


Marcus Herrmann

Accessible Vue

A stylized graphic of a mountain range composed of several overlapping triangles of varying heights and shades of blue and green, set against a light green background.

**The Why and How of building
inclusive apps with Vue.js**

Accessible Vue

The Why and How of building inclusive apps with Vue.js

Marcus Herrmann

This book is for sale at <http://leanpub.com/accessible-vue>

This version was published on 2021-03-25



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2020 - 2021 Marcus Herrmann

Contents

About this book	1
What are the contents of this book?	3
What are the limits of this book?	4
Who is this book for?	4
How do code examples in this book work?	5
Who wrote this book?	5
Cover accessibility basics first	6
Web app accessibility is web accessibility	6
Law of the instrument	7
The sad status quo	8
The search for root causes	10
Where to start to make a change	11
Ask users, do User Testing	15
A special look on screen-readers	16
Action Steps for this chapter	21
Basic web app accessibility concepts	23
Disclosure widgets	23
Focus management	25
Modality	28
Hiding and deactivating elements accessibly	31

CONTENTS

Links and Buttons	35
WAI-ARIA Authoring practices	38
Action Steps for this chapter	41
Using Vue’s strengths	42
Conveying context with props	42
Requiring props	45
Circumventing Vue 2’s one-root-element rule	49
Facilitate focus management with \$refs	53
Controlling where attributes get applied to	54
Visibility helper components	55
Accessible Base Components	58
Action Steps for this chapter	62
Make typical components accessible	63
Modal dialogs	63
Skip links	71
Slide-In Navigation	75
Menus	84
Tab Component	89
Aside: Component libraries	101
Action Steps	103
Convey changes of state to screen-readers	104
Live regions	104
Accessible routing	110
Action Steps	115
Testing for accessibility	117
Automated tests are no miracle workers	118

CONTENTS

Accessibility Testing as education	118
Linting	119
Checks in your browser	121
Unit tests	126
End-to-end, or inter-component testing	128
The most important testing of it all	131
Action Steps	133
Keep on learning	134
Follow these people	134
Look into React	137
Follow the development of standards	138
Join the accessibility community	139
Action Steps	140
Conclusion	141

About this book

In one regard, Vue is very accessible: When it comes to the developer experience and learning new shiny JavaScript frameworks, its often quoted “developer experience” is one of a kind. At least, in my opinion, getting to write your first lines of Vue code is far easier than writing your first lines of code in a React or Angular project. If you are using direct imports of the script, you don’t even need a bundler or “create Xy app” script in the first place! Not even a node.js setup necessary!

In another regard, Vue is considered not very accessible: When it comes to the output it produces. This is not a problem that is exclusive to Vue, mind you. All three of the current big players in JavaScript framework land - Vue, React, Angular - have this problem. The issue is **not** that it is impossible to create inclusive experiences with these systems: all three of them are very well capable of producing code that can be accessed by a maximum of users (and I hope this book succeeds in helping you to churn out accessible Vue apps). There are some specialities that JavaScript frameworks and especially the concept of client-side rendering bring to the table, but there are strategies to tackle them (and in this book you are going to learn how). No, in my mind, inaccessible apps built with a large framework are first and foremost an awareness and education problem. In order to create inclusive apps, the **actual output** of your JSX, Typescript, Hooks, Composables, State of whatever has to be moved much more into the limelight than it is now. Because this output part is what counts for a large and always growing user group.

Our industry must learn and understand: Websites and apps are not always consumed in a setting which is (I assume) envisioned by the developer: Rendered visually and used with either mouse and keyboard or touch. The reality is: there are more type of web usages around that the average developer can imagine: zoomed in, with changed

fonts, being output non-visually through a screen reader voice, utilized with a mouth-controlled mouse, seen with color vision impairments, used with an impairment that prevents fine motor movements, read by people who have the language your app is in as a foreign language, populated with people who use websites and apps with their voice only, visited by people with learning impairments - or simply operated by people who are in a hurry, outside in the sun, have trouble concentrating because they are distracted by whoever, can't move their mouse because they broke an arm - and so on, and so on. This list is as infinite as human variety. At the same time, I consider this the first pillar of the problem: Non-awareness of diversity.

The second part of the educational problem is not exclusive to apps built in or with JavaScript – it is also valid for an old-school website (whatever that may be exactly). It is the mismatch that there is so much good content around on the internet to make the web accessible, but people either aren't aware of it or are overwhelmed by it and don't manage to find a point of entry into this topic.

