Web Performance Made Easy

Addy Osmani
Ewa Gasperowicz
What makes a web page feel heavy?

1.5MB

Images: 800KB
JavaScript: 350KB

15s to load & get interactive

HTTP Archive
Lean and yet full of content
Why does performance matter?
How important is speed to users?

- 75% The speed it takes to load the page
- 66% How easy it is to find what I’m looking for
- 61% How well the site fits my screen
- 58% How simple the site is to use
- 24% How attractive the site looks

UX HIERARCHY
“Send less stuff!”
Paul Irish in your pocket.
New Lighthouse Web Performance Audits

bit.ly/lighthouse-perf
New Lighthouse Audits

- JavaScript Boot-up Time
- Preload key requests
- Avoid multiple, costly round trips to any origin (preconnect)
- Use a video format for animated content (instead of GIF)
- Main thread work breakdown
- Unminified JavaScript & CSS
- Unused CSS rules
- All text remains visible during webfont loads
- Uses efficient cache policy on static assets
Fixing web performance is as easy as drawing a horse

1. Draw 2 circles
2. Draw the legs
3. Draw the face
4. Draw the HAIR
5. Add small details
Google Doodles
Let’s start our journey.
Lighthouse.

Performance
These encapsulate your web app's current performance and opportunities to improve it.

Metrics
These metrics encapsulate your web app's performance across a number of dimensions.

- First meaningful paint: 15,800 ms
- First Interactive (beta): 15,800 ms
- Consistently Interactive (beta): 15,900 ms
- Perceptual Speed Index: 14.968
- Estimated Input Latency: 16 ms

Score: 23
TTI: 15s
Unnecessary resources
Send less stuff and fewer bytes
Minify and compress JavaScript and CSS

Before

<table>
<thead>
<tr>
<th>Minify CSS</th>
<th>1,790 ms</th>
<th>329 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable text compression</td>
<td>12,430 ms</td>
<td>2,286 KB</td>
</tr>
</tbody>
</table>

```javascript
const UglifyJsPlugin = require('uglifyjs-webpack-plugin')
const webpackConfig = {
  plugins: [
    new UglifyJsPlugin({})
  ]
}
```

$ firebase deploy

After

<table>
<thead>
<tr>
<th>Minify CSS</th>
<th>30 ms</th>
<th>5 KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable text compression</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inefficient cache policies

Short cache lifetimes can impact repeat visits

- Uses inefficient cache policy on static assets: 8 assets found

Firebase.json:

```
"source": "**/.*\(jpg|jpeg|gif|png|woff2\)",
"headers": [{
  "key": "Cache-Control",
  "value": "max-age=31536000"
}]
```

After

- Uses efficient cache policy on static assets: 1 asset found
Remove unnecessary bytes and don’t send things twice

Solution

- Minimize code
- Automate minification
  - UglifyJS
  - Cloudflare
  - mod_pagespeed
- Compress (gzip, brotli)
- Cache wherever possible
JS and CSS Code Unused

Scripts with unused code slowing page load
Unused code can surprise us
Check Code Coverage

Unused CSS rules

- /vendor.9b557ae0909ea1917622.js: JS, 21619 bytes, 2013610 unused (93.7%)
- /app.416e289bce96f2c386e94dd85.js: CSS, 193546 bytes, 185262 unused (95.7%)
- https://use.fontawesome.com/.../all.js: JS, 700147 bytes, 11282 unused (1.6%)
- h.../app.4c993baff29561fc6551.js: JS, 12234 bytes, 6202 unused (50.7%)
- /manifest.2ae2e69a05c33dfc65f8.js: JS, 857 bytes, 97 unused (11.3%)

Total: 2,280 ms, 419 KB

95% unused
Vue MDC Adapter

If we drop MVC adapter our styles drop to 10KB!

We have almost no unused CSS now

Hash: 4c0525cc18122141e4cf
Version: webpack 3.11.0
Time: 24134ms

Hash: 15bd9c13b98e73648db4
Version: webpack 3.11.0
Time: 25719ms

Unused CSS rules

30 ms
6 KB
Sanity check: Perf is up, how about UX?

