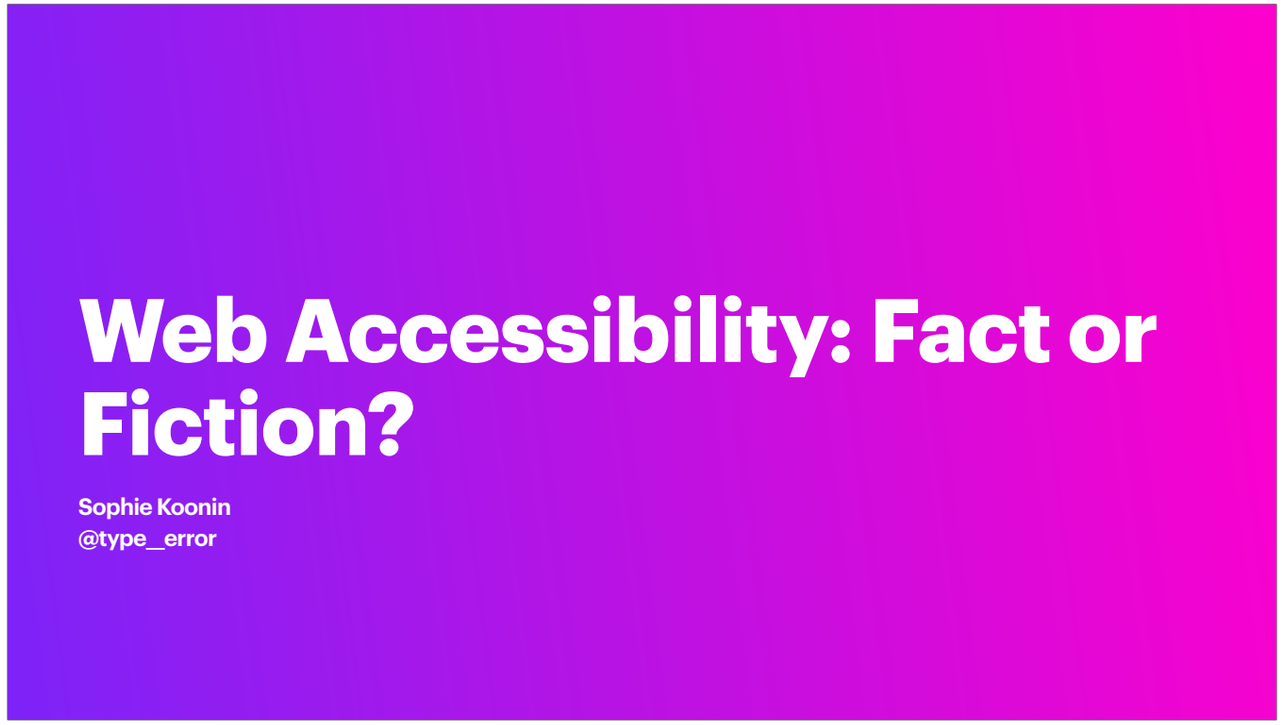


Some useful resources:

The original blog post this talk is based on - https://localghost.dev/2020/10/7-myths-designers-and-developers-believe-about-web-accessibility/

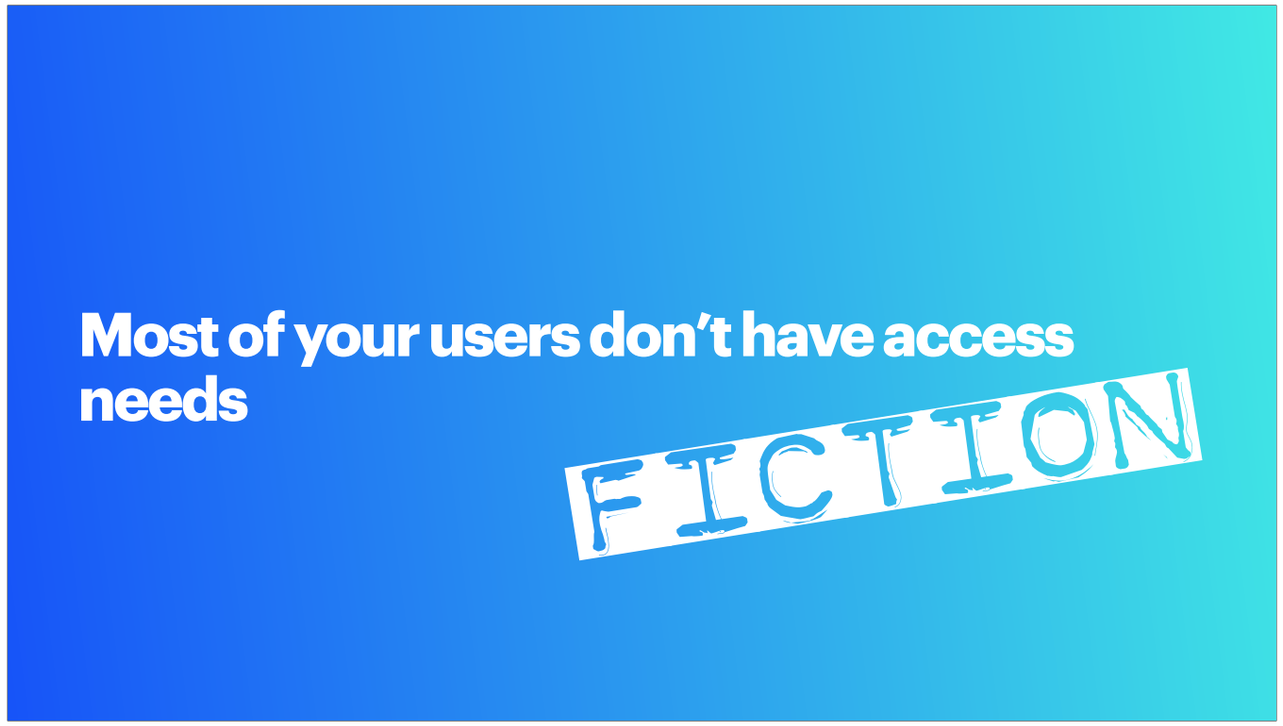
Microsoft Inclusive Design Toolkit - https://www.microsoft.com/design/inclusive/

7 things every designer needs to know about accessibility - https://medium.com/salesforce-ux/7-things-every-designer-needs-to-know-about-accessibility-64f105f0881b

Semantic HTML: what and why - https://www.aleksandrhovhannisyan.com/blog/semantic-html-accessibility/

