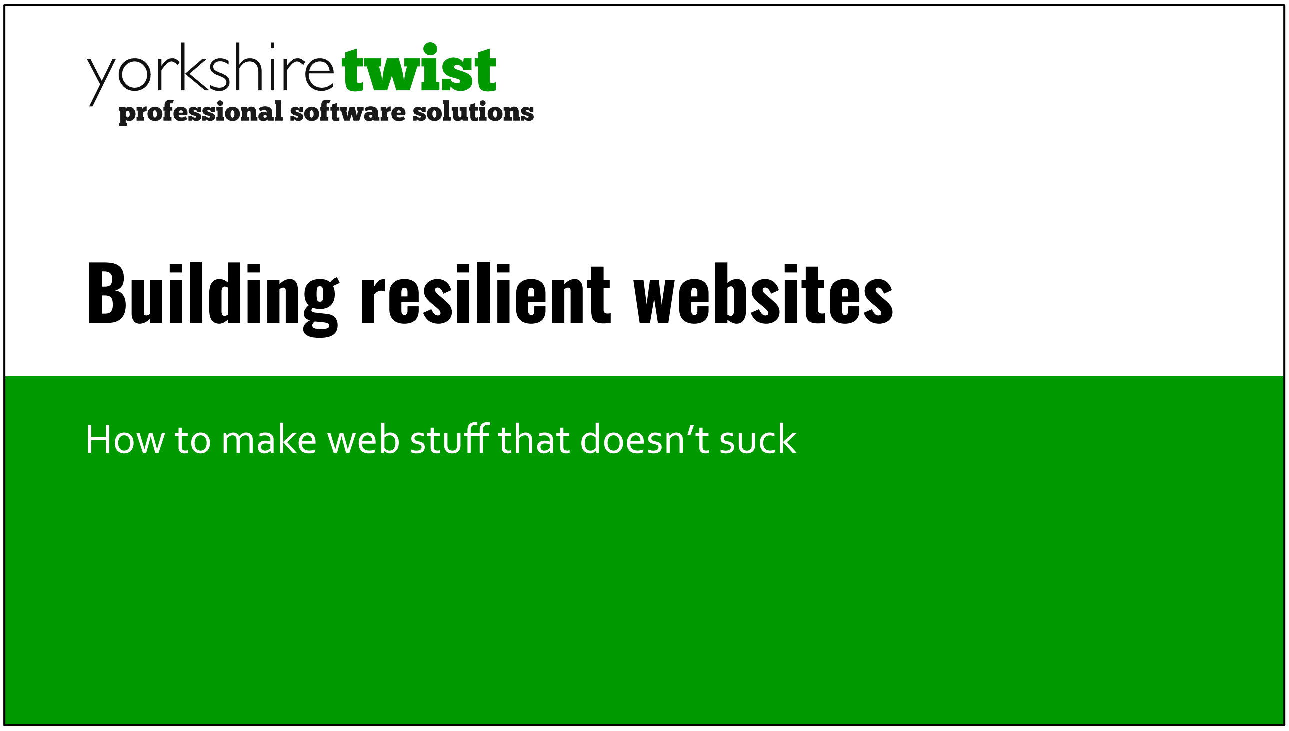


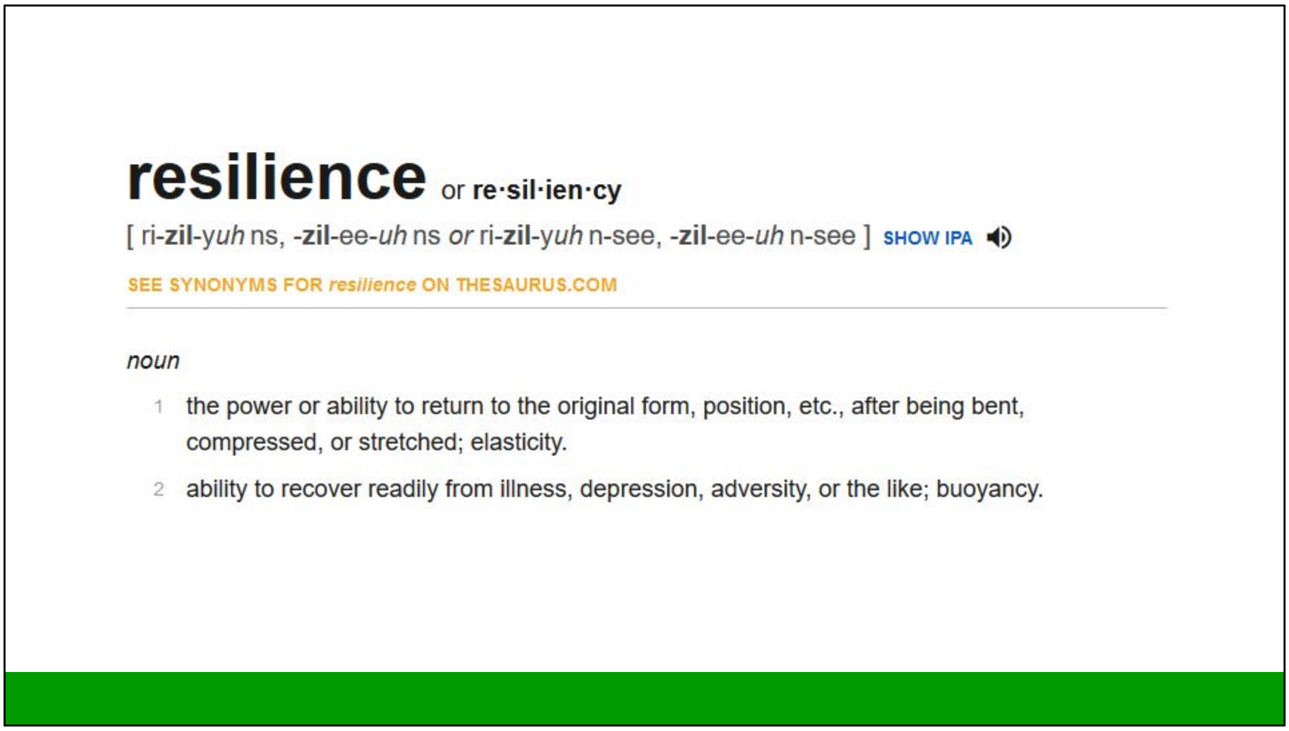
yorkshire**twist**
professional software solutions

# Building resilient websites

How to make web stuff that doesn‘t suck

I guess most people here are involved professionally in the development of software. And most of the time that means stuff which is on, or uses, the Internet - particularly the web. But we all know just as well as the average punter that the web isn’t the nirvana of user experience it’s cracked up to be. Why? How can we make it better?

That’s what we’re going to explore in this talk.



resilience or re·sil·ien·cy

[ ri-**zil**-y*uh* ns, -**zil**-ee-*uh* ns *or* ri-**zil**-y*uh* n-see, -**zil**-ee-*uh* n-see ] SHOW IPA

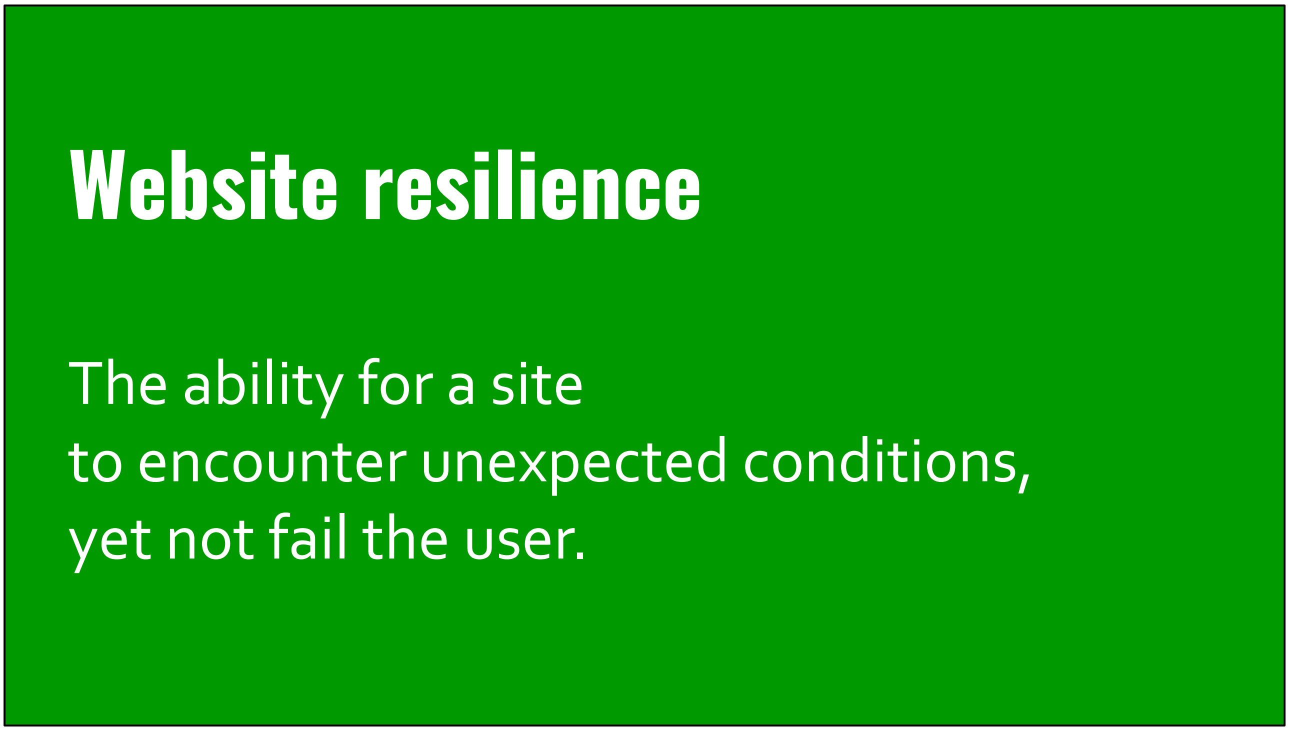
SEE SYNONYMS FOR *resilience* ON THESAURUS.COM

*noun*

1 the power or ability to return to the original form, position, etc., after being bent, compressed, or stretched; elasticity.

2 ability to recover readily from illness, depression, adversity, or the like; buoyancy.

But first let's define what we mean by 'resilience'. The dictionary definition seems to be nothing to do with the web. Let's try to paraphrase this so we have a handle on it in terms of websites.



## Website resilience

The ability for a site
to encounter unexpected conditions,
yet not fail the user.

Here’s a definition I’ll be using. Sounds great, right? We all know that unexpected conditions occur all the time on the web - we’ll be looking at some of those shortly - so if a site can recover from those, bounce back, and not fail the user, that’s got to be a good thing, right?

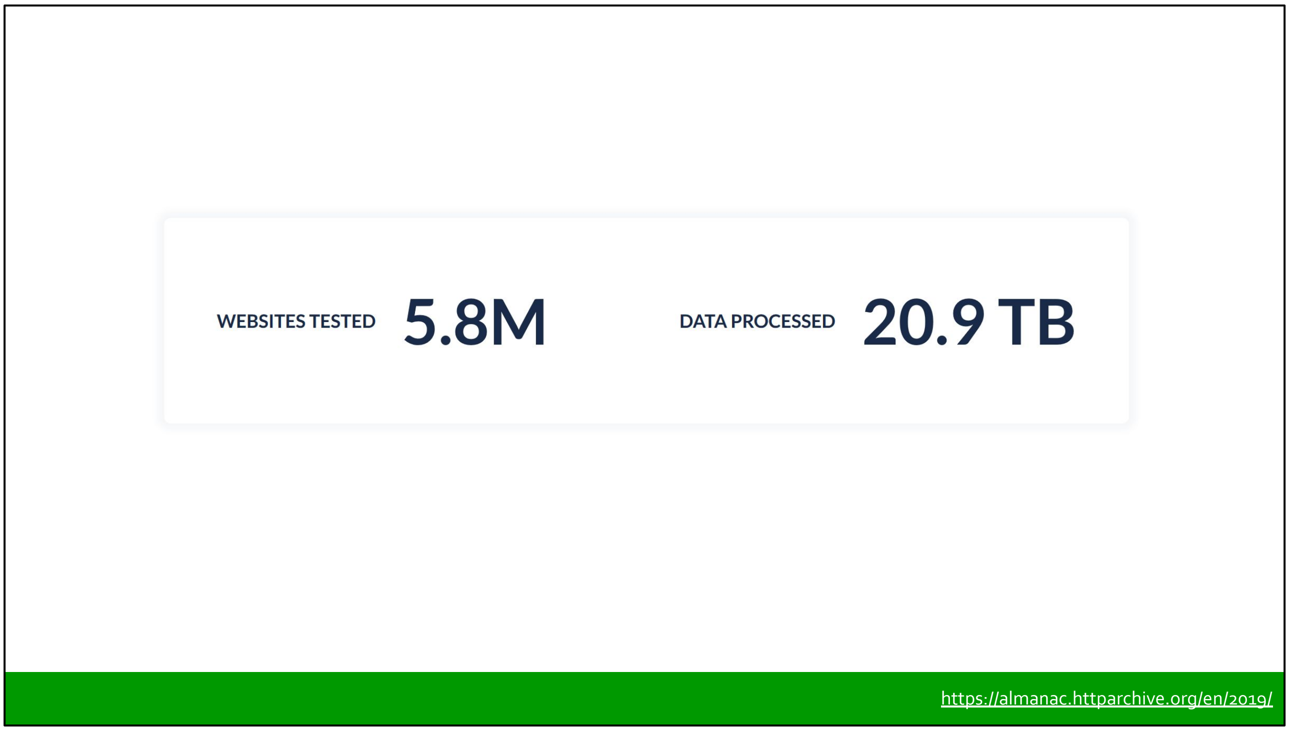
These unexpected conditions normally result in a sucky experience for our users and customers. If we can mitigate those unexpected conditions, reduce the suck, we’ll end up with happier users and we’ll improve the bottom line.

## Web development

With great power
comes great responsibility

These conditions are sometimes out of our control. But more often than not they are in our control, so we should do something about them. because make stuff that could affect millions of lives, so we should be responsible with that power.

Wherever we can control how our sites respond to unexpected conditions, we should do so.


WEBSITES TESTED 5.8M

DATA PROCESSED 20.9 TB

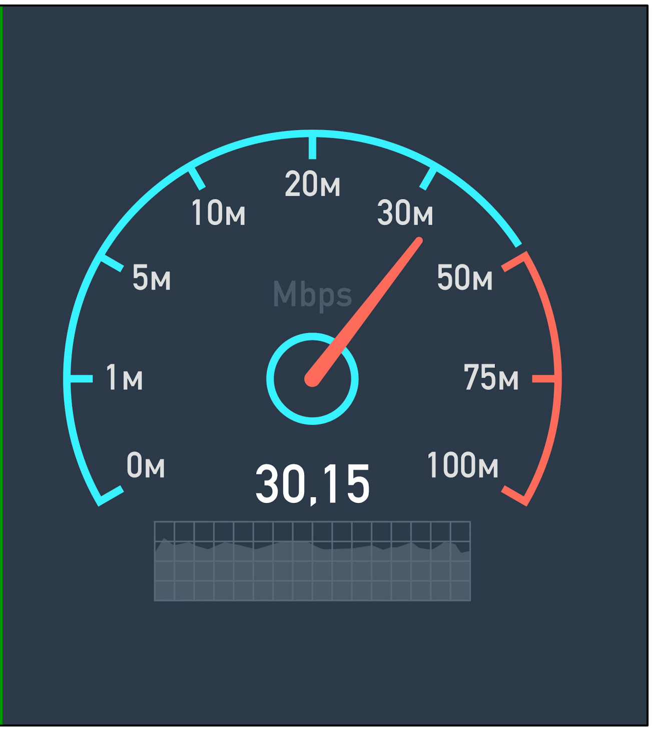
https://almanac.httparchive.org/en/2019/

We're going to be using data from the HTTP Archive, which has been gathering this information for over a decade. Their yearly Web Almanac offers fascinating insights in to a huge amount of data.

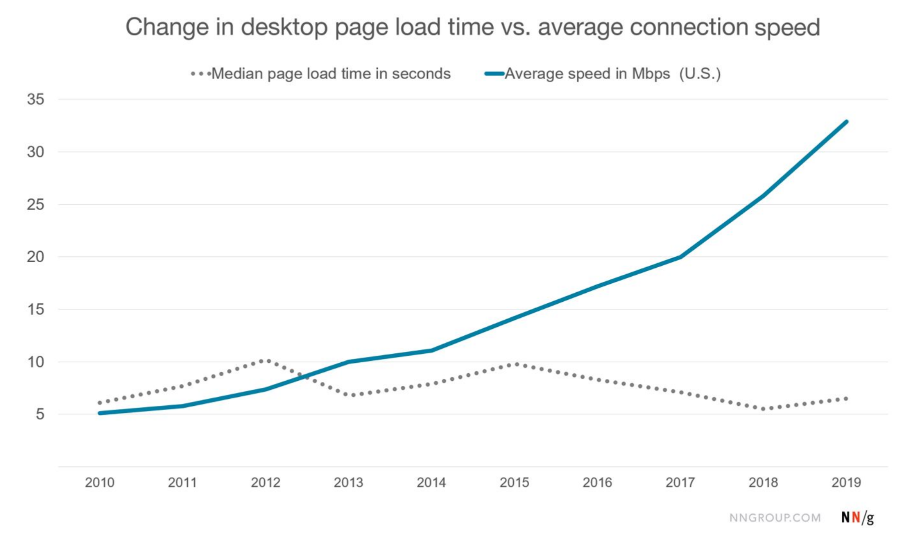
The first aspect we're going to look at is:

## Networks

The need for speed

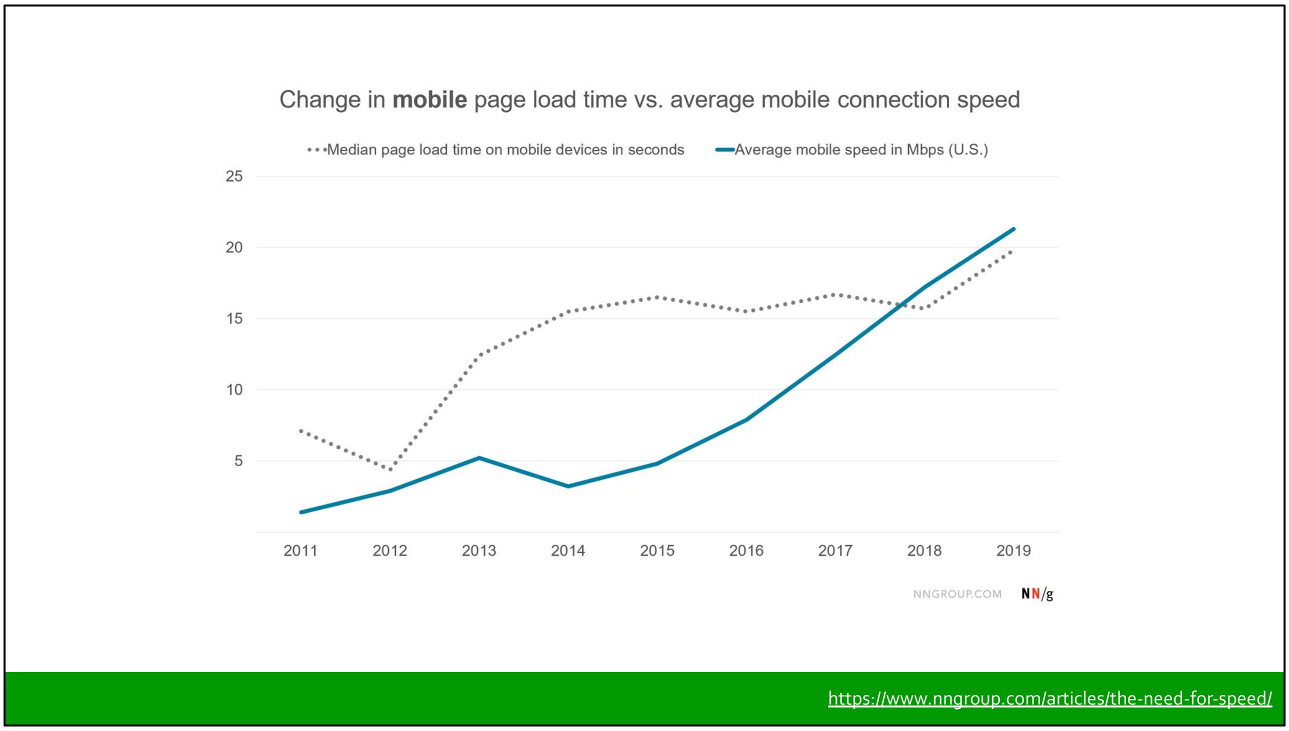


Networks. Or, as some people call them, “notworks”. We all know that network speeds are getting faster - many of us here will have moved on from 3G to 4G for our mobile devices, some perhaps even to 5G. And on desktop many of us are now on fibre broadband, perhaps even fibre-to-the-home. But does the web feel fast, most of the time? No.



https://www.nngroup.com/articles/the-need-for-speed/

The Nielsen Norman group, who have been doing research into user experience since 1998, have analysed the data from the HTTP Archive to pull out a few sobering facts. This first graph shows that in the US on desktop despite network speeds increasing greatly since 2010, the median page load time has stayed relatively static.



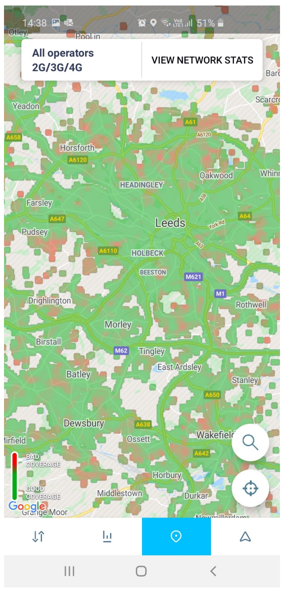
https://www.nngroup.com/articles/the-need-for-speed/

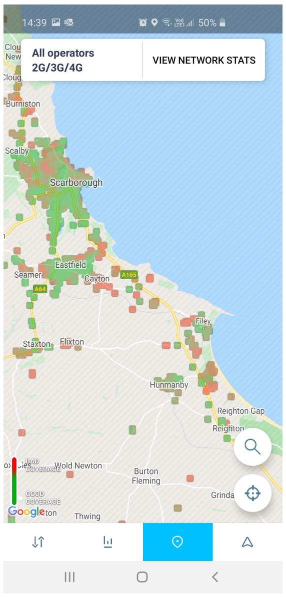
For mobile devices the situation is even worse. Median page load time has got slower over the last 10 years despite the network speed for many of us being faster than we had to our desktops just a few years ago. The load time here is measured in seconds. That’s right: the median load time for web pages is approaching 20 seconds.

This, I’m sure, will ring true for many people here. Often, the web just feels slow. What does that mean for users?

## This site sucks

They think the site sucks.

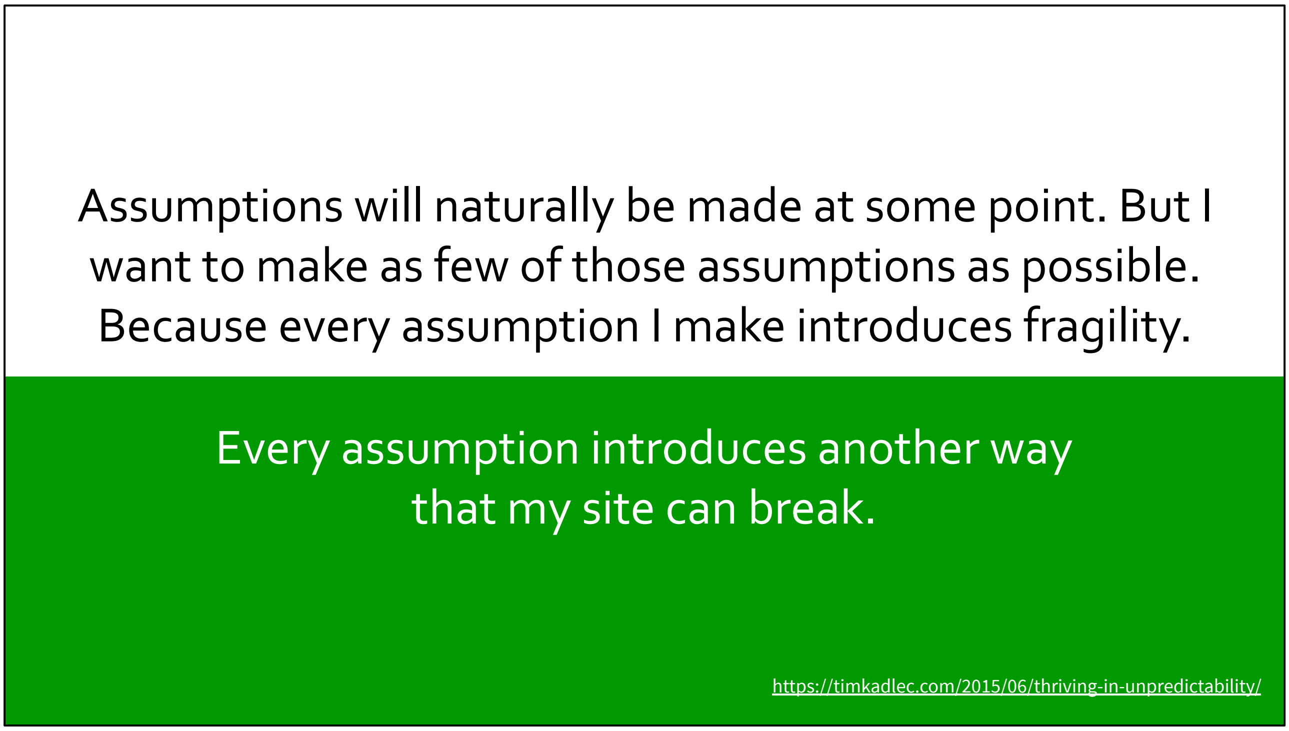




https://opensignal.com/

At this point we should recognise that there are a lot of moving parts which go into delivering a web page. For mobile devices, network coverage, latency and congestion play a major part. This data from OpenSignal shows that even close to a big city like Leeds there are plenty of weak or dead spots. And if we travel to the beautiful Yorkshire coast the situation is even worse.

The biggest problem with networks is we assume they are going to work. We assume they are going to be relatively predictable and consistent. But they aren't - mobile networks in particular.



Assumptions will naturally be made at some point. But I want to make as few of those assumptions as possible. Because every assumption I make introduces fragility.

Every assumption introduces another way that my site can break.

https://timkadlec.com/2015/06/thriving-in-unpredictability/

These assumptions are bad. Bad for us, bad for our sites, bad for our clients, users and customers. Every assumption - especially assumptions about the network - is a “fingers-crossed” wish for the best.

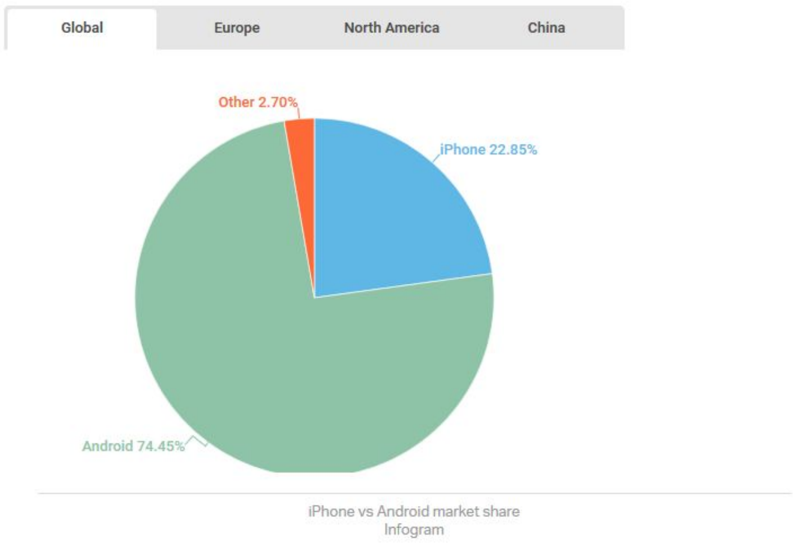
We can’t do much about making the network connection between a user and our sites better, but we can definitely change how we use that connection. So that brings me onto another subject over which we have no control.

## Devices

You never know what you’re gonna get

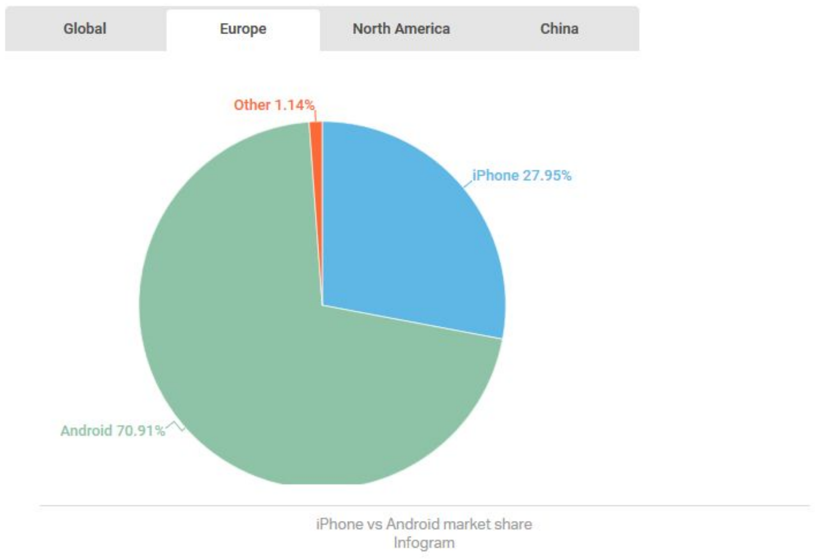
Devices. In most circumstances we have absolutely no control over the devices used to access our websites. We hope - that is, we often assume - that most people are on a reasonably new Apple or Android device. And if your audience is mainly young, mainly middle class, mainly urban people then you may well be partly right.

But there are a lot of people who use devices you and I wouldn’t want to be stuck with.



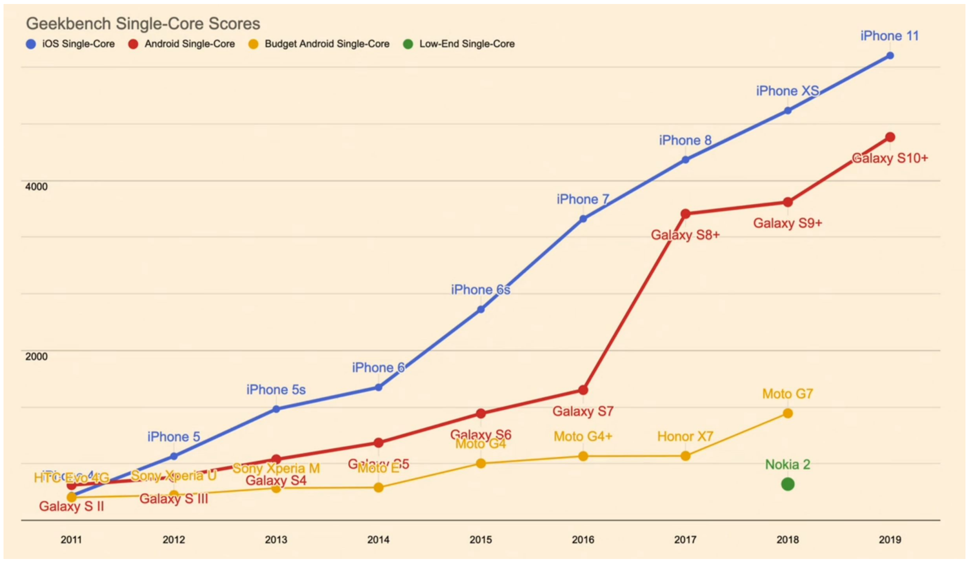
https://www.macworld.co.uk/feature/iphone/iphone-vs-android-market-share-3691861/

Worldwide sales data from 2019 shows that Android has a market share three times bigger than Apple. Is this what you expected? Well, that's worldwide so includes a bunch of countries where paying the Apple premium isn't an option. How about Europe?



https://www.macworld.co.uk/feature/iphone/iphone-vs-android-market-share-3691861/

In Europe Apple have only a 5% bigger market share. So if your users and customers are based in Europe, which for most of us is true, this is what you have to deal with.



https://vimeo.com/364402896

It's true that mobiles are getting faster - or at least the top-end ones are. Alex Russell from Google has done amazing research into the performance of mobile devices, I highly recommend you checkout his presentation linked from this slide.