**Before**

<table>
<thead>
<tr>
<th>Performance</th>
<th>Progressive Web App</th>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>36</td>
<td>88</td>
</tr>
</tbody>
</table>

**Performance**
These encapsulate your web app's current performance and opportunities to improve it.

**Metrics**
These metrics encapsulate your web app's performance across a number of dimensions.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First meaningful paint</td>
<td>6,800 ms</td>
</tr>
<tr>
<td>First Interactive (beta)</td>
<td>6,800 ms</td>
</tr>
<tr>
<td>Consistently Interactive (beta)</td>
<td>6,800 ms</td>
</tr>
<tr>
<td>Perceptual Speed Index</td>
<td>7,628</td>
</tr>
<tr>
<td>Estimated Input Latency</td>
<td>16 ms</td>
</tr>
</tbody>
</table>

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

- Enable text compression

**After**

<table>
<thead>
<tr>
<th>Performance</th>
<th>Progressive Web App</th>
<th>Best Practices</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>36</td>
<td>88</td>
</tr>
</tbody>
</table>

**Performance**
These encapsulate your web app's current performance and opportunities to improve it.

**Metrics**
These metrics encapsulate your web app's performance across a number of dimensions.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First meaningful paint</td>
<td>4,850 ms</td>
</tr>
<tr>
<td>First Interactive (beta)</td>
<td>4,850 ms</td>
</tr>
<tr>
<td>Consistently Interactive (beta)</td>
<td>4,850 ms</td>
</tr>
<tr>
<td>Perceptual Speed Index</td>
<td>6,311</td>
</tr>
<tr>
<td>Estimated Input Latency</td>
<td>16 ms</td>
</tr>
</tbody>
</table>

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

- Reduce render-blocking stylesheets
Did we miss anything? One component was using MDC.

We can manually copy/paste the lines needed to fix.
Remove unused JavaScript & CSS from the critical path.

Solution

- Code Coverage in DevTools
  - Page load
  - Runtime
- Lighthouse Coverage Audit
- Remove unused code to improve page load time
- Test for regressions
CHALLENGE

Bloated Web Pages

A “modern” web page
Detect enormous network payloads

Diagnostics
More information about the performance of your application.

Has enormous network payloads: Total size was 3,256 KB
Large network payloads cost users real money and are highly correlated with long load times. Learn more.

View Details

<table>
<thead>
<tr>
<th>URL</th>
<th>Total Size</th>
<th>Transfer Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>...js/vendor.9b557ae....js (127.0.0.1)</td>
<td>2,122 KB</td>
<td>11,540 ms</td>
</tr>
<tr>
<td>...js/all.js (use.fontawesome.com)</td>
<td>290 KB</td>
<td>1,580 ms</td>
</tr>
<tr>
<td>...css/app.416e289....css (127.0.0.1)</td>
<td>189 KB</td>
<td>1,030 ms</td>
</tr>
<tr>
<td>...img/seats.png (127.0.0.1)</td>
<td>187 KB</td>
<td>1,020 ms</td>
</tr>
</tbody>
</table>

3.2MB payload
The fastest request is the one not made.
JavaScript Bundle Auditing

Webpack Bundle Analyzer

So.js

import-cost for Visual Code

```javascript
import slug from 'slug' 1.7M (gzipped: 98.6K)

export default {
  name: 'DoodleSlider',
}
```

BundlePhobia

npm run build --report
Auditing JavaScript bundles paid off.

Diagnostics
More information about the performance of your application.

- Has enormous network payloads: Total size was 3,256 KB
  Large network payloads cost users real money and are highly correlated with long load times. Learn more.

  View Details

<table>
<thead>
<tr>
<th>URL</th>
<th>Total Size</th>
<th>Transfer Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>...js/vendor.9b557ae...js (127.0.0.1)</td>
<td>2,122 KB</td>
<td>11,540 ms</td>
</tr>
</tbody>
</table>

- Avoids enormous network payloads: Total size was 1,569 KB
  Large network payloads cost users real money and are highly correlated with long load times. Learn more.