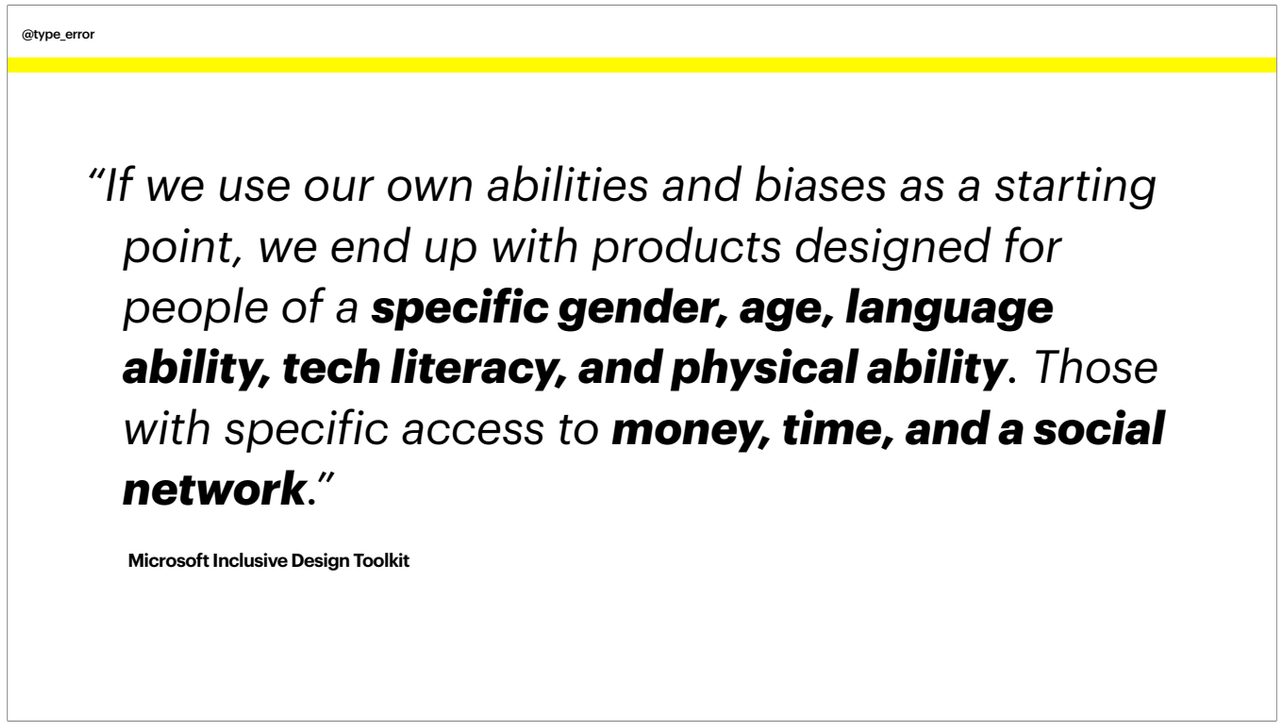
Marcy Sutton - https://marcysutton.com/ - brilliant accessibility expert

What we found when we tested tools on the world's least accessible webpage - Accessibility in Government Blog - https://accessibility.blog.gov.uk/2017/02/24/what-we-found-when-we-tested-tools-on-the-worlds-least-accessible-webpage/



This is something that’s often used to excuse not taking accessibility into account when designing or building a product.
“What’s the point of making all these adjustments for the sake of accessibility, if only a tiny proportion of our customers actually use screenreaders or whatever?”

It’s easy to assume that your users are just like you (and I use ‘you’ generally here). You might not even think about it. You build a website, you think “yeah, that’s recognisably a menu icon. Yeah, that animation is a cool thing to play when you land on the site for the first time.” and for people like you, it probably is.



My own abilities, experiences and knowledge shape how I interact with a website, and my experience may be totally different from yours. So even if you know instinctively that a particular icon means “download”, others might not make that connection.

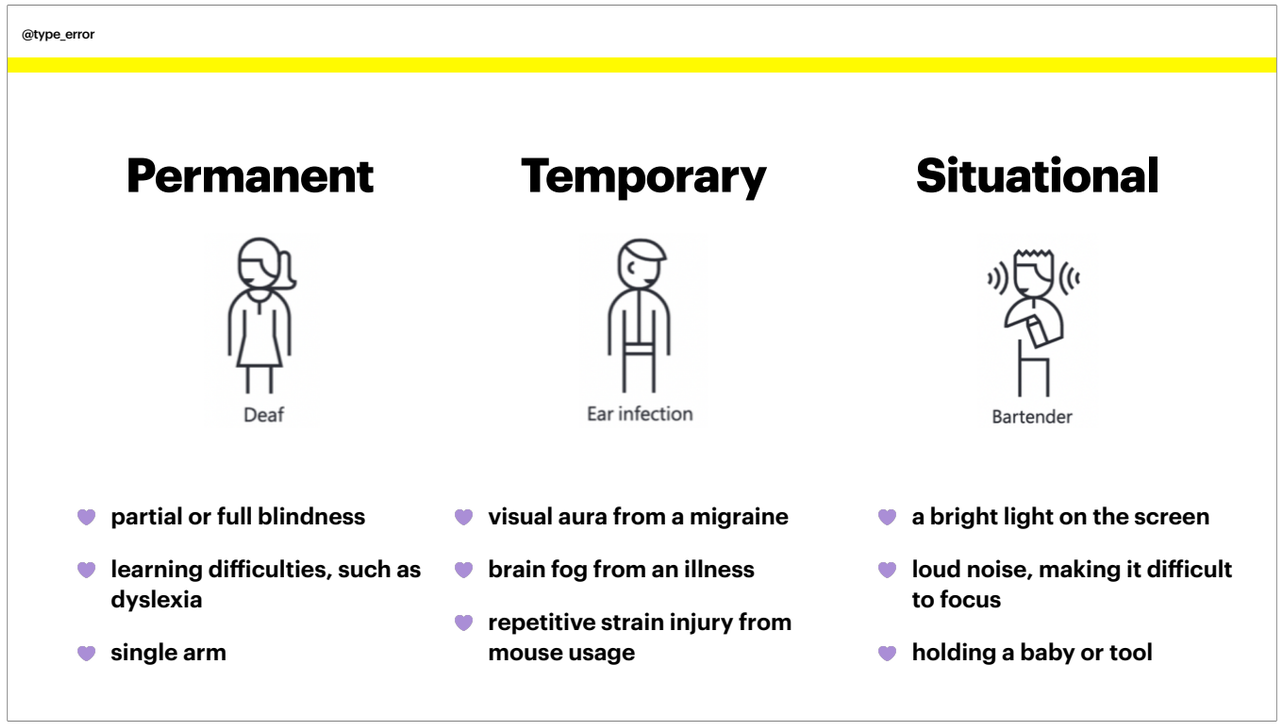
I have fibre internet, I have a nice macbook, I have a university degree and access to a lot of valuable resources. But the people accessing the apps I’m building aren’t all going to be in the same boat.

@type_error

> *“When it comes to people, there’s* ***no such thing as “normal.”*** *The interactions we design with technology depend heavily on what we can see, hear, say, and touch. Assuming all those senses and abilities are fully enabled all the time creates the potential to* ***ignore much of the range of humanity****.”*
>
> **Microsoft Inclusive Design Toolkit**

and they go on to say…

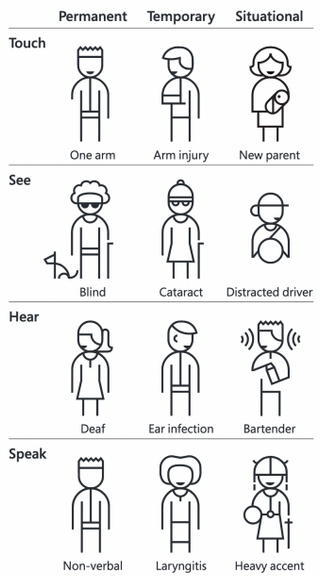
It’s important to remember that access needs can cover a whole host of things. Often when we think about web accessibility we think of blind people using screenreaders, but there’s so much more to it.