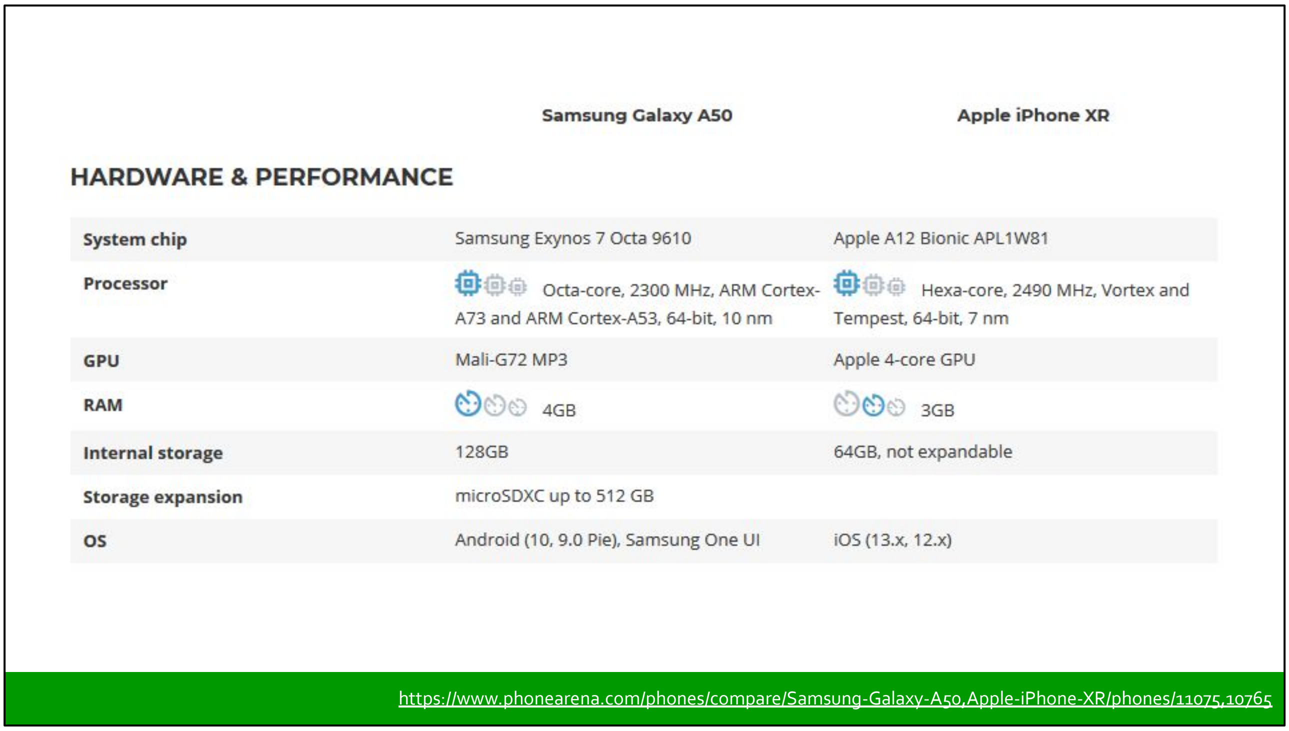
But even top-end devices use only a fraction of their power, and only for a fraction of the time? Why? Heat. If you ran a decent mobile device at full-CPU-power for any length of time it would become as hot as a light bulb in your hand. So device manufacturers do loads of trickery inside those devices to ensure the hot bits run, basically, as little as possible.

Bu most people don't have top-end devices. Remember, we're probably all affluent geeks - power users with money to spend on expensive phones. We search for things like "best smartphone 2020". Most people don't.

1. Samsung Galaxy A50 (£309)
2. Samsung Galaxy A40 (£219)
3. Samsung Galaxy A20e (£169)
4. Redmi Note 7 (£168)
5. Apple iPhone XR (£629)
6. Samsung Galaxy A10 (£139)
7. Samsung Galaxy S10 (£799)
8. Apple iPhone 8 (£479)
9. Samsung Galaxy A70 (£369)
10. Samsung Galaxy S10+ (£899)

https://www.standard.co.uk/tech/top-smartphones-2019-kantar-samsung-a-series-a4268926.html

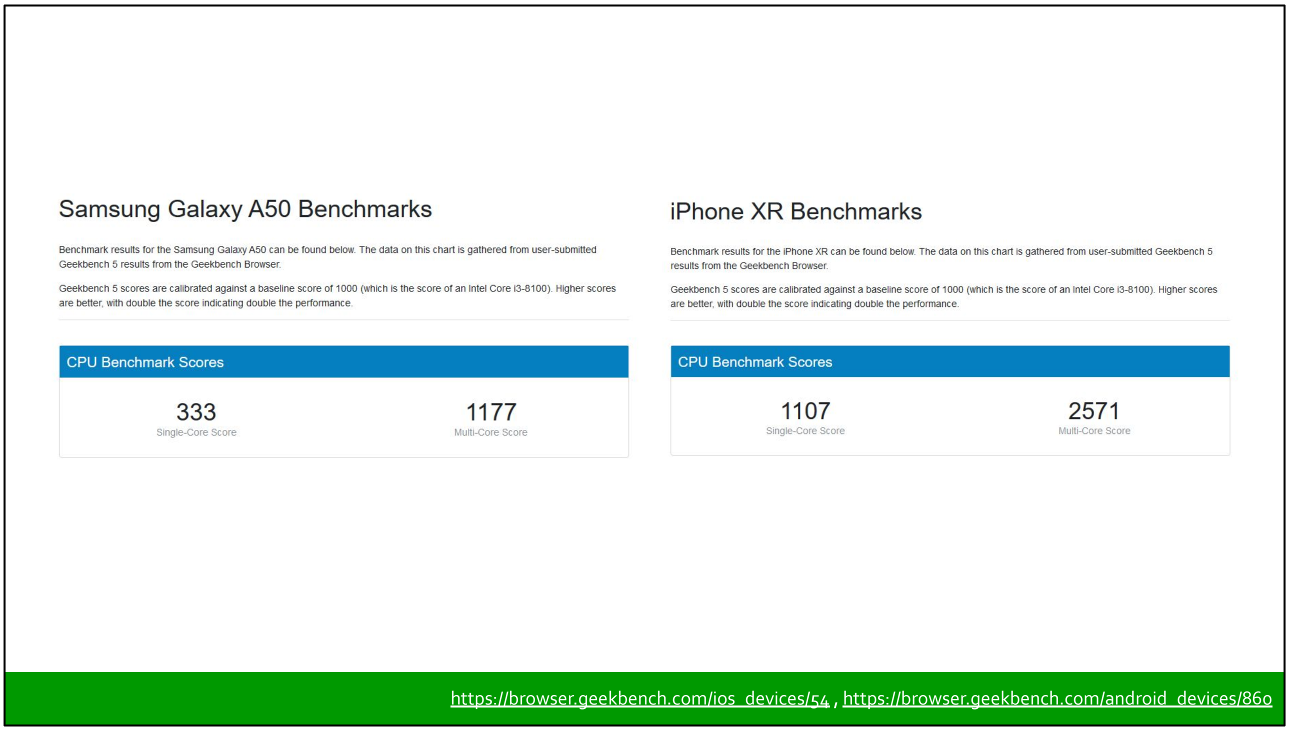
According to the Standard newspaper the best-selling smartphone in 2019 was the Samsung Galaxy A50. In fact there are only two Apple devices in the top 10. What do we see if we compare the specs of the A50 to the iPhone XR, which was the iPhone model launched around the same time?



| | Samsung Galaxy A50 | Apple iPhone XR |
|---|---|---|
| **HARDWARE & PERFORMANCE** | | |
| **System chip** | Samsung Exynos 7 Octa 9610 | Apple A12 Bionic APL1W81 |
| **Processor** | Octa-core, 2300 MHz, ARM Cortex-A73 and ARM Cortex-A53, 64-bit, 10 nm | Hexa-core, 2490 MHz, Vortex and Tempest, 64-bit, 7 nm |
| **GPU** | Mali-G72 MP3 | Apple 4-core GPU |
| **RAM** | 4GB | 3GB |
| **Internal storage** | 128GB | 64GB, not expandable |
| **Storage expansion** | microSDXC up to 512 GB | |
| **OS** | Android (10, 9.0 Pie), Samsung One UI | iOS (13.x, 12.x) |

https://www.phonearena.com/phones/compare/Samsung-Galaxy-A50,Apple-iPhone-XR/phones/11075,10765

Looks OK, the Samsung has an octa-core processor, and the Apple has a hexa-core. There's not much in terms of megahertz between them. But if we look at the actual geekbench scores:


Samsung Galaxy A50 Benchmarks

Benchmark results for the Samsung Galaxy A50 can be found below. The data on this chart is gathered from user-submitted Geekbench 5 results from the Geekbench Browser.

Geekbench 5 scores are calibrated against a baseline score of 1000 (which is the score of an Intel Core i3-8100). Higher scores are better, with double the score indicating double the performance.

CPU Benchmark Scores

| 333 | 1177 |
|---|---|
| Single-Core Score | Multi-Core Score |

iPhone XR Benchmarks

Benchmark results for the iPhone XR can be found below. The data on this chart is gathered from user-submitted Geekbench 5 results from the Geekbench Browser.

Geekbench 5 scores are calibrated against a baseline score of 1000 (which is the score of an Intel Core i3-8100). Higher scores are better, with double the score indicating double the performance.

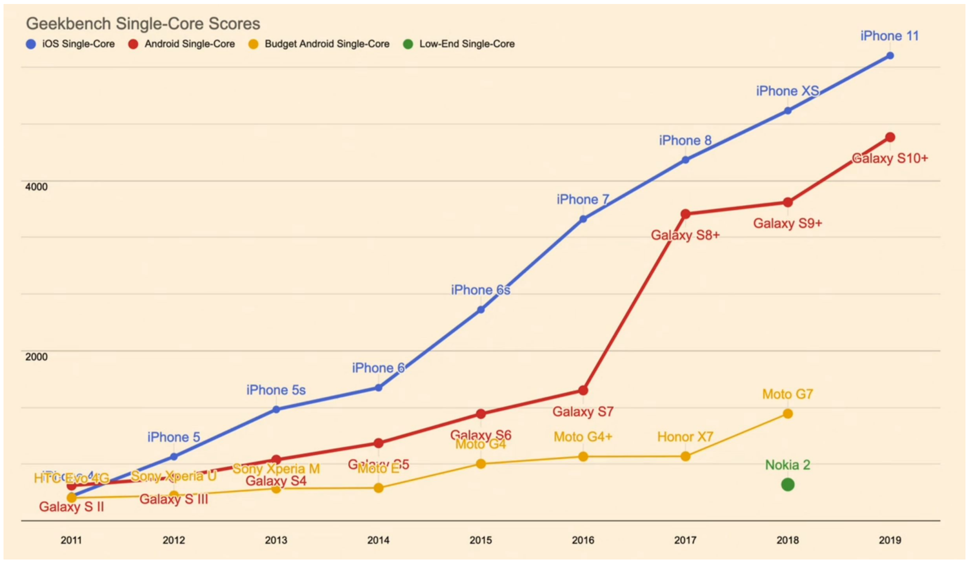
CPU Benchmark Scores

| 1107 | 2571 |
|---|---|
| Single-Core Score | Multi-Core Score |

https://browser.geekbench.com/ios_devices/54 , https://browser.geekbench.com/android_devices/860

Oh dear, the Apple wipes the floor with the Samsung.

This is just one example, but there are countless others. The spectrum of Android devices out there is huge, and even mid-range ones like this A50 are no match for the top-end devices many of us working in web development have in our pockets. In fact, according to Alex Russell (seriously - check out his talks, they're amazing) many popular phone models now have the same performance as top-of-the-line models from nearly a decade ago.

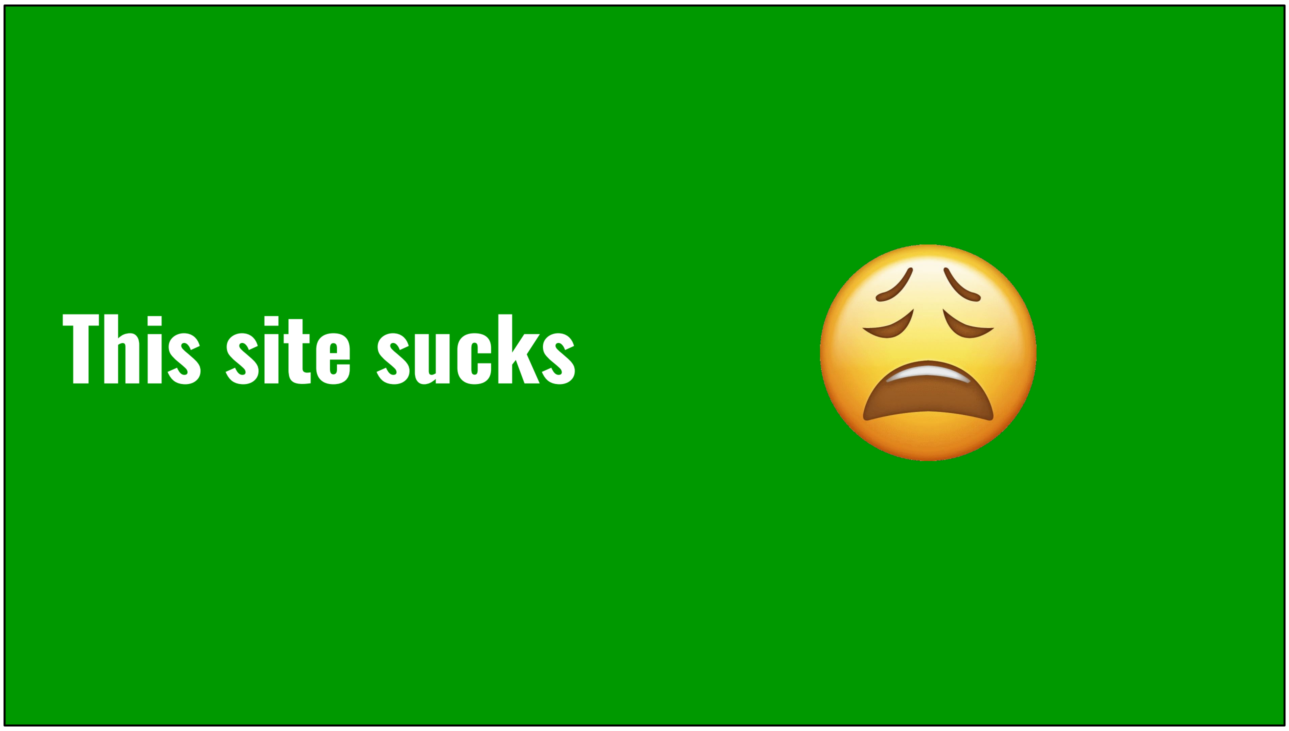


https://vimeo.com/364402896

Just look at that Nokia 2 at the bottom right. It was released in June 2019 and costs just £94 on Amazon, and has the same performance as an iPhone 4 or Galaxy S3.

As manufacturers look for ways to sell more devices, especially in emerging markets, these low-end devices are going to become more and more common.

If a user with one of these devices visits your website, which you’ve tested and works great on your iPhone 11 or Galaxy S10, and they have a bad, janky experience, what are they going to think?

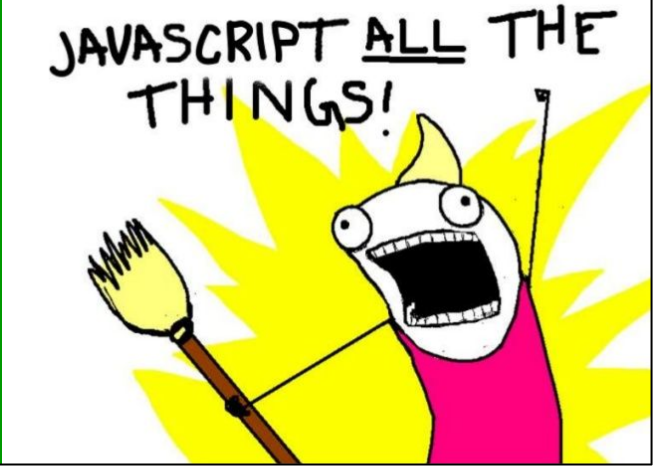


## This site sucks 😩

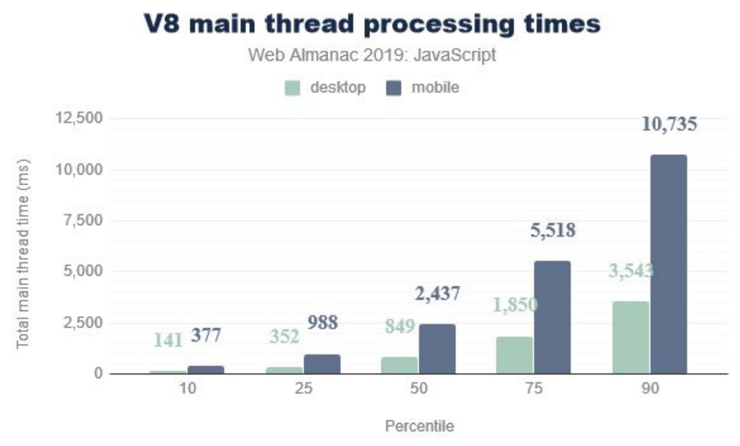
They think the site sucks.

What’s the part of your site that is going to choke these slower devices? Yes, it’s the language everyone loves to hate - or hates to love:

## Scripts



We all know JavaScript is a big part of the web these days. But JavaScript, unlike other types of assets such as images, requires parsing and executing before it does anything. So what effect does that processing have on page performance? Back to the HTTP Archive data we go.



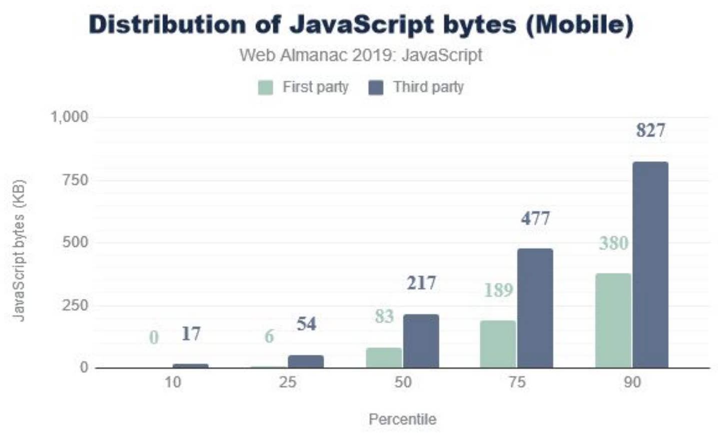
Show description of Figure 3

*Figure 3. V8 Main thread processing times by device.*

https://almanac.httparchive.org/en/2019/javascript

The HTTP Archive found that the processing time for scripts varies wildly, with the median around 849ms on desktop and 2.4 seconds on mobile.

Think about that. 2.4 seconds of … nothing. In terms of web page performance that's an eternity. The 90th percentile is over 10 seconds! That's a whole lot of waiting around for a lot of users. Where does all this JavaScript come from?

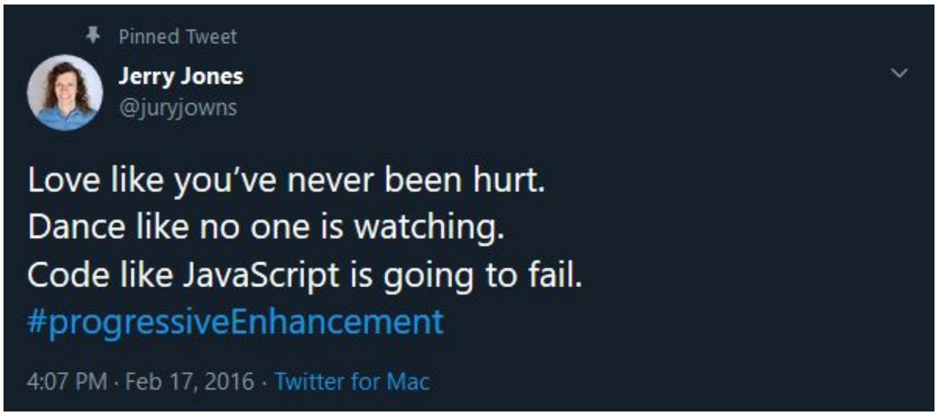


Show description of Figure 9

*Figure 9.* Distribution of total JavaScript downloaded on mobile.

https://almanac.httparchive.org/en/2019/javascript

It comes mainly from 3rd parties. And every connection to a 3rd party means not only do your users have to parse and execute their scripts, but you're putting the performance of your pages at the mercy of other sites over whom you have little or no control. And that's assuming all these scripts play nicely together. What if things go wrong? What if there are clashes? What if dependencies fail to load, parse or execute correctly?


Pinned Tweet

Jerry Jones
@juryjowns

Love like you've never been hurt.
Dance like no one is watching.
Code like JavaScript is going to fail.
#progressiveEnhancement

4:07 PM · Feb 17, 2016 · Twitter for Mac

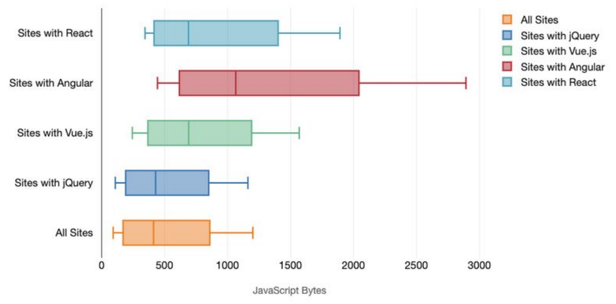
https://mobile.twitter.com/juryjowns/status/699988463773298688

Jerry Jones, a developer at Automattic, puts it like this. Automattic, by the way, develops WordPress - which powers by some counts over 30% of all websites, so they know a thing or two about scale.

This dependency on JavaScript is fashionable at the moment.

**JavaScript Bytes Served to Mobile Devices, by Percentile**

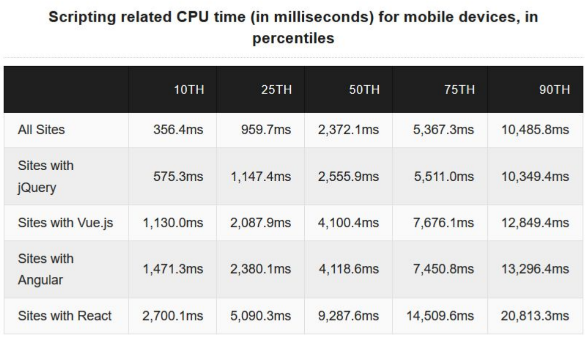
| | 10TH | 25TH | 50TH | 75TH | 90TH |
|---|---|---|---|---|---|
| All Sites | 93.4kb | 196.6kb | 413.5kb | 746.8kb | 1,201.6kb |
| Sites with jQuery | 110.3kb | 219.8kb | 430.4kb | 748.6kb | 1,162.3kb |
| Sites with Vue.js | 244.7kb | 409.3kb | 692.1kb | 1,065.5kb | 1,570.7kb |
| Sites with Angular | 445.1kb | 675.6kb | 1,066.4kb | 1,761.5kb | 2,893.2kb |
| Sites with React | 345.8kb | 441.6kb | 690.3kb | 1,238.5kb | 1,893.6kb |



https://timkadlec.com/remembers/2020-04-21-the-cost-of-javascript-frameworks/

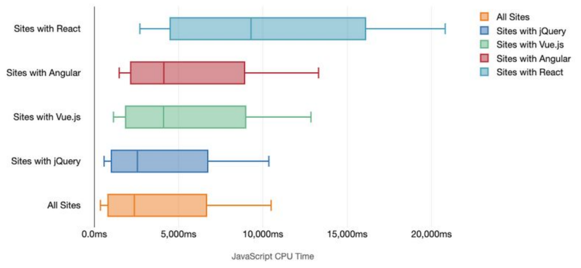
Tim Kadlec did some analysis on HTTP Archive data and found some startling facts. He compared sites using jQuery, Angular, React and Vue. Here you can see that the number of bytes of JavaScript sent to mobile devices varies a lot.

But it’s the processing time where it gets really scary.



**Scripting related CPU time (in milliseconds) for mobile devices, in percentiles**

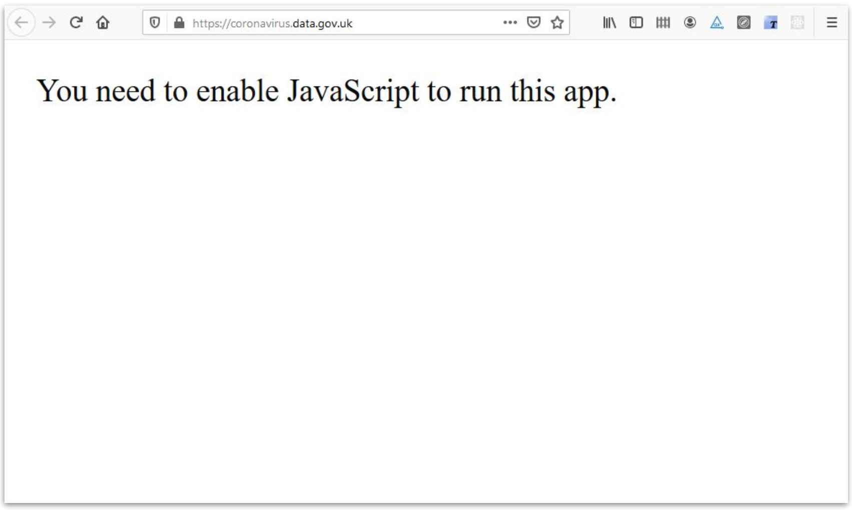
| | 10TH | 25TH | 50TH | 75TH | 90TH |
|---|---|---|---|---|---|
| All Sites | 356.4ms | 959.7ms | 2,372.1ms | 5,367.3ms | 10,485.8ms |
| Sites with jQuery | 575.3ms | 1,147.4ms | 2,555.9ms | 5,511.0ms | 10,349.4ms |
| Sites with Vue.js | 1,130.0ms | 2,087.9ms | 4,100.4ms | 7,676.1ms | 12,849.4ms |
| Sites with Angular | 1,471.3ms | 2,380.1ms | 4,118.6ms | 7,450.8ms | 13,296.4ms |
| Sites with React | 2,700.1ms | 5,090.3ms | 9,287.6ms | 14,509.6ms | 20,813.3ms |



https://timkadlec.com/remembers/2020-04-21-the-cost-of-javascript-frameworks/

All those milliseconds spent on parsing and executing JavaScript. This is yet more evidence to show that you shouldn't believe that everyone has the same experience as you on your fast device when visiting sites using frameworks like these. Many - most people - have a slow and janky experience.

But the use of JavaScript that really confuses me is when it's entirely unnecessary.

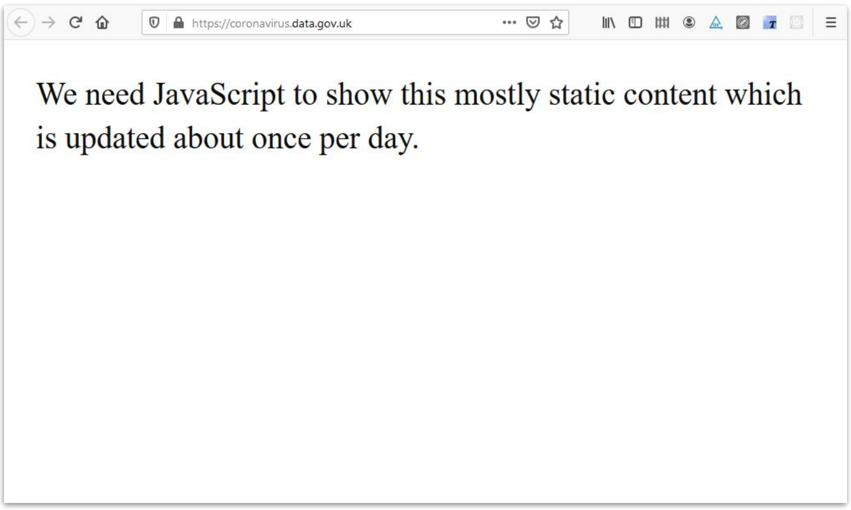


https://coronavirus.data.gov.uk

You need to enable JavaScript to run this app.

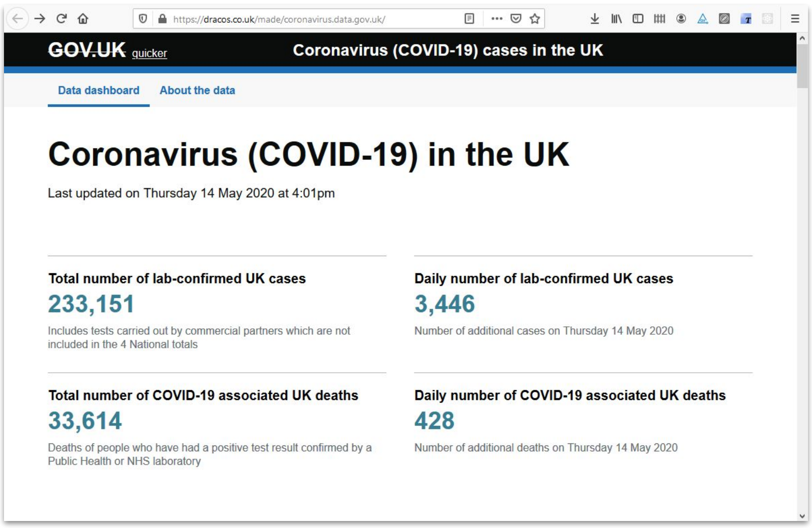
https://coronavirus.data.gov.uk/

Here's a very current example. The corona virus dashboard from gov.uk. Anyone seen this? It relies on JavaScript. But here's the thing; the page is updated about once per day. So instead of this error message, perhaps they should show this:


https://coronavirus.data.gov.uk

We need JavaScript to show this mostly static content which is updated about once per day.

I’ve seen too many examples of static content relying on hugely complex and powerful front-end frameworks, where a set of HTML pages would have done the job.



Coronavirus (COVID-19) cases in the UK

Coronavirus (COVID-19) in the UK

Last updated on Thursday 14 May 2020 at 4:01pm

**Total number of lab-confirmed UK cases**
233,151
Includes tests carried out by commercial partners which are not included in the 4 National totals

**Daily number of lab-confirmed UK cases**
3,446
Number of additional cases on Thursday 14 May 2020

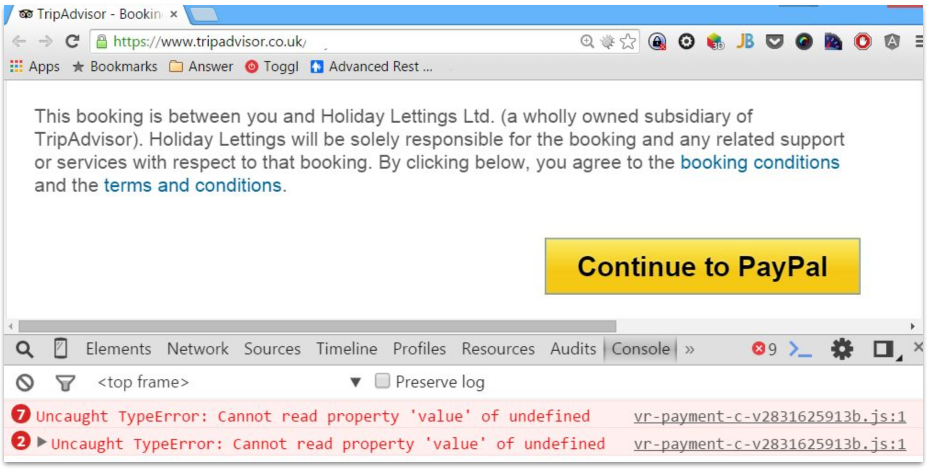
**Total number of COVID-19 associated UK deaths**
33,614
Deaths of people who have had a positive test result confirmed by a Public Health or NHS laboratory

**Daily number of COVID-19 associated UK deaths**
428
Number of additional deaths on Thursday 14 May 2020

https://dracos.co.uk/wrote/coronavirus-dashboard/

Which is why Matthew Somerville, a developer from Birmingham, did a version of the coronavirus dashboard which is just HTML, enhanced with a bit of JavaScript. On a desktop his version is 87% smaller than the official gov.uk site - but provides the same data and functionality.

Then there are actual errors, genuine production runtime bugs.



This booking is between you and Holiday Lettings Ltd. (a wholly owned subsidiary of TripAdvisor). Holiday Lettings will be solely responsible for the booking and any related support or services with respect to that booking. By clicking below, you agree to the booking conditions and the terms and conditions.