  View Details

<table>
<thead>
<tr>
<th>URL</th>
<th>Total Size</th>
<th>Transfer Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>...js/vendor.3bbfac6...js (127.0.0.1)</td>
<td>435 KB</td>
<td>2,370 ms</td>
</tr>
</tbody>
</table>

Bonus: Saved another 320KB discovering an unused dependency!

2.1MB saved
65% smaller
Eliminate unnecessary downloads.

Solution

- Make an inventory of all assets
- Measure value & impact of assets
- Audit your assets regularly
JavaScript Boot-up Time Is High
JavaScript Boot-up time is high

- JavaScript boot-up time is too high: 1,850 ms
  Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.

- View Details

<table>
<thead>
<tr>
<th>URL</th>
<th>Script Evaluation</th>
<th>Script Parsing &amp; Compile</th>
</tr>
</thead>
<tbody>
<tr>
<td>...js/app.1b5c8ae....js</td>
<td>1,516 ms</td>
<td>120 ms</td>
</tr>
<tr>
<td>...js/all.js</td>
<td>81 ms</td>
<td>10 ms</td>
</tr>
</tbody>
</table>

Before code-splitting (static import)

```javascript
import DoodleHome from './DoodleHome'
import DoodleBrowse from './DoodleBrowse'
import DoodleFullscreen from './DoodleFullscreen'
import DoodleOffline from './DoodleOffline'
```
Use JavaScript Code-splitting

Split by route
Split by component
Code-splitting reduced JavaScript Boot-up Time

After code-splitting (dynamic import)

```javascript
const DoodleHome = () => import('./DoodleHome')
const DoodleBrowse = () => import('./DoodleBrowse')
const DoodleFullscreen = () => import('./DoodleFullscreen')
const DoodleOffline = () => import('./DoodleOffline')
```

![Asset Size](image)

- JavaScript boot-up time: 780 ms
- Consider reducing the time spent parsing, compiling, and executing JS. You may find delivering smaller JS payloads helps with this. Learn more.
- 0.78s boot-up
- 56% faster
Fast JS = fast at

- Download
- Parse
- Compile
- Execute

Solution

- Only send code users need.
- Use code-splitting
  - Split routes
  - Split components
  - Split vendor bundles
- Consider tree-shaking
- Serve modern, smaller JS bundles to modern browsers
- Remove unused library code with bit.ly/webpack-libs
I'd tell you a load performance joke, but
CHALLENGE

Unoptimized Images
Images that are large, inefficient or unnecessary
Lighthouse Image Audits

- **Optimize images**
  
  Optimized images load faster and consume less cellular data. Learn more.

- **Properly size images**
  
  Serve images that are appropriately-sized to save cellular data and improve load time. Learn more.

- **Serve images in next-gen formats**
  
  Image formats like JPEG 2000, JPEG XR, and WebP often provide better compression than PNG or JPEG, which means faster downloads and less data consumption. Learn more.
Image Optimization Tools

GUI

- ImageOptim (Mac)
- XNConvert (Cross-platform)

Build Process

- imagemin
- libvips

Roll your own CDN

- Thumbor
- ImageFlow

CDN

- Cloudinary
- Imgix
- Fastly
- Akamai
Optimize Images

Before

```
static/img/strip_light.png  6.82 kB
static/img/oodle.png  190 kB
static/img/background.png  1.21 MB
static/img/strip_small_light.png  3.09 kB
static/img/seats.png  192 kB
```

After

```
static/img/this_week.png  9.21 kB
static/manifest.json  297 bytes
static/ponyexpress15.htm  30.9 kB
static/img/background.png  105 kB
```

Lighthouse is happy too

- Optimize Images
- Properly size images
- Serve images in next-gen formats
Animated GIFs can be expensive

Large GIFs are inefficient for delivering animated content. Consider using MPEG4/WebM videos for animations and PNG/WebP for static images instead of GIF to save network bytes. [Learn more](#)

<table>
<thead>
<tr>
<th>URL</th>
<th>Transfer Size</th>
<th>Byte Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>v152.../doodle-theatre_es8oc9.gif (res.cloudinary.com)</td>
<td>7,360 KB</td>
<td>7,286 KB</td>
</tr>
</tbody>
</table>
Replace Animated GIFs with `<video>`