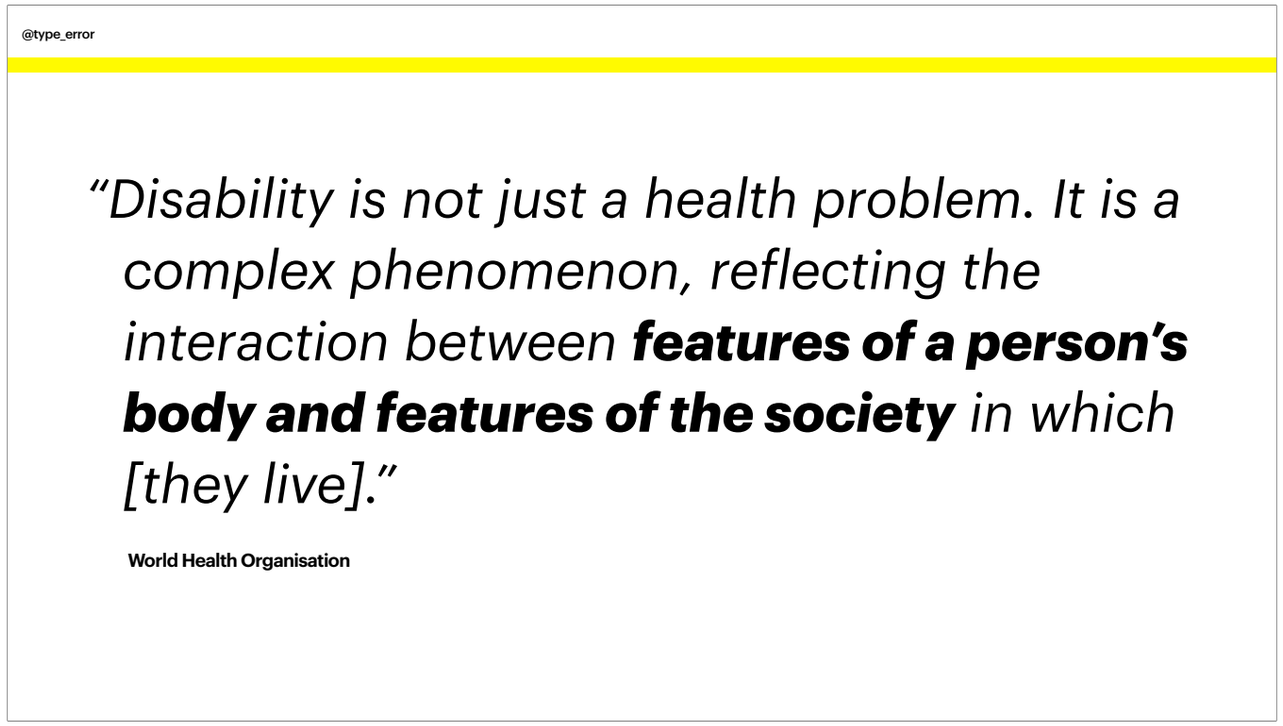
The Microsoft Inclusive Design toolkit identifies three categories of disability: permanent, temporary and situational. Temporary impairments might be due to a medical condition, and situational impairments may result from the environment around us - a situation we're in.

Even if you consider yourself not to have any form of disability, you could find yourself with a temporary or situational impairment at any point.

@type_error



Here’s a great illustration from the Inclusive Design Toolkit of some of the kinds of permanent, temporary, and situational impairments people might have.



The important thing to note is that it doesn’t necessarily translate to some kind of personal health condition.

(read quote)

For example, we don’t tend to think of people who need glasses as having a disability, because society has accommodated them. Disability is caused by the society we live in not accommodating a person’s access needs - it’s society that is disabling the person. Imagine if we accommodated all access needs as well as we did long or short-sightedness.

It is indeed the law here in the UK.

Equality Act 2010

In the UK, the equality act 2010 says: A person ... concerned with the provision of a service to the public or a section of the public (for payment or not) must not discriminate against a person requiring the service by not providing the person with the service.

So, for example, having an inaccessible website and therefore not providing a service to someone with a disability because they can’t use it would be a breach of this law.

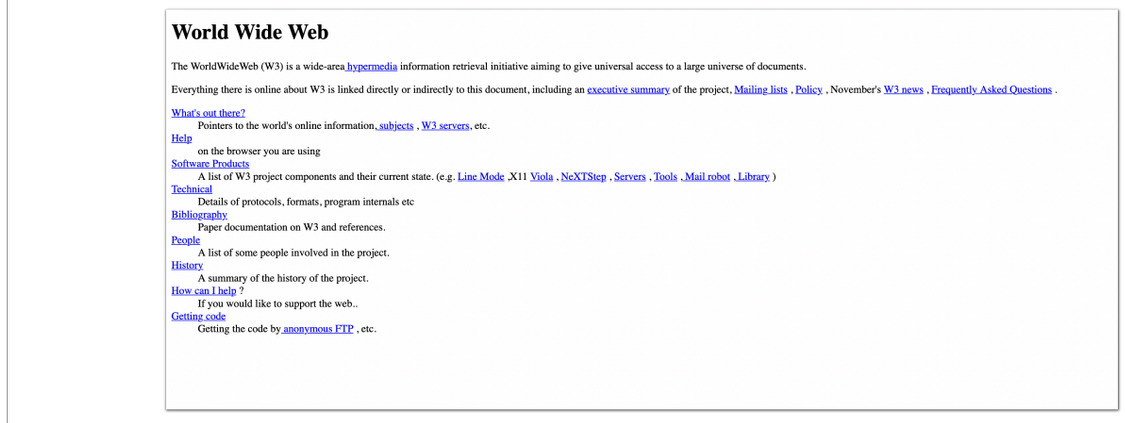
> *"[An organisation] is responsible for ensuring that **reasonable adjustments** have been made where needed, for example by changing the size of the font, to ensure that **disabled users are able to get the information**, without being placed at a substantial disadvantage (even if the [organisation] employs an **external organisation** to build and maintain its website)."*
>
> **Equality and Human Rights Commission**

The Equality Act doesn't explicitly mention websites, but the Equality and Human Rights Commission's Code of Practice for the Act goes into bit more detail.

The EHRC may:

Conduct formal investigations
Serve non-discrimination notices
Act over persistent discrimination
Help someone prosecute a company - this requires someone to actually sue the company, which there's no case law for at the moment - but that could change any time.

Here’s one I hear a lot. And when I hear it from senior designers it makes me sad.



World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

It’s easy to imagine that in order for a website to be really accessible, it has to look something like this. That’s really not the case.