The third part of the educational problem is a delicate one, and I'm really not trying to enter the blame game. Modern web development professionalized more and more. Which is a good thing, because the web is becoming more and more infrastructure of our world. This professionalization led to an influx of diverse developer groups (which is, in the big picture, a great thing). What you can do on the web, what functionality is available online is mind-blowing and was only possible on some dedicated machines, on proprietary and expensive systems, years ago. Part of that boost of functionality and complexity had something to do with, let's call them, programming veterans. The gate-keeping aspect of the "brave new web dev" world put aside: The respect for well-designed, elegant and dry backend parts of apps has definitely increased one hundredfold, where the respect for the actual output (HTML, CSS) has stalled (or did even shrink). And this is a problem. The thinking of "framework first", as [Eric W. Baley puts it](https://avue.link/ff)¹, needs to give way to thinking "user-first". And the profound realization that user are far more diverse than most web

¹<https://avue.link/ff>

developers, old and new, think.

I wrote this book to help with educational problems 2 and 3. Regarding problem number one, there are a plethora of great books and resources around (which I'll try to link in the first chapter).

What are the contents of this book?

This book is about web app and Vue accessibility in general. However, “normal” accessibility is about 80% of web app accessibility, and I aim to give you pointers on where to get a basic understanding, where you can start your journey towards inclusivity (chapter 1).

Then, in Chapter 2, I will try to present you building blocks for an accessible web that are especially useful in client-side driven situations.

Chapter 3 is about how you can use Vue's core concepts and architectural strengths to achieve accessible output. JavaScript-based frameworks have, despite their bad reputation, plenty of advantages when it comes to accessible code. Alas, just a minority of developers know and use them.

The next chapter (number 4) deals with making some of the most typical components you have in your app or JavaScript-enhanced page accessible.

What really differentiates web apps from web pages (and where Virtual-DOM frameworks can really shine) is state and asynchronicity. One can't deny the problems of inaccessibility this potentially leads to, especially with screen readers. Chapter 5 is about that.

Chapter 6 is about testing, and about how you can use the power of automated checks to improve and (in part) ensure accessible output. At the same time, it tries to convey the limits of automated or semi-automated testing and tries to nudge you towards creating infrastructure for the most important kind of testing at all.

Finally, chapter 7 is about the fact that accessibility is not only a topic that can't be covered aptly in a short book like this (even more if it limits itself to Vue). So, this last chapter is about the reader's further learning journey. I tried to gather some interesting articles, books, videos and teachers to do just that in this last part.

What are the limits of this book?

I have the urge to put an emphasis on this: I can't and won't repeat web accessibility basics here, although they make up the vast majority of accessibility of your app. Other, better and larger books have been written especially about this topic, and the ones that I know and love are recommended in chapter 1. This is a book about accessibility (a11y) in Vue, not about a11y in general. But: since the very aim of any JavaScript-driven framework is to output HTML, many contexts are the same in other frameworks (or user interface libraries, if you want).

Due to the vast amount of accessibility topics covered, this book only can introduce most of the tools I recommend on a surface level. Being too detailed in a book dealing with a moving target like JavaScript is an error in itself, I think. For example, the proper usage of `eslint`, `jest` and `Cypress` in the testing chapter deserves a whole book each. I, thus only can supply starting points, prime you towards the existence of those tools, try to convey the message why they are useful when building accessible apps.

Who is this book for?

This book is for individuals who are familiar with Vue and care about not excluding users from the pages and apps they built with it. Potential readers ideally have a basic understanding of web accessibility. Otherwise, they should follow the pointers in chapter 1. Knowing more about accessibility is a skill that pays off. Not only for further reading of

the book! The topic of web app accessibility and especially accessibility with Vue is a tad under-documented. But nonetheless very important.

How do code examples in this book work?

There are plenty of ways to use vue (with `vue-cli`, by just adding a `<script>` tag, full client-side or using Vue just as the JS-icing of the serverside cake).

In the coming chapters, I will write example code in the “classic” way:

- By using single file components
- When I refer to a complete project, I assume it was created with Vue CLI.
- I will consequently use the Options API (if you don’t know what this is: the way you write the script parts of Vue 2 components).
- Code examples are provided both in Vue 3 and Vue 2 syntax - in the form of a CodeSandbox instance for every code sample.

Who wrote this book?