80%+ savings

```
<video autoplay muted playsinline>
  <source src="video.webm" type="video/webm"/>
  <source src="video.mp4" type="video/mp4"/>
</video>
```

```
ffmpeg -i animation.gif -b:v 0 -crf 40 -vf scale=600:-1 video.mp4
```
Adapt based on user’s effective network connection

navigator.connection.effectiveType

4G+ get <video>
~1MB

2G-3G get static <img>
~30KB
Carousels often load unnecessary images
LazySizes

1. Include library

```python
import lazysizes from 'lazysizes'
```

```html
<script src="lazysizes.min.js"></script>
```

2. Use

```html
<!-- non-responsive: -->
<img data-src="image.jpg"
    class="lazyload"/>

<!-- responsive: -->
<img class="lazyload"
    data-sizes="auto"
    data-src="image2.jpg"
    data-srcset="image1.jpg 300w,
                image2.jpg 600w,
                image3.jpg 900w"/>
```
Don’t serve unoptimized or unnecessary images to your users.

Solution

- Optimize images
- Use responsive Images
  - `<img srcset>`, `<picture>`
  - Media Queries
  - Client-hints
- Use lighter formats (SVG, video)
- Lazy-load offscreen images
Image Optimisation Book

Balance quality & size  Choose the right format  Size appropriately  Compress carefully

Adapt intelligently  Prioritize critical images  Lazy-load offscreen images  Remove image metadata

https://images.guide
Resources Discovered & Delivered Late

<table>
<thead>
<tr>
<th>Type</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>stylesheet</td>
<td>Highest</td>
</tr>
<tr>
<td>png</td>
<td>Low</td>
</tr>
<tr>
<td>png</td>
<td>Low</td>
</tr>
<tr>
<td>script</td>
<td>Medium</td>
</tr>
<tr>
<td>script</td>
<td>Medium</td>
</tr>
<tr>
<td>script</td>
<td>Medium</td>
</tr>
</tbody>
</table>
Let the browser know what’s important to fetch.

<link rel=preconnect/>
<link rel=preload/>
<link rel=prefetch/>
Avoid multiple, costly round trips to any origin

<table>
<thead>
<tr>
<th>Origin</th>
<th>Potential Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://fonts.googleapis.com">https://fonts.googleapis.com</a></td>
<td>300 ms</td>
</tr>
<tr>
<td><a href="https://fonts.gstatic.com">https://fonts.gstatic.com</a></td>
<td>300 ms</td>
</tr>
</tbody>
</table>

**Google Fonts** stylesheets + Web Fonts

Mask connection latency with rel="preconnect"

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

0.3s faster
### Preload key requests

<table>
<thead>
<tr>
<th>URL</th>
<th>Potential Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>../Montserrat.woff2 (fonts.gstatic.com)</td>
<td>790 ms</td>
</tr>
<tr>
<td>../Teko.woff2 (fonts.gstatic.com)</td>
<td>800 ms</td>
</tr>
</tbody>
</table>

**Web Fonts**

```html
<link rel="preload" as="font" href="webfont.woff2" type="font/woff2" crossorigin="anonymous">
```
Web Fonts: with and without `<link rel=preload>`

Diagram is for illustrative purposes only

1s saved
Self-host Web Fonts for maximum control

Works around Google Fonts URLs expiring or changing

```html
url('https://fonts.gstatic.com/s/montserrat/v12/JTUSjIg1_i6t8kCHKm 459WRhyyTh89ZnPQ.woff2') format('woff2');
```

**Pros**
- `<link rel=preload>`
- `font-display`
- `unicode-range`
- Easier subsetting

**Cons**
- Lose Google Fonts cache-hit rate
- Lose Google Fonts server optimizations
- Have to check for updates
Help browsers deliver critical resources early.