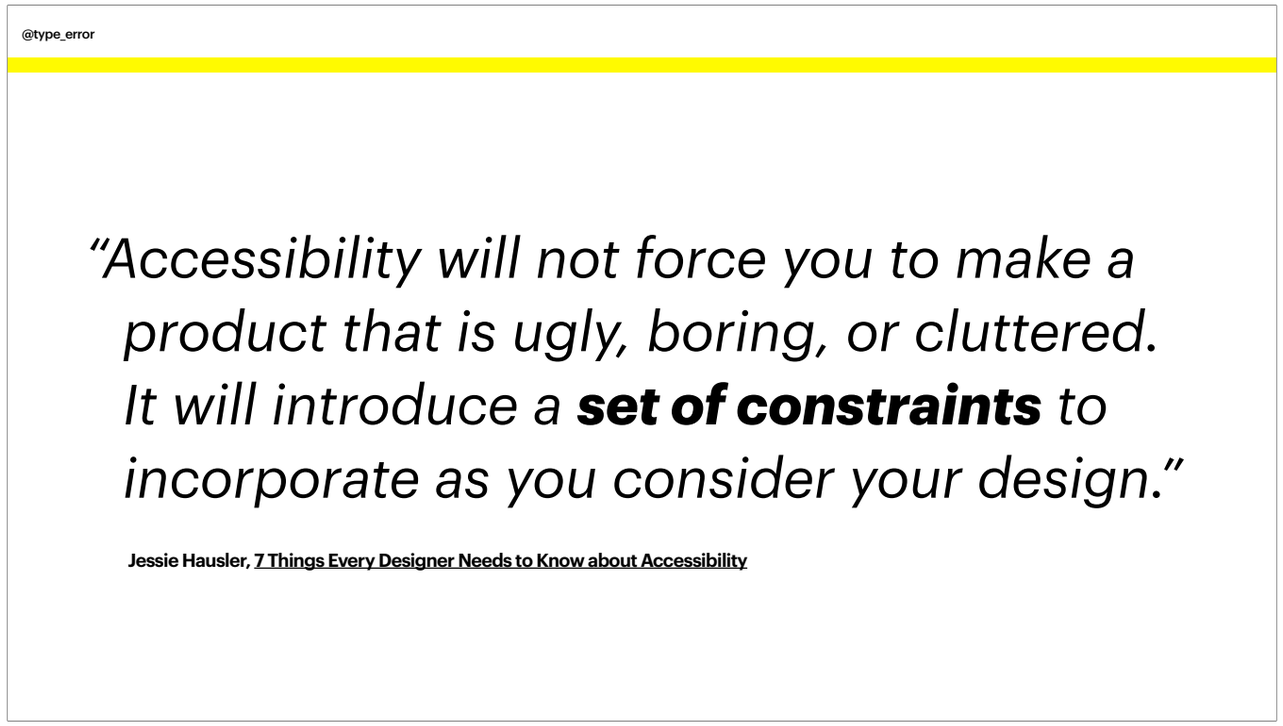
@type_error

Accessible design is good design.

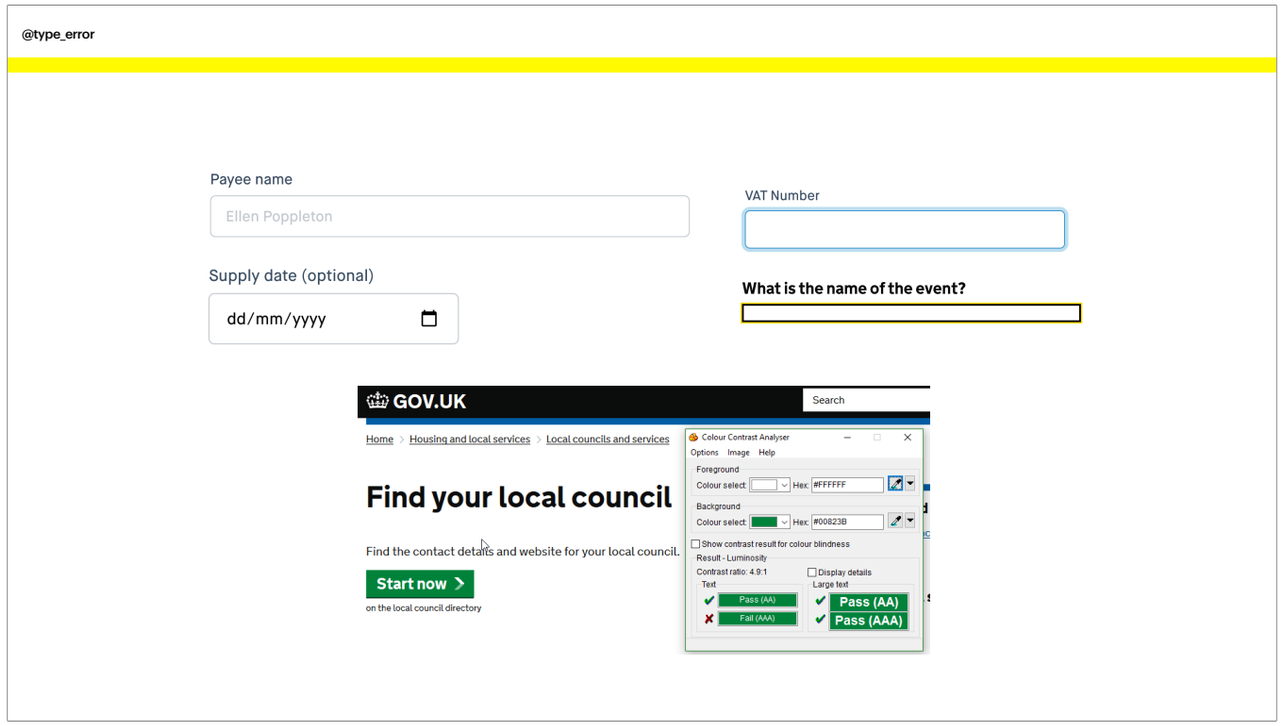
(AND VICE VERSA)

In reality, accessible design IS good design. Is a design really good if it doesn't work for everyone?

Web accessibility is everyone's job. It's not just the engineer's job, it starts at the design level. It's crap to have to be the engineer who goes back to the designer and says "we can't do this because it's not accessible". If your designer doesn't have the knowledge about accessibility yet, work with them to come up with something that works for everyone.