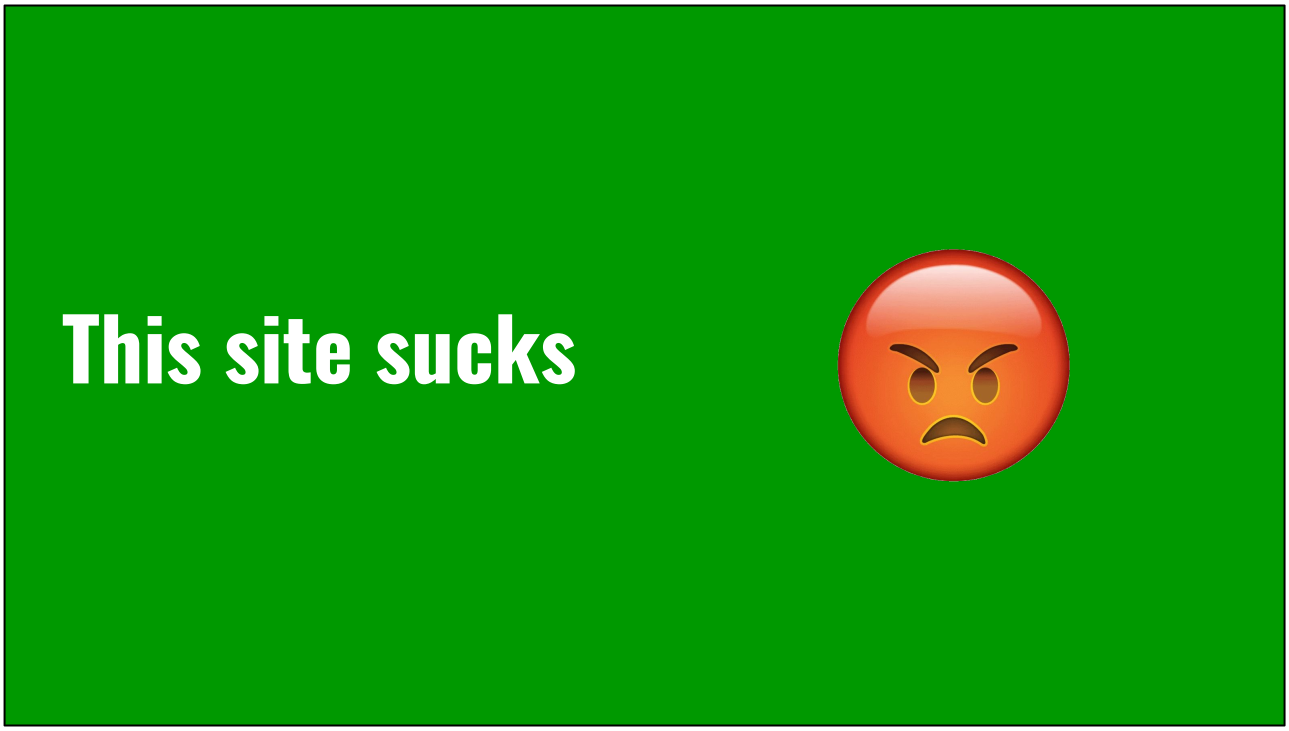
Continue to PayPal

```
7 Uncaught TypeError: Cannot read property 'value' of undefined    vr-payment-c-v2831625913b.js:1
2 ▶Uncaught TypeError: Cannot read property 'value' of undefined   vr-payment-c-v2831625913b.js:1
```

https://www.stillbreathing.co.uk/2015/06/29/progressive-enhancement-matters

OK, who here has never seen a JavaScript error in the wild? This was one I saw a few years ago which I've used as an example of the worst time for a JavaScript error to happen - as a customer (in this case me) was about to purchase something.

A normal person - by which I mean someone who doesn't browse the web with devtools open - would click repeatedly, then eventually give up in despair. What's the outcome? That's right.



## This site sucks

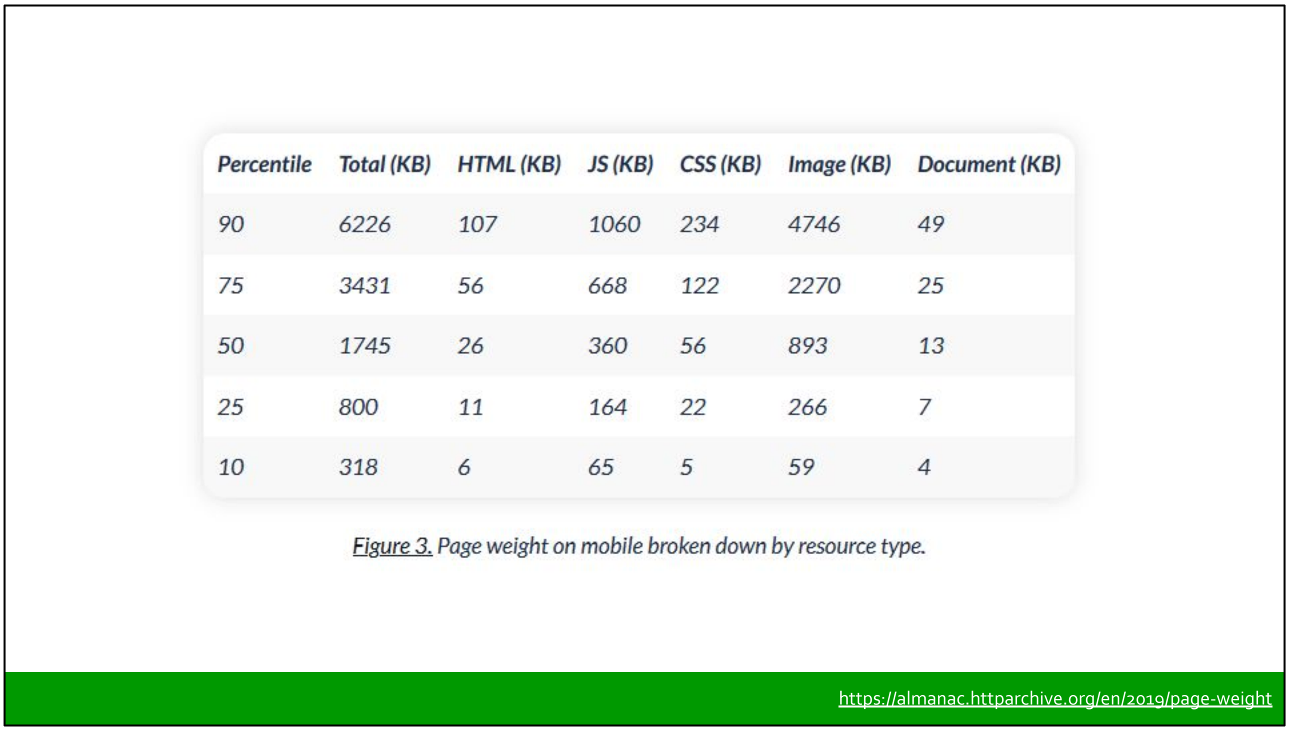
They think the site sucks.

Clearly we can do a lot to fix many of these performance problems. One of the ways is just to send less stuff down the wire. Yes, I’m talking about assets.

## Assets

They should be called "liabilities"

Or, as some have said, they should be called "liabilities" - and you'll see why. One of the reasons web pages are slow is because of the assets that have to be downloaded. Let's dive back into the HTTP Archive Web Almanac.



| Percentile | Total (KB) | HTML (KB) | JS (KB) | CSS (KB) | Image (KB) | Document (KB) |
|---|---|---|---|---|---|---|
| 90 | 6226 | 107 | 1060 | 234 | 4746 | 49 |
| 75 | 3431 | 56 | 668 | 122 | 2270 | 25 |
| 50 | 1745 | 26 | 360 | 56 | 893 | 13 |
| 25 | 800 | 11 | 164 | 22 | 266 | 7 |
| 10 | 318 | 6 | 65 | 5 | 59 | 4 |

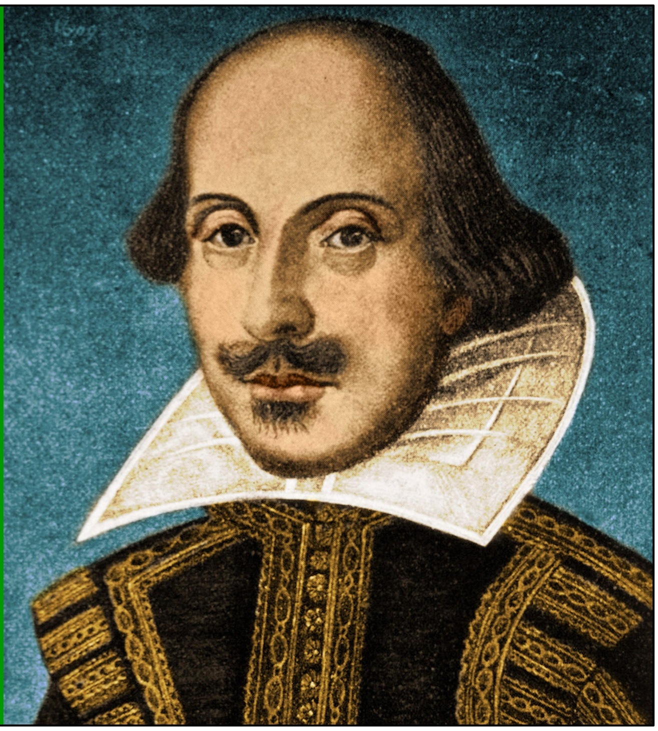
Figure 3. Page weight on mobile broken down by resource type.

https://almanac.httparchive.org/en/2019/page-weight

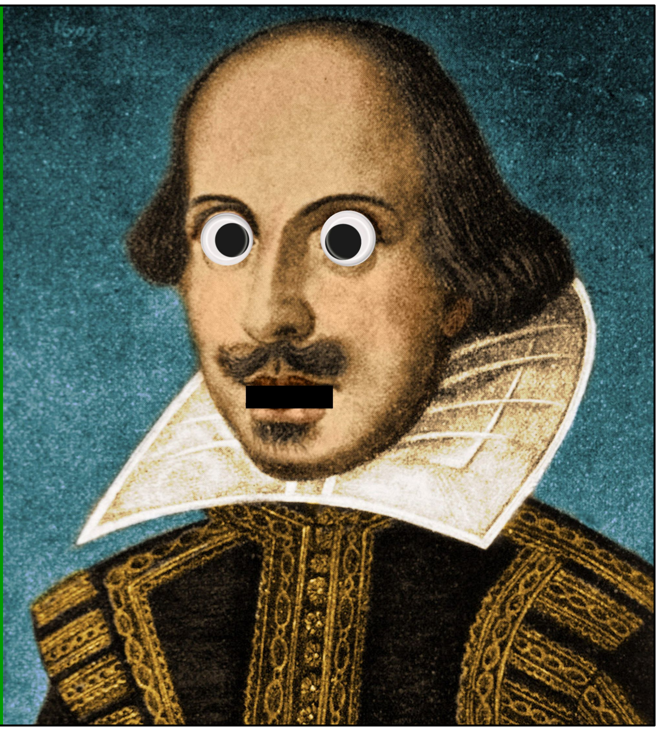
From the data gathered in 2019, the HTTP Archive has found that most web pages have a total weight - taking into account scripts, images, CSS and everything else - of multiple megabytes. On our beefy desktops we lose track of just how huge a megabyte is in terms of raw data.

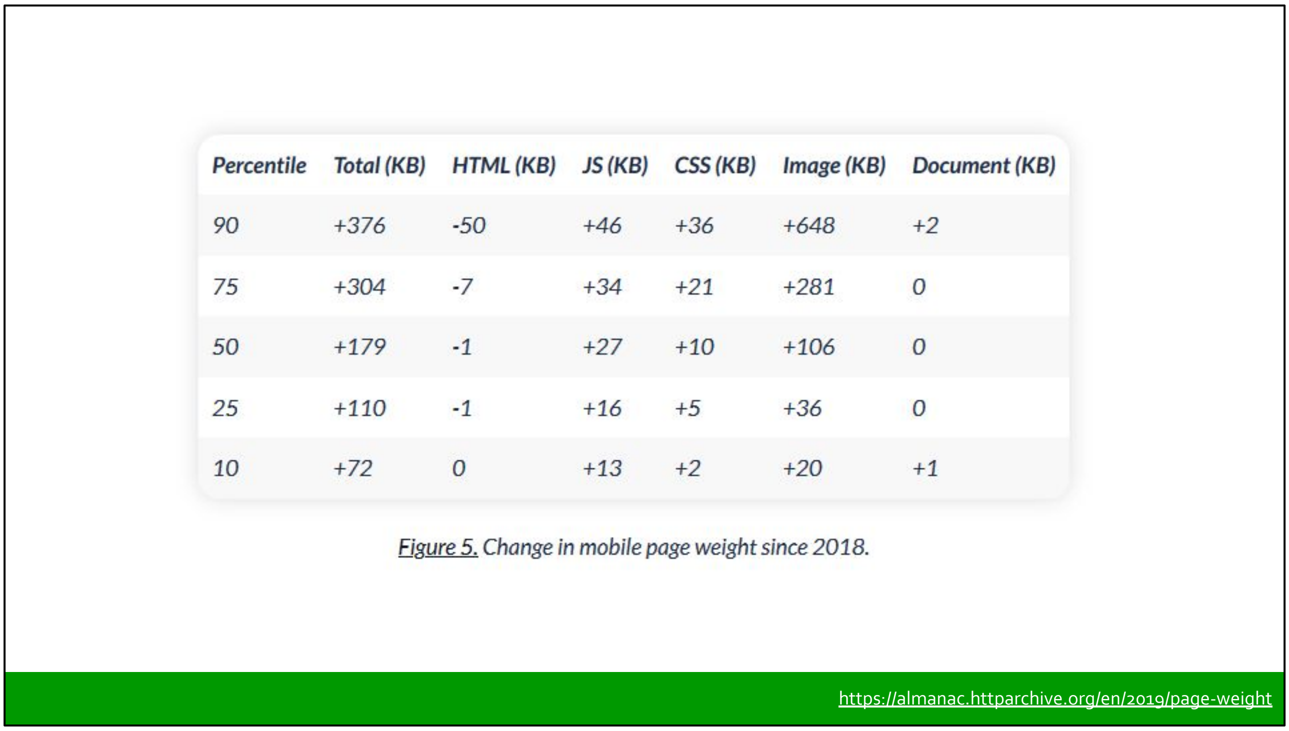
Did you know that the complete works of Shakespeare consists of about 3.5 million characters, which is about 3.4MB. Let me make that clear.

25% of web pages are larger than the complete works of Shakespeare



## 25% of web pages are larger than the complete works of Shakespeare





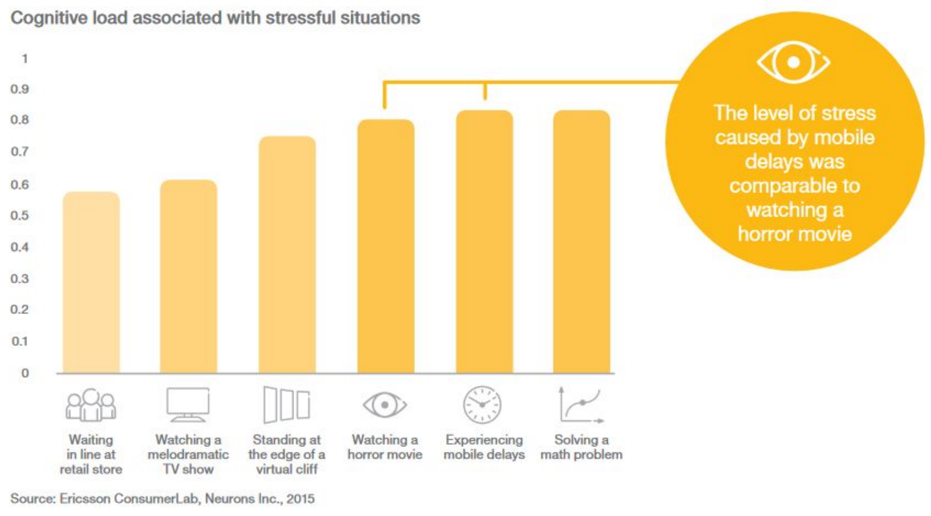
| Percentile | Total (KB) | HTML (KB) | JS (KB) | CSS (KB) | Image (KB) | Document (KB) |
|---|---|---|---|---|---|---|
| 90 | +376 | -50 | +46 | +36 | +648 | +2 |
| 75 | +304 | -7 | +34 | +21 | +281 | 0 |
| 50 | +179 | -1 | +27 | +10 | +106 | 0 |
| 25 | +110 | -1 | +16 | +5 | +36 | 0 |
| 10 | +72 | 0 | +13 | +2 | +20 | +1 |

Figure 5. Change in mobile page weight since 2018.

https://almanac.httparchive.org/en/2019/page-weight

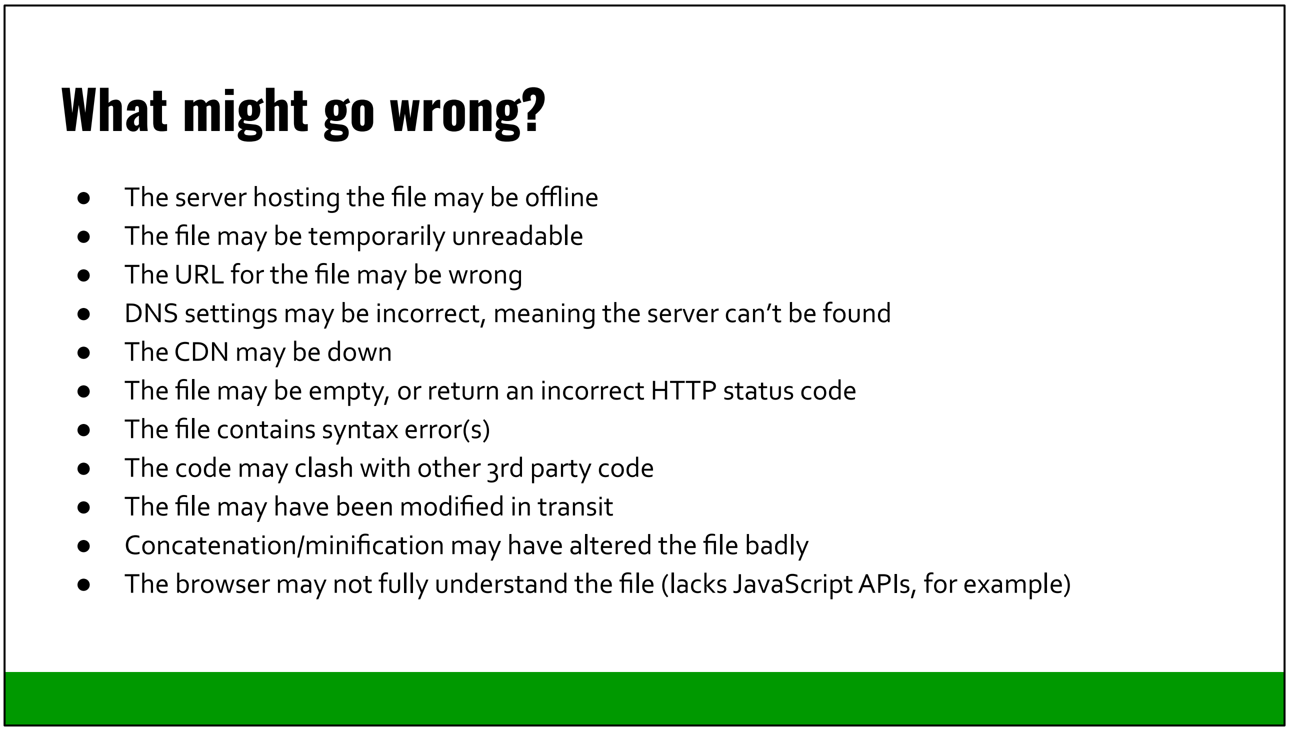
And this huge size is increasing year on year. The change for the heaviest sites has increased by more than 350KB in the year from 2018 to 2019.

One thing I hate, and I guess you do too, is when you think a page has finished loading so you try to click or press on something but the page “jumps” as the layout shifts because some other asset affecting layout has been downloaded. That makes my blood boil.



https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-feb-2016-interim.pdf

In fact Ericsson found that the stress caused by waiting for a slow web page on a mobile device is comparable to watching a horror movie.



## What might go wrong?

- The server hosting the file may be offline
- The file may be temporarily unreadable
- The URL for the file may be wrong
- DNS settings may be incorrect, meaning the server can't be found
- The CDN may be down
- The file may be empty, or return an incorrect HTTP status code
- The file contains syntax error(s)
- The code may clash with other 3rd party code
- The file may have been modified in transit
- Concatenation/minification may have altered the file badly
- The browser may not fully understand the file (lacks JavaScript APIs, for example)

When it comes to assets, what are the kind of unexpected conditions we might encounter? Here's a non-exhaustive list. In some of these cases we can do our best to ensure they can't happen, but on some cases we have no control.

What's the outcome for users when they wait ages for stuff to download, or any of these problems occur?

## This site sucks

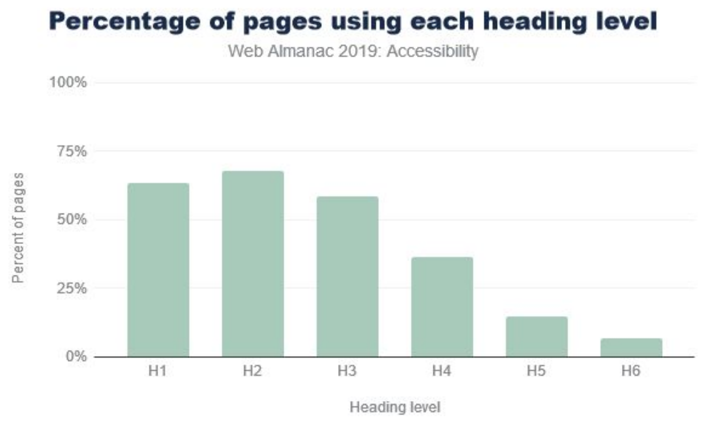
They think the site sucks.

All these different aspects of performance are clearly a big problem. But there's another unexpected condition which degrades the experience users have of our sites. That unexpected condition is the needs of the user for the site to be accessible.

## Accessibility

You must be this tall
to use this website

I’m not talking about really advanced stuff, even the basics are being missed too often.



Show description of Figure 3

*Figure 3. Popularity of heading levels.*

https://almanac.httparchive.org/en/2019/accessibility

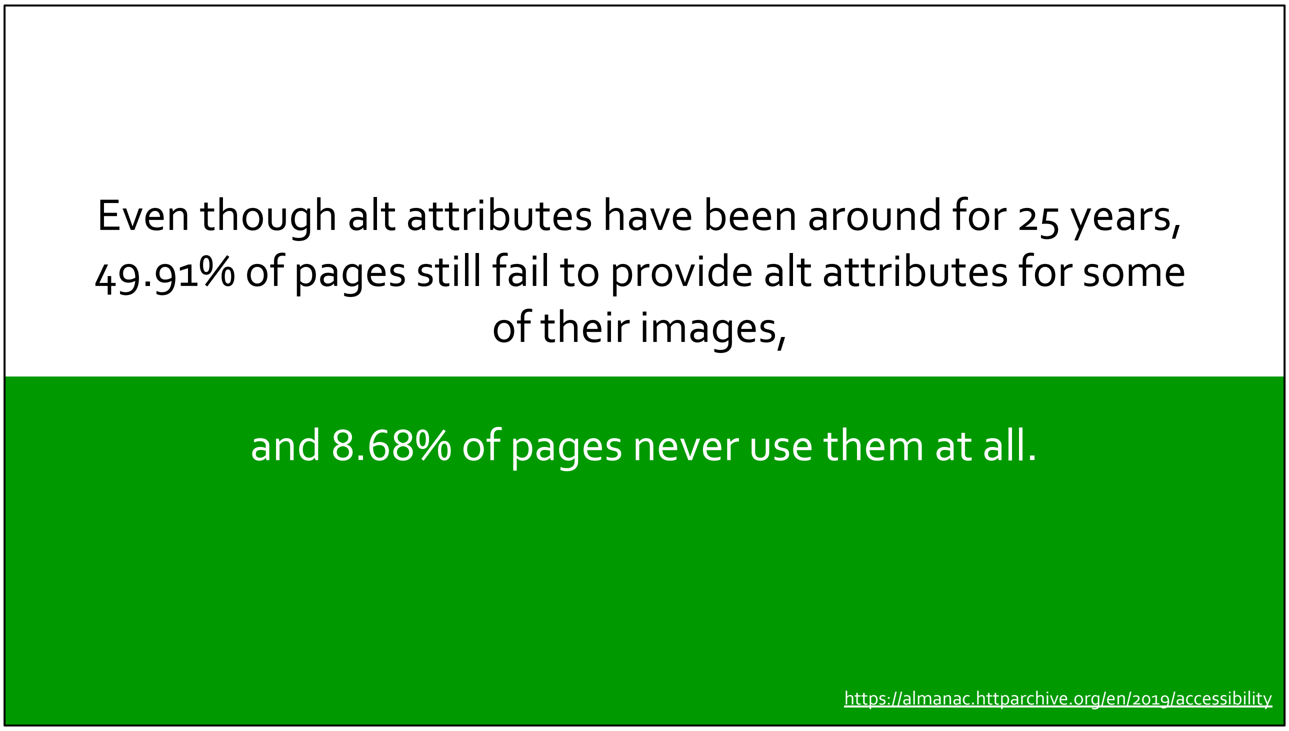
The Web Almanac highlights the scale of the problem, showing even something as simple as semantic HTML - in this case the 6 heading levels - aren't used on many sites. These headings, along with other "landmark" HTML elements, help people who use assistive technologies such as screenreaders to navigate the page. Oh, they also help search engines understand the structure of the page, so put a tick in the box for search engine optimisation as well.

There are 140 HTML elements

The average 2 year-old knows 200-300 words

After all, the full range of HTML elements isn’t too hard to learn. A 2 year old could do it.

Thanks to the great Bruce Lawson for correlating these two facts. OK, let’s look at something simpler.



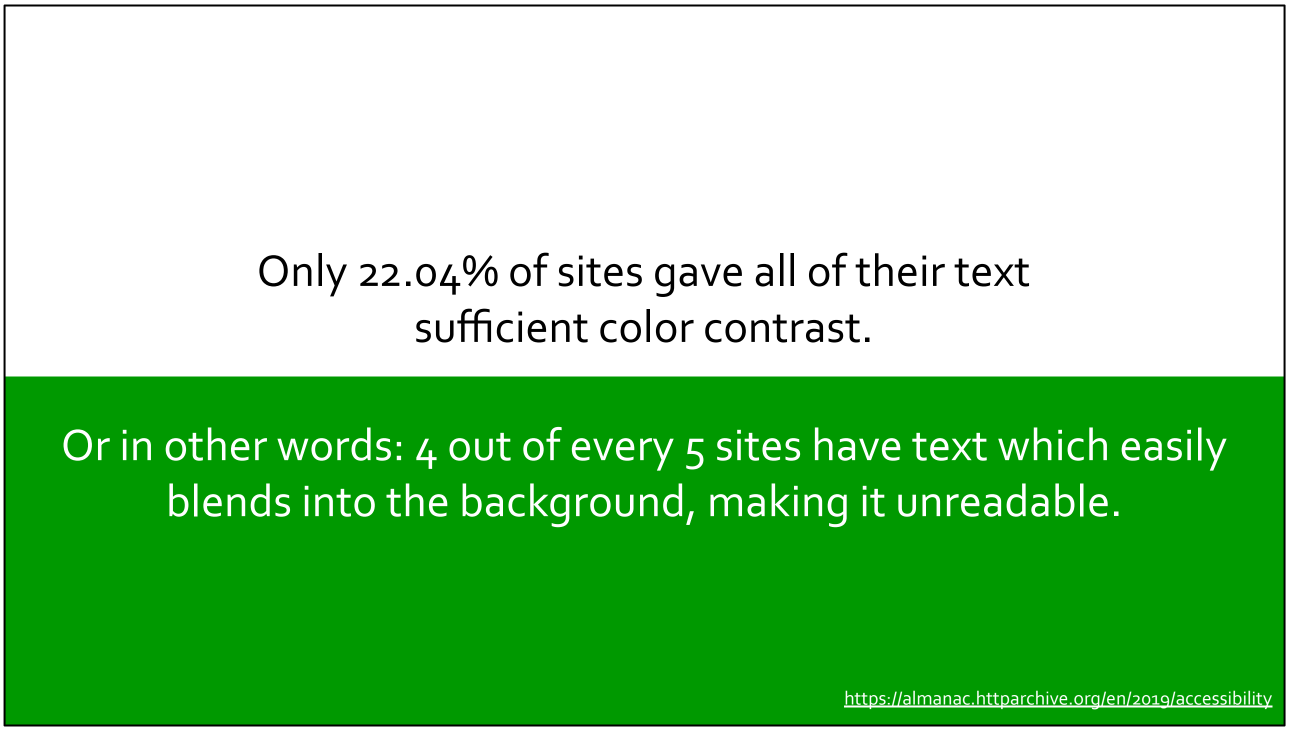
Even though alt attributes have been around for 25 years, 49.91% of pages still fail to provide alt attributes for some of their images,

and 8.68% of pages never use them at all.

https://almanac.httparchive.org/en/2019/accessibility

Everyone knows that alt attributes need to be added to images, yes? Yet many pages don’t.

So how about visual accessibility:


Only 22.04% of sites gave all of their text sufficient color contrast.

Or in other words: 4 out of every 5 sites have text which easily blends into the background, making it unreadable.

https://almanac.httparchive.org/en/2019/accessibility

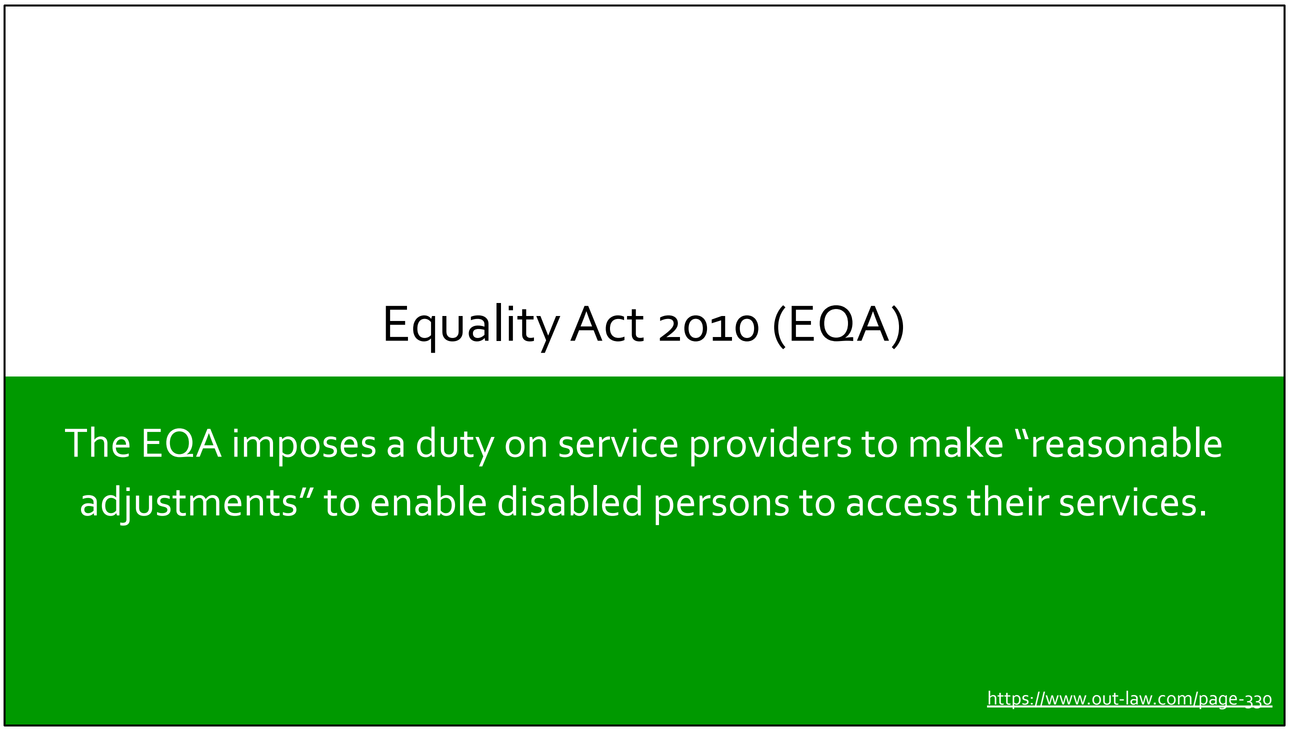
Colour contrast is very important, not just for our users and customers but for ourselves. Eyesight gets worse in most humans as they age. The ability to distinguish between colours reduces. It might not be fashionable, but increasing colour contrast can even save battery life on mobile devices as users don’t have to turn the screen contrast up as much.

https://twitter.com/petervangrieken/status/1268171535736541184

I won’t dwell on accessibility too much longer, I have a whole other talk about that if people are interested. I do want to say that increasing the accessibility of your pages benefits everyone, in the same way that increasing the accessibility of a physical space - in this example getting on or off a train - benefits more than just people with disabilities.

This is catering for the human needs of people using our site - whether they have visual or physical disabilities, or any of the wide range of cognitive problems.

And the other reason to do it, of course, is it’s the law.



