THE SAME WORK, DONE PROPERLY

A PRACTICAL GUIDE TO WEB ACCESSIBILITY



Chris Smith

Table of Contents

Introduction

Part 1 - Getting Started

1. Who You're Designing and Building For
2. POUR in Brief
3. Making the Case to Your Team

Part 2 - Design Practices

4. Colour, Contrast and Visual Design
5. Layout, Spacing and Responsive Design
6. Content Design and Plain Language

Part 3 - Forms

7. Why Forms Matter
8. Labels, Instructions and Grouping
9. Errors, Validation and Feedback
10. Complex Forms - Multi-Step, Dynamic Fields and Autosave
11. A Complete Example - Accessible Contact Form

Part 4 - Accessible Components

12. Why Components Are Where Accessibility Breaks Most Often
13. Keyboard Navigation and Focus Management
14. Menus, Tabs and Accordions
15. Modals and Dialogs
16. Carousels, Autocomplete and Other Troublemakers

Part 5 - Data Visualisation

17. Accessible Charts
18. Accessible Tables
19. Alternatives and Summaries

Part 6 - Development Practices

20. Semantic HTML and ARIA, Used Sparingly and Correctly
21. Progressive Enhancement and Robustness

Part 7 - AI and Accessibility

22. Using AI to Help Design and Build Accessible Interfaces
23. Using AI to Test and Audit

Part 8 - Making It Stick

24. Building Accessibility Into Your Workflow
25. Common Pitfalls and Fast Fixes
26. Where to Go Next

A Bit About Me

Resources

Introduction

Most developers and designers understand, at some level, that accessibility matters. You're probably already convinced. You wouldn't have picked up this book otherwise.

But being convinced isn't the same as knowing what to do or how to explain it to the people around you - the stakeholders, project managers and teammates who see accessibility as a nice-to-have, a compliance checkbox or someone else's problem. That gap, between caring about it and being able to act on it and advocate for it, is what this book is here to close.

The title says it all, really. Accessible web development isn't a separate discipline layered on top of the real work. It's the same work, done properly. The same HTML, the same CSS, the same components - written with more care, more awareness of how people actually use the web and more respect for the full range of people who might use what you build.

Who this book is for

This book is written for front-end developers and designers who build things for the web. It assumes you're comfortable with HTML, CSS and at least some JavaScript. It doesn't assume any prior knowledge of accessibility but it also won't spend long trying to persuade you it matters. You're already here.

What you'll walk away with

By the end of this book you'll know how to build accessible interfaces from the ground up - forms that work properly with a keyboard and a screen reader, components that don't fall apart when the browser's doing something unexpected, data visualisations that don't exclude users because they can't see a chart.

You'll be able to evaluate third-party tools and libraries before adopting them, test as you build rather than auditing at the end and make a clear, practical case for accessibility to the people who need convincing.

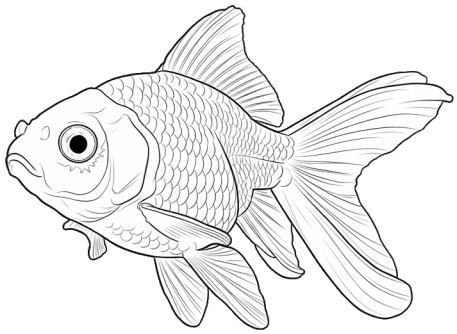
How this book is structured

The book is split into several parts. It opens with the foundations - the standards and concepts that underpin everything else. From there it moves into building accessible components, a dedicated section on forms (which are large enough and problematic enough to deserve their own part), data visualisation and finally a look at how AI tools fit into an accessible workflow.

Each chapter stands on its own. If you work primarily with forms, jump to that part. If you're trying to get data tables right, go there. There's no running example threading through the book and no assumed reading order.

The approach is also framework-agnostic. There's no React chapter, no Vue chapter. The principles and patterns here work regardless of your stack and the code examples reflect that.

Part 1 - Getting Started



Chapter 1: Who You're Designing and Building For

Who actually uses the thing you're building? Ask most teams this question and you'll get an answer shaped like a persona document - a name, a job title, a device. Rarely does it include someone using a screen reader, someone who can't grip a mouse or someone reading your interface with one eye closed on a cracked phone screen on a bus. But they're there. They've always been there.

I'm not going to go deep into every type of disability and assistive technology - there are whole books dedicated to that. What you need, before you design or build a single component, is just enough of a practical overview to make good decisions. Let's get through it quickly.