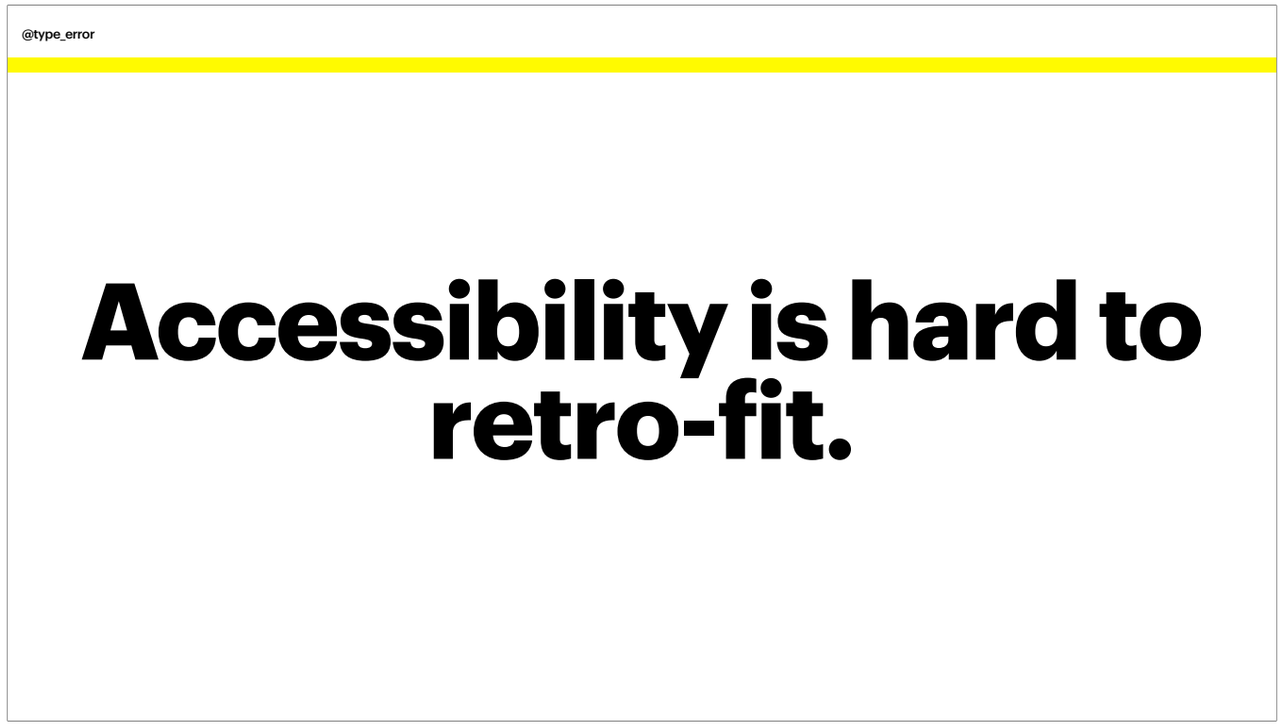


I recommend sharing this article with your designer colleagues as well.



Good designs should factor in things like:
form labels
identifying that things are optional, rather than identifying what's required
focus states for links, buttons and inputs
dimensions in rem or other standardised units rather than pixels
colour contrast between text and background
avoiding potentially distracting animations or making them manually triggered


Web Accessibility is hard to implement
FICTION



The truth is, it's hard to retro-fit. If you're dealing with a massive site that's really inaccessible, yes, it's going to be a bit of a nightmare to try and make it all compliant. But that's no more of a nightmare than doing a massive refactor for some other reason.

The key is to start accessible. Establish some guidelines that designers and developers stick to as the website or web app grows. And if you're working on something that's already pretty inaccessible, make sure that every new thing you add to it *is* as accessible as it can be, so you're not just adding to the existing problem. Accessibility by default is a lot easier than accessibility after the fact.

@type_error

Progressive Enhancement

Progressive enhancement is the approach of building something very simple that works for everyone in all browsers, and then adding extra shiny things that'll enhance the experience for people who can use them, so people using more modern browsers. The idea is, the core functionality of your app should work for everyone, regardless of the technology they're using. This applies to assistive technology as well, for example screenreaders or Braille displays. You can use things like feature detection in CSS and JavaScript to check if a user's browser supports a certain feature before displaying it for them.

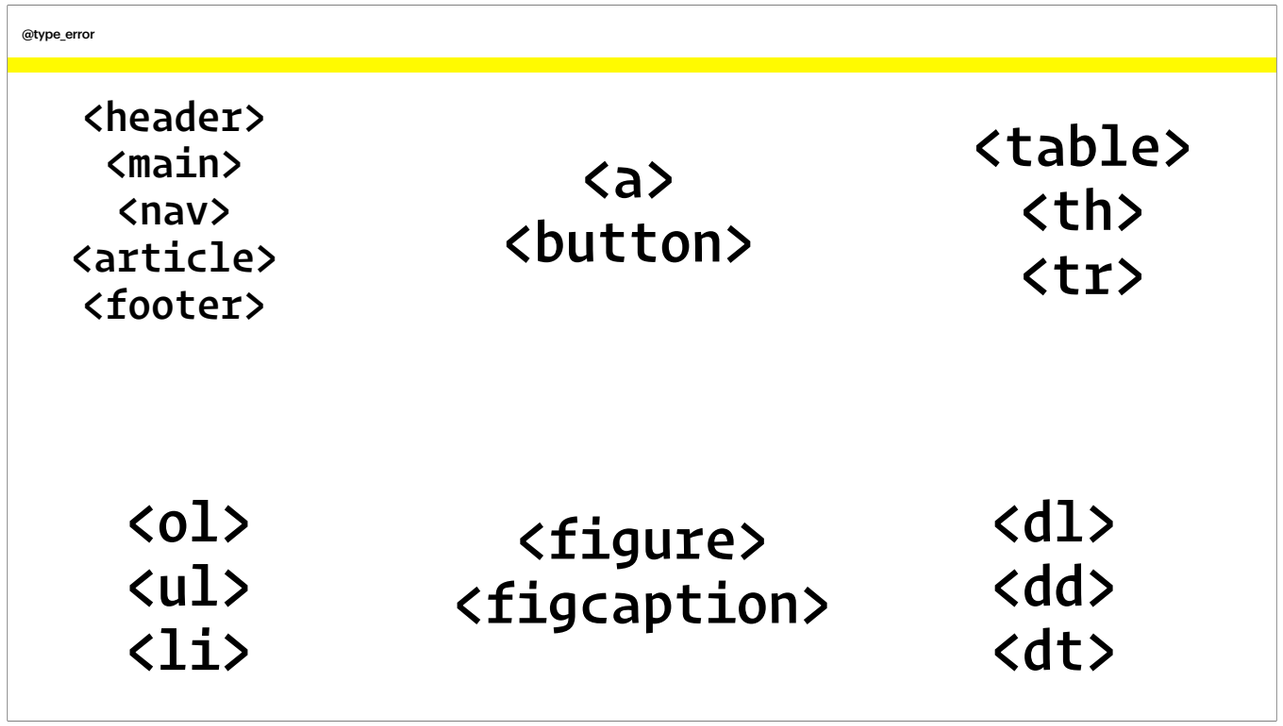
This is a great principle to adopt when starting out a new web project.