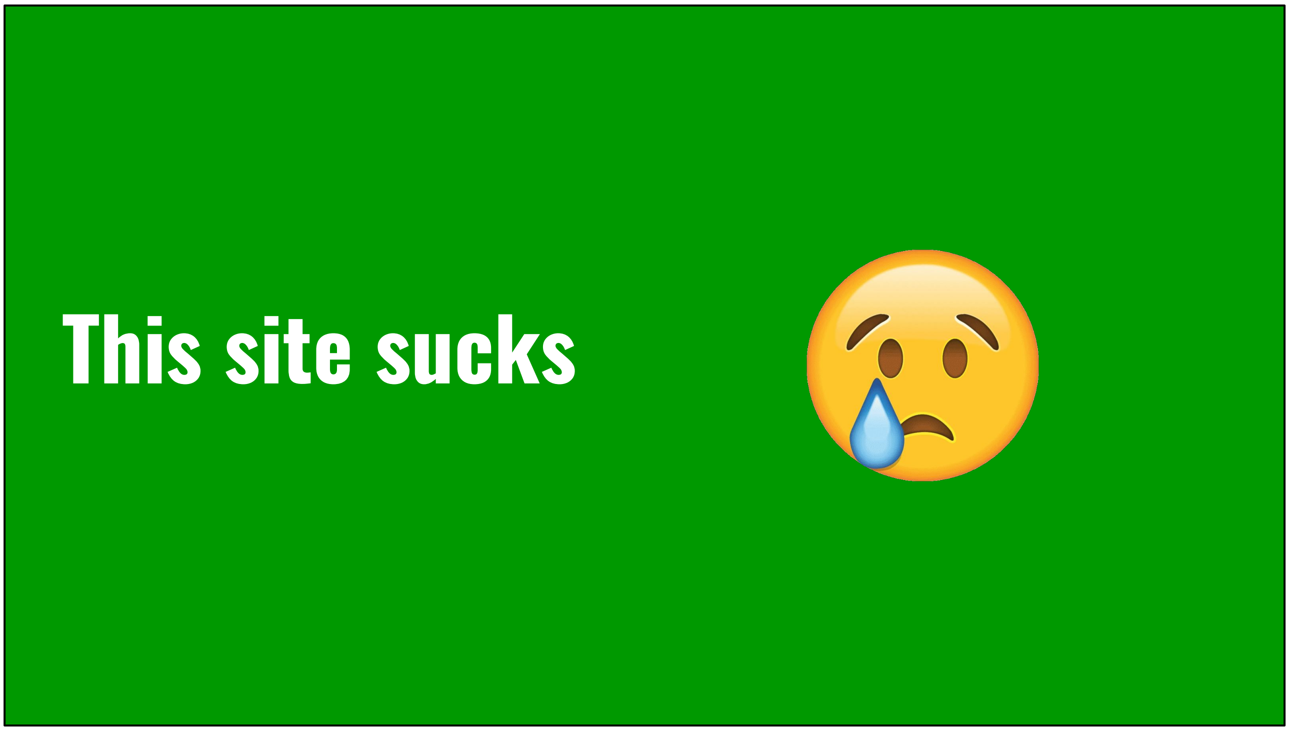
## Equality Act 2010 (EQA)

The EQA imposes a duty on service providers to make “reasonable adjustments” to enable disabled persons to access their services.

https://www.out-law.com/page-330

The Equality Act of 2010 in the UK gives clear guidance to service providers - which includes businesses with a web site or app - to make “reasonable adjustments” for disabled persons.

What happens if we fail to provide an accessible experience for users? You’ve guessed it.



## This site sucks

They think the site sucks.

Now, you might be thinking at this point:

## Does it matter?

Works On My Machine

Certification of Attainment

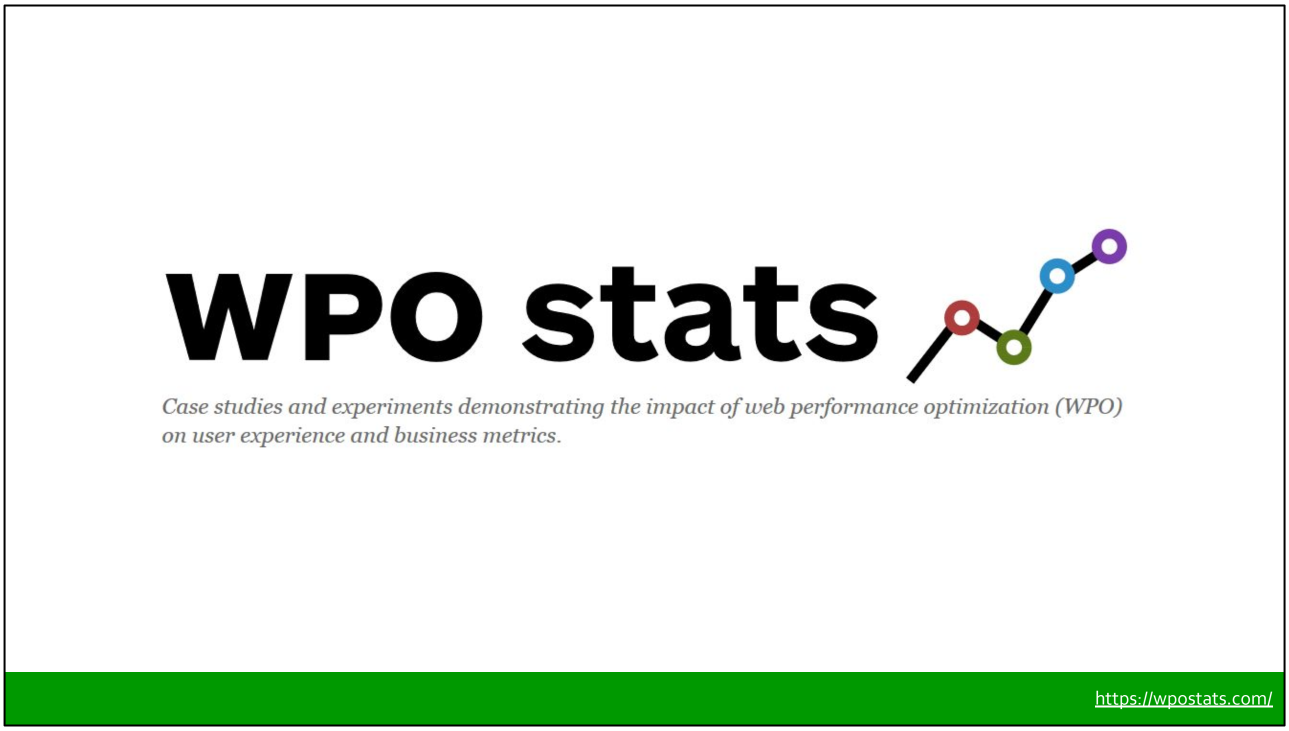
Dennis Nedry

For outstanding achievement

11 June 2020

Does any of this matter? What effect do a few performance or accessibility problems actually have on users?

There's a great website that tells us all about what effect performance has:

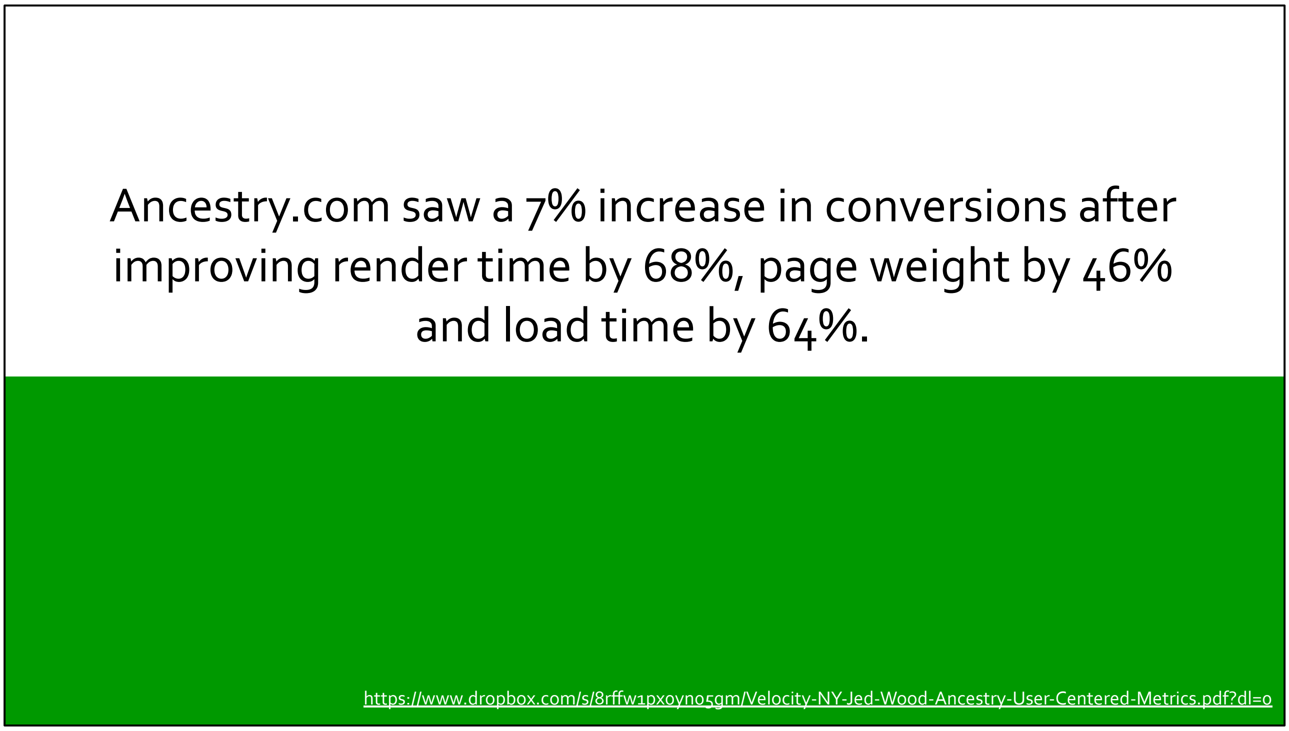


WPO stats

*Case studies and experiments demonstrating the impact of web performance optimization (WPO) on user experience and business metrics.*

https://wpostats.com/

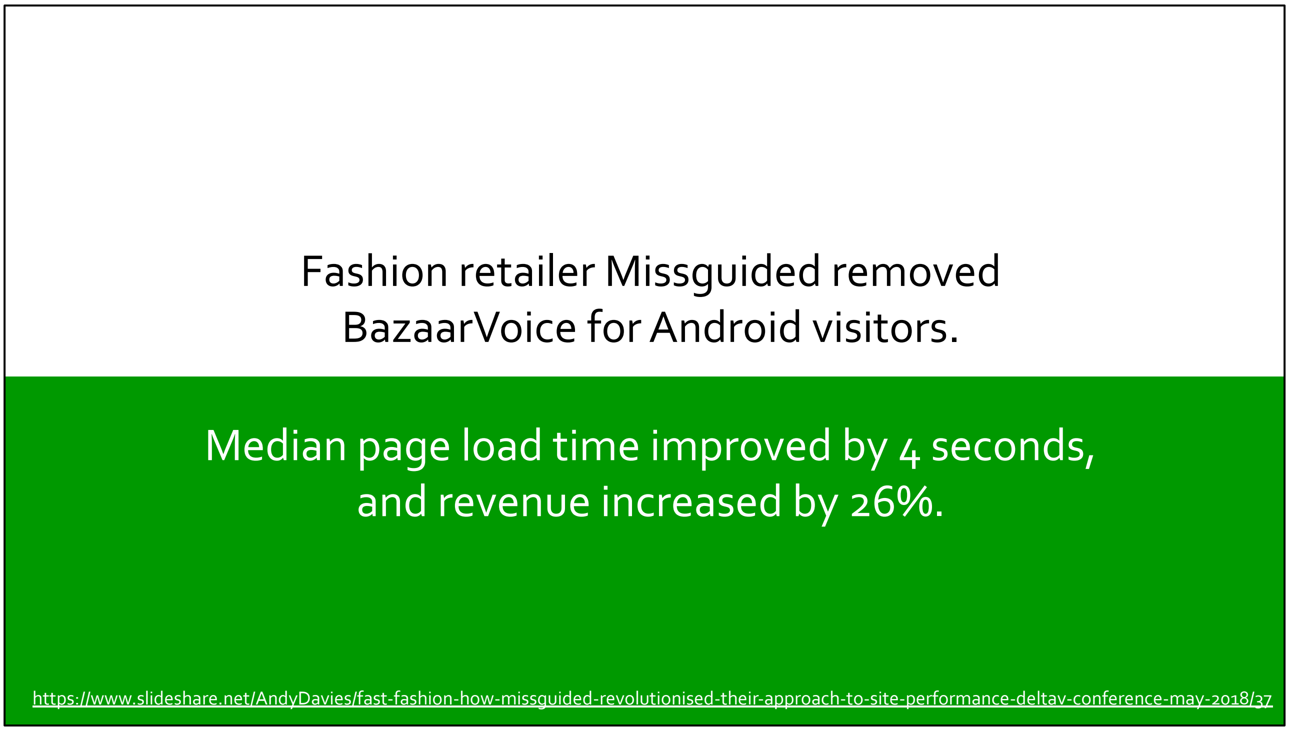
Wpostats.com. Web performance optimisation stats collects examples showing the effect of web performance for all kinds of organisations. Let's run through a few of them here.



Ancestry.com saw a 7% increase in conversions after improving render time by 68%, page weight by 46% and load time by 64%.

https://www.dropbox.com/s/8rffw1px0yno5gm/Velocity-NY-Jed-Wood-Ancestry-User-Centered-Metrics.pdf?dl=0

Improving performance by reducing render time and page weight increases conversions.

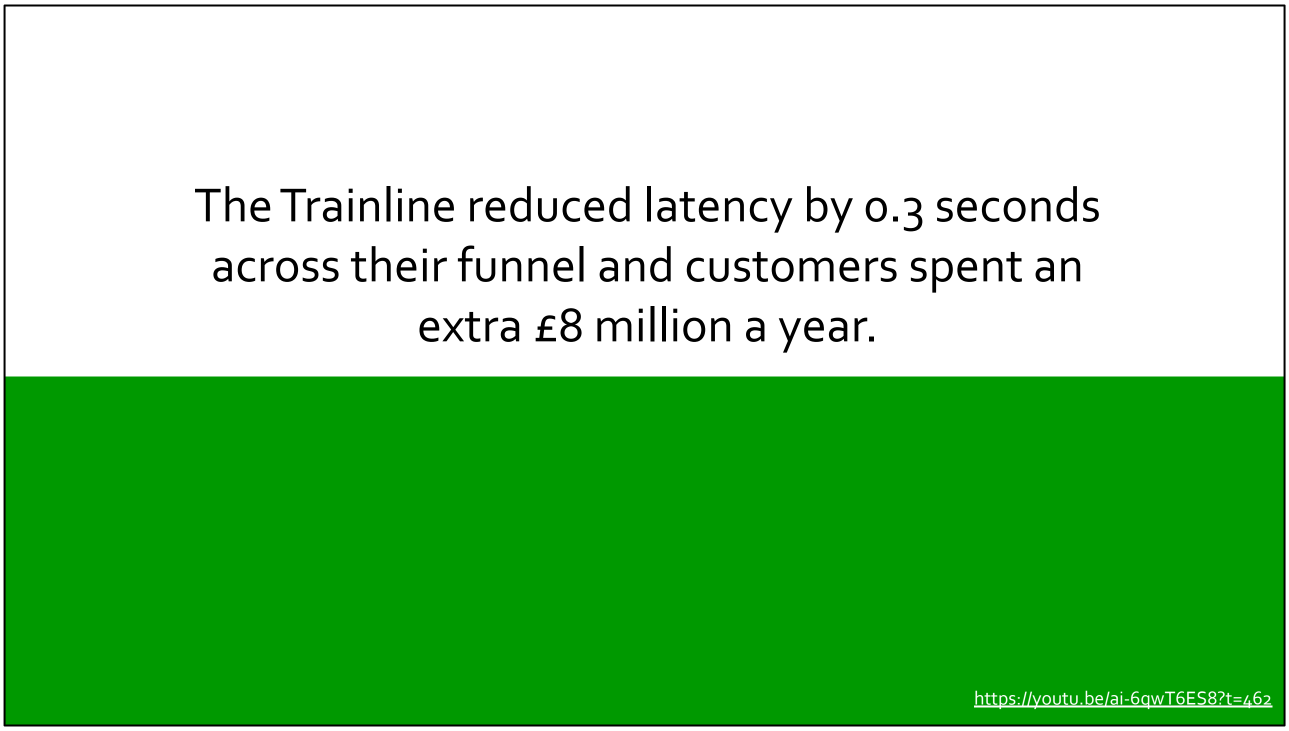


Fashion retailer Missguided removed BazaarVoice for Android visitors.

Median page load time improved by 4 seconds, and revenue increased by 26%.

https://www.slideshare.net/AndyDavies/fast-fashion-how-missguided-revolutionised-their-approach-to-site-performance-deltav-conference-may-2018/37

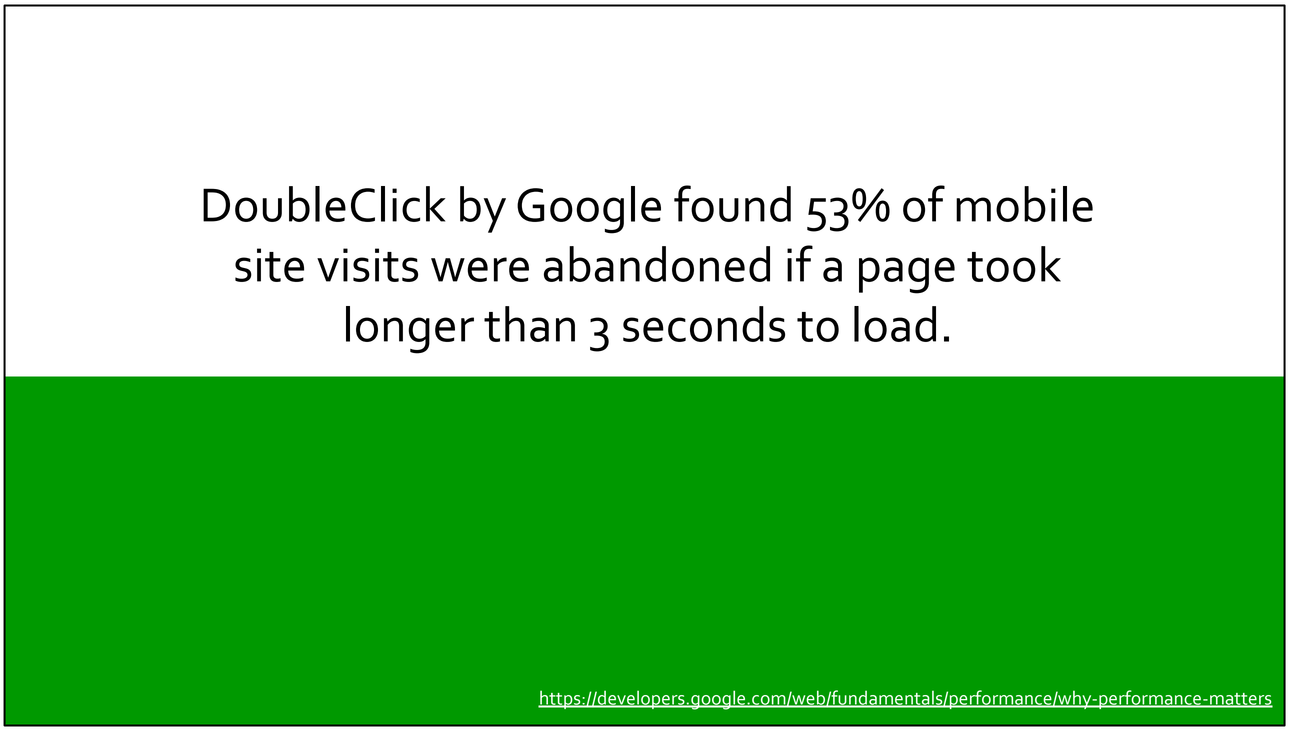
Increasing performance by removing 3rd party assets improves revenue.



The Trainline reduced latency by 0.3 seconds across their funnel and customers spent an extra £8 million a year.

https://youtu.be/ai-6qwT6ES8?t=462

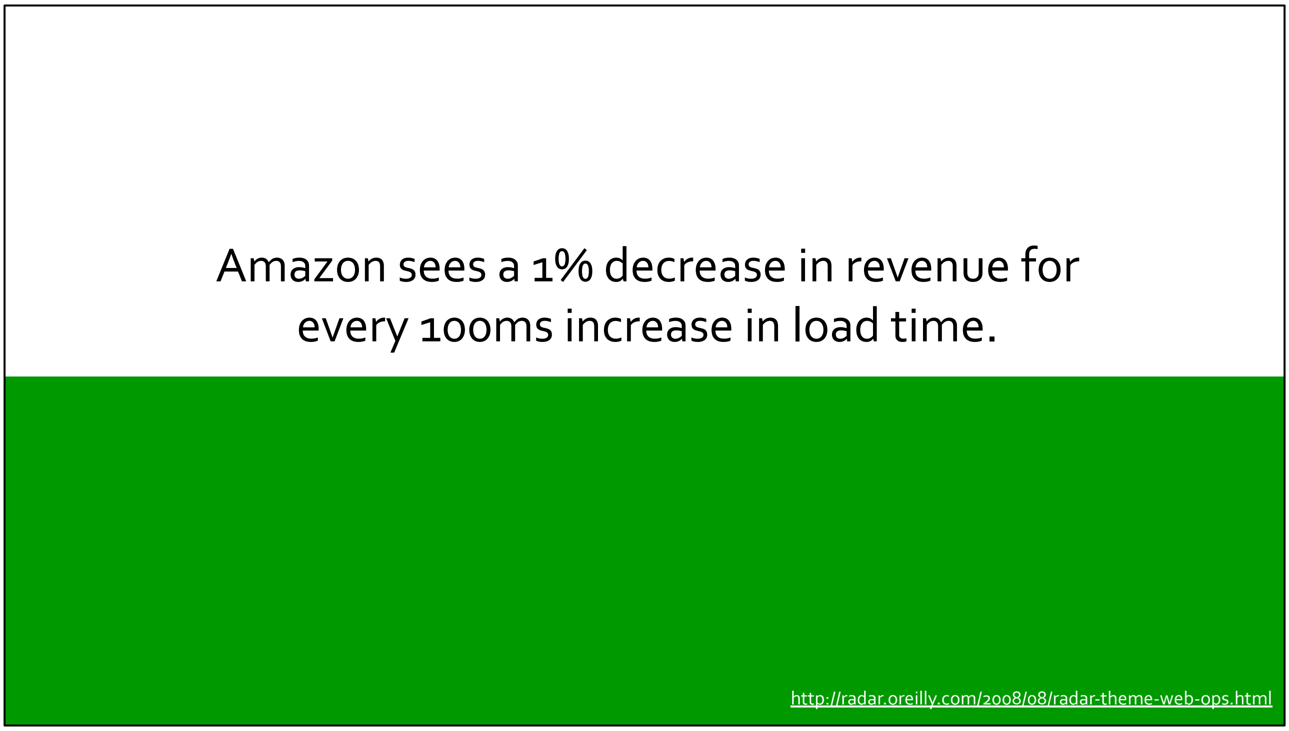
Reducing latency makes customers spend more.



DoubleClick by Google found 53% of mobile site visits were abandoned if a page took longer than 3 seconds to load.

https://developers.google.com/web/fundamentals/performance/why-performance-matters

Users will abandon your site if it's too slow.

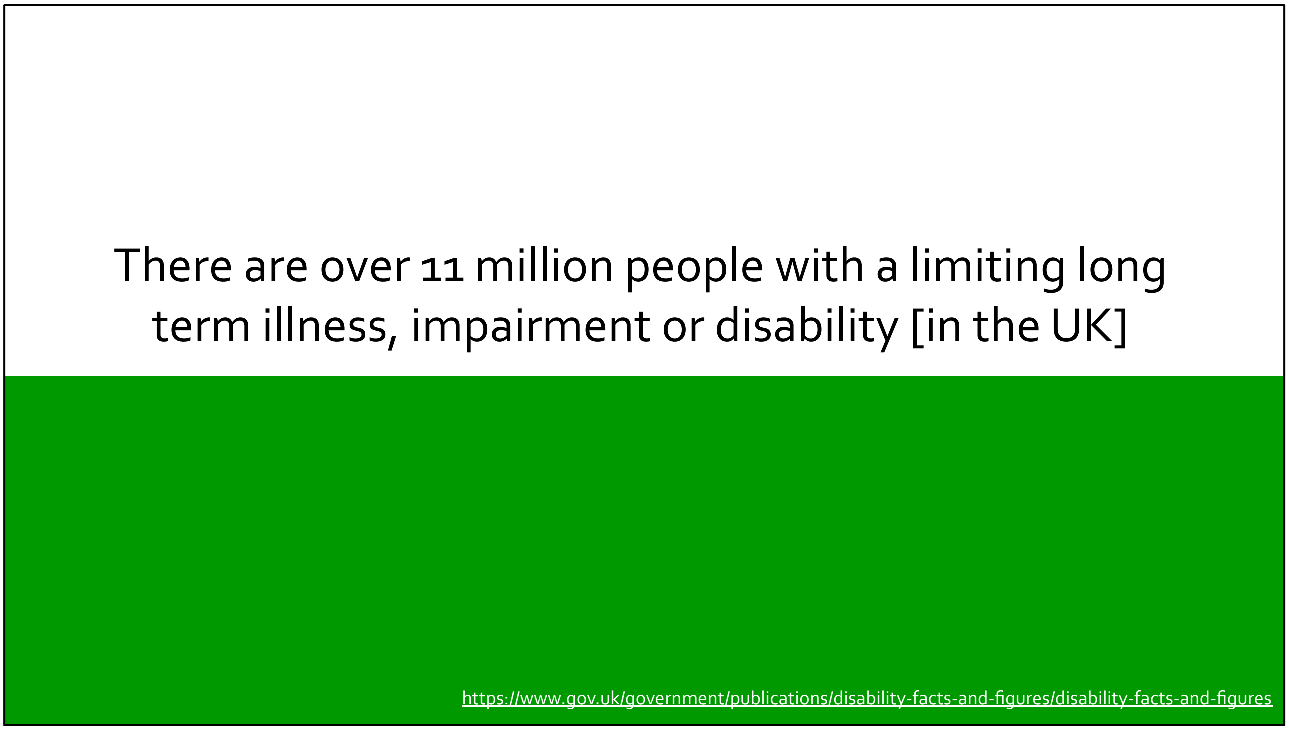

Amazon sees a 1% decrease in revenue for every 100ms increase in load time.

http://radar.oreilly.com/2008/08/radar-theme-web-ops.html

Increasing load time reduces revenue.

I could go on, but you get the picture. Company after company, organisation after organisation have found that improving performance - handling the unexpected conditions their sites encounter - consistently improves business metrics. Some of these studies are from a while ago, but you'll agree with me that good performance isn't going to go out of style any time soon.

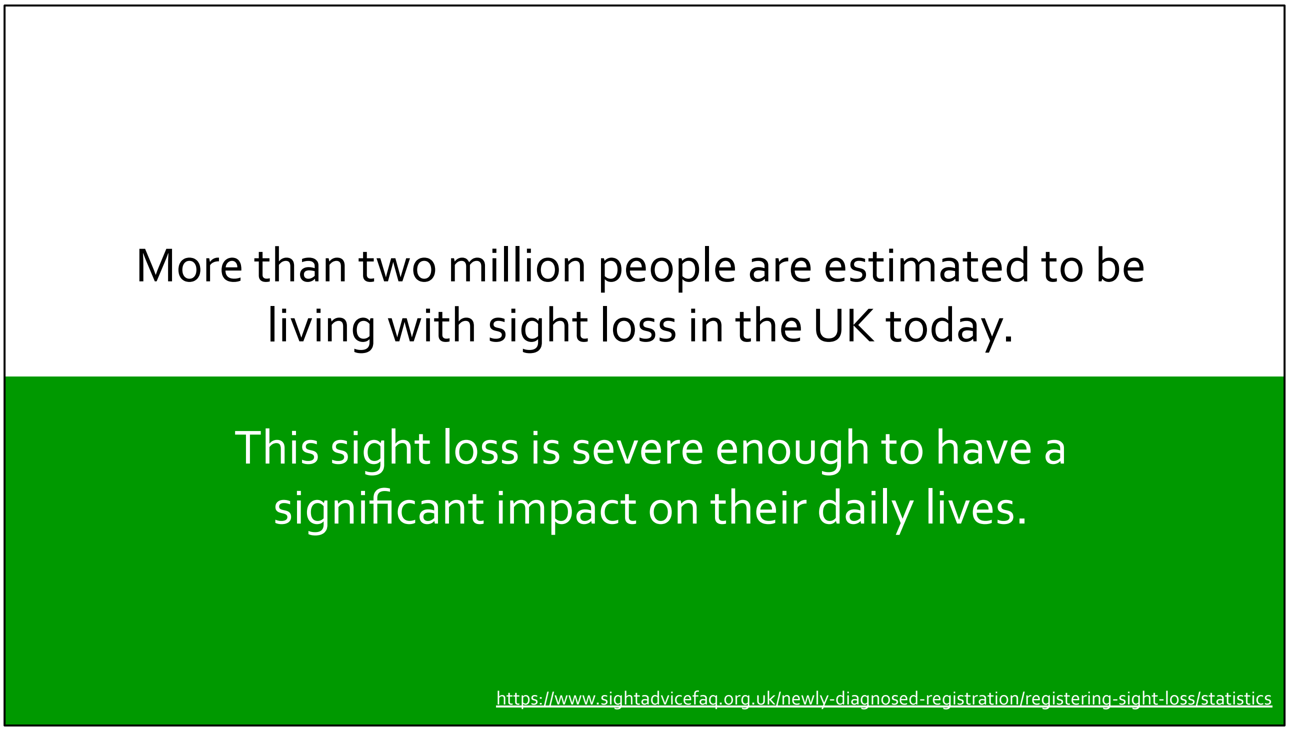
OK, what about accessibility.



## There are over 11 million people with a limiting long term illness, impairment or disability [in the UK]

https://www.gov.uk/government/publications/disability-facts-and-figures/disability-facts-and-figures

Firstly the number of people who have a long-term illness, impairment or disability is probably much higher than you thought. Many of these things affect how people use the web. Did you know:

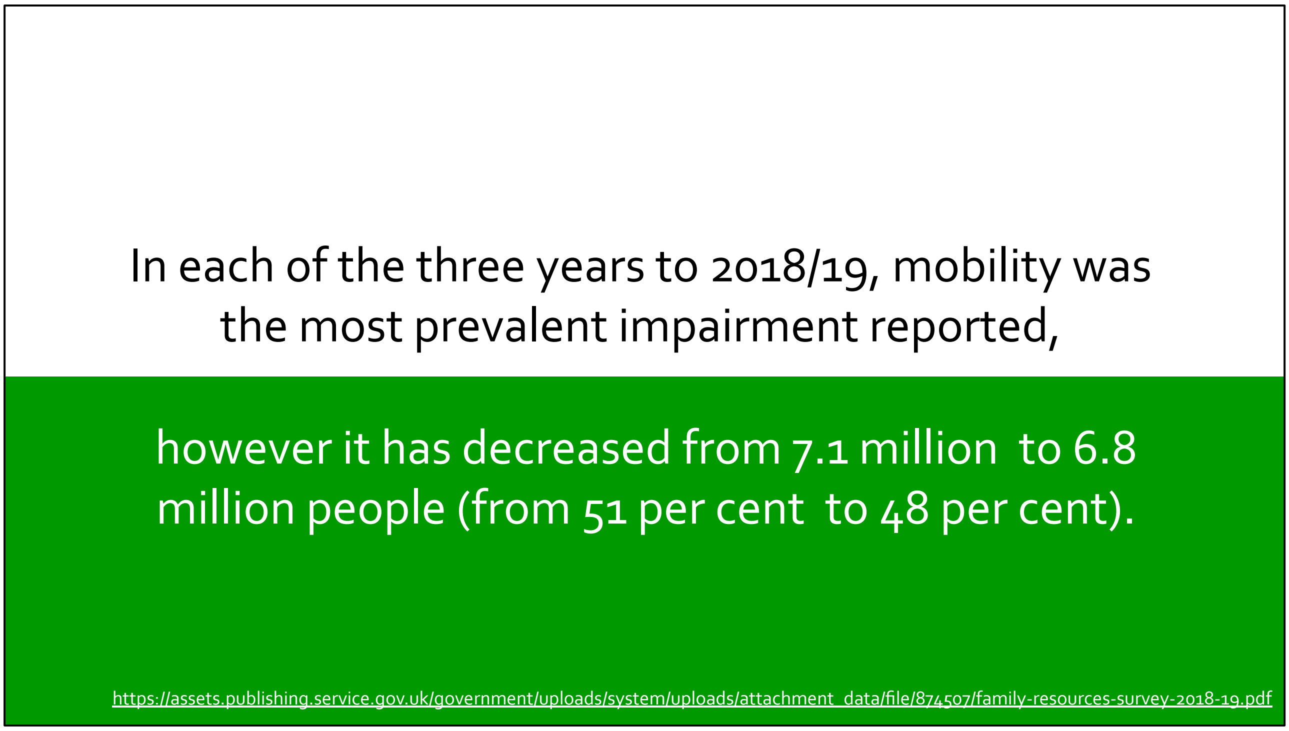


More than two million people are estimated to be living with sight loss in the UK today.

This sight loss is severe enough to have a significant impact on their daily lives.

https://www.sightadvicefaq.org.uk/newly-diagnosed-registration/registering-sight-loss/statistics

More than 2 million people in the UK have some form of sight loss.



