEAoUkJvovQqvYYuVbARkkBCiSEhqNiRRQXWgooFK7IqYlsLIIsNCxYWwd4fFlSUdbFgQ+VNEkBXv
/fe906+uffPmTNnzjmZO7kDgGosWyTKQdUAyBXmi+NCA5gTUlKZpIcAkX8ogMDmSET+sbGRAMrw/Z/
y7jq0hHLFTubr5/7/
KupcnoQDABILOZ0r4eRCPgQA7soRifMBIPRCven0fBFkIowSaIphgJDNZJypYHcZpys4Um6TEBcIOQ0AJSqbLc4EQE
UWF7OAkwn9qJRDthdyBULIzZB9OHw2F/JnyKNzc6dBVrWCbJX+nZ/Mf/hMH/HJZmeOsCIXuSgFCSSiHPbM/7Mc/
1tyc6TDc5jCRuWLw+JkOcvqlj0tQsZUyOeE6dExkDUgXxVw5fYyfsKXhiUO2X/
gSAJhzQADAJTKZQdFQNaHbCLMiY4c0vtkCEJYkGHt0QRBPitBMRbliqfFDflHZ/AkwfHDzBbL55LZlEqzE/
2HfG7i81jDPpsK+QnJijjRjgJBUjRkFch3JdnxEUM2zwv5gdHDNmJpnCxm+JtjIEMcEqewwcxyJcN5YZ58ASt6iCPz
+QlhirHYFA5bHpsO5CyeZELkcJxcXlCwIi+siCdMHIofqxDlB8QN2deIcmKH7LFmXk6oTG8CuV1SED88ti8fLjZFvj
gQ5ccmKGLDNbPY4bGKGHAbEAkCQRBgAils6WAayAKC9t6GXvhN0RMC2EAMMgEP2A1phkcky3uE8BoPCsFfkHhAMjIu
QN7LAwVQ/
2VEq7jagQx5b4F8RDZ4AjkXRIAc+F0qHyUcmS0JPIYawU+zc2CsObDJ+n7SMVWHdcRgYhAxjBhCtMb1cB/cC4+EVz/
YHHF33GM4rm/2hCeETsJDwjVCF+HWVEGR+IfImSAKdMEYQ4ayS/8+O9wCenXBA3Bv6B/6xhm4HrDDneFM/
rgvnNsFar+PVTqS8bdaDvki25NRsjbZj2z1YwQqNiouI15klfq+Foq40keqFTjS82Megd/
VjwvvET9aYouxg1grdhI7jzVjDYCJHccasTbsqIxH1sZj+doYni1OHk829CP4aT720Jyyqkns6+x77D8P9YF83ox82
cMSOE00UyzI5Ocz/eFuzWOyhJwxo5mO9g5wF5Xt/Yqt5Q1DvqcjjAvfdHknAPAohcrMbzo23IOOPAGA/
u6bzvQ1XPbLATjawZGKCxQ6XHYhwP8TVfik6AJDuHdZwYwcgSvwAn4gGISDGJAAUsAUWGc+XKdiMB3MBgtACSgDy8F
qsB5sBtvATrAHHAANoBmcBGfBRdABroE7cK10gxegD7wDAwiCkBAaQkd0ESPEHLFFHBF3xAcJRiKROCQFSUMyESEiR
WYjC5EypAJZj2xFapHfkSPISeQ80oncQh4gPchr5BOKoVRUEzVALdCxqDvqj0agCehkNBPNQwvRYnQpuhatRnej9eh
J9CJ6De1CX6D9GMCUMQZmjNlh7lggFoOlYhmYGJuLlWKVWDW2F2uCv/QVrAvrxT7iRJyOM3E7uF7D8EScg+fhc/
FyfD2+E6/HT+NX8Ad4H/
6VQCPoE2wJngQWYQIhkzCdUEKoJGwnHCacgc9ON+EdkUhkEC2JbvDZSyFmEWcRy4kbifuIJ4idxEfEfhKJpEuyJXmT
YkhsUj6phLSOtJt0nHSZ1E36oKSsZKTkqBSilKokVCpSqlTapXRM6bLSU6UBshrZnOxJjiFzyTPJy8g15CbyJXI3eY
CiTrGkeFMSKFmUBZS1lL2UM5S7lDfKysomyh7K45UFyvOV1yrvVz6n/
ED5I1WDakMNpKZRRdSl1B3UE9Rb1Ddqamomar5qqWpCtaVqtWqn1O6rfVCnq9urh6pz1eep
RLVStVD2oekm1V42sZqEWqMZWm6tWpXZE7YZavzpd3UE9Rj1XvVx9l/
p59WcaJA0LjWANrkaxxjaNUxqP6BjdlB5I59AX0mvoZ+jdmkRNS02WZpZmmeYezXbNPi0NLWetJK0ZWlVaR7W6GBjD
gsFi5DCWMQ4wrjM+aRto+2vztJdo79W+rP1eZ5SOnw5Pp1Rnn841nU+6TN1g3WzdFboNuvf0cD0bvfF60/
```

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

## AVOID IMAGES

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

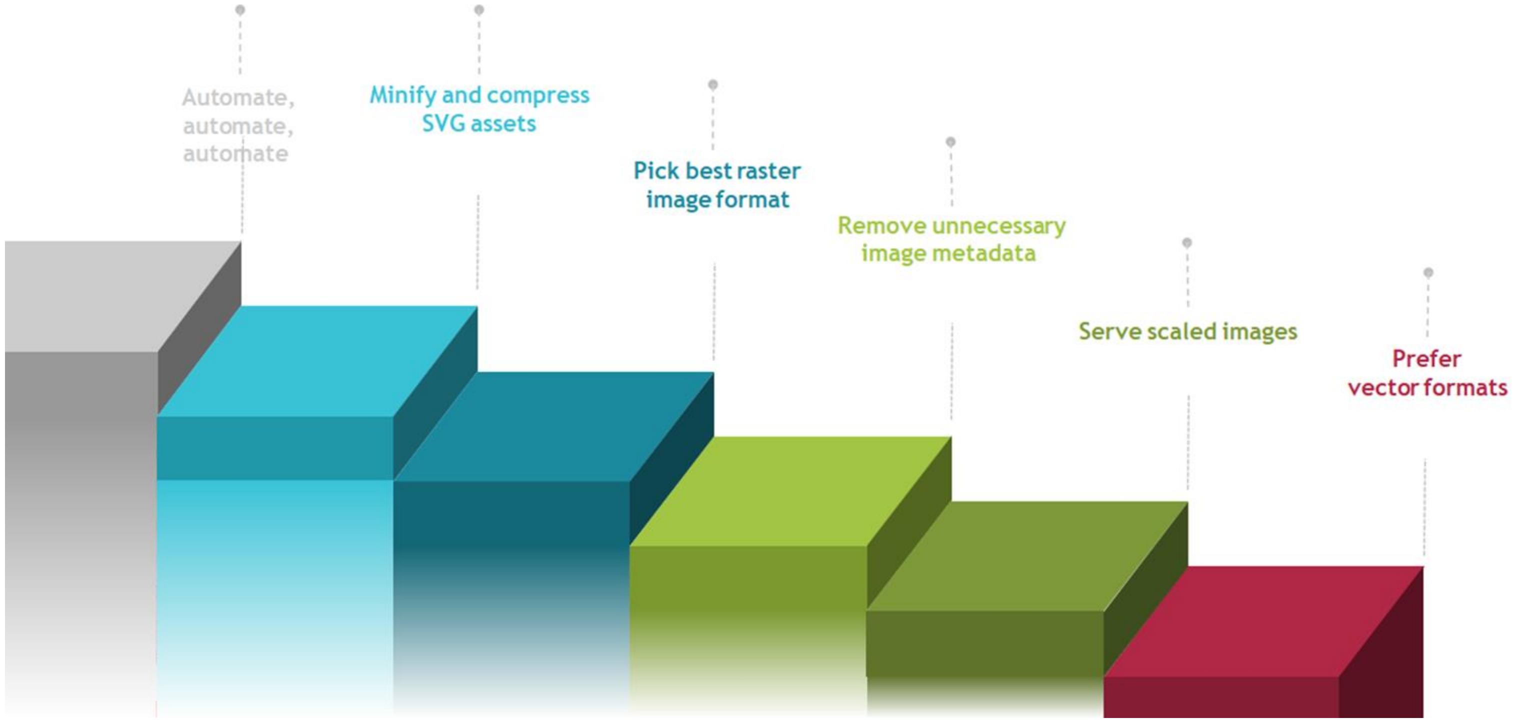
Wherever possible, text should be  text, not embedded into images:

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