Solution

- Connecting to critical origins?
  - `<link rel=preconnect>`

- Asset for current page?
  - `<link rel=preload>`
  - `preload-webpack-plugin`

- Asset for future navigation?
  - `<link rel=prefetch>`
  - `webpack 4.6 (prefetch, preload)`

Review the legal playbook to learn how to source imagery, amongst other important legal guidelines: [go/iospeakerplaybook](#). For every image used in the deck, add source details in the “For Legal Review” slide.
<link rel="preload" as="script" href="critical-script.js"/>
<link rel="preload" as="style" href="theme.css" importance="low" onload="this.rel=stylesheet"/>

<!-- critical-path styles */\n
 */ critical-path styles */

<style>
/* critical-path styles */
</style>

<!-- superfluous fetch requests -->

<script>
fetch('/api/related.json', { importance: 'low' });
</script>

<!-- scripts at the end of the document -->

<script src="critical-script.js"></script>
Browser prioritized images

Fix using `<img importance>`
Invisible Text While Web Fonts Load

PROBLEM

Request page → Get HTML → Get CSS → Get Web Font

Render blocking → Blocked text painting
Avoid invisible text while Web Fonts are loading
Leverage the font-display CSS feature to ensure text is user-visible while webfonts are loading. Learn more.

View Details

<table>
<thead>
<tr>
<th>Font URL</th>
<th>Font download time</th>
</tr>
</thead>
<tbody>
<tr>
<td>...teko-v7-latin/teko-v7-latin-regular.woff2 (oodle-demo.firebaseapp.com)</td>
<td>947 ms</td>
</tr>
<tr>
<td>...montserrat-v12-latin/montserrat-v12-latin-regular.woff2 (oodle-demo.firebaseapp....</td>
<td>1,128 ms</td>
</tr>
<tr>
<td>...teko-v7-latin/teko-v7-latin-300.woff2 (oodle-demo.firebaseapp.com)</td>
<td>786 ms</td>
</tr>
<tr>
<td>...montserrat-v12-latin/montserrat-v12-latin-600.woff2 (oodle-demo.firebaseapp.com)</td>
<td>859 ms</td>
</tr>
</tbody>
</table>

Flash of invisible text

Fully loaded Web Font
@font-face {
font-family: 'Montserrat';
font-style: normal;
font-display: swap;
font-weight: 400;
src: local('Montserrat Regular'), local('Montserrat-Regular'),
    /* Chrome 26+, Opera 23+, Firefox 39+ */
url('montserrat-v12-latin-regular.woff2') format('woff2'),
    /* Chrome 6+, Firefox 3.6+, IE 9+, Safari 5.1+ */
url('montserrat-v12-latin-regular.woff') format('woff');
}
Have a Web Font loading strategy.

Solution

- `font-display: swap` or `optional`
- `<link rel=preload>`
- Web Font subsetting
- Font Loading API
- Use SVGs instead of icon fonts
- For more, see Zach Leatherman’s Web Font recipes at bit.ly/webfont-recipes
Render-blocking Scripts

External stylesheets block first paint of your page
External stylesheets impacted our metrics

Perceived performance could be improved

Reduce render-blocking stylesheets opportunity
Critical path optimisation

```html
<head>
  <style>/!* Inlined critical styles */</style>
  <link href="app.css" rel="preload" as="style"
       onload="this.onload=null;this.rel='stylesheet'">
</head>

<body>
  <div id="app">
    <!-- AppShell markup -->
  </div>
  <noscript>
    <link href="app.css" rel="stylesheet">
  </noscript>
</body>
```

Tools
Critical Penthouse loadCSS
Critical path optimisation

Before

After

1.2s saved on FMP + TTI
Reduce render-blocking scripts and stylesheets.

Solution

- Inline critical styles in `<head>` and preload/async load the rest
- Split styles into separate files organized by media query
- Mark non-critical scripts with the defer attribute or lazy-load

Review the legal playbook to learn how to source imagery, amongst other important legal guidelines: go/iospeakerplaybook.
18 Passed Audits

- Reduce render-blocking stylesheets
- Reduce render-blocking scripts
- Properly size images
- Offscreen images
- Minify CSS
- Minify JavaScript
- Unused CSS rules
- Optimize images
- Serve images in next-gen formats
- Enable text compression
- Avoids page redirects: 0 ms
- Preload key requests: 0 ms
- Avoids enormous network payloads: Total size was 178 KB
- Avoids an excessive DOM size: 43 nodes
- User Timing marks and measures: 0
- JavaScript boot-up time: 520 ms
- Main thread work breakdown: 790 ms
- All text remains visible during webfont loads
The final score

Performance
These encapsulate your web app’s current performance and opportunities to improve it.