In each of the three years to 2018/19, mobility was the most prevalent impairment reported,

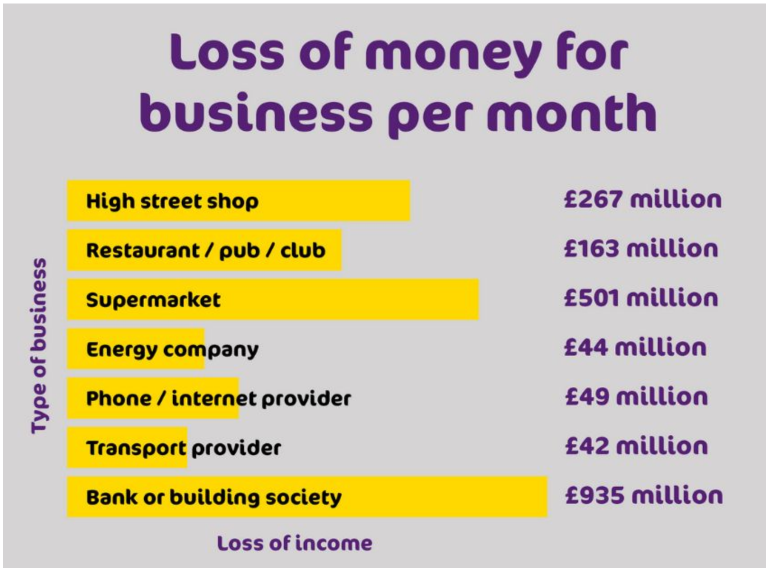
however it has decreased from 7.1 million to 6.8 million people (from 51 per cent to 48 per cent).

https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/874507/family-resources-survey-2018-19.pdf

6.8 million people are living with some form of mobility problem, which especially for older people can affect their manual dexterity such as using a mouse.

http://www.sightandsound.co.uk/hardware/braille-notetakers/braille-sense-u2-qt.html

People with disabilities often rely on software or specialist hardware, such as this Braille keyboard. If your site doesn't provide them with a good experience they'll take their custom to somewhere that does.

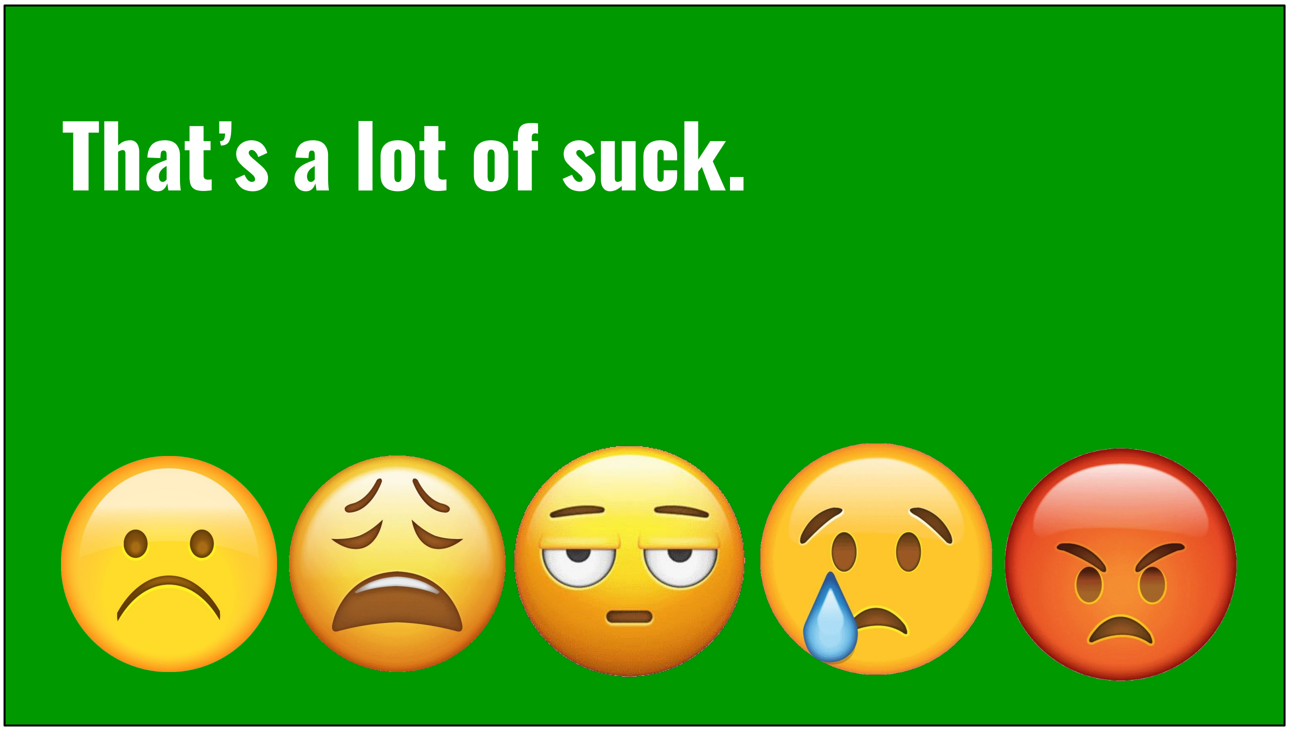


https://wearepurple.org.uk/the-purple-pound-infographic/

The “purple pound” refers to the spending power of disabled households. If your bosses don’t want to make your sites more accessible because it’s the right thing to do, or because it’s the law, maybe showing them there’s money on the table will persuade them.

So we’ve seen several examples of unexpected conditions that degrade the experience of people using our sites. These conditions can happen at any layer in the stack. Plus there are human needs for accessible sites.

And we’ve seen statistics showing that these things directly impact the success of the site.



## That’s a lot of suck.

That’s a lot of suck. But I guess a lot of it has rung true for many of us.

The question is:

## Can we fix it?

Yes, we can!

I bet you'd thought I'd forgotten about my Bob the Builder threat, right? Nope.

There are many things we can do to fix these problems and ensure our sites are more resilient to the unexpected conditions I've described. But I've got to warn you.

Some of the things I suggest aren't fashionable. But I believe they are necessary to improve our sites and ensure they are resilient to the many different ways they may fail our users and customers.

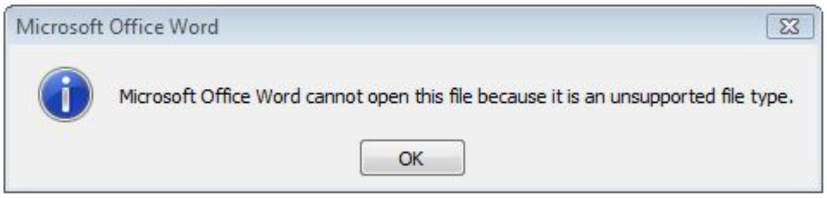
I have six tips to suggest for you. The first tip is:

## Tip 1:

## Learn how browsers work

Chickens go in, pies come out

Learn how browsers work. It’s always surprising to me how many developers working on the web have never looked into how the runtime environment they use actually functions. After all, the web is vastly different to desktop or server software.


Microsoft Office Word

Microsoft Office Word cannot open this file because it is an unsupported file type.

With desktop and server software you either get everything or nothing. Either you have the software or you don’t. And when you’re trying to open an old file format you need the right version of the software.

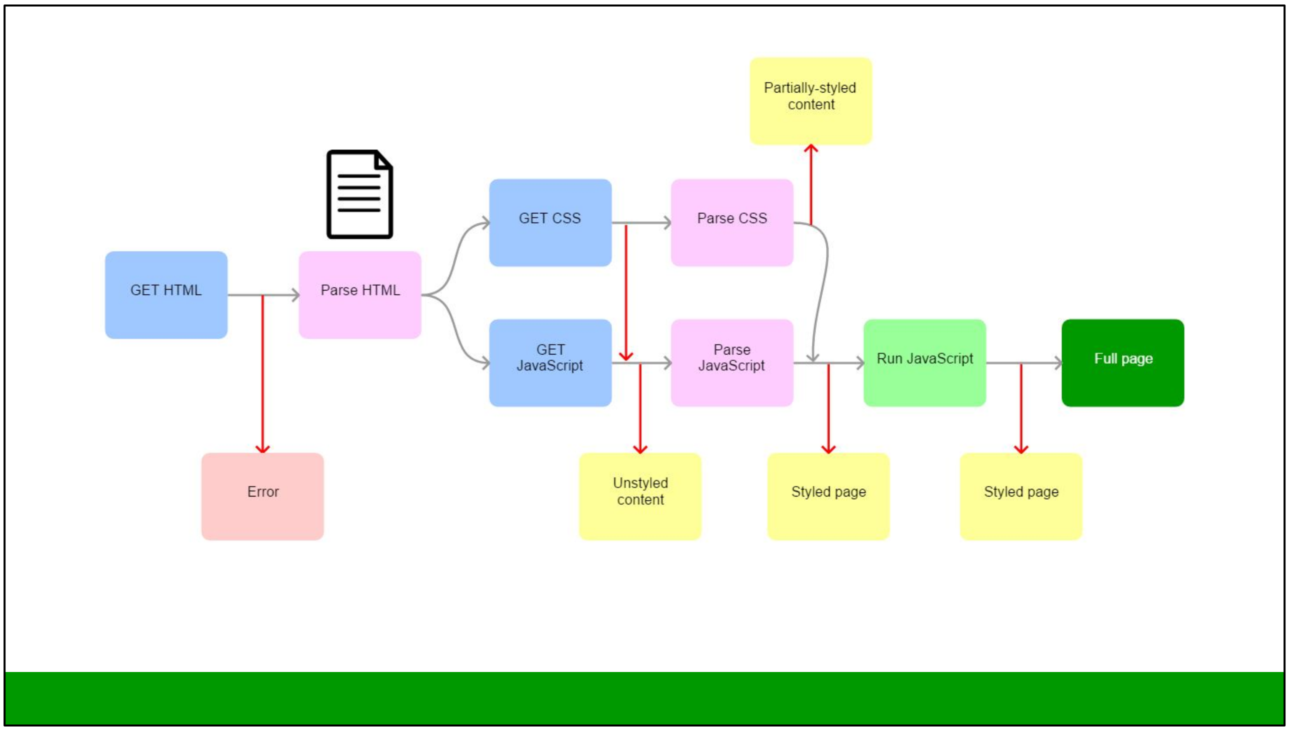
macromedia® FLASH™ ENABLED
macromedia® SHOCKWAVE® ENABLED
Java™

Many years ago the web copied this ‘all or nothing’ approach. Does anyone remember Flash, Shockwave or Java applets? If you had the right plugin you got the full experience, but if not you got nothing. It was all or nothing.

https://en.wikipedia.org/wiki/All_or_Nothing_(Milli_Vanilli_album)

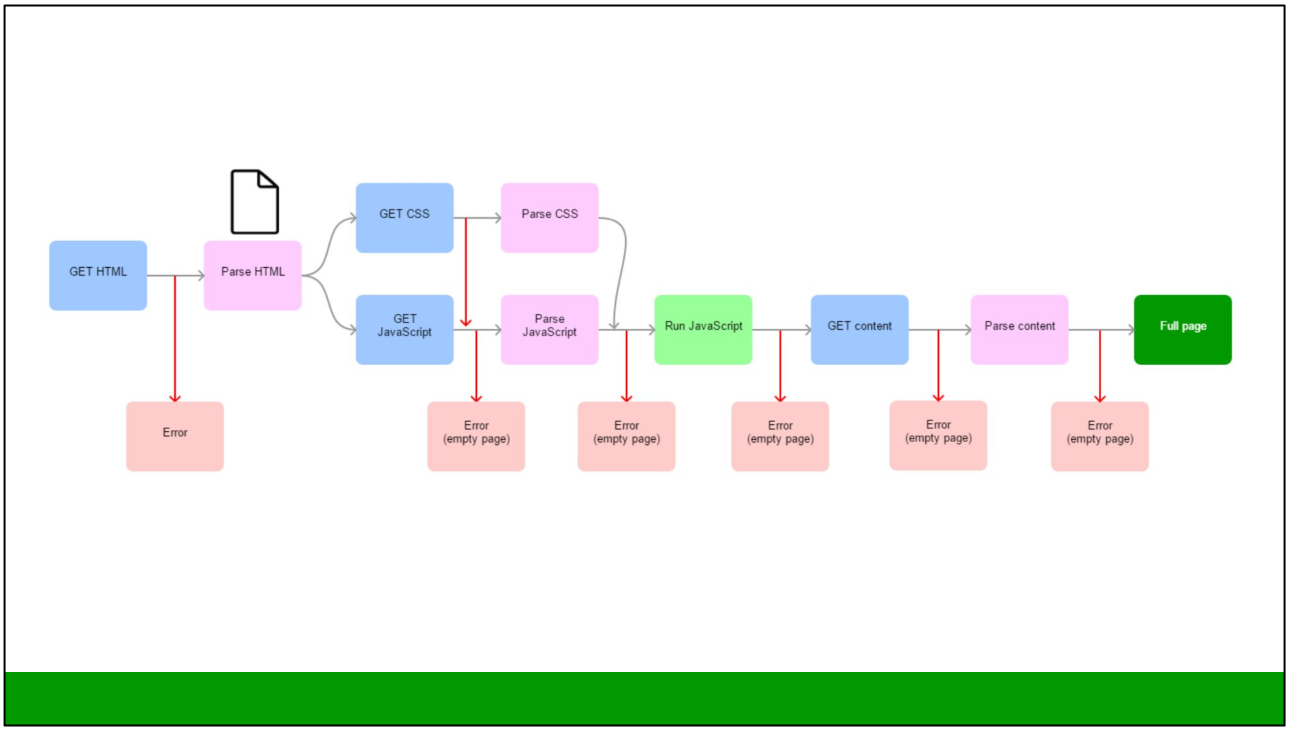
Milli Vanilli sang - and I use the term loosely - about this on their 1988 album ‘All or Nothing’, which was included in Q magazine's 2006 list of the 50 worst albums ever made.

But ‘all or nothing’ is not how the web was designed. Let’s have a look at a process diagram for a browser.



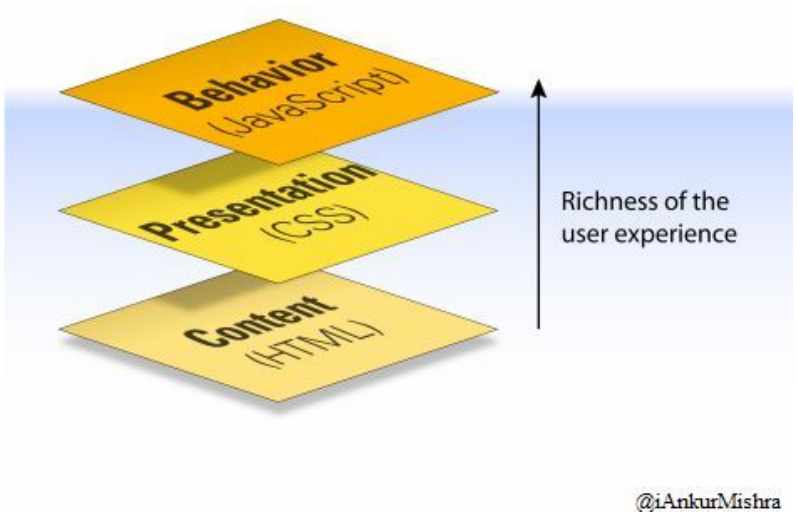
Requesting a web page begins by getting some HTML. If an error at that point happens you get the status of that error - a 404 ‘Page Not Found’, 500 ‘Server Error’, some kind of authentication error maybe. Once you get some HTML the browser parses it and kicks off two different pipelines - one getting and parsing CSS, the other getting, parsing and executing JavaScript.

If errors happen during those pipelines - assets not available, script execution problems, whatever - the user still has the HTML. Yes, it’s not a pretty or fancy web page, but they get something.



With SPA frameworks where content is loaded by JavaScript calls to APIs, there are more places that errors can occur. In this model, everything has to work before the user gets anything. Because if the script which fetches the content fails for some reason, the user has nothing. This is 'all or nothing', in the same way that Flash and Java applets were.

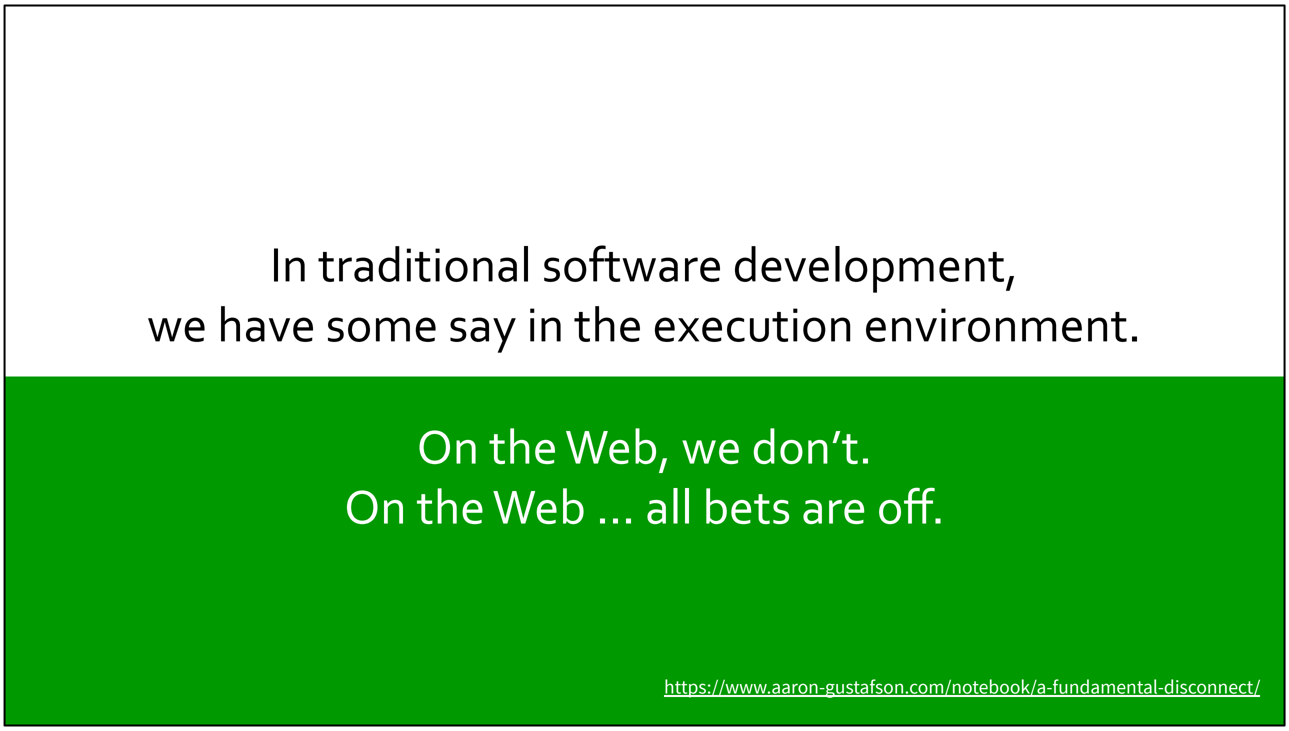
Of course, there's a lot we can do to protect against this - server-rendering is a great start.



https://www.c-sharpcorner.com/UploadFile/79037b/better-webapp-presentation-with-css/

Built into the design principles of the web is this simple idea: that these layers build on each other to increase the richness of the user experience. HTML comes first, both figuratively and literally - as when the web was being designed there was only HTML. CSS and JavaScript came later and built on it.

So you need to understand at least to a reasonably level how browsers work. Learn about rendering pipelines, blocking and non-blocking resources - there's loads of information out there to help you.


In traditional software development,
we have some say in the execution environment.

On the Web, we don't.
On the Web … all bets are off.

https://www.aaron-gustafson.com/notebook/a-fundamental-disconnect/

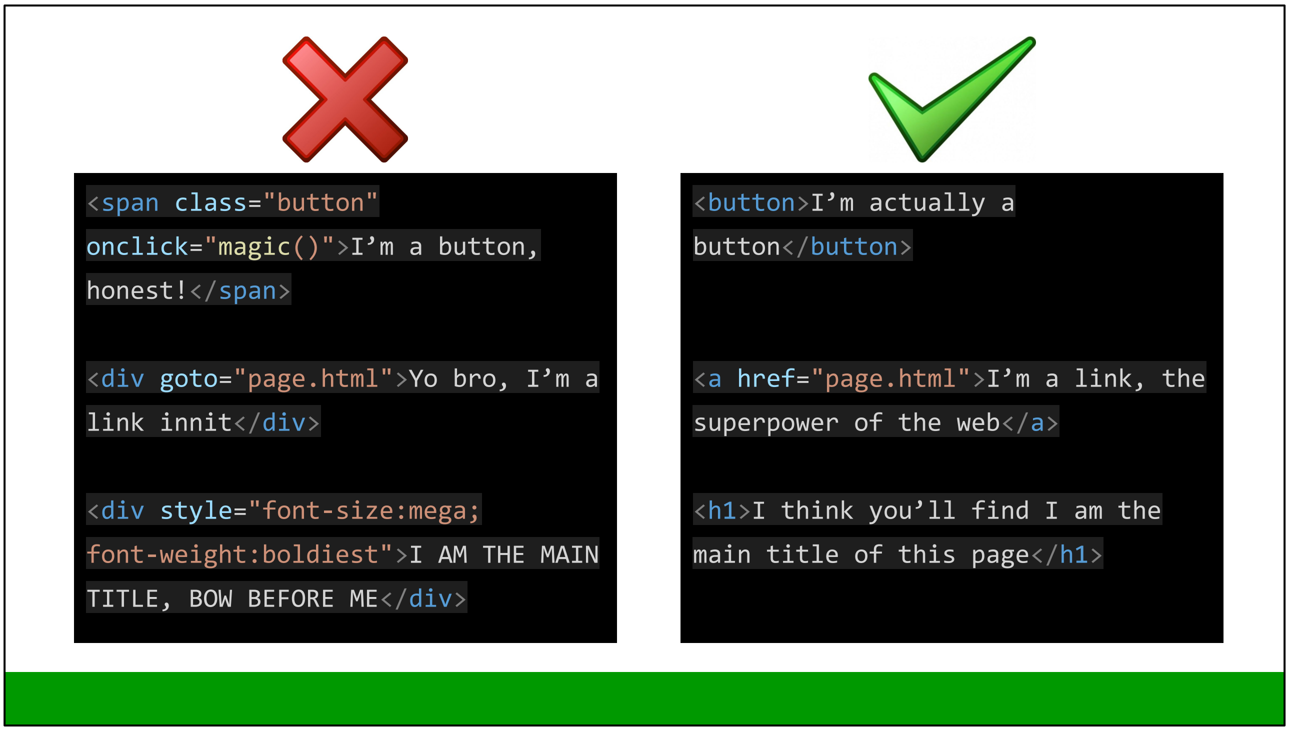
Because the reality is we don't control the execution environment our sites run in. So we should try to understand how it works so we can make our sites more resilient.

And the simplest practical way to get started is:

## Tip 2:

## Learn and use semantic HTML

Semantic HTML. By which I mean HTML which describes what the content is. We’ve already seen there are around 140 HTML elements, so let’s use them.



```
<span class="button"
onclick="magic()">I’m a button,
honest!</span>

<div goto="page.html">Yo bro, I’m a
link innit</div>

<div style="font-size:mega;
font-weight:boldiest">I AM THE MAIN
TITLE, BOW BEFORE ME</div>
```

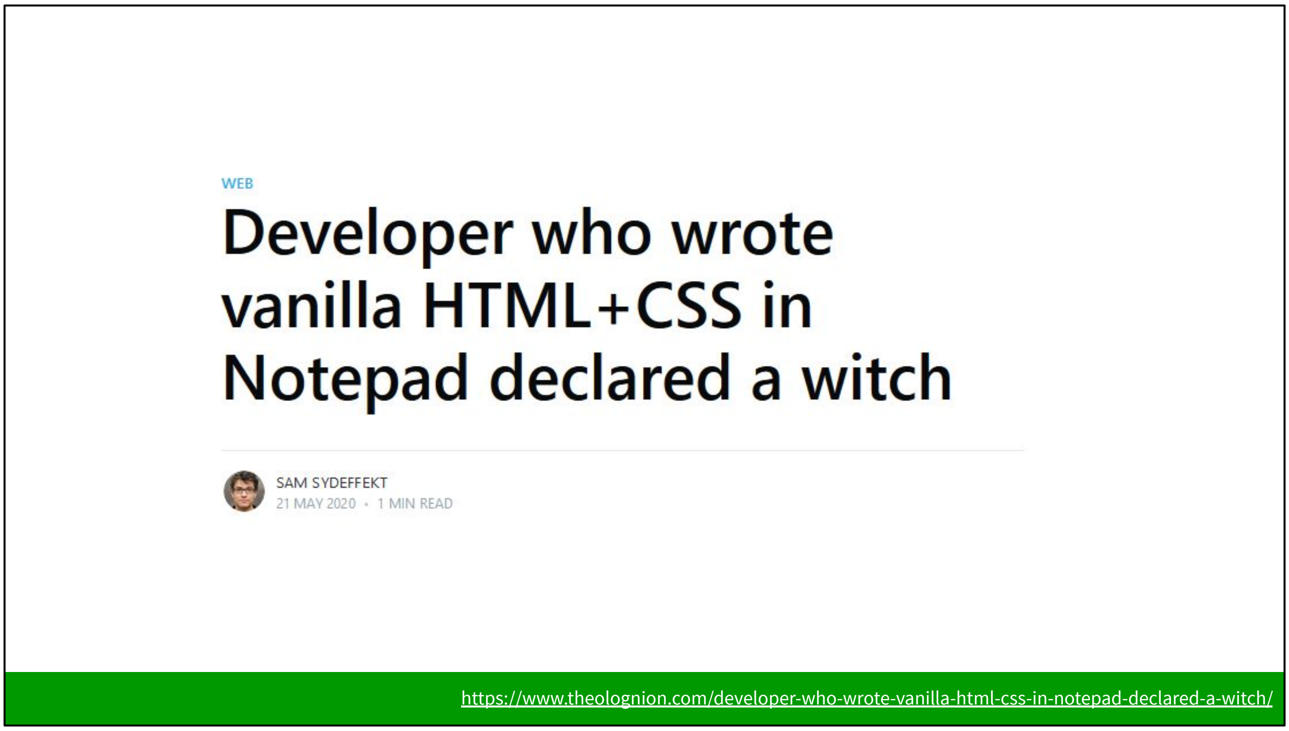
```
<button>I’m actually a
button</button>

<a href="page.html">I’m a link, the
superpower of the web</a>

<h1>I think you’ll find I am the
main title of this page</h1>
```

That means using buttons for buttons, links for links, proper headings. Doing this not only fixes many accessibility problems, but helps with search engine optimisation and saves you writing code. For example, the functionality built into the humble button element means it’s focusable, can be triggered by various keypresses as well as the mouse. That’s all functionality that you don’t have to write.

I’m not saying don’t use div and span elements at all - they have their uses - but you should look for a more suitable element first.



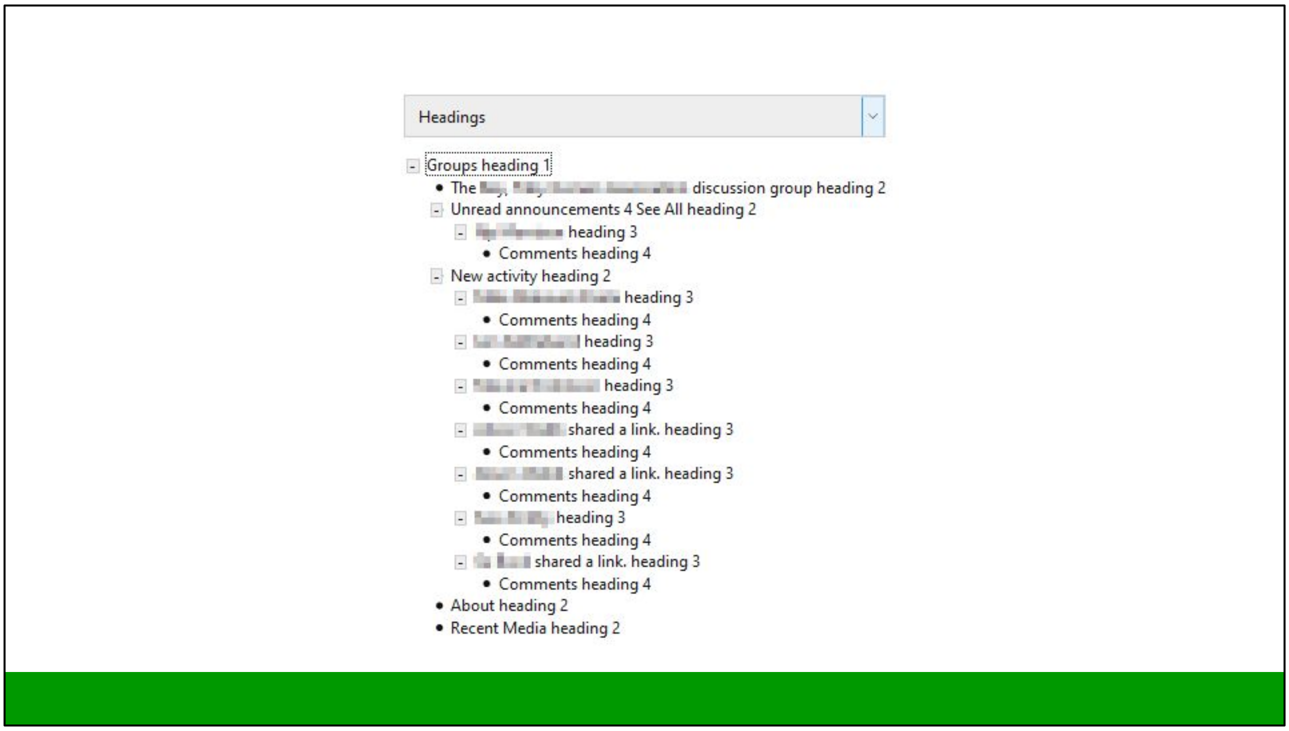
WEB

Developer who wrote vanilla HTML+CSS in Notepad declared a witch

SAM SYDEFFEKT
21 MAY 2020 • 1 MIN READ

https://www.theolognion.com/developer-who-wrote-vanilla-html-css-in-notepad-declared-a-witch/

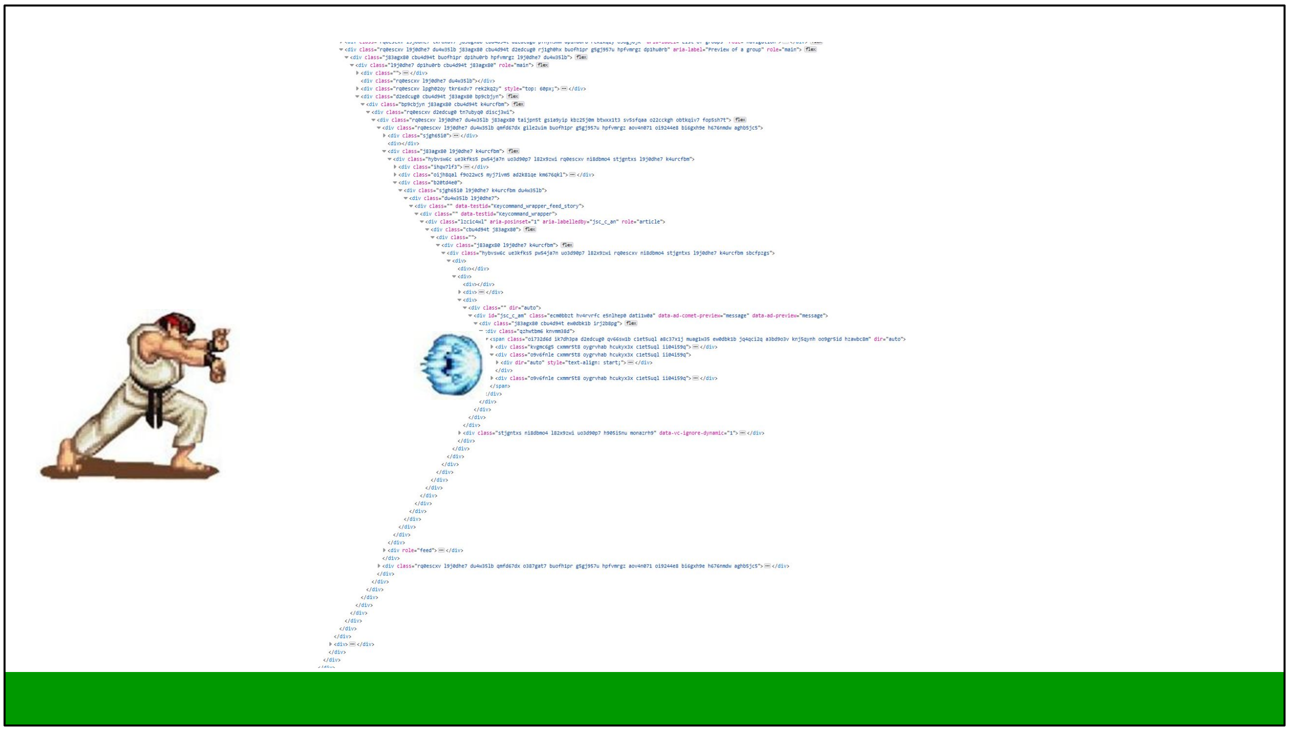
Knowing HTML isn't fashionable, but I'd argue it should be. Because semantic HTML means that your pages have a 'flow' to them which is discoverable programmatically.