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.

# IMAGE OPTIMIZATION: CHECKLIST


Automate, automate, automate
Minify and compress SVG assets
Pick best raster image format
Remove unnecessary image metadata
Serve scaled images
Prefer vector formats

# IMAGE OPTIMIZATION: USE CASES

- 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 are lazyloaded.
- Ensure to serve images that are close to your display size.
  - Create different image sizes for the devices you want to target.
  - Use *srcset* and picture to deliver multiple variants of each image.

## JS OPTIMIZATION

## JS Optimization

There are several challenges that can impact JavaScript performance.  Here are common issues associated with JavaScript performance:

- Poor quality event handling
- Unorganized code
- Too many dependencies
- Inefficient iterations

Tips for JavaScript performance optimization:

- Batching the DOM changes
- Learning the methods of asynchronous programming
- Utilizing gzip compression
- Utilize HTTP/2
- Lazy loading
- Restrict library dependencies
- Bundling
- Minification

# JS OPTIMIZATION: USE CASES

**Only sending the code a user needs**

- Use code-splitting to break up your JavaScript into what is critical and what is not. Module bundlers like webpack support code-splitting.
- Lazily loading in code that is non-critical.

**Caching code to minimize network trips.**

- Use HTTP caching to ensure browsers cache responses effectively. Determine optimal lifetimes for scripts (max-age) and supply validation tokens (ETag) to avoid transferring unchanged bytes.
- Service Worker caching can make your app network resilient and give you eager access to features like V8's code cache.
- Use long-term caching to avoid having to re-fetch resources that haven't changed. If using Webpack, see filename hashing.

**Minification**

- Use UglifyJS for minifying ES5 code.
- Use babel-minify or uglify-es to minify ES2015+.

**Compression**

- At minimum, use gzip to compress text-based resources.

**Removing unused code.**

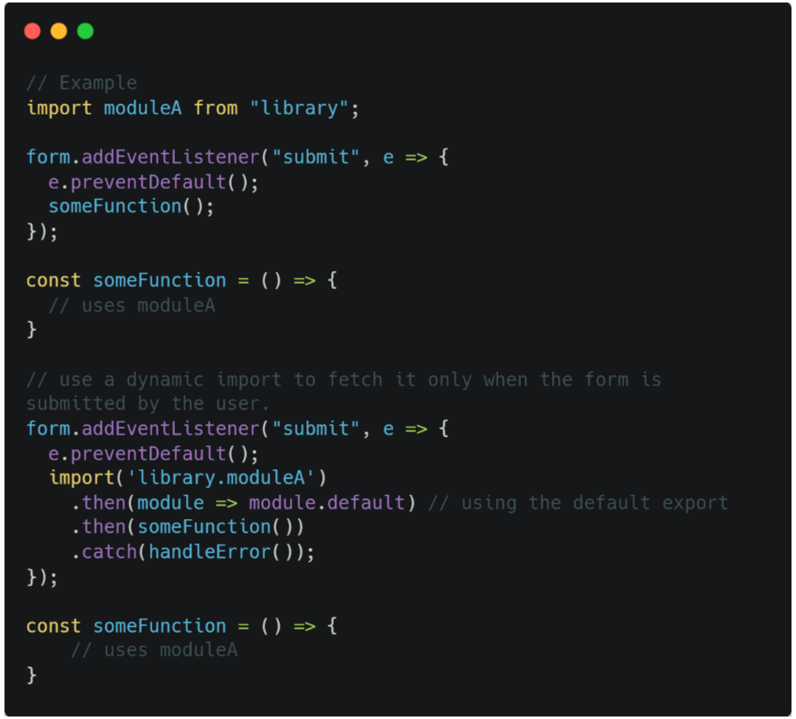
- Identify opportunities for code that can be removed or lazily loaded in with DevTools code coverage.
- Use babel-preset-env and browserlist to avoid transpiling features already in modern browsers. Advanced developers may find careful analysis of their webpack bundles helps identify opportunities to trim unneeded dependencies.