Metrics
These metrics encapsulate your web app’s performance across a number of dimensions.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First meaningful paint</td>
<td>2,570 ms</td>
</tr>
<tr>
<td>First Interactive (beta)</td>
<td>2,570 ms</td>
</tr>
<tr>
<td>Consistently Interactive (beta)</td>
<td>2,570 ms</td>
</tr>
<tr>
<td>Perceptual Speed Index</td>
<td>2,424</td>
</tr>
<tr>
<td>Estimated Input Latency</td>
<td>16 ms</td>
</tr>
</tbody>
</table>

2.6s TTI
Case Study
Nikkei
NIKKEI

+230% Organic traffic
+58% Conversion rate
+40% Daily active users
+28% Page-views

Performance: 23
Progressive Web App: 27
Accessibility: 55
Best Practices: 44
SEO: 90

r.nikkei.com

14s faster
Nikkei - Optimizations

- Compression
- HTTP Caching
- Image Optimization
- Lazy-load non-critical resources
- Pre-resolve DNS for key origins
- Have a Web Font loading strategy
- Route-based JavaScript chunking
- JS bundling (webpack, RollUp)
- Serve ES2015+ to modern browsers
- Service Workers
- Inline Critical-path CSS in <head>

75% faster loading w/prefetch
94% cache-hit ratio
43% smaller JS bundles
<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>700 ms</td>
<td>636 ms</td>
</tr>
<tr>
<td>1.4 s</td>
<td>1.3 s</td>
</tr>
<tr>
<td>2.1 s</td>
<td>1.9 s</td>
</tr>
<tr>
<td>2.8 s</td>
<td>2.5 s</td>
</tr>
<tr>
<td>3.5 s</td>
<td>3.2 s</td>
</tr>
<tr>
<td>4.2 s</td>
<td>3.8 s</td>
</tr>
<tr>
<td>4.9 s</td>
<td>4.5 s</td>
</tr>
<tr>
<td>5.6 s</td>
<td>5.1 s</td>
</tr>
<tr>
<td>6.3 s</td>
<td>5.7 s</td>
</tr>
<tr>
<td>7 s</td>
<td>6.4 s</td>
</tr>
</tbody>
</table>

- **First meaningful paint**
  - Before: 2,150 ms
  - After: 1,110 ms
## PRPL Pattern

<table>
<thead>
<tr>
<th>Request</th>
<th>Push the minimal code for the initial route</th>
</tr>
</thead>
<tbody>
<tr>
<td>index.html</td>
<td>parsing</td>
</tr>
<tr>
<td>entry</td>
<td>parse/compile</td>
</tr>
<tr>
<td>vendor</td>
<td>parse/compile</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Push/Preload critical scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Render route &amp; get interactive</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Navigate to another route</th>
</tr>
</thead>
<tbody>
<tr>
<td>route-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Navigate next route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-cache using Service Workers</td>
</tr>
<tr>
<td>scripts</td>
</tr>
<tr>
<td>images</td>
</tr>
<tr>
<td>styles</td>
</tr>
</tbody>
</table>