Headings

- Groups heading 1
  - The [redacted] discussion group heading 2
  - Unread announcements 4 See All heading 2
    - [redacted] heading 3
      - Comments heading 4
  - New activity heading 2
    - [redacted] heading 3
      - Comments heading 4
    - [redacted] heading 3
      - Comments heading 4
    - [redacted] heading 3
      - Comments heading 4
    - [redacted] shared a link. heading 3
      - Comments heading 4
    - [redacted] shared a link. heading 3
      - Comments heading 4
    - [redacted] heading 3
      - Comments heading 4
    - [redacted] shared a link. heading 3
      - Comments heading 4
  - About heading 2
  - Recent Media heading 2

In this example you can see how headings used properly allow the page to be navigated by assistive technologies. Users can skip over entire sections if they don’t contain content they’re interested in.

The over use of div elements for layout is a particular bugbear of mine. It’s been called ‘divitis’, and you can see why:

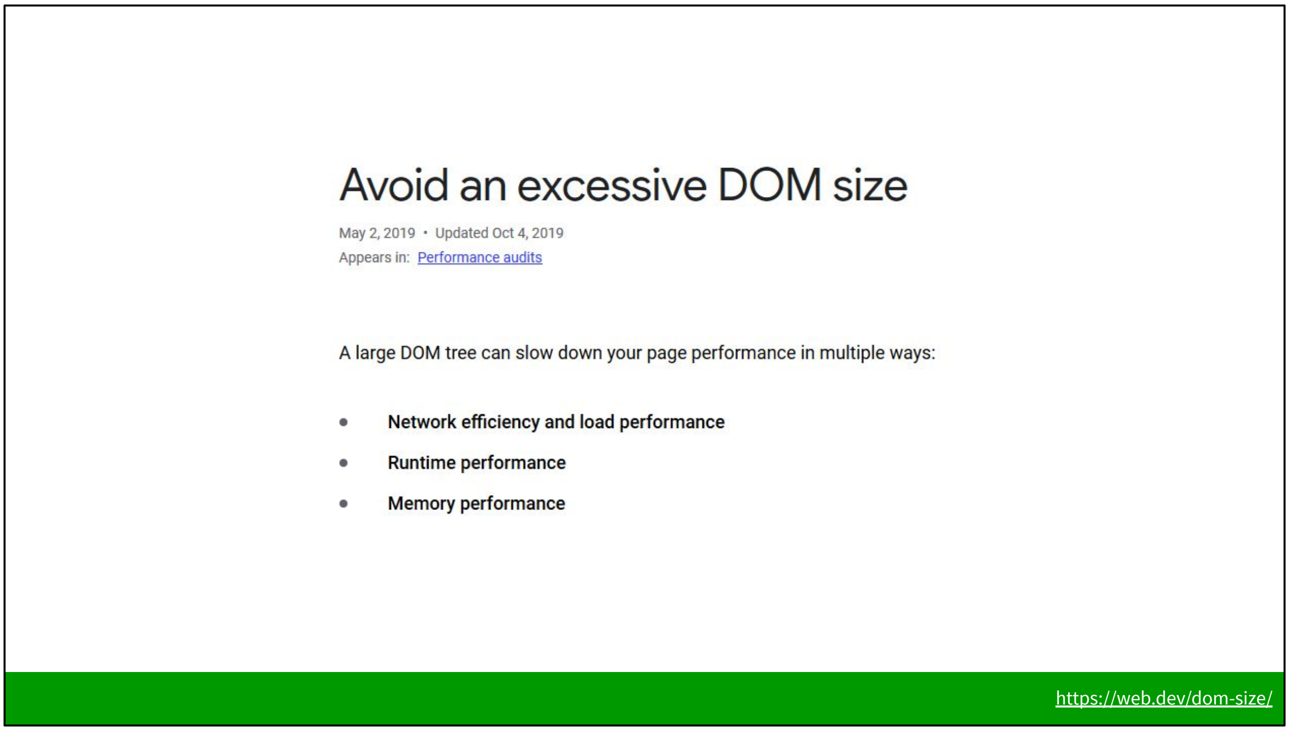


```
          <div class="rq0escxv l9j0dhe7 du4w35lb j83agx80 cbu4d94t d2edcug0 rj1gh0hx buofh1pr g5gj957u hpfvmrgz dp1hu0rb" aria-label="Preview of a group" role="main"> flex
            <div class="j83agx80 cbu4d94t buofh1pr dp1hu0rb hpfvmrgz l9j0dhe7 du4w35lb"> flex
              <div class="l9j0dhe7 dp1hu0rb cbu4d94t j83agx80" role="main"> flex
                <div class="">…</div>
                <div class="rq0escxv l9j0dhe7 du4w35lb"></div>
                <div class="rq0escxv lpgh02oy tkr6xdv7 rek2kq2y" style="top: 60px;">…</div>
                <div class="d2edcug0 cbu4d94t j83agx80 bp9cbjyn"> flex
                  <div class="bp9cbjyn j83agx80 cbu4d94t k4urcfbm"> flex
                    <div class="rq0escxv d2edcug0 tn7ubyq0 discj3wi">
                      <div class="rq0escxv l9j0dhe7 du4w35lb j83agx80 taijpn5t gs1a9yip kbz25j0m btwxx1t3 sv5sfqaa o22cckgh obtkqiv7 fop5sh7t"> flex
                        <div class="rq0escxv l9j0dhe7 du4w35lb qmfd67dx gile2uim buofh1pr g5gj957u hpfvmrgz aov4n071 oi9244e8 bi6gxh9e h676nmdw aghb5jc5">
                          <div class="sjgh65i0">…</div>
                          <div></div>
                          <div class="j83agx80 l9j0dhe7 k4urcfbm"> flex
                            <div class="hybvsw6c ue3kfks5 pw54ja7n uo3d90p7 l82x9zwi rq0escxv ni8dbmo4 stjgntxs l9j0dhe7 k4urcfbm">
                              <div class="ihqw7lf3">…</div>
                              <div class="oijh8qal f9o22wc5 myj7ivm5 ad2k81qe km676qkl">…</div>
                              <div class="b20td4e0">
                                <div class="sjgh65i0 l9j0dhe7 k4urcfbm du4w35lb">
                                  <div class="du4w35lb l9j0dhe7">
                                    <div class="" data-testid="Keycommand_wrapper_feed_story">
                                      <div class="" data-testid="Keycommand_wrapper">
                                        <div class="lzcic4wl" aria-posinset="1" aria-labelledby="jsc_c_an" role="article">
                                          <div class="cbu4d94t j83agx80"> flex
                                            <div class="">
                                              <div class="j83agx80 l9j0dhe7 k4urcfbm"> flex
                                                <div class="hybvsw6c ue3kfks5 pw54ja7n uo3d90p7 l82x9zwi rq0escxv ni8dbmo4 stjgntxs l9j0dhe7 k4urcfbm sbcfpzgs">
                                                  <div>
                                                    <div></div>
                                                    <div>
                                                      <div></div>
                                                      <div>…</div>
                                                      <div>
                                                        <div class="" dir="auto">
                                                          <div id="jsc_c_am" class="ecm0bbzt hv4rvrfc e5nlhep0 dati1w0a" data-ad-comet-preview="message" data-ad-preview="message">
                                                            <div class="j83agx80 cbu4d94t ew0dbk1b irj2b8pg"> flex
                                                              :div class="qzhwtbm6 knvmm38d">
                                                                <span class="oi732d6d ik7dh3pa d2edcug0 qv66sw1b c1et5uql a8c37x1j muag1w35 ew0dbk1b jq4qci2q a3bd9o3v knj5qynh oo9gr5id hzawbc8m" dir="auto">
                                                                  <div class="kvgmc6g5 cxmmr5t8 oygrvhab hcukyx3x c1et5uql ii04i59q">…</div>
                                                                  <div class="o9v6fnle cxmmr5t8 oygrvhab hcukyx3x c1et5uql ii04i59q">
                                                                    <div dir="auto" style="text-align: start;">…</div>
                                                                  </div>
                                                                  <div class="o9v6fnle cxmmr5t8 oygrvhab hcukyx3x c1et5uql ii04i59q">…</div>
                                                                </span>
                                                              :/div>
                                                            </div>
                                                          </div>
                                                        </div>
                                                      </div>
                                                      <div class="stjgntxs ni8dbmo4 l82x9zwi uo3d90p7 h905i5nu monazrh9" data-vc-ignore-dynamic="1">…</div>
                                                    </div>
                                                  </div>
                                                </div>
                                              </div>
                                            </div>
                                          </div>
                                        </div>
                                      </div>
                                    </div>
                                  </div>
                                </div>
                              </div>
                            </div>
                          </div>
                          <div role="feed">…</div>
                        </div>
                        <div class="rq0escxv l9j0dhe7 du4w35lb qmfd67dx o387gat7 buofh1pr g5gj957u hpfvmrgz aov4n071 oi9244e8 bi6gxh9e h676nmdw aghb5jc5">…</div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
    <div>…</div>
  </div>
</div>
```

Hadouken!

Almost everything you see here in this code from Facebook is a div.

I get it, modern sites can be complex, but having a huge number of DOM elements can cause problems.


## Avoid an excessive DOM size

May 2, 2019 • Updated Oct 4, 2019
Appears in: Performance audits

A large DOM tree can slow down your page performance in multiple ways:

- **Network efficiency and load performance**
- **Runtime performance**
- **Memory performance**

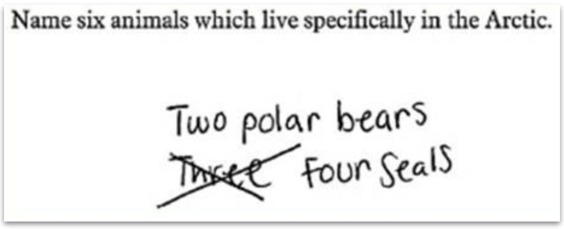
https://web.dev/dom-size/

In fact, Google's Lighthouse tool (which we'll look at in a moment) penalises sites where the DOM tree is too large.

As developers we should get familiar with the output our systems are producing. That means we need to get good at investigation and testing.

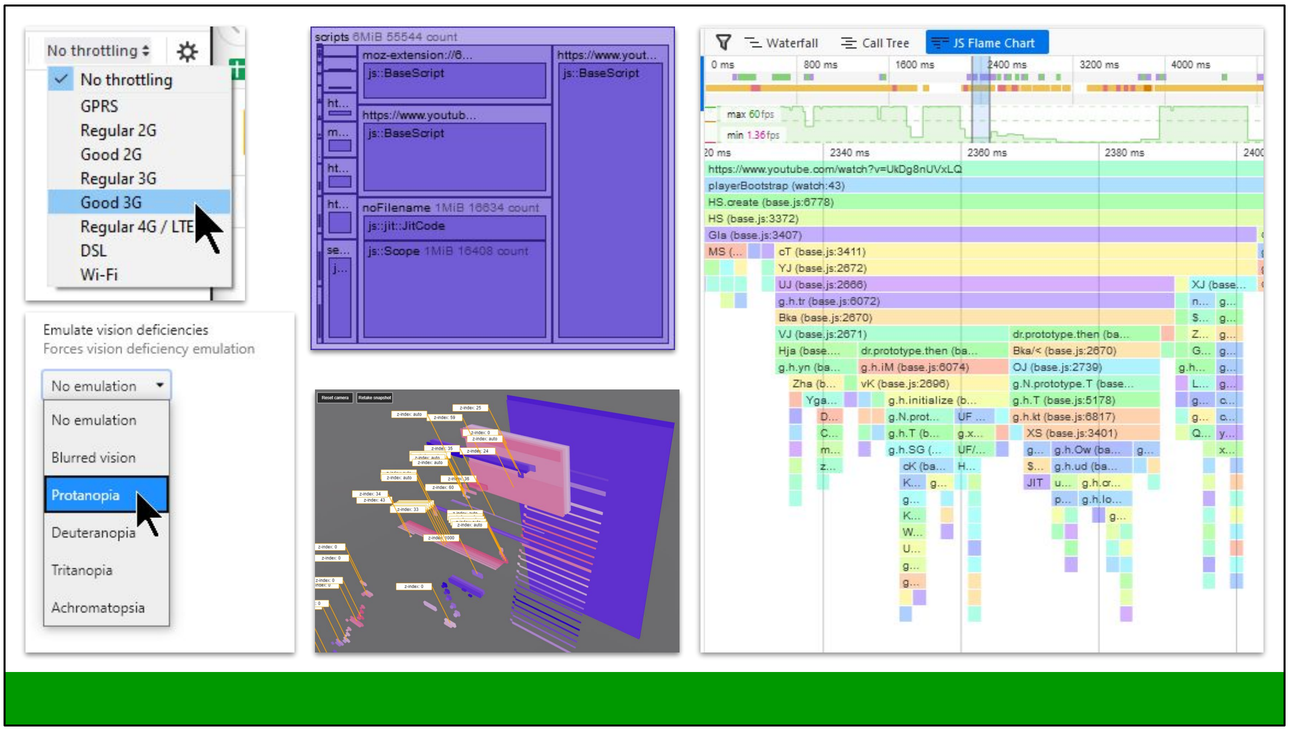
## Tip 3:

## Make testing tools your best friends


Name six animals which live specifically in the Arctic.

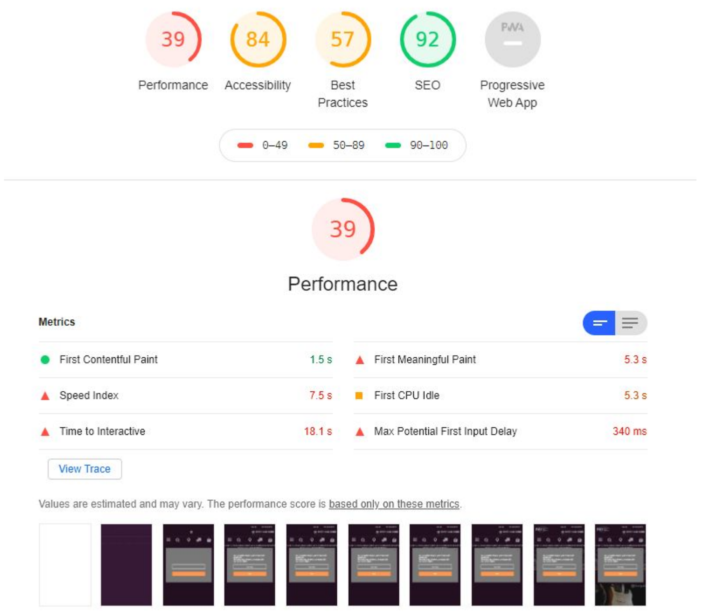
Two polar bears
~~Three~~ Four Seals

Fortunately, there are loads of great tools to help us. The first one you already use, devtools.



Devtools in all of the major browsers are amazingly powerful. I remember the days before devtools, dropping console.log calls everywhere and hoping for the best. If it wasn't for people like Chris Pederick who paved the way by creating browser extensions for web developers I probably would have gone bonkers by now.

As well as the usual features you're already using, like DOM inspection and adding breakpoints, devtools offer other features you can see here such as network throttling simulation, emulation of vision deficiencies, flame charts of JavaScript calls and Edge is bringing back the 3D view. All great stuff.



39 Performance

84 Accessibility

57 Best Practices

92 SEO

PWA – Progressive Web App

🟥 0–49 🟧 50–89 🟩 90–100

39

Performance

Metrics

| | | | |
|---|---|---|---|
| ● First Contentful Paint | 1.5 s | ▲ First Meaningful Paint | 5.3 s |
| ▲ Speed Index | 7.5 s | ■ First CPU Idle | 5.3 s |
| ▲ Time to Interactive | 18.1 s | ▲ Max Potential First Input Delay | 340 ms |

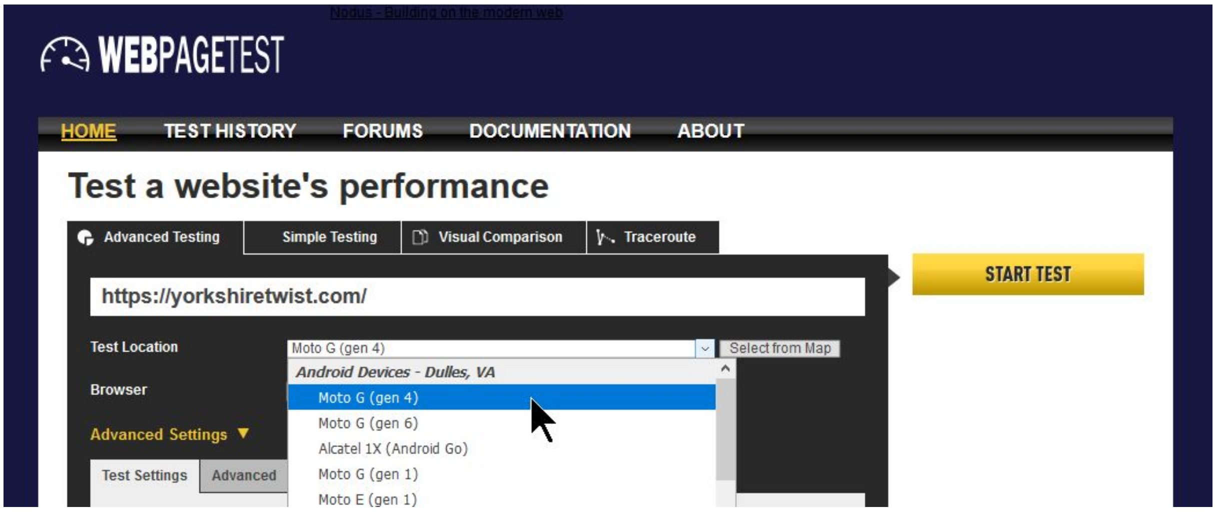
View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

https://developers.google.com/web/tools/lighthouse/

Built into the devtools for Chromium-based browsers is Lighthouse. This is an automated tool for checking the quality of web pages, based on rules set by Google. It gives a good high-level view of how well a page is working in terms of performance, accessibility and SEO. It's something you should run regularly, but as it's built into your own browser and can only simulate network conditions, it's not a proper test of what your users actually experience.

For that you need a tool which actually loads your pages from different locations.



Test a website's performance

https://yorkshiretwist.com/

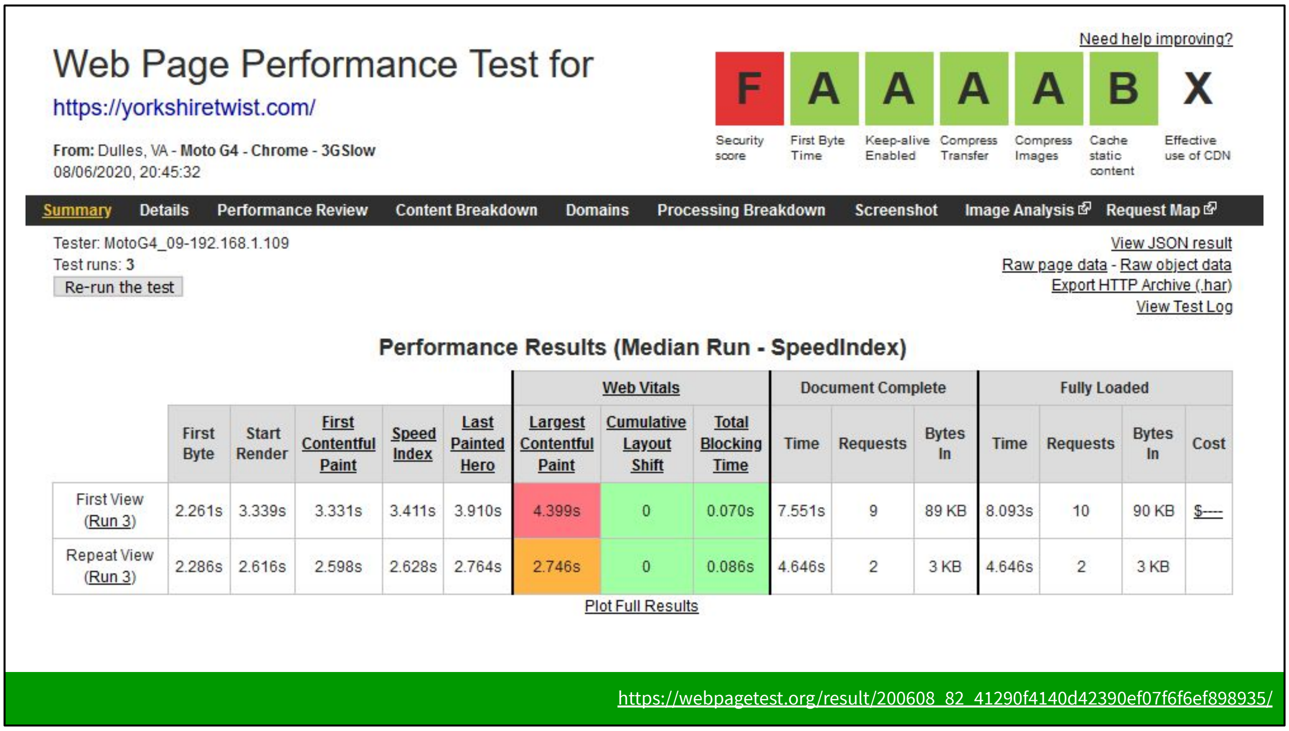
Test Location: Moto G (gen 4)

Browser

*Android Devices - Dulles, VA*
- Moto G (gen 4)
- Moto G (gen 6)
- Alcatel 1X (Android Go)
- Moto G (gen 1)
- Moto E (gen 1)

https://webpagetest.org/

So next on your list should be webpagetest.org. It may not be the prettiest of sites, but for testing performance it's a free tool of astounding quality. Here I'm testing my site with a Moto G device - yes, that's a real phone available for free from Dulles, in the US state of Virginia. Actual devices are available from just a couple of locations worldwide, but there are dozens of testing locations offering desktop browsers all over the planet.



Web Page Performance Test for
https://yorkshiretwist.com/

**From:** Dulles, VA - **Moto G4 - Chrome - 3GSlow**
08/06/2020, 20:45:32

Need help improving?

| F | A | A | A | A | B | X |
|---|---|---|---|---|---|---|
| Security score | First Byte Time | Keep-alive Enabled | Compress Transfer | Compress Images | Cache static content | Effective use of CDN |

Summary · Details · Performance Review · Content Breakdown · Domains · Processing Breakdown · Screenshot · Image Analysis · Request Map

Tester: MotoG4_09-192.168.1.109
Test runs: **3**
Re-run the test

View JSON result
Raw page data - Raw object data
Export HTTP Archive (.har)
View Test Log

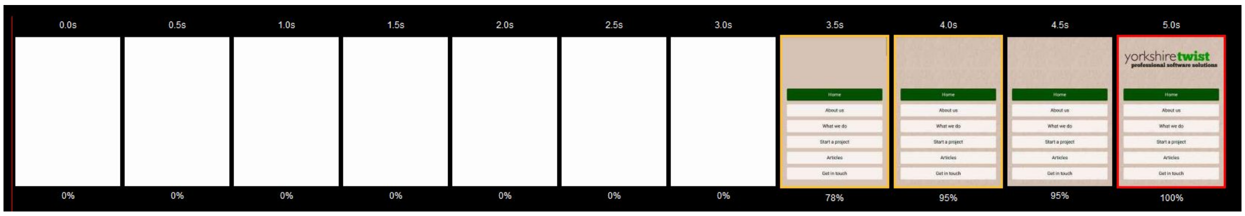
**Performance Results (Median Run - SpeedIndex)**

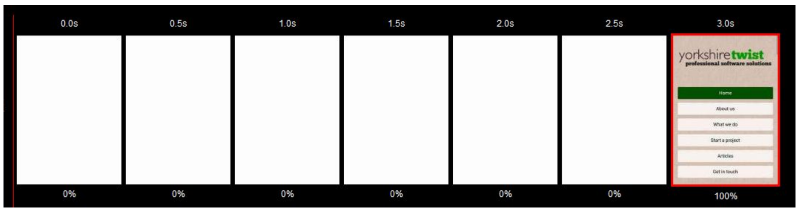
| | | | | | | Web Vitals | | | Document Complete | | | Fully Loaded | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | First Byte | Start Render | First Contentful Paint | Speed Index | Last Painted Hero | Largest Contentful Paint | Cumulative Layout Shift | Total Blocking Time | Time | Requests | Bytes In | Time | Requests | Bytes In | Cost |
| First View (Run 3) | 2.261s | 3.339s | 3.331s | 3.411s | 3.910s | 4.399s | 0 | 0.070s | 7.551s | 9 | 89 KB | 8.093s | 10 | 90 KB | $---- |
| Repeat View (Run 3) | 2.286s | 2.616s | 2.598s | 2.628s | 2.764s | 2.746s | 0 | 0.086s | 4.646s | 2 | 3 KB | 4.646s | 2 | 3 KB | |

Plot Full Results

https://webpagetest.org/result/200608_82_41290f4140d42390ef07f6f6ef898935/

Web Page Test shows loads of deep information about the performance of your site. One of my favourites is the filmstrip view:

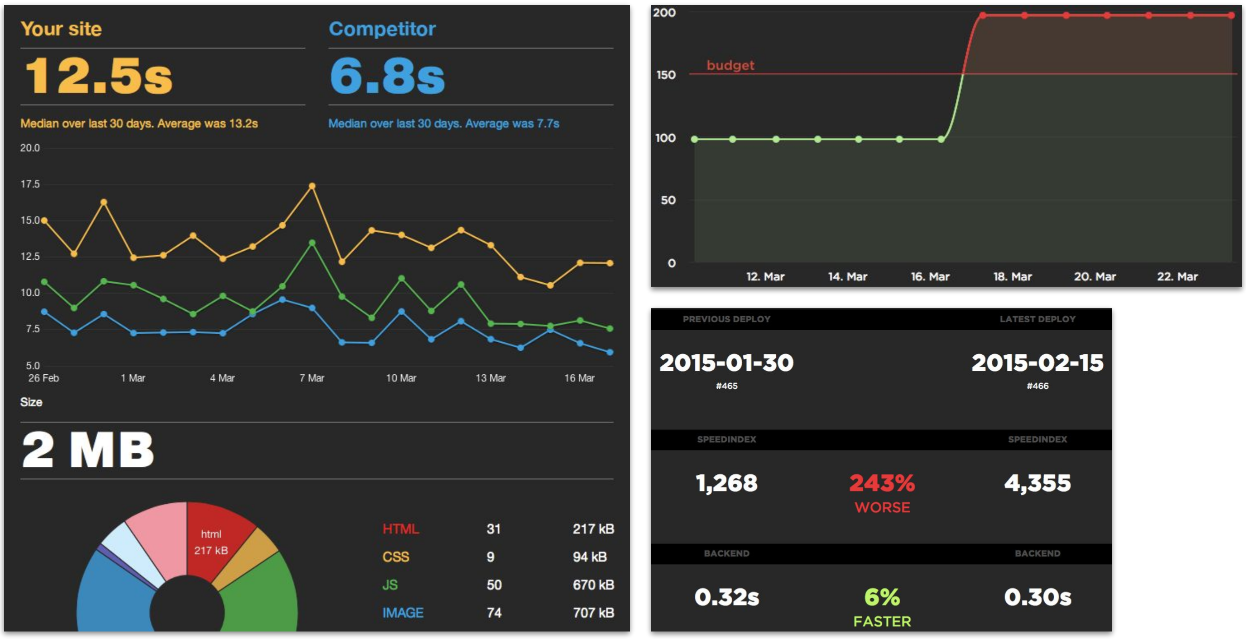




https://webpagetest.org/result/200608_82_41290f4140d42390ef07f6f6ef898935/

The top view here is the first load with an empty cache, the bottom view is a repeat visit with a primed cache. I've had great success showing these filmstrips to managers and other non-technical people to show what the performance of their sites actually looks like. And when you compare your slow site to a competitor's fast site, there's no better way to get buy-in for making performance improvements.

Please do take some time to have a look around Web Page Test at everything it does. It can be quite daunting to a newbie, and as it's free you sometimes have to wait in line behind lots of other tests. If your organisation is happy paying a bit of money then do look at Speedcurve.

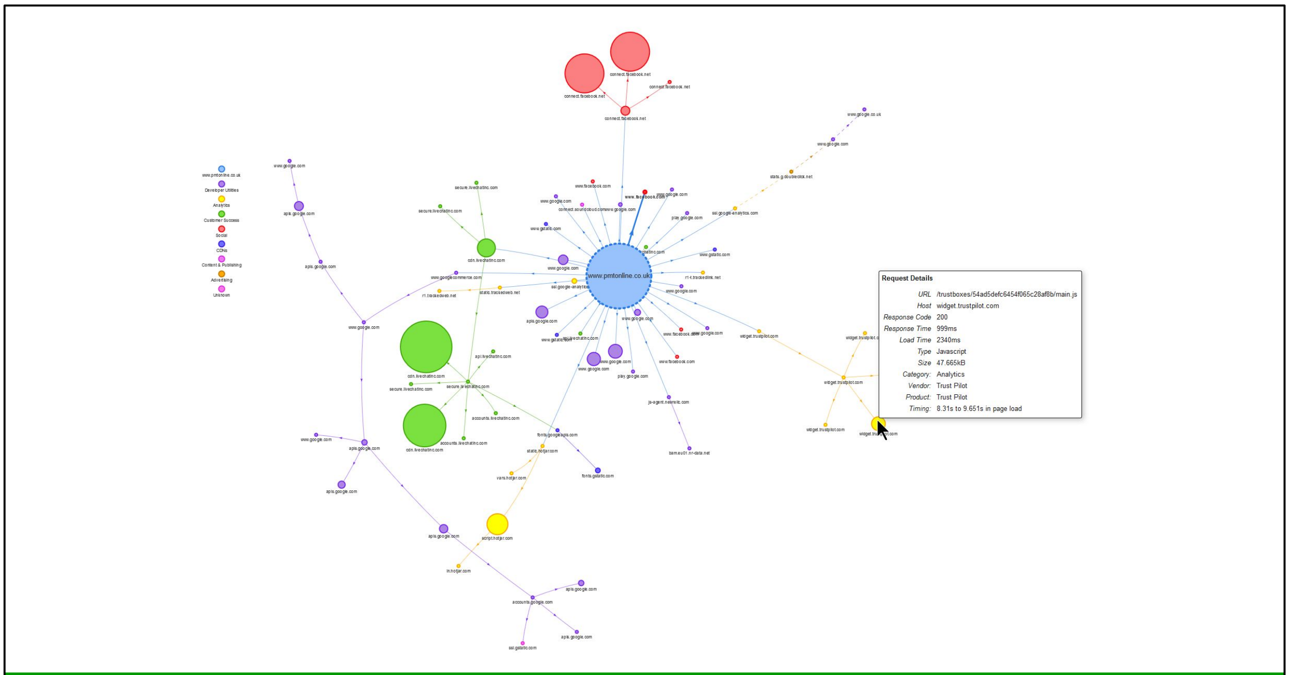


| | | |
|---|---|---|
| HTML | 31 | 217 kB |
| CSS | 9 | 94 kB |
| JS | 50 | 670 kB |
| IMAGE | 74 | 707 kB |

| PREVIOUS DEPLOY | | LATEST DEPLOY |
|---|---|---|
| 2015-01-30 #465 | | 2015-02-15 #466 |
| SPEEDINDEX | | SPEEDINDEX |
| 1,268 | 243% WORSE | 4,355 |
| BACKEND | | BACKEND |
| 0.32s | 6% FASTER | 0.30s |

https://speedcurve.com/

SpeedCurve is built on top of Web Page Test and allows you to set up scheduled tests, comparisons to other sites, performance budgets (which will tell you if your site breaches those budgets - we'll discuss this more in a bit) and much more. It's all really gorgeous, perfect for putting on a dashboard, and the SpeedCurve team comprises of some of the best web performance engineers in the business.

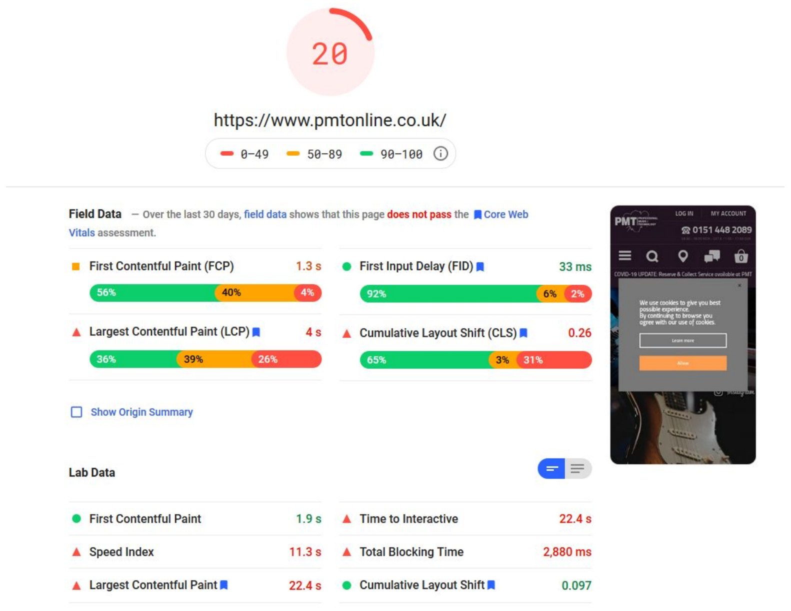


**Request Details**

| | |
|---|---|
| *URL* | /trustboxes/54ad5defc6454f065c28af8b/main.js |
| *Host* | widget.trustpilot.com |
| *Response Code* | 200 |
| *Response Time* | 999ms |
| *Load Time* | 2340ms |
| *Type* | Javascript |
| *Size* | 47.665kB |
| *Category:* | Analytics |
| *Vendor:* | Trust Pilot |
| *Product:* | Trust Pilot |
| *Timing:* | 8.31s to 9.651s in page load |

https://requestmap.webperf.tools/

Linked from Web Page Test, and using its results, is a tool called Request Map. This visually shows the relationship a site has with third party domains. Here I've tested the site for a large UK music equipment retailer, and you can see there's a whole lot more 3rd party calls than you might expect. Request Map traces calls as far as it can, so you can see branches of calls going off into the distance. This is why 3rd party calls are a performance killer.