HTML gives us accessibility for free

@type_error

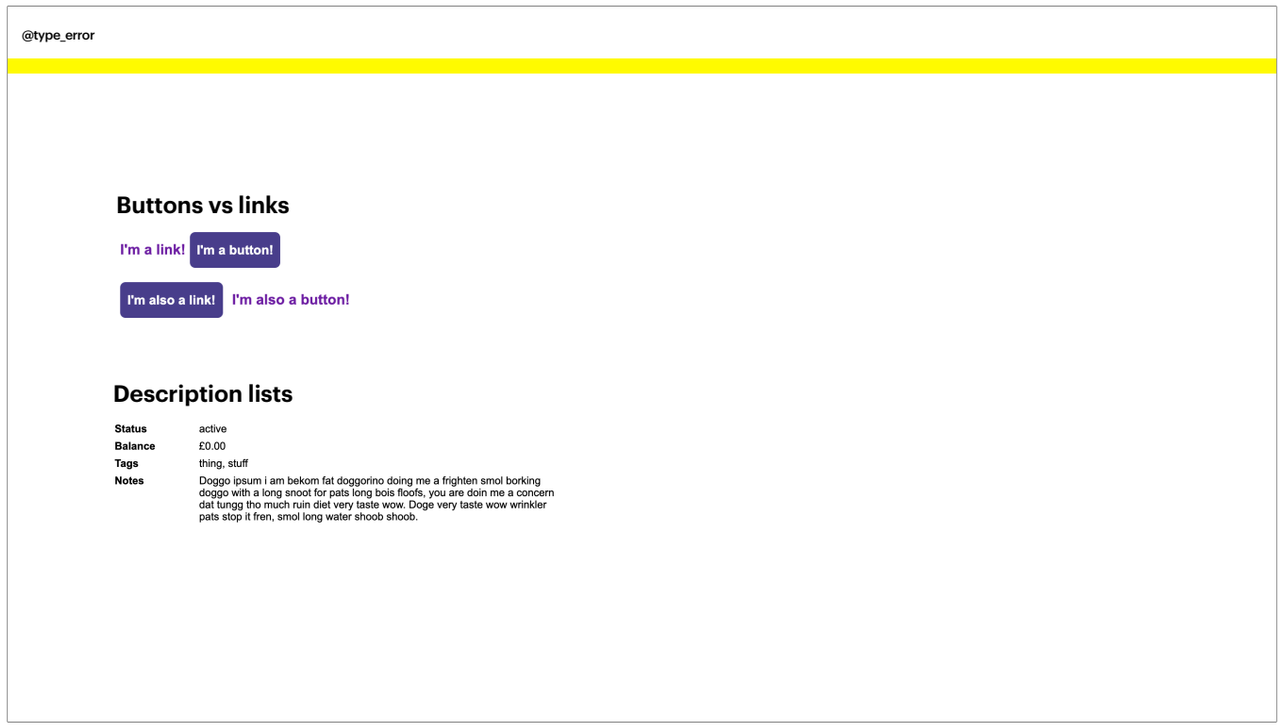
HTML is accessible

Good old trustworthy HTML is totally accessible! A screenreader will see some HTML tags and understand exactly what it’s doing. and since HTML5 was introduced quite a long time ago now, there’s pretty much a tag for everything. It’s what we call semantic HTML - HTML tags that convey meaning about what they contain.



we’ve got tags that indicate different parts of a document, tags that mark up links and buttons, tables, lists, figures and captions, a description list for key-value pairs. All of these tags will tell assistive technology exactly what each thing on the page is for, and that’s so important.

Ultimately you can style almost anything with CSS, so you don’t have to sacrifice design for semantics here.



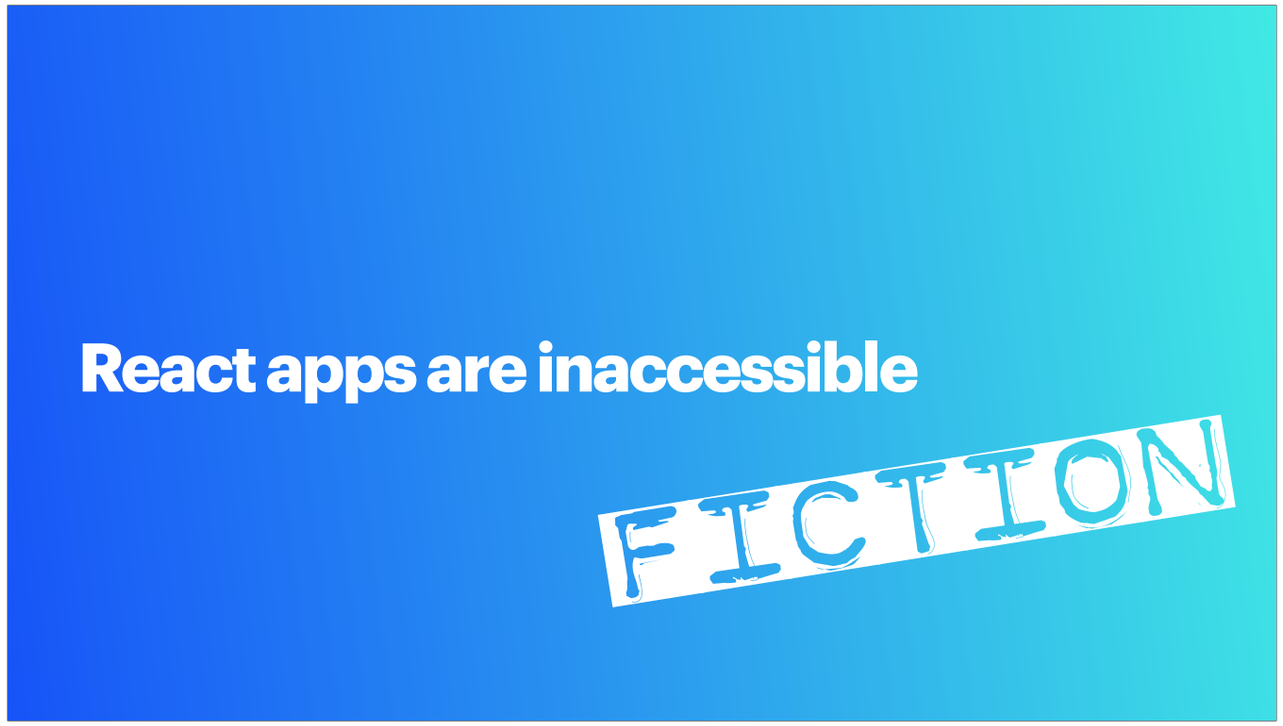
If you have a link that executes some javascript when you click it, it should be a button element. You can make it look like a link using CSS.

If you've got a list of key value pairs conveying information, you can use a description list or an unordered list so that a screenreader knows it's not just unstructured text.