- For stripping code, see tree-shaking, Closure Compiler's advanced optimizations and library trimming plugins like lodash-babel-plugin or webpack's ContextReplacementPlugin for libraries like Moment.js.

# Reduce JavaScript payloads with code splitting

Sending large JavaScript payloads impacts the speed of your site significantly. Instead of shipping all the JavaScript to your user as soon as the first page of your application is loaded, split your bundle into multiple pieces and only send what's necessary at the very beginning.

Split the JavaScript bundle to only send the code needed for the initial route when the user loads an application. This minimizes the amount of script that needs to be parsed and compiled, which results in faster page load times.

Popular module bundlers like webpack, Parcel, and Rollup allow you to split your bundles using dynamic imports. For example, consider the following code snippet that shows an example of a someFunction method that gets fired when a form is submitted.



```
// Example
import moduleA from "library";

form.addEventListener("submit", e => {
  e.preventDefault();
  someFunction();
});

const someFunction = () => {
  // uses moduleA
}

// use a dynamic import to fetch it only when the form is
submitted by the user.
form.addEventListener("submit", e => {
  e.preventDefault();
  import('library.moduleA')
    .then(module => module.default) // using the default export
    .then(someFunction())
    .catch(handleError());
});

const someFunction = () => {
    // uses moduleA
}
```

## How do you load third-party script efficiently?

If a third-party script is slowing down your page load, you have several options to improve performance:

- Load the script using the async or defer attribute to avoid blocking document parsing.
- Consider self-hosting the script if the third-party server is slow.
- Consider removing the script if it doesn't add clear value to your site.
- Consider Resource Hints like *<link rel=preconnect>* or *<link rel=dns-prefetch>* to perform a DNS lookup for domains hosting third-party scripts.

Third-party script optimization should be followed by on-going real-time performance monitoring of your scripts and communication with your third-party providers. The web is evolving at a rapid pace and a script’s locally observed performance gives no guarantees that it will perform as well in the future or in the wild.

THANK YOU!