The four broad categories

It helps to think about disability in four rough buckets. Real people often don't sit neatly in one but it's a useful starting point.

Visual

Blindness, low vision, colour blindness. This is the one most people think of first but there's a lot more to consider.

Auditory

Deafness and hearing loss. If your interface plays a sound to signal success or failure and offers nothing else, you've already lost part of your audience.

Motor

Limited fine motor control, tremor, paralysis, missing limbs, temporary injury. Someone might not be able to use a mouse at all. Someone else might have one hand in a cast this week.

Cognitive

Dyslexia, ADHD, autism, memory or processing differences. This is the category most often ignored entirely and it's arguably the one your content and layout decisions affect the most.

None of these are edge cases. They're just cases you haven't been designing for yet.

Situational and temporary - the bit everyone forgets

We've touched on it already but it's important to recognise that disability isn't only permanent.

Broken your arm and you're typing one-handed for six weeks? That's temporary motor impairment. Standing on a train platform trying to read your phone in bright sunlight? That's a situational visual impairment. Trying to follow a video with the sound off because you're in a quiet office? Well, you now need captions just as much as a deaf user does.

If you design only for permanent disability, you build for a narrower group than you think. If you design for the full spectrum - permanent, temporary and situational - you end up building something that's just better, for everyone.

Assistive technology, briefly

You don't need to become an expert operator of every assistive technology out there. You do need to know what exists, so you understand what your HTML and ARIA choices are actually talking to.

Screen readers

JAWS, NVDA, VoiceOver and TalkBack read out the content and structure of a page. They don't "see" your layout. They hear your semantics. A `<div>` styled to look like a button is invisible to a screen reader as a button - it's just a div that happens to be clickable if you're lucky enough to have wired up the right events.

Screen magnifiers

Zoom in on parts of the screen. If your layout breaks or content gets clipped at 200% zoom, this is who you've broken it for.

Voice control

Lets users navigate and activate elements by speaking commands, often matched against visible labels. If your button says "Submit" but the accessible name is something else entirely, voice control users can't find it by saying what they see.

Non-pointer devices

A single button or a couple of them, used to step through interactive elements one at a time. This is where keyboard focus order stops being an abstract concern and becomes the entire interface.

Try unplugging your mouse for an hour of normal work and use just the keyboard. You'll learn more about your interface's accessibility than by reading up on theory.

Why this chapter comes first

Everything later in this book - forms, components, charts, tables - comes back to the people and tools in this chapter. When we talk about focus order in Part 4, this is who it's for. When we talk about colour contrast in Part 2, this is who benefits. You don't need to memorise assistive technology names. You need to remember that behind every guideline is somebody actually trying to get something done.

Keep that person in mind and most of what follows in this book will feel more like common sense than a compliance checklist.

Chapter 2: POUR in Brief

Designers and developers have different mental models of "accessibility". Some think about colour contrast. Some think screen readers. Some think legal requirements. They are all valid but the whole picture is much broader.

POUR is the framework that fixes that. It comes from the Web Content Accessibility Guidelines (WCAG), the standard the World Wide Web Consortium (W3C) maintains. However, you don't need to wade through the full spec to get the value from it. It's four words and once they're in your head, you'll find yourself running everything you build through them without even trying.

Perceivable

Can people actually perceive the content, in whatever way is available to them?

This one seems obvious until you realise how many ways content can quietly become imperceivable. An image with no alt text is invisible to a screen reader user. A video with no captions is inaccessible to a deaf user. Text at a 2:1 contrast ratio against its background is functionally invisible to someone with low vision, even though it "renders fine" on your high quality monitor in good light.

Perceivable isn't about one sense. It's about making sure the same information gets through no matter which sense someone is relying on.

Operable

Can people actually operate the interface - trigger it, move through it, control it?

This is where mouse-first thinking quietly breaks things. A dropdown that only opens on hover. A custom button built from a `<div>` with a click handler and nothing else, so it's a black hole to keyboard users. A carousel that auto-advances every four seconds with no way to pause it, for someone who needs longer than four seconds to read.

Operable is asking: if I take away the mouse, does this still work? If someone needs twice as long to interact, have I made that possible or have I built something with a timer running against them?

Understandable

Can people understand what's happening, what's expected of them and what just happened?

This one's easy to underrate because it feels like a content problem, not a code problem. But it's both. It's the error message that says "Error: 0x8007045D" instead of telling someone what went wrong and what to do about it. It's the form that clears itself silently when validation fails. It's jargon where plain language would do the same job in half the words.