| Cache remaining resources |
Machine Learning + Web Performance

One More Thing
Data-driven User Experiences
### Predict Page Navigations

<table>
<thead>
<tr>
<th>Page</th>
<th>Pageviews</th>
<th>Next Pageviews</th>
<th>Exits</th>
<th>% Exits</th>
<th>Top Next Page</th>
<th>Pageviews (top next page)</th>
<th>Likelihood of visiting top next page</th>
</tr>
</thead>
<tbody>
<tr>
<td>/web/updates/2018/01/devtools</td>
<td>923311</td>
<td>72389</td>
<td>762023</td>
<td>91.32%</td>
<td>/web/</td>
<td>16502</td>
<td>1.98%</td>
</tr>
<tr>
<td>/web/updates/2018/02/devtools</td>
<td>294281</td>
<td>28107</td>
<td>235925</td>
<td>89.35%</td>
<td>/web/</td>
<td>4485</td>
<td>1.70%</td>
</tr>
<tr>
<td>/web/tools/chrome-devtools/network-performance/reference</td>
<td>139891</td>
<td>9671</td>
<td>121152</td>
<td>92.61%</td>
<td>/web/tools/chrome-devtools/network-performance/</td>
<td>785</td>
<td>0.60%</td>
</tr>
<tr>
<td>/web/</td>
<td>131612</td>
<td>70407</td>
<td>51636</td>
<td>42.31%</td>
<td>/web/fundamentals/</td>
<td>16592</td>
<td>13.60%</td>
</tr>
<tr>
<td>/web/progressive-web-apps/</td>
<td>129943</td>
<td>48279</td>
<td>76655</td>
<td>61.36%</td>
<td>/web/progressive-web-apps/checklist</td>
<td>15560</td>
<td>12.45%</td>
</tr>
<tr>
<td>/web/tools/chrome-devtools/</td>
<td>128151</td>
<td>60105</td>
<td>63798</td>
<td>51.49%</td>
<td>/web/tools/lighthouse/</td>
<td>8572</td>
<td>6.92%</td>
</tr>
<tr>
<td>/web/fundamentals/web-app-manifest/</td>
<td>120616</td>
<td>11804</td>
<td>99842</td>
<td>89.43%</td>
<td>/web/fundamentals/app-install-banners/</td>
<td>1985</td>
<td>1.78%</td>
</tr>
<tr>
<td>/web/tools/chrome-devtools/remote-debugging/</td>
<td>99821</td>
<td>23436</td>
<td>64588</td>
<td>73.38%</td>
<td>/web/tools/chrome-devtools/</td>
<td>9606</td>
<td>10.91%</td>
</tr>
<tr>
<td>/web/tools/lighthouse/</td>
<td>92571</td>
<td>34053</td>
<td>52087</td>
<td>60.47%</td>
<td>/web/tools/chrome-devtools/</td>
<td>12062</td>
<td>14.00%</td>
</tr>
</tbody>
</table>

predictjs.firebaseapp.com
Data-driven Loading for Web Sites

Google Analytics

Analyze User Navigations

Model Next Page Predictions
(Markov Chains, Neural Networks, TensorFlow)

Prefetch next pages a user is likely to visit as they browse.

<link rel=prefetch>
Data-driven Bundling for Web Apps

- Google Analytics
  - Analyze User Navigations
  - Page
  - Previous Page Path
  - Page Views
  - Exits Metrics

- Model Next Page Predictions
  - (Markov Chains, Neural Networks, TensorFlow)

- Map “Pages” to JavaScript router
- Bundle JavaScript routes into chunks

- Prefetch next JavaScript chunks a user is likely to visit as they browse.
Guess.js: Predictive Fetching

- **GA module**
  - Google Analytics
  - Analyze User Navigations

- **Guess Parser**
  - Map “Pages” to JavaScript router

- **Guess Webpack Plugin**
  - Model Next Page Predictions (Markov Chains)
  - Bundle JavaScript routes into chunks
  - Prefetch next JavaScript chunks a user is likely to visit as they browse.

**+ Experimental support for predictive prefetching for static sites**

By Minko Gechev, Addy Osmani, Kyle Mathews & Katie Hempenius
Guess.js/Catsty Wikipedia Demo

For other uses, see Chocolate (disambiguation)

<table>
<thead>
<tr>
<th>Type</th>
<th>Course</th>
<th>Ingredients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dessert</td>
<td>Cheesecake</td>
<td>cherries, lemon, chocolate, and many more</td>
</tr>
</tbody>
</table>

Hit `⌘ R` to reload and capture filmstrip.
Improving performance is a journey. Lots of small changes can lead to big gains.
Thank you

Addy Osmani
Google
@addyosmani

Ewa Gasperowicz
Google
@devnook

Helpful resources
developers.google.com/web/fundamentals/performance
developers.google.com/web/tools/lighthouse

With special thanks to
Ward Peeters, Minko Gechev, Kyle Mathews, Katie Hempenius, Dom Farolino, Yoav Weiss, Susie Lu, Yusuke Utsunomiya, Lighthouse & Google Doodles.
We want to hear from you!

Provide feedback for this session by signing in on google.com/io/schedule