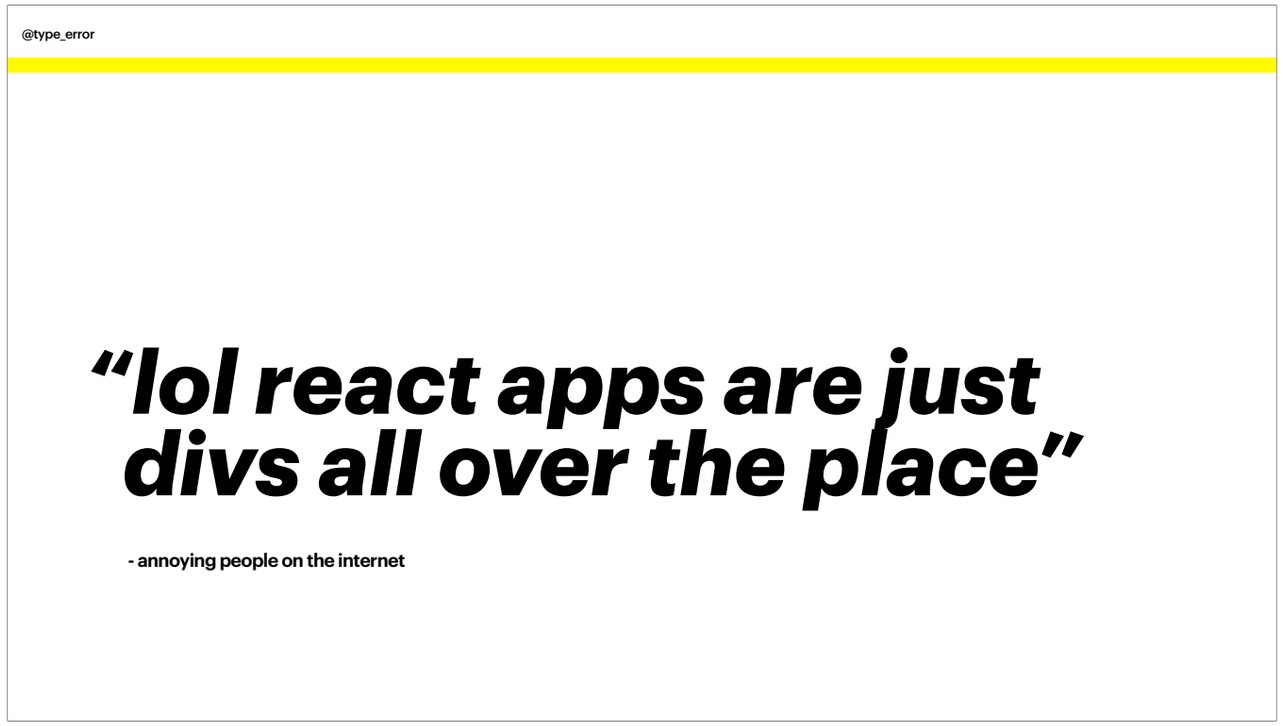
Semantic HTML: everyone's a winner!

Many of the things that will help some groups of users with access needs will also benefit others: for example, good semantic HTML is great for screenreader users, but also helpful for keyboard or adaptive switch users, as with the right tags the browser knows what should be focusable and what shouldn't. Labels on forms make them easier to read and understand for people of all cognitive abilities.

Add an accessibility checklist to your JIRA tickets or pull request templates: make it part of your definition of done. Look out for best practices in your code reviews.



I seem to follow two camps of people on Twitter: people in the React community, and people who hate the React community. I use it every day at work and I think it’s just as good as some of the alternatives out there.



I don’t know about you but this is something I’ve heard a lot over the last few years of using React. Some people seem to think that React apps are all inaccessible piles of divs.

The truth is, if there are a million divs in a React app, it’s because the developer put them there, not because of some intrinsic feature of React.

Even the days of wrapping multiple elements in divs to return them from components are long gone, as we’ve had React Fragments for quite a long time.

JSX is basically fancy HTML

If you've written JSX, which I expect you probably have since you're here, you'll know it's a bit like writing HTML with some extra stuff in it. And all those lovely semantic HTML tags we just saw work perfectly in React.

@type_error

ARIA attributes can add context for assistive technology

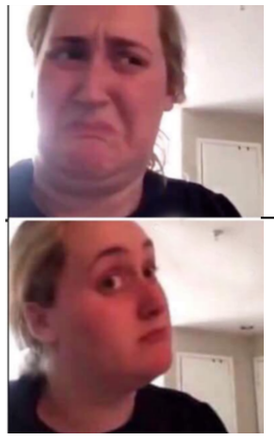
```
<button onClick={showPreferences} aria-haspopup="true">Change your preferences</button>
```

We can add ARIA attributes to elements in the DOM to indicate to assistive technology what function those elements are playing.

For example, we can signal that clicking a button will trigger some kind of popup with aria-haspopup, or indicate whether or not a menu is expanded with aria-expanded.

@type_error

ARIA roles are no substitute for semantic HTML!

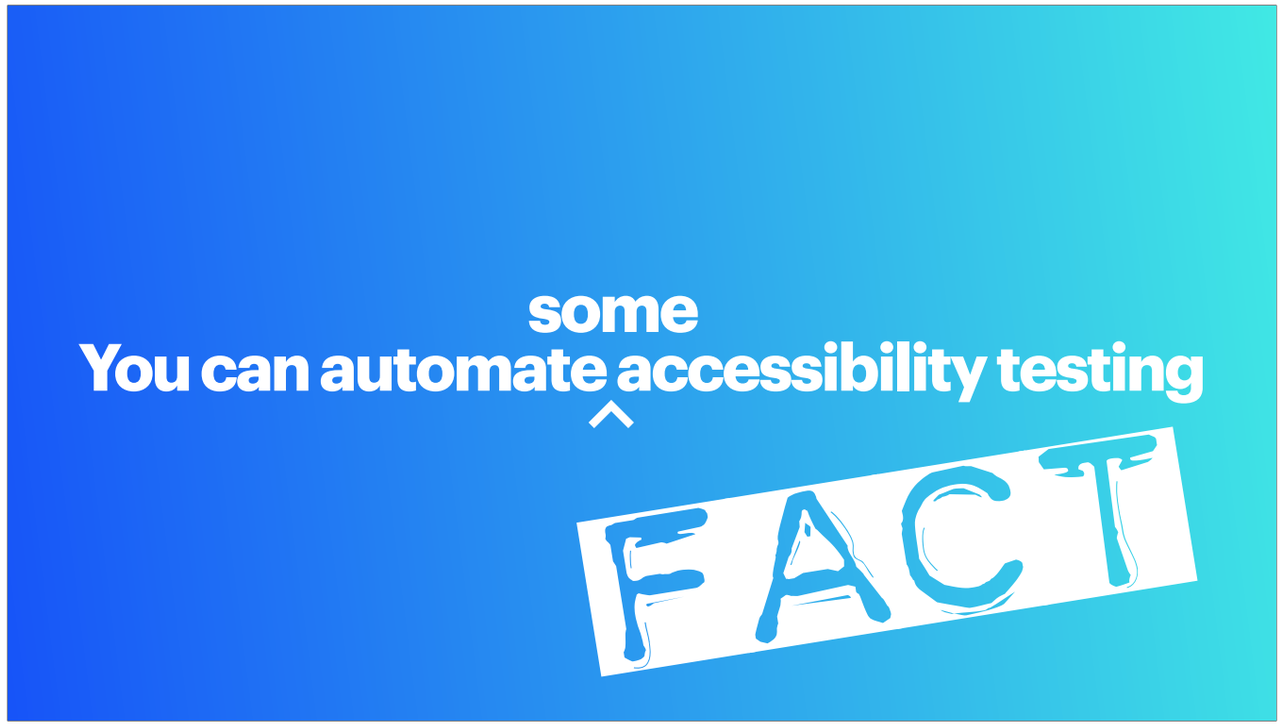


```
<a role="button" onClick={()⇒{}}>Cancel</a>
```



```
<button onClick={()⇒{}} className="button-as-link">Cancel</button>
```

And you should never rely on aria roles when you can use semantic HTML elements instead. When in doubt, use good old fashioned HTML.



To be more specific, you can automate *some* of your accessibility testing.