Understandable isn't about dumbing anything down. It's about respecting the reader's time and attention.

Robust

Will this keep working, reliably, across the tools people actually use to access it?

Robust is the one that ages fastest, because it depends on things outside your control - browser updates, assistive technology updates, devices you've never tested on. A `<div>` with `role="button"` bolted on might work today in the screen reader you tested with. Robust is choosing semantic HTML and standard patterns so it keeps working next year, in a screen reader you've never heard of, on a device that doesn't exist yet.

Why this is the model we'll keep coming back to

You'll notice these four aren't independent boxes to tick - they overlap constantly. A poorly labelled form field is a **perceivable** problem (screen readers can't announce it properly) and an **understandable** one (sighted users aren't sure what it's asking either). A custom dropdown built without keyboard support is an **operable** problem and, if it's built from non-semantic elements, a **robust** one too.

That overlap is the point. Every chapter from here on - forms, components, charts and tables - is really just POUR applied to a specific piece of UI. When something feels off later in the book and you're not sure why, come back to these four words and ask which one you've missed. It's almost always one of them.

Chapter 3: Making the Case to Your Team

You're convinced. You wouldn't be reading a book on accessibility otherwise. The harder problem is usually not you - it's the PM who wants the feature shipped by Friday, the designer who thinks contrast requirements will "kill the aesthetic," or the exec who's never heard a business reason to care. This chapter is about them, not you.

Lead with the argument that lands for the person in front of you

Different people respond to different framings and it's worth knowing which one you're reaching for before the conversation starts.

For a PM or exec, lead with reach and risk

Depending on the source, somewhere around 15-20% of people have some form of disability. That's not a niche audience, it's a market segment most competitors are ignoring by accident. On the risk side, accessibility lawsuits and complaints are a real and growing cost - not something worth scaring anyone with but worth having as a fact ready if "why does this matter commercially" comes up.

For a designer, lead with the false trade-off

The instinct is that accessibility and good design are in tension. They're not. High contrast, clear focus states and readable type are design decisions that make an interface better for everyone, not compromises made for a minority. Some of the most acclaimed products in recent years are also some of the most accessible - it's not a coincidence, it's cause and effect.

For a developer, lead with "it's cheaper now than later"

Retrofitting accessibility into a shipped product is always more expensive than building it in from the start, for the same reason retrofitting any missed requirement is expensive - you're now unpicking decisions instead of making them.

Reframe "extra work" as "the same work, done properly"

A phrase worth having ready: accessibility isn't a feature you add, it's a quality bar the feature has to clear. Nobody asks "should this button also work?" as an optional extra. Nobody asks "should this page also load?" as a stretch goal. Keyboard support and screen reader compatibility deserve the same status - not a nice-to-have, part of "done."

This reframing matters because "extra work" implies extra time and extra budget, which is exactly the fight you don't want to keep having every sprint. "The same work, done properly" doesn't need a separate line item.

Bring a demonstration, not just an argument

Nothing convinces a sceptical colleague faster than a live demo. Sit next to them, unplug the mouse and try to complete your own product's checkout flow or sign-up form, using only the keyboard. If it takes you four minutes of frustrated tabbing to find the submit button, you've made the argument better than any statistic could.

Screen readers work the same way. Turn one on, close your eyes and try to understand a page you built. It's humbling and it's persuasive in a way that slide decks aren't.

Use the language your organisation already uses

If your company talks about "quality," frame accessibility as a quality issue. If it talks about "risk," frame it as risk. If it talks about "growth" or "market size," frame it there. You're not being dishonest by choosing the framing - all of these are true simultaneously. You're just meeting people where they already think.

Start small, without waiting for permission

You don't need a company-wide accessibility initiative to fix a missing alt attribute or add a focus style to a button. Fix what's in front of you and let the visible improvement be part of the pitch for doing more. Momentum is easier to build from a handful of small, obvious wins than from a proposal sitting in someone's inbox.

What not to do

Don't lead with guilt. Telling a colleague they're "excluding disabled users" as an opening line puts people on the defensive. Save the human argument for after you've got buy-in, when it reinforces the decision.

A quick script, if you need one

"I want us to make sure this works for keyboard and screen reader users too - it's the same standard we already hold ourselves to for browser support, just extended to how people actually navigate. I can show you what I mean in two minutes if that helps."

That's it. Short, non-confrontational and backed by a demo, not a lecture. The rest of this book gives you everything you need to make good on it.