I’m Marcus Herrmann, certified Web Accessibility Specialist (IAAP) and freelance web developer from Berlin, Germany. As a developer, I reach out - happily - for VueJS, when I need should build a more complex user interface with JavaScript. At the same time, I like things that I built to be robust. This means I don’t build every project in Vue.

But for some other projects, using a client-side-driven framework is the best choice. Since I’m fascinated by web accessibility, Vue and especially the intersection of both, I decided to share what I know about it in this book and in [my blog](#)². I hope you can take something away from the reading.

²<https://avue.link/marcus>

Cover accessibility basics first

Web app accessibility is web accessibility

Let's start this book about accessibility in Vue with quotes about React:

Building an accessible React app is, at its core, not about React at all. The key is mastering the fundamentals of web accessibility: semantic document structure, appropriate labeling, and managing focus.

[Netlify's article 'Accessibility is not a "React Problem"'](#)³

80% of your React is HTML, so learn it. It's much more than just divs and spans.

[Artem Sapegin on Twitter](#)⁴

In the quotes above, you could replace "React" with "Vue" (or "Angular", or "Svelte", etc.) and the sentiments would still hold true.

This means: Although there are special idiosyncrasies of web apps that create challenges for accessibility (more on these specifically in chapter 5), one thing remains true: Everyone committed to JavaScript Framework accessibility has to make sure that they understand the basics of web accessibility first. **Accessibility for a specific framework can only be a second step in a process where the first one is to learn the basics.** You could go even as so far as to refer to the famous Pareto principle: 80% of web app

³<https://avue.link/reacta11y>

⁴<https://avue.link/react80>

accessibility is taking care of basics, while only 20% is a special web app accessibility matter.

Unlike material on JavaScript framework accessibility, you will find plenty of good educational content on web accessibility online – if you look for it. I wrote this first chapter to help you look and learn, hopefully providing you with many starting points to tackle “the 80%”. A word of caution, though: If you are a developer new to web accessibility, then merely reading some of the resources linked below would not be enough to get a grasp on the accessibility fundamentals. It is a much more effective way to learn by building little experiments and projects, apply what you have learned, and try to use it assistive technology (a screen-reader, for example). On the other hand, when you are a seasoned and accessibility-savvy developer and are looking to employ your knowledge in Vue.js, you can safely skip this chapter.

Law of the instrument

I am not only starting the first chapter about a book on Vue with statements about React, but I will also go even further in saying: **Check if you need Vue in the first place for the project at hand.** Just because you know a particular tool very well and have profound experience does not mean that it is the right tool for the job. This principle is attributed to the US psychologist Abraham Maslow and dubbed Maslows’s hammer: “If all you have is a hammer, everything looks like a nail”. Reflect and ask yourself:

- Is Vue (or any other Virtual DOM Framework) your hammer?
- Is Vue (or any other Virtual DOM Framework) really the best fit for this project?

Tools made to create applications with should be **only** used for exactly that. Building a simple landing page or blog clientside-rendered is overkill. Why engage in solving (accessibility) problems like asynchronous updates in screen readers and accessible

routing without any need? Especially when you could have chosen a tool that outputs static pages (nuxt, eleventy, Hugo, even WordPress) and you could reap the fruits of standard page loading browser behaviour – and of more of robustness (see [Jeremy Keith’s talk “Layers of the web”⁵](#))?

The sad status quo

Did the last paragraph either makes you rage-deleting the ebook file and asking for a refund, or are you really plan to build an *app*? In the second case: let’s move on (in the first case, we’re probably both writing each other an email right now).

The sad truth is that even in projects that are not web apps (wherever you draw the line between web page and app), things are not great when it comes to following the most basic accessibility advice. As a consequence, the web is full of websites that exclude people.

The “WebAim Million” is an often-quoted piece of research where the service provider WebAim investigated the accessibility of 1,000,000 pages via automated testing. While the source of the data, automated testing of accessibility, has its limits (more on that in chapter 7), the study still shows tendencies what goes wrong in the web due to the sheer number of sites examined. The conclusion of WebAim is devastating:

...the results paint a rather dismal picture of the current state of web accessibility for individuals with disabilities.

Aside: Web Content Accessibility Guidelines (WCAG)

W3 Consortium’s collection of testable steps (success criteria), ensuring the minimum limit of website accessibility. The WCAG in version 2.1 is referenced in accessibility

⁵<https://avue.link/layers>

legislation worldwide.