PMT
LOG IN
MY ACCOUNT
0151 448 2089
COVID-19 UPDATE: Reserve & Collect Service available at PMT
We use cookies to give you best possible experience. By continuing to browse you agree with our use of cookies.
Learn more
Allow

20

https://www.pmtonline.co.uk/

▬ 0-49 ▬ 50-89 ▬ 90-100

**Field Data** — Over the last 30 days, field data shows that this page does not pass the Core Web Vitals assessment.

| Metric | Value | Good | Needs improvement | Poor |
|---|---|---|---|---|
| First Contentful Paint (FCP) | 1.3 s | 56% | 40% | 4% |
| First Input Delay (FID) | 33 ms | 92% | 6% | 2% |
| Largest Contentful Paint (LCP) | 4 s | 36% | 39% | 26% |
| Cumulative Layout Shift (CLS) | 0.26 | 65% | 3% | 31% |

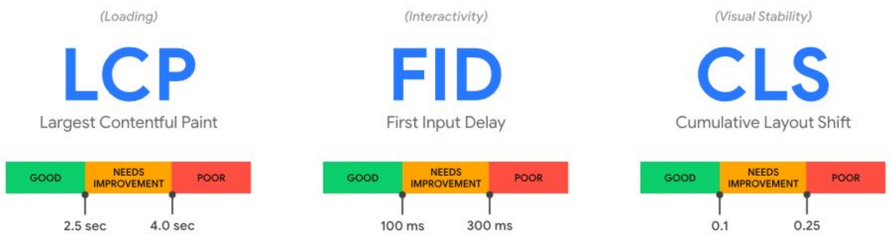
Show Origin Summary

**Lab Data**

| | | | |
|---|---|---|---|
| First Contentful Paint | 1.9 s | Time to Interactive | 22.4 s |
| Speed Index | 11.3 s | Total Blocking Time | 2,880 ms |
| Largest Contentful Paint | 22.4 s | Cumulative Layout Shift | 0.097 |

https://developers.google.com/speed/pagespeed/insights/

It's also worth mentioning PageSpeed Insights, which is another Google tool providing much the same information as Lighthouse, but using Google's infrastructure rather than your own browser. What I want to draw your attention to here is the mention of "Core Web Vitals".



- Largest Contentful Paint (LCP): measures *loading* performance. To provide a good user experience, LCP should occur within **2.5 seconds** of when the page first starts loading.
- First Input Delay (FID): measures *interactivity*. To provide a good user experience, pages should have a FID of less than **100 milliseconds**.
- Cumulative Layout Shift (CLS): measures *visual stability*. To provide a good user experience, pages should maintain a CLS of less than **0.1**.

For each of the above metrics, to ensure you're hitting the recommended target for most of your users, a good threshold to measure is the **75th percentile** of page loads, segmented across mobile and desktop devices.

https://web.dev/vitals/

Core Web Vitals is a new initiative from Google which tests three major aspects of performance: how quickly a page looks like it loads (known as largest contentful paint), how quickly a user can interact with it (called first input delay) and whether the layout shifts, which as I mentioned earlier is something which really annoys people.

These metrics are built into PageSpeed Insights, Lighthouse and Web Page test, so they are a useful high-level set of metrics to use.

I mentioned testing from real devices is much better than just simulating a slow network connection. So if you have old phones you can put a pay as you go SIM card in that would be fantastic. Perhaps your organisation is willing to pay for a few popular cheap smartphones and tablet devices to use for testing. A ‘device lab’ like this can be shared between multiple teams.

Welcome to the wonderful world of Web Performance

Sitespeed.io is a set of Open Source tools that makes it easy to monitor and measure the performance of your web site.

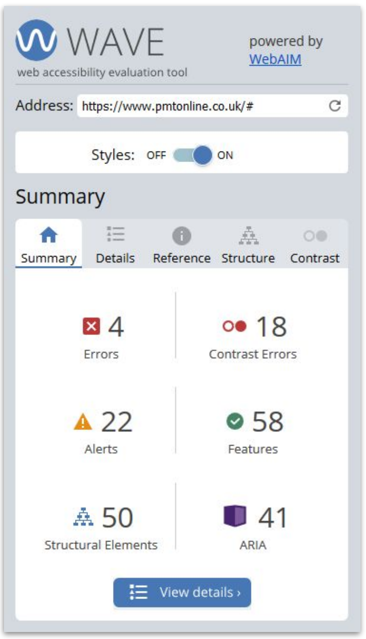
Measuring performance shouldn't be hard: you should be able to have full control of your metrics, own your own data and you should be able to do it without paying top dollars.

That's why we created sitespeed.io.

https://www.sitespeed.io/, https://webperformance.slack.com

Web performance is really easy to get into, there are loads of resources and tools to help you, and a thriving community on social media. There's even a Slack workspace at webperformance.slack.com where you can talk to other performance engineers. Sitespeed.io also has a great collection of tools and lots of guides to get you started. There's no excuse not to make your sites fast.

OK, that's a lot about testing performance. What about accessibility.


WAVE
web accessibility evaluation tool
powered by WebAIM

Address: https://www.pmtonline.co.uk/#

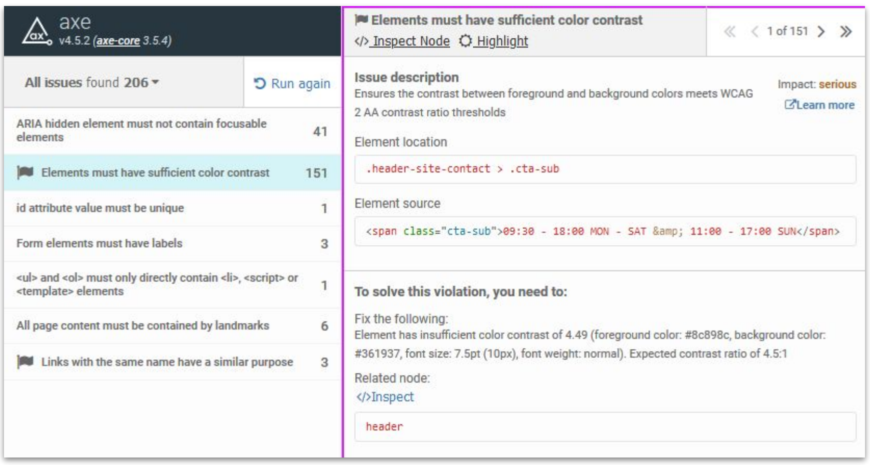
Styles: OFF ON

Summary

Summary | Details | Reference | Structure | Contrast

| | |
|---|---|
| 4 Errors | 18 Contrast Errors |
| 22 Alerts | 58 Features |
| 50 Structural Elements | 41 ARIA |

View details ›


axe
v4.5.2 (axe-core 3.5.4)

All issues found 206 ▾ | Run again

| | |
|---|---|
| ARIA hidden element must not contain focusable elements | 41 |
| Elements must have sufficient color contrast | 151 |
| id attribute value must be unique | 1 |
| Form elements must have labels | 3 |
| <ul> and <ol> must only directly contain <li>, <script> or <template> elements | 1 |
| All page content must be contained by landmarks | 6 |
| Links with the same name have a similar purpose | 3 |

Elements must have sufficient color contrast
</> Inspect Node  Highlight
1 of 151

Issue description
Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds
Impact: serious
Learn more

Element location
```
.header-site-contact > .cta-sub
```

Element source
```
<span class="cta-sub">09:30 - 18:00 MON - SAT &amp; 11:00 - 17:00 SUN</span>
```

To solve this violation, you need to:

Fix the following:
Element has insufficient color contrast of 4.49 (foreground color: #8c898c, background color: #361937, font size: 7.5pt (10px), font weight: normal). Expected contrast ratio of 4.5:1

Related node:
</>Inspect
```
header
```

https://wave.webaim.org/ , https://www.deque.com/axe/

First up has to be WAVE - the web accessibility evaluation tool. You can use this from the website linked here, or there are extensions for Chrome and Firefox. Also displayed here is the aXe extension from Deque. Both will highlight common accessibility issues.

NV Access

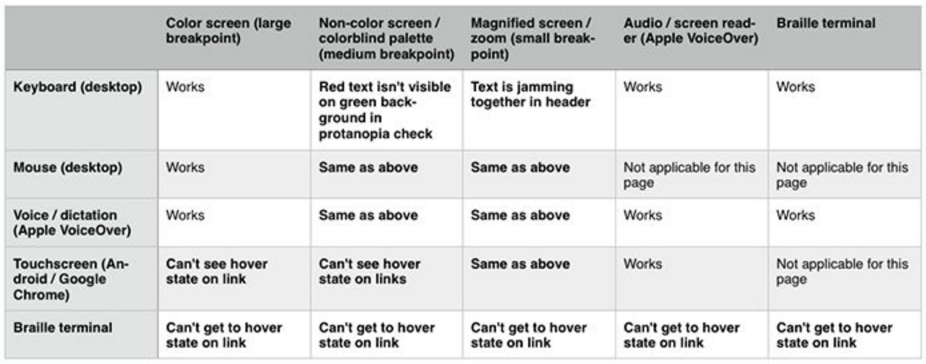
Empowering lives through non-visual access to technology

WE BELIEVE THAT every Blind + Vision Impaired person DESERVES THE RIGHT TO freely & easily access a computer!

https://www.nvaccess.org/

For a deeper look into possible accessibility problems you should test your pages using screenreader software. There are extensions for browsers, but it's also worth using one of the "real" ones - NVDA from NV Access is free and open source.

Screenreaders can be very daunting if you've never used one before. But persevere with them, they are incredibly useful. It's also worth trying to navigate and use your site using just a keyboard - that's easy to do and very enlightening.

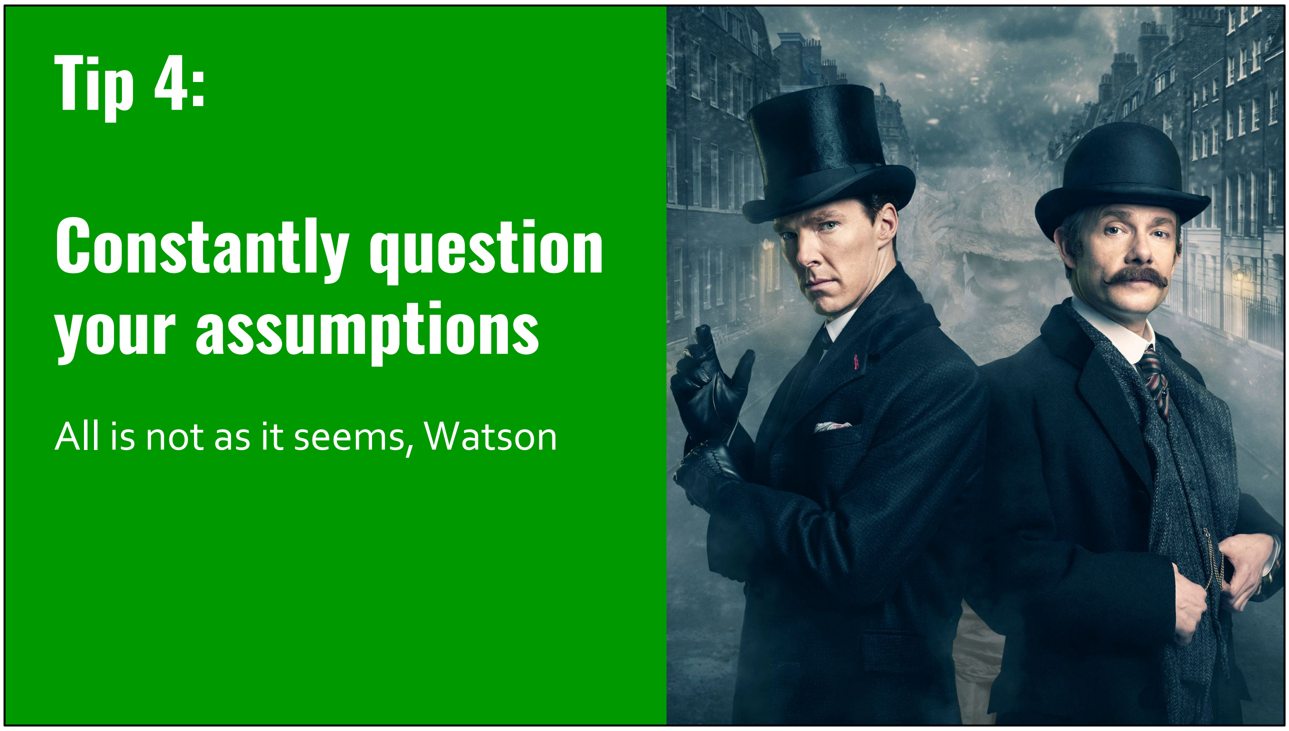


| | Color screen (large breakpoint) | Non-color screen / colorblind palette (medium breakpoint) | Magnified screen / zoom (small breakpoint) | Audio / screen reader (Apple VoiceOver) | Braille terminal |
|---|---|---|---|---|---|
| Keyboard (desktop) | Works | Red text isn't visible on green background in protanopia check | Text is jamming together in header | Works | Works |
| Mouse (desktop) | Works | Same as above | Same as above | Not applicable for this page | Not applicable for this page |
| Voice / dictation (Apple VoiceOver) | Works | Same as above | Same as above | Works | Works |
| Touchscreen (Android / Google Chrome) | Can't see hover state on link | Can't see hover state on links | Same as above | Works | Not applicable for this page |
| Braille terminal | Can't get to hover state on link | Can't get to hover state on link | Can't get to hover state on link | Can't get to hover state on link | Can't get to hover state on link |

https://alistapart.com/article/reframing-accessibility-for-the-web/

Finally, and most importantly, if you're serious about making your site accessible your should investigate testing with real users. People who use assistive technologies, who can give you the wisdom of their lived experience.

According to world-renowned accessibility experts the Paciello Group, automated accessibility tools will only catch about 30% of issues. Get serious about the other 70%.

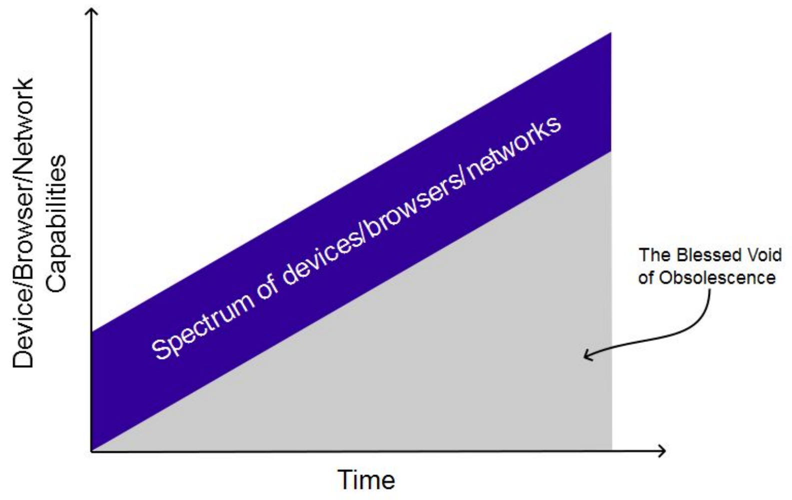


## Tip 4:

## Constantly question your assumptions

All is not as it seems, Watson

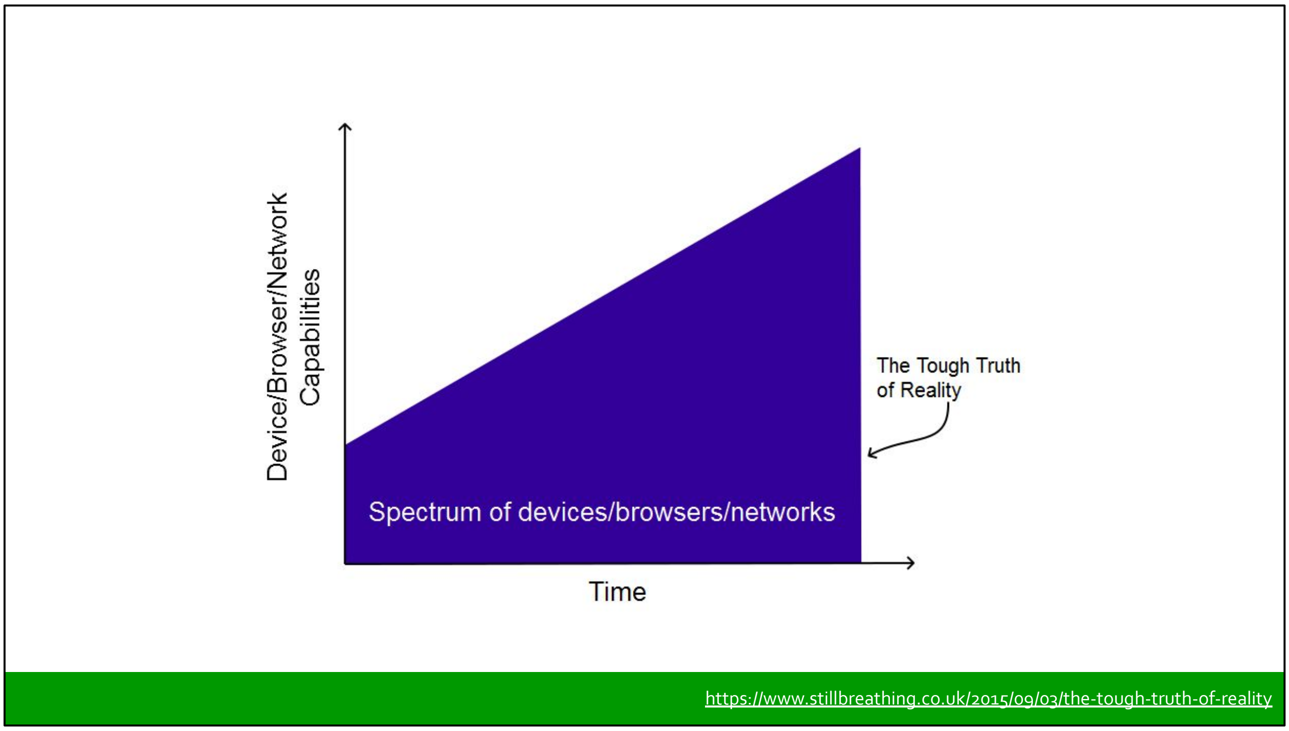
The next step is to question all your assumptions. Earlier we looked at assumptions about the network, but there are many other assumptions we make.



Device/Browser/Network Capabilities
Spectrum of devices/browsers/networks
The Blessed Void of Obsolescence
Time

https://www.stillbreathing.co.uk/2015/09/03/the-tough-truth-of-reality

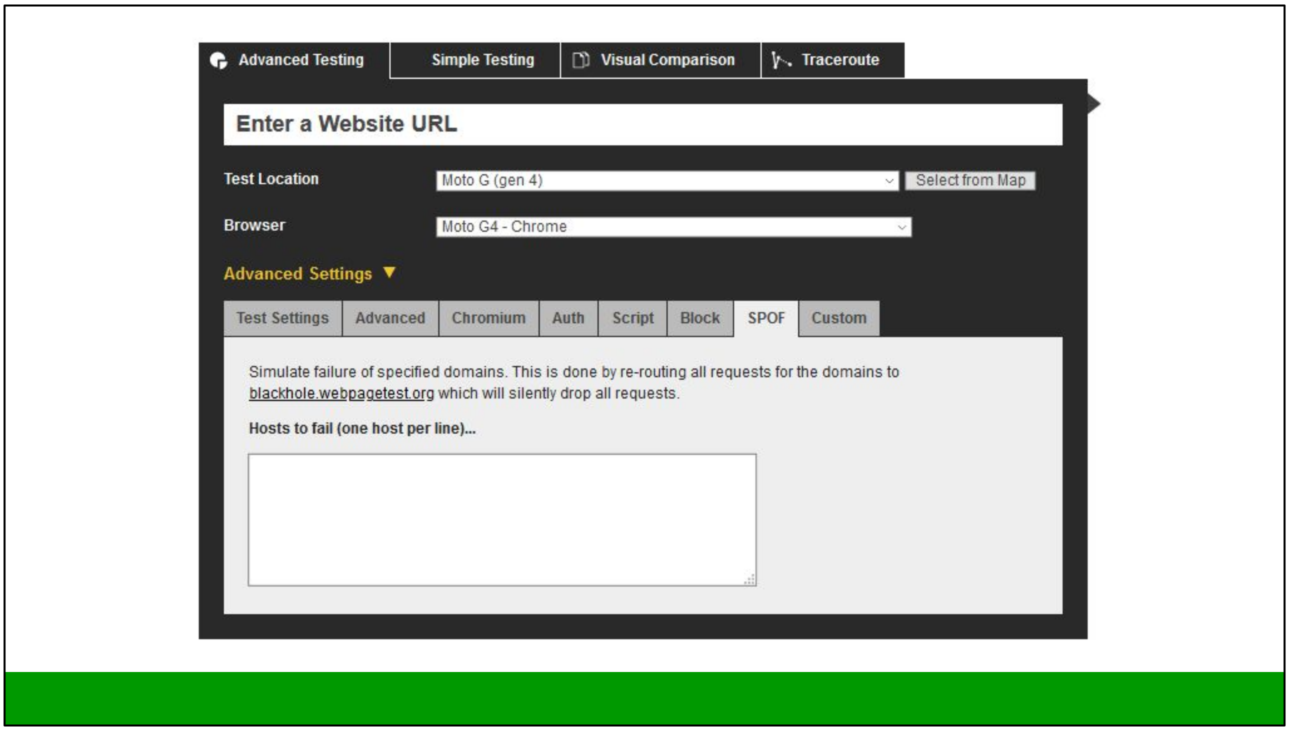
For example we can assume which browsers and devices people are using. As we've seen, the range of devices people may be using is huge. Developers often assume that over time the capabilities and quality of things like networks, browsers and devices increase - and they do. We believe that the old, outdated, slow stuff is left behind in what I've called the Blessed Void of Obsolescence.


Device/Browser/Network Capabilities
Spectrum of devices/browsers/networks
Time
The Tough Truth of Reality

https://www.stillbreathing.co.uk/2015/09/03/the-tough-truth-of-reality

But the tough truth of reality is that we never really leave the old stuff behind. For example, as we've seen, cheap new devices get released which have the same performance profile of top-end devices from nearly a decade ago.

You should check the analytics for your site regularly see whether the assumptions you make about devices, browsers, screen sizes, and more are still valid.



Test Location: Moto G (gen 4)

Browser: Moto G4 - Chrome

Simulate failure of specified domains. This is done by re-routing all requests for the domains to blackhole.webpagetest.org which will silently drop all requests.

Hosts to fail (one host per line)...

Web Page Test can help with one other assumption - that 3rd parties can be relied on. When you start a test, under the ‘advanced’ section there’s a tab called ‘SPOF’ which stands for ‘single point of failure’. Here you can simulate what would happen if certain domains - for example your CDN, or a 3rd party you rely on - were to fail. Yes, this is your very own chaos monkey.

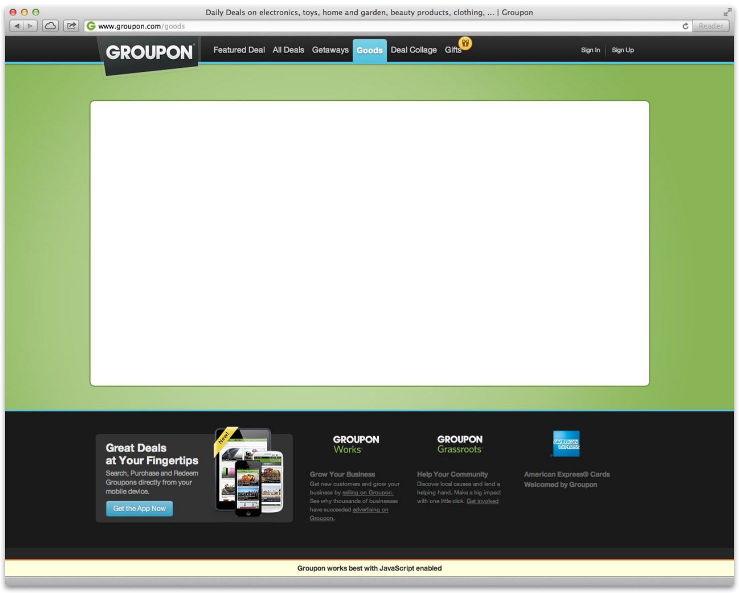
Chaos monkey, if you didn’t know, is something invented by Netflix to test how resilient their systems are. Basically it’s a bot which randomly turns services off to see whether Netflix as a whole keeps running. This feature in Web Page Test allows you to do that as well.

## Tip 5:

## Do you really need all that JavaScript?

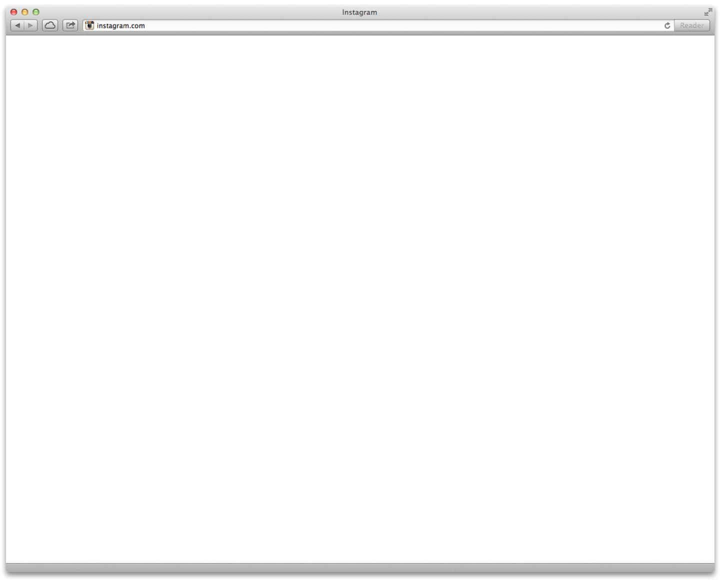
Say when…

The next tip is the least fashionable suggestion I’ll make. JavaScript is cool, but sometimes you don’t need it - or at least not so much of it, as we saw with the coronavirus dashboard earlier. Some sites seem to lose sight of the content they are actually serving up. For example, how about a site that shows coupons - pretty static content, right?


Daily Deals on electronics, toys, home and garden, beauty products, clothing, ... | Groupon
www.groupon.com/goods
GROUPON
Featured Deal
All Deals
Getaways
Goods
Deal Collage
Gifts
Sign In
Sign Up
Great Deals at Your Fingertips
Search, Purchase and Redeem Groupons directly from your mobile device.
Get the App Now
GROUPON Works
GROUPON Grassroots
Grow Your Business
Help Your Community
American Express® Cards Welcomed by Groupon
Groupon works best with JavaScript enabled

http://sighjavascript.tumblr.com/

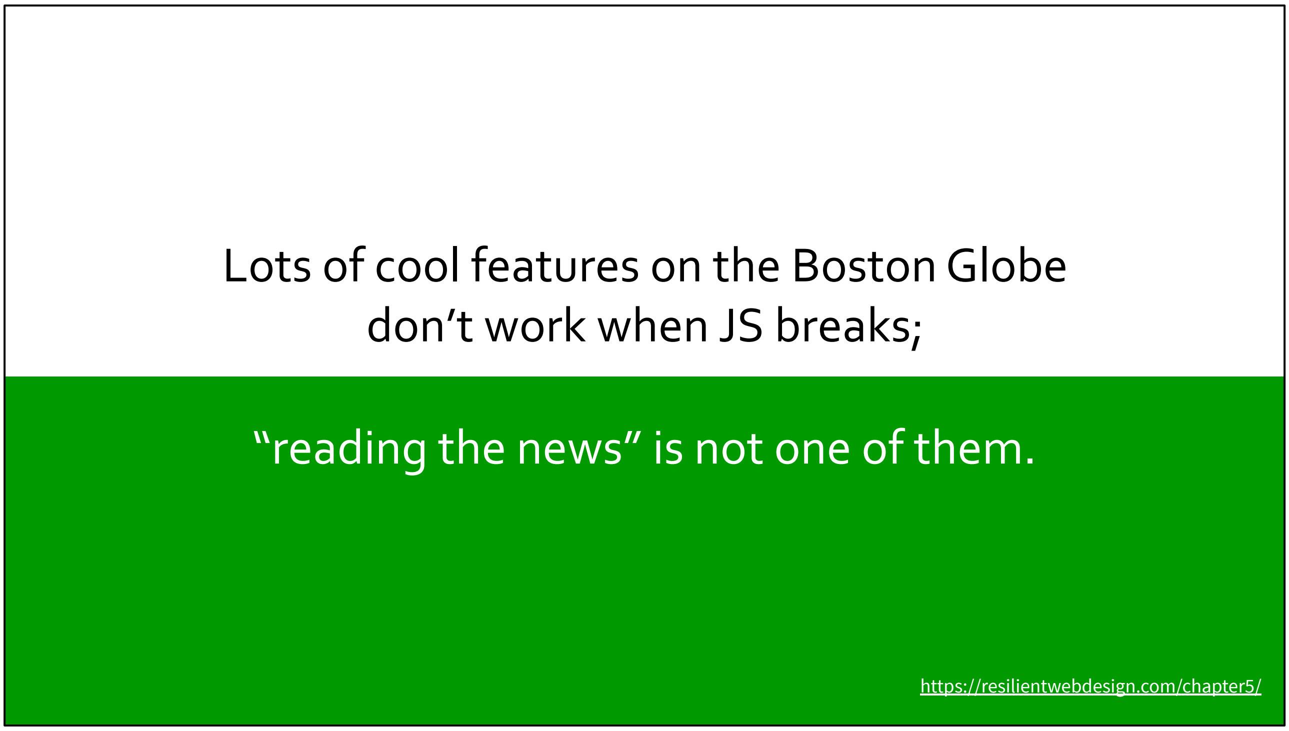
Except when the JavaScript fails you get only this beautiful template. How about an image gallery like instagram? That’s got to give the user _something_ if the script fails, right?



http://sighjavascript.tumblr.com/

Nope, nothing. Don’t get me wrong, I’m not saying JavaScript is bad - but relying on it too much can lead to negative outcomes for users.

Several years ago a team was put together to rebuild a newspaper site in the US called the Boston Globe. It was one of the first big responsive designs. One of the team members said this about how they used JavaScript.



## Lots of cool features on the Boston Globe don’t work when JS breaks;

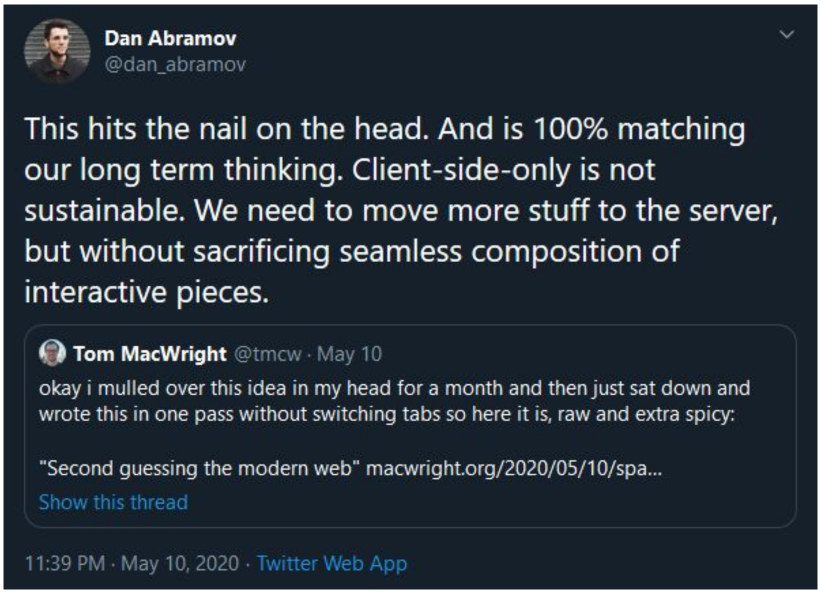
“reading the news” is not one of them.

https://resilientwebdesign.com/chapter5/

This, for me, is a pragmatic approach. The core functionality of that site is to show people the news, so the team made that functionality as resilient as it could be. Other features were considered enhancements - they aren’t the core functionality, so even if they break the user still gets what they visited the site for.

This quote, by the way, is from the book ‘Resilient Web Design’ by Jeremy Keith. It’s a fantastic book, not just for the content but because it’s online, free, and it’s a progressive web app. The book practices what it preaches.

This isn’t a fashionable view, but it’s not just me saying that a total dependence on JavaScript needs to be considered very carefully. Other developers have the same opinion.


Dan Abramov
@dan_abramov

This hits the nail on the head. And is 100% matching our long term thinking. Client-side-only is not sustainable. We need to move more stuff to the server, but without sacrificing seamless composition of interactive pieces.