@type_error

Lighthouse

a11y by Addy Osmani

@axe-core/react

Test your React application with the axe-core accessibility testing library.

axe-core for React

Some examples of automated testing tooling. Lighthouse is built into Chrome, and can be plugged into your continuous integration pipeline. Axe will warn you in the console about any accessibility problems, and A11y is a CLI that does the same.

They can do things like spot incorrectly nested headings, images without correct alt attributes, or forms without labels. All of these things are quite easy mistakes for humans to make, so it’s really helpful to have something scanning your page to pick up on it.

@type_error

Lighthouse Badges

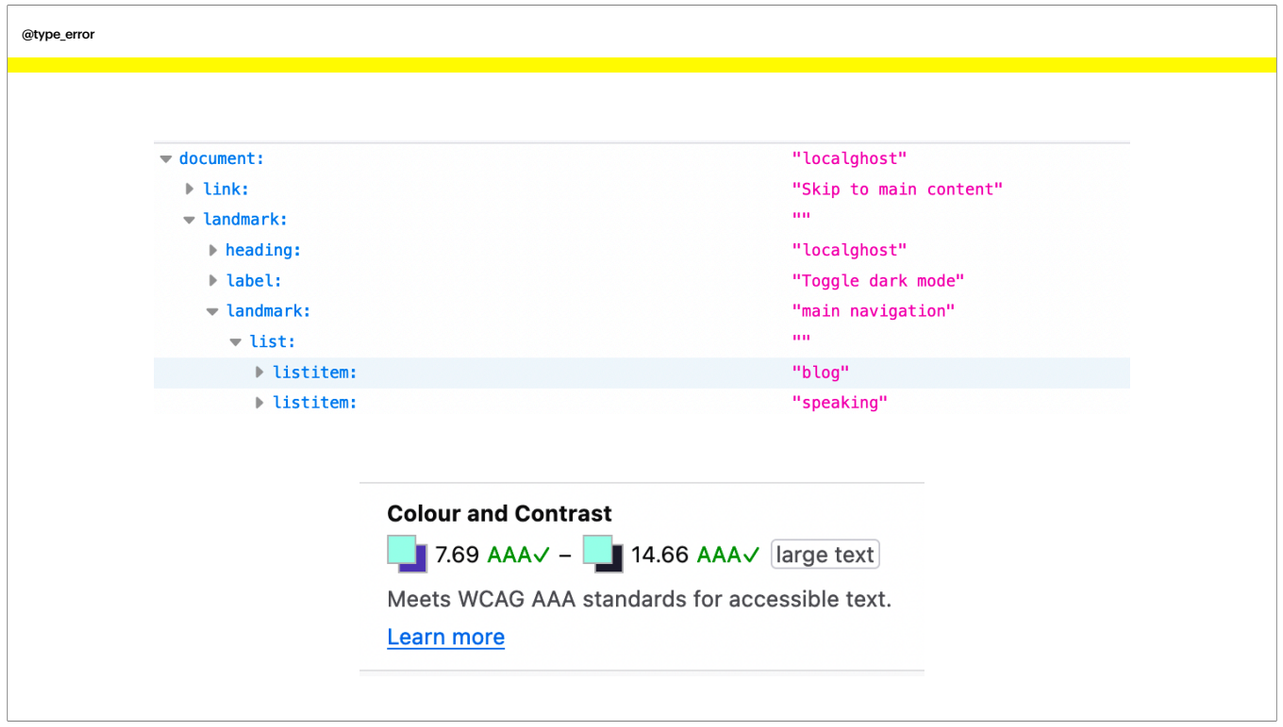
This package allows you to easily create Lighthouse badges for all Lighthouse categories.
Ever wanted to brag about your sites's awesome Lighthouse performance? Then this is the package for you!

Examples

All Badges

People LOVE to show off their lighthouse scores, and that’s all well and good, but it doesn’t tell the whole story.

Lighthouse checks for performance, accessibility, progressive web app performance, best practices and SEO. It’s a great tool, but unfortunately it’s often treated as a silver bullet. A high Lighthouse score is a good thing, but it doesn’t mean that you’ve “done” accessibility - there are always things that automated tools won’t be able to detect.



Firefox and Chrome both have an “Accessibility” section in Dev Tools that give you an outline of how assistive technology might “see” your site. It’ll help you to spot anything that’s being accidentally ignored, or picked up when it shouldn’t be. slightly less automated, but not entirely manual.
These browsers can also identify colour contrast issues.

Automated testing is not a substitute for manual testing!

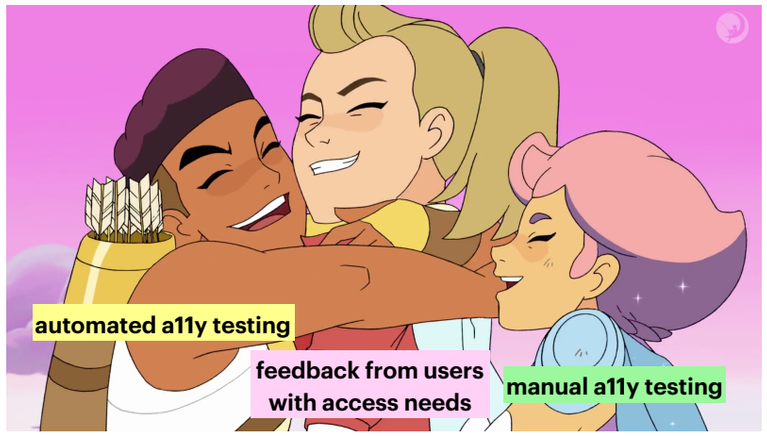
which is not a substitute for real user feedback!

The best way to test accessibility is to approach it from all sides. Automated tooling such as Lighthouse or axe) can and should be part of your development process, as it'll provide a quick feedback loop for common accessibility problems in your markup and CSS. But what these tools don't account for are things like the quirks of different screenreader software (and boy, do they have quirks), or the other kinds of assistive technology people will use to access your website. Lighthouse can't tell you whether your site is still legible when it's zoomed in 600%, or whether you can interact with the various parts of the app with a keyboard as you would with a mouse. The Accessibility in Government blog has a great article about what they found when they tested automated tooling on the world's least accessible webpage. I'll share some useful links after the talk.

Before you merge your PR, check how your new feature behaves with screenreaders: Macs and iPhones come with VoiceOver, Android has TalkBack, and you can download the open-source NVDA screenreader for Windows. Does it read out everything it's supposed to? Is it reading out anything it's not supposed to? Are the headings in the right order?

Ultimately, there's no substitute for getting people with actual access needs to use the website or app. You can do some user testing, commission a formal audit, and/or make sure you have ways for users to give you feedback. Include disabled users in your UX research upfront, as well, so that you're building something accessible right from the start.

@type_error



These three approaches are the best of friends and should be used together for maximum accessibility power. People with access needs may have their own ways of doing things on the internet and assistive technology you might not have thought about.

If you encounter any of these myths, you can (and should) challenge them. A lot of the time, these myths stick around because accessibility just isn’t being talked about. Be the one to bring it up, and get others on board too. Bake accessibility into your design systems and your ways of working.

Thanks!

@type_error
localghost.dev

Resources are in the notes for the title slide!