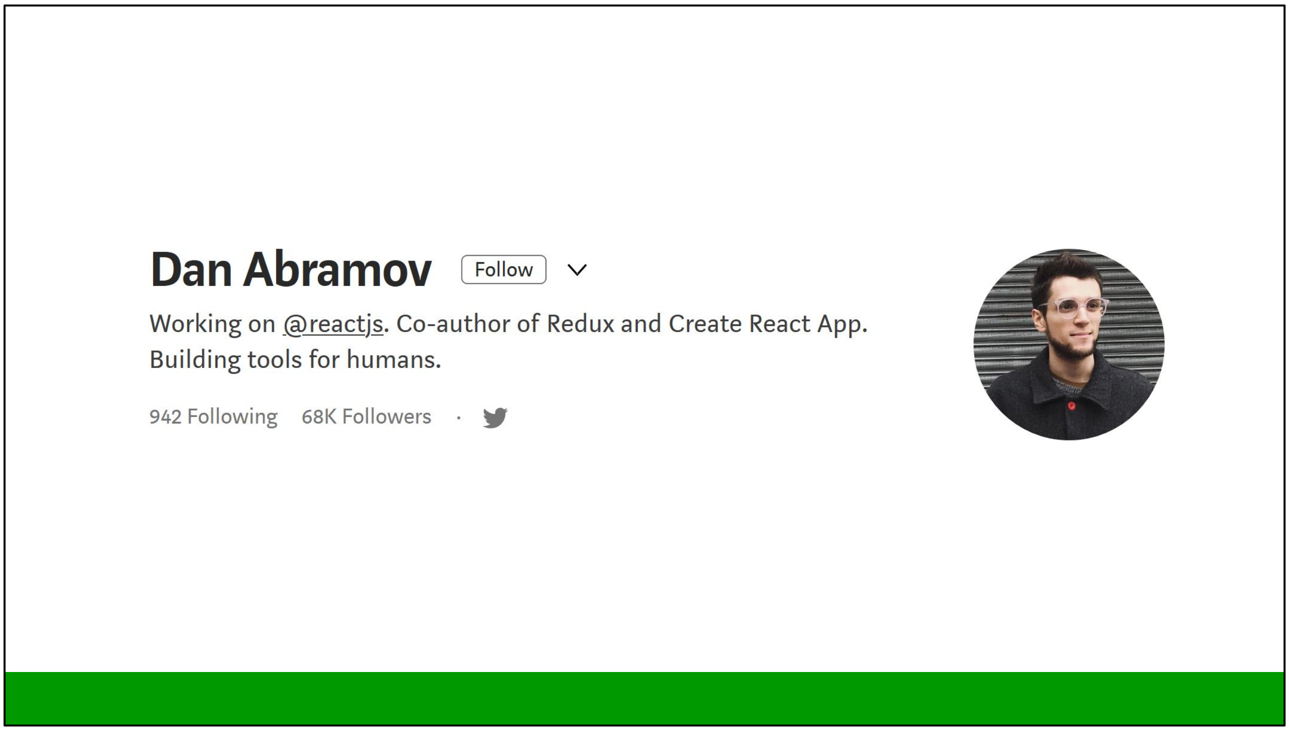
Tom MacWright @tmcw · May 10
okay i mulled over this idea in my head for a month and then just sat down and wrote this in one pass without switching tabs so here it is, raw and extra spicy:

"Second guessing the modern web" macwright.org/2020/05/10/spa...
Show this thread

11:39 PM · May 10, 2020 · Twitter Web App

https://mobile.twitter.com/dan_abramov/status/1259614150386425858

For example Dan Abramov, who recently said this: “Client-side only is not sustainable”. Who is Dan Abramov?


Dan Abramov

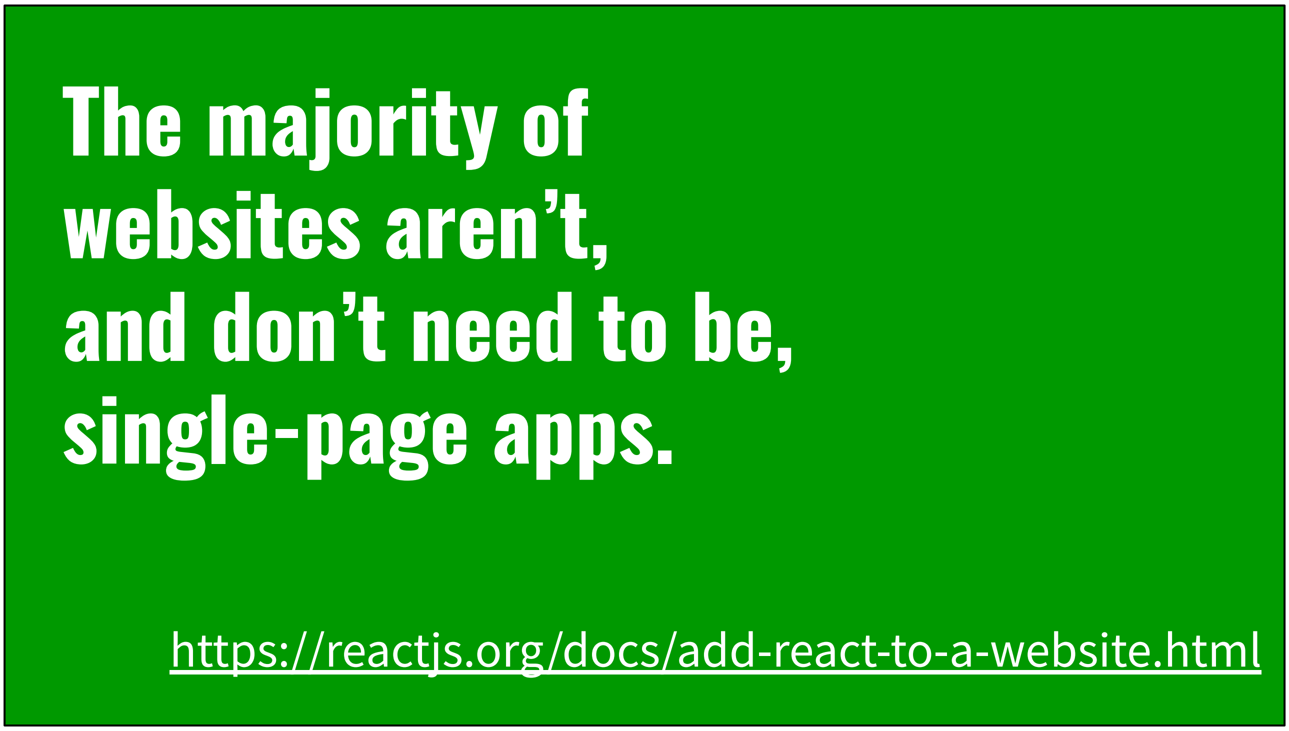
Working on @reactjs. Co-author of Redux and Create React App. Building tools for humans.

942 Following 68K Followers ·

The co-author of redux and create-react-app, who works on the react team.

## The majority of websites aren't, and don't need to be, single-page apps.

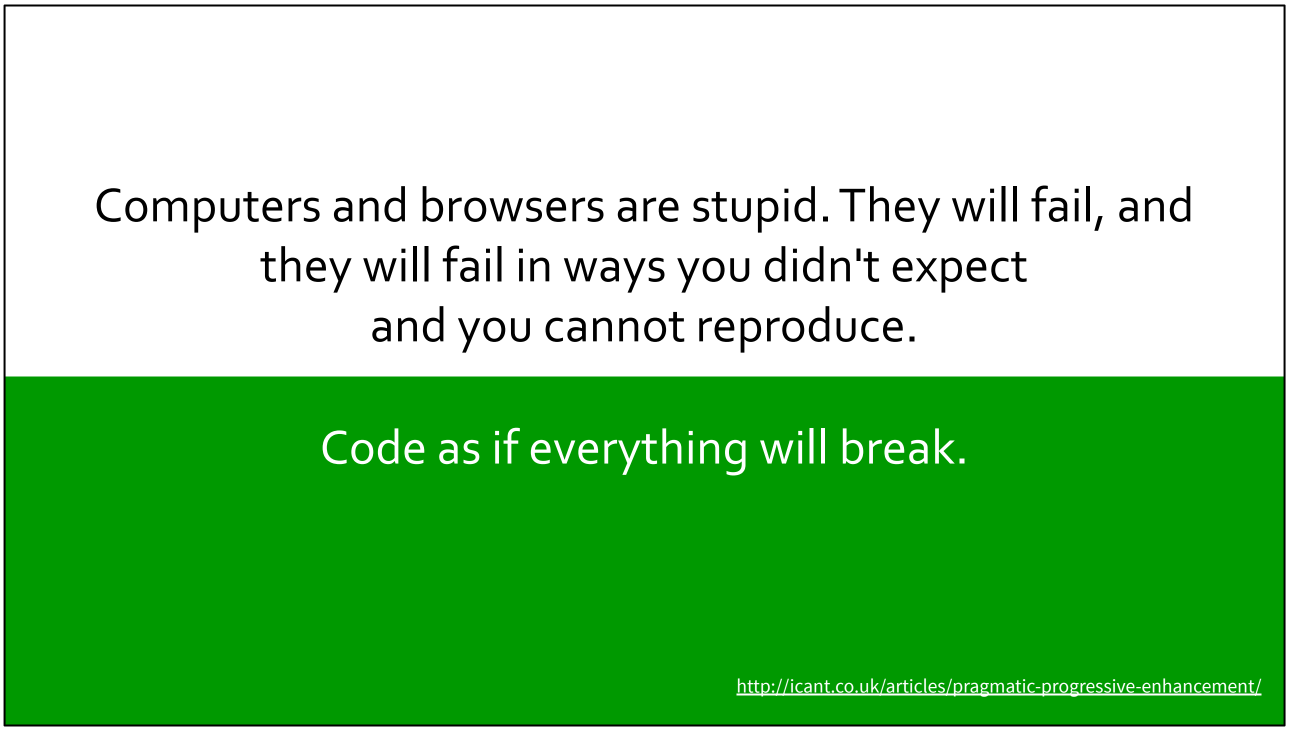
And not just Dan and me. Guess where you'll find this phrase?



## The majority of websites aren’t, and don’t need to be, single-page apps.

https://reactjs.org/docs/add-react-to-a-website.html

On the react site.

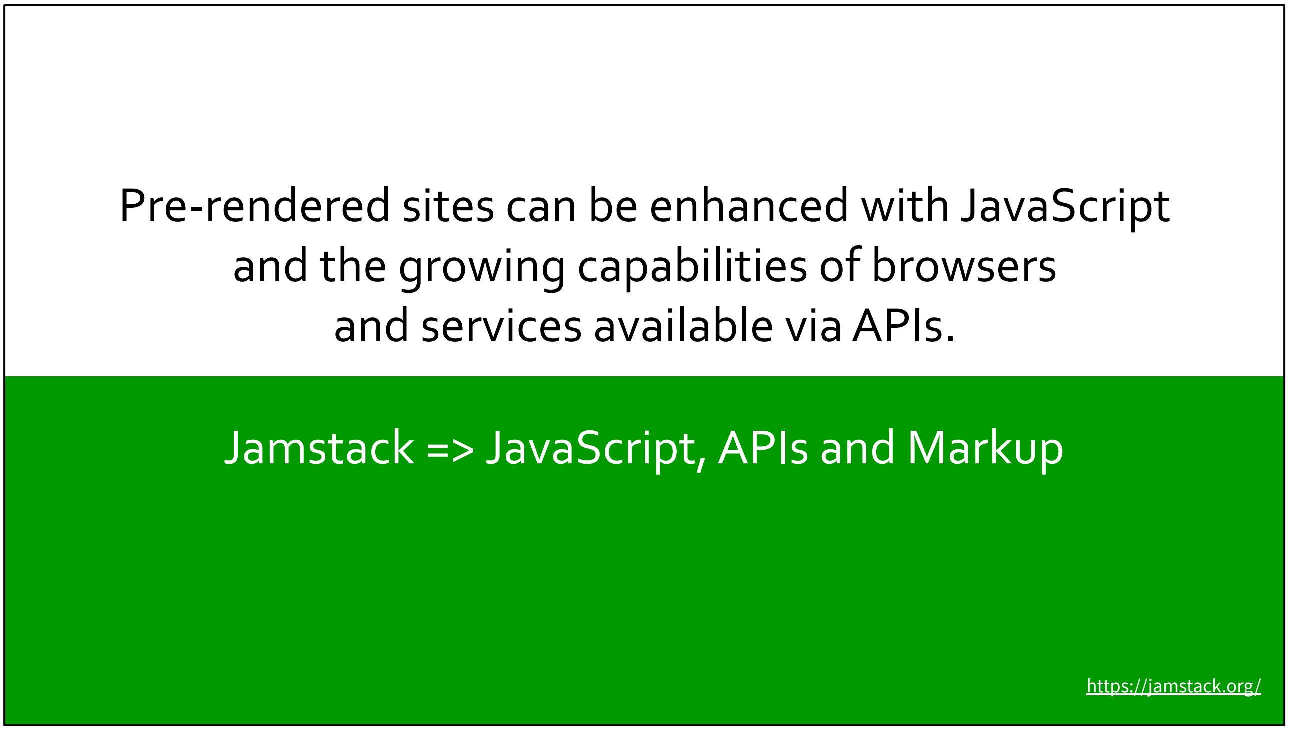


Computers and browsers are stupid. They will fail, and they will fail in ways you didn't expect and you cannot reproduce.

Code as if everything will break.

http://icant.co.uk/articles/pragmatic-progressive-enhancement/

If we're focussed on ensuring that our users are successful in what they're trying to do - read some content, fill in a form, search for some information - then we should code to make that successful outcome as likely as possible. Where appropriate, JavaScript is a great tool. But where it's not needed it can cause problems for users that could be avoided.



Pre-rendered sites can be enhanced with JavaScript
and the growing capabilities of browsers
and services available via APIs.

Jamstack => JavaScript, APIs and Markup

https://jamstack.org/

There's an approach gaining in popularity called Jamstack - that stands for JavaScript, APIs and markup. Essentially it describes pre-rendered information that is enhanced with judicious use of JavaScript. So you get all the performance and stability of server-rendered HTML, and all the whizz-bang of client-side interactivity.

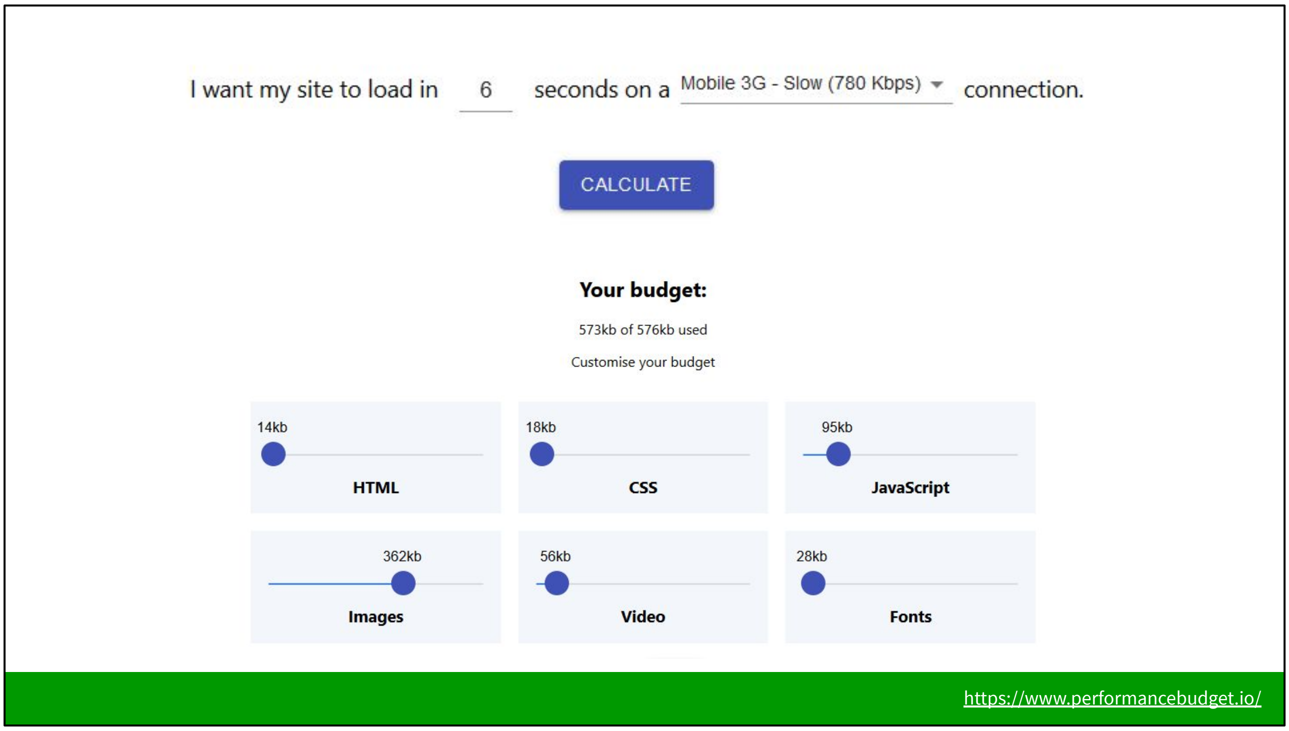
And when this is combined with a service worker, which can provide advanced caching and offline support - not to mention the ability for your site to be "installed" to a device - it's a compelling approach.

## Tip 6:

## Create a performance culture

My final tip is to create a performance culture. In the same way we implement CI/CD pipelines and automate tests to catch regression bugs at code level, we should do the same at user experience level.

All the tools I mentioned earlier help with this, but nothing beats getting close to the experience real users have with our sites. That may be through a real user monitoring system, or web analytics - these are often owned by a marketing department, so you may need to bridge that gap.



I want my site to load in 6 seconds on a Mobile 3G - Slow (780 Kbps) connection.

**Your budget:**

573kb of 576kb used

Customise your budget

| | |
|---|---|
| HTML | 14kb |
| CSS | 18kb |
| JavaScript | 95kb |
| Images | 362kb |
| Video | 56kb |
| Fonts | 28kb |

https://www.performancebudget.io/

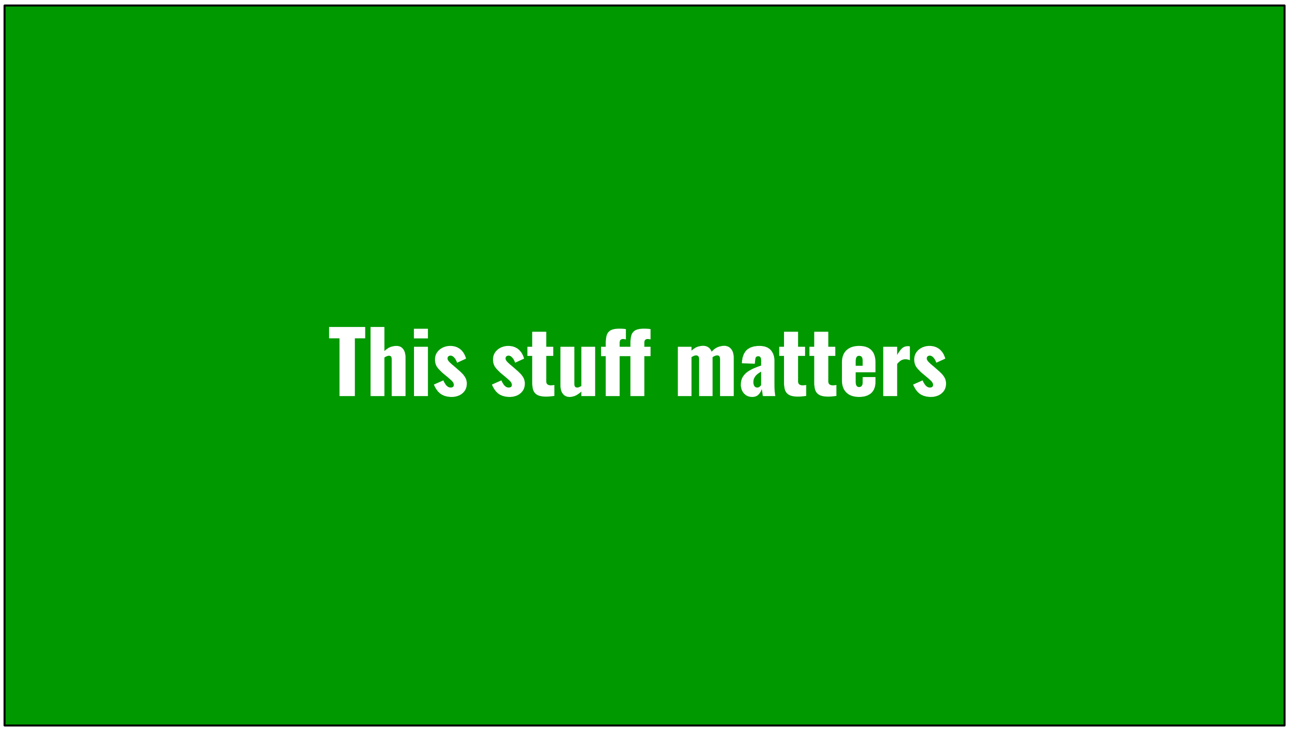
One great way to start is to create a performance budget. This defines upper limits for asset sizes and various metrics to ensure you keep your pages fast. The performancebudget.io tool helps you craft a budget suitable for your site - but be warned, you'll have MUCH less to play with than you would like.

Often, the secret of the most successful digital companies in the world is their obsession with their customers.

https://gerrymcgovern.com/

Gerry McGovern is a master at helping organisations focus on what customers actually want. If you visit his website linked here you can watch a selection of his talks, I guarantee you'll be glad you did.

Gerry often talks about 'top tasks' - the things that visitors most often want to accomplish on our sites. Perhaps it's find a product, or download some information, or make an insurance claim. Whatever those top tasks are, we should strive to remove as many barriers as possible which stop the user achieving their goal. Resilience of the site, ensuring it doesn't fail the user when it encounters unexpected conditions, is a big part of that.

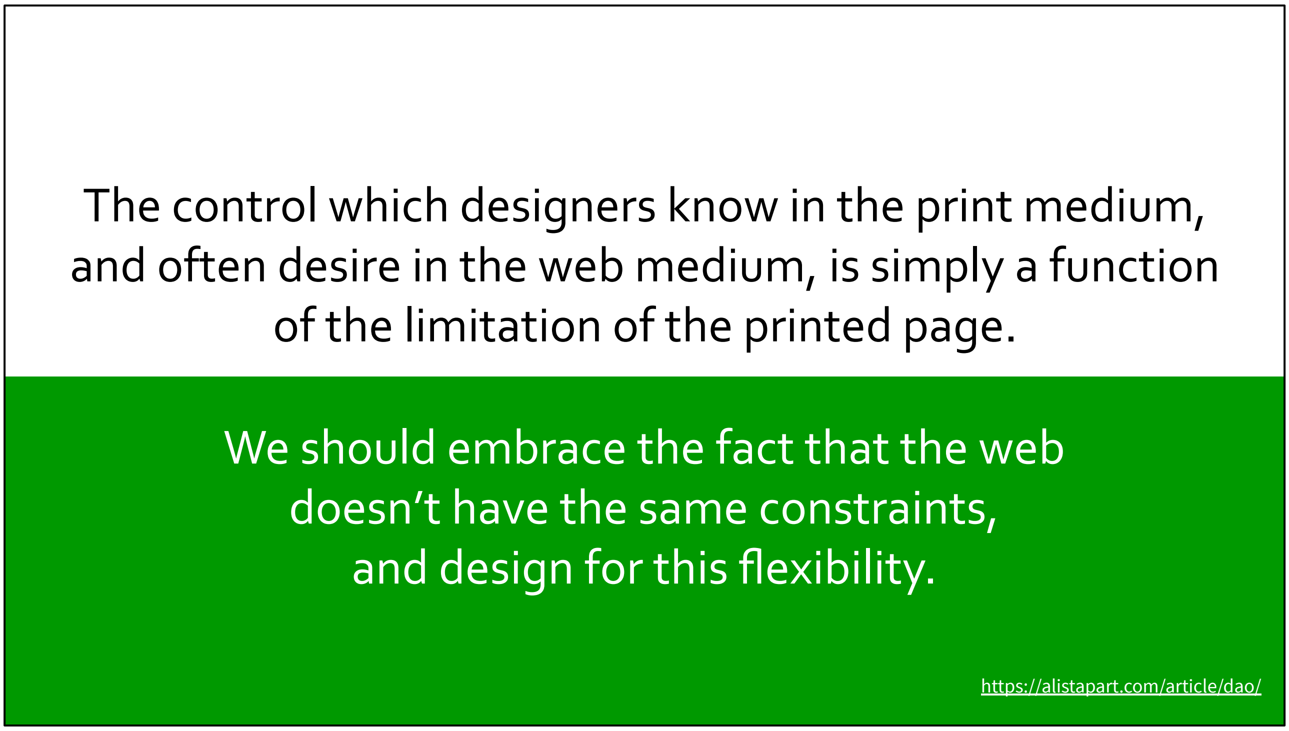


## This stuff matters

Because this stuff - performance, accessibility, this resilience I've talked about - matters. It matters to our users and customers, and when they are happy we'll get better outcomes from the systems we build.

Back in April 2000 an article was published by John Allsop on the A List Apart magazine. Called 'A Dao of Web Design' it was a call to understand and design for the web medium - which at the time was still fairly young. The article is incredibly visionary, alluding to responsive design ten years before Ethan Marcotte coined the term, and encouraging us to think about different browsers, platforms and screens - six years before the first iPhone revolutionised where the web could be used.

In that article John wrote this:

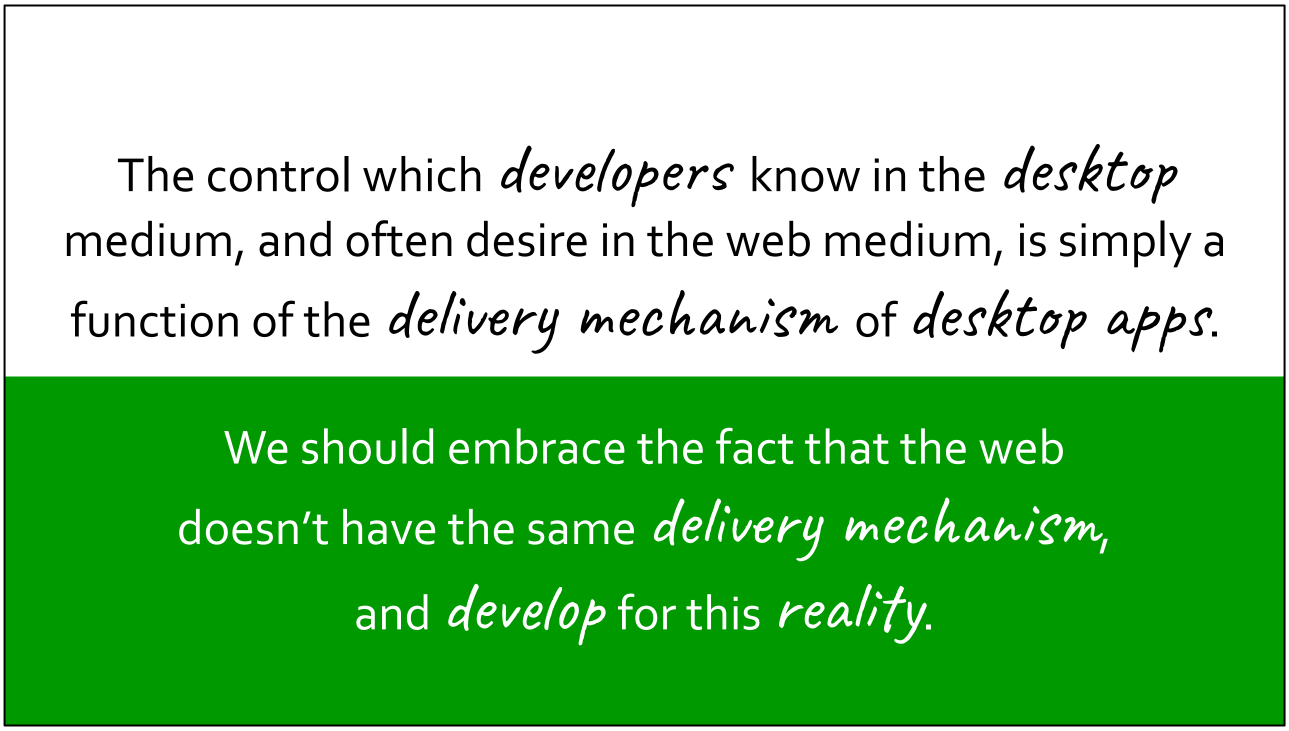


The control which designers know in the print medium, and often desire in the web medium, is simply a function of the limitation of the printed page.

We should embrace the fact that the web doesn't have the same constraints, and design for this flexibility.

https://alistapart.com/article/dao/

While this is absolutely true, I think we can paraphrase that to be more suitable for web developers, like this:



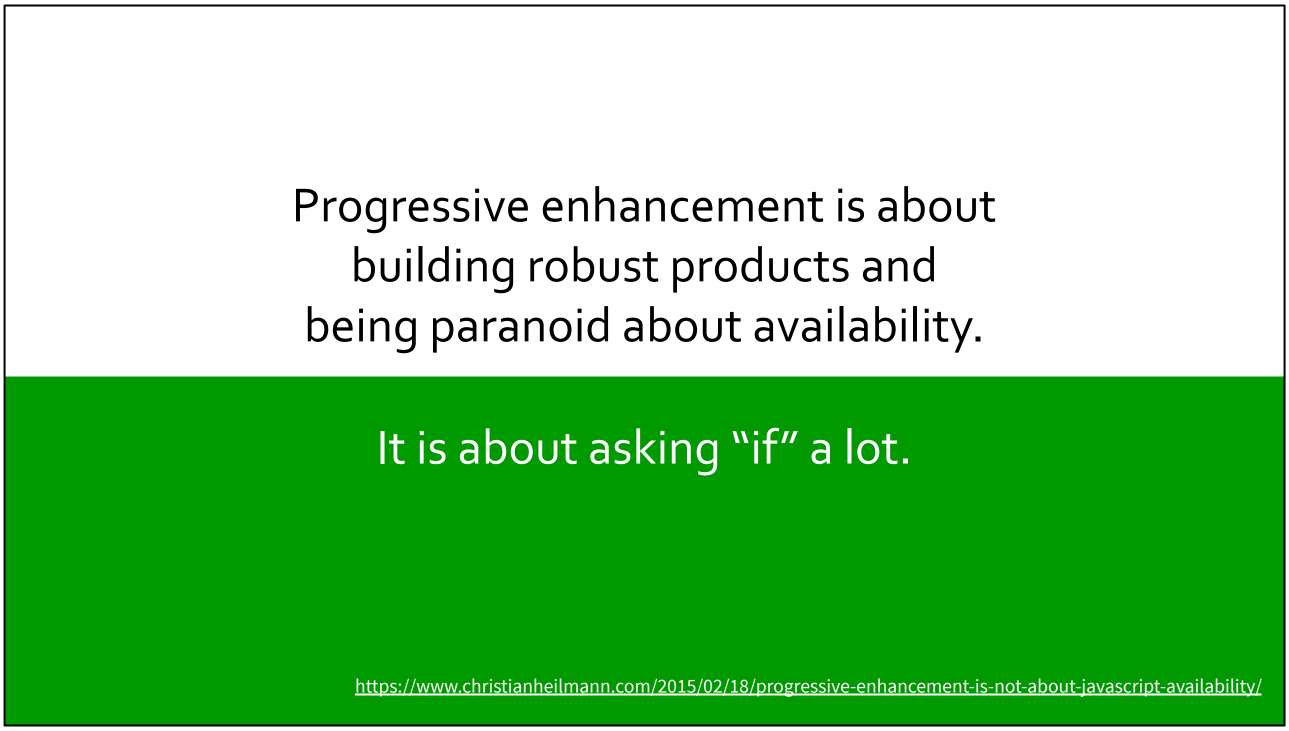
The control which *developers* know in the *desktop* medium, and often desire in the web medium, is simply a function of the *delivery mechanism* of *desktop apps*.

We should embrace the fact that the web doesn't have the same *delivery mechanism*, and *develop* for this *reality*.

If you remember back 3 or 4 weeks ago when I started this presentation, I talked about an 'all or nothing' approach, which is what you get with desktop apps and with plugins like Flash and Java applets.

The web wasn't designed to work like that, and we've seen that the layering of different technologies to build a complete page - HTML, CSS, JavaScript - means there are many potential points of failure. But we can and should try to make our sites resilient to these unexpected conditions, and not fail our users and customers.

There's a name for this mindset of building websites: progressive enhancement.



Progressive enhancement is about building robust products and being paranoid about availability.

It is about asking “if” a lot.

https://www.christianheilmann.com/2015/02/18/progressive-enhancement-is-not-about-javascript-availability/

It means asking “if” a lot. If this file is missing, if this API call fails, if the browser doesn’t execute this script correctly - what then. It’s only when we consider those things, question our assumptions, that we can truly make resilient websites.

expect

the

unexpected

We as developers need to expect the unexpected on the web.

## Thank you

Let’s go fix it!