When examining the results, WebAim was able to identify and group the following main sources of error in the pages they checked:

WCAG Failure Type	% of home pages
Low contrast text	86.3%
Missing alternative text for images	66.0%
Empty links	59.9%
Missing form input labels	53.8%
Empty buttons	28.7%
Missing document language	28.0%

(Data from February 2020)

All in all, a whopping 98.1% of homepages had detectable WCAG failures. There are two very sad things about this: Firstly, most of the errors (or error groups) listed above are relatively easy to fix. This must lead to the conclusion: Web developers either don't know about these issues, don't care, or aren't allowed to fix.

Secondly, the picture painted by the research project is most likely incomplete, and the actual situation is even worse. As mentioned before, the WebAim Million mass test was conducted in an automated way. Automated testing is only one way of auditing a web project for accessibility and can only detect circa about 30% of the barriers present. The other way is the more thorough manual test. Some decisions can only be made by a human being with their judgement and knowledge of the context. For this reason, automated testing is in the best-case scenario kept within a manual audit process. Both human and machine can play out their respective strengths in this scenario.

The search for root causes

Now, why is the situation the way it is? I do not claim to be able to answer the question. But I think theories are legitimate. Here are my two guesses as to what the crucial root causes are:

Theory 1: It's an education problem

Whichever way you look at it, accessibility is very rarely part of web development tutorials. The majority of authors or content creators concentrate on the main learning content (naturally). When you are not aware of the underlying problem of unsemantic HTML, but want to convey how some library X fetches data from end-point Y you do not really mind about the trigger of it all being a `` button with a click event listener on it. Because everything seems to be working! So the people who teach are not always aware that their knowledge of the craft is incomplete (“unknown unknowns”, as Donald Rumsfeld put it). A chain reaction of inaccessible code follows (but hey, your state library now purrs like a cat!).

The lack of accessibility as a topic in coding boot camps can be considered as the other side of the coin. Due to recruiting needs, the hottest frameworks are put in the window displays (and course agendas) of boot camp providers. As a result, the necessary accessibility basics are skipped or ignored. Your state library still has soft fur.

Theory 2: Too much focus on tooling, not on the output

Developers have always loved abstracting, and to be frank, I consider this one of our superpowers. And abstraction is not bad in itself! Sometimes it is the only way to deal with both large code bases and to work with your omnipotent framework for web applications. Before there's a misunderstanding: excellent developer experience (DX)

and following concepts like DRY (Don't repeat yourself) are not bad per se! We just have to remember the end-user. What they experience using the result of our work is not the awesome DX, the dogmatically correct code. It's the resulting HTML, CSS and JavaScript. Ultimately, the user (rightly) only assesses projects from the concrete result in front of them, and whether they can use it or not. I strongly feel web developers need to keep spending less mental energy to keep the framework environment happy and need to be more invested in the output that your miracle framework generates.

Where to start to make a change

So it comes down to learning about “the lived experiences of disabled people as a context for understanding the need for design and code created with access in mind” (to quote [Eric Bailey on Smashing Magazine⁶](#)). The Million \$yourCurrency question is now: Where to start? If you ask me, it makes great sense to start with a big round of questions. Here are the first general but important ones to ask yourself:

- Do I know the variety of ways people use websites? Have I heard of some of their assisting technologies and strategies for doing that?
- Do I know which standards for web accessibility do exist?
- Do I know HTML, CSS, WAI-ARIA and JavaScript well enough to build accessible experiences?

Only then it is reasonable to ponder about the following, much more concrete questions:

- Does my HTML document's page title convey the topic or purpose of the app or document?
- Do present headlines help to structure its content, and is their hierarchy arranged in a logical way?

⁶<https://avue.link/equivalent>

- Are there so-called “bypass blocks”, meaning: Navigation helpers such as skip links (sending focus to other parts of a page), or structuring elements such as landmark regions (like `<main>`)?
- Is information not only conveyed through color only? For example: does a red outline alone show that a form input validation has gone wrong?
- In your texts, do you avoid referring to content or controls by their visual position on the site, their shape or color? Does your copy say “Click on the triangular/green button to proceed” or the like?
- Do you use good text contrasts in your document, especially regarding (but not limited to) links and other interactive elements?
- Is the currently focused element marked clearly? Meaning: do you either have a visible outline, or a dedicated focus styling in your CSS?
- Is the page zoomable, or can users adapt the text display without breaking your layout and making parts of the content unreachable?
- Does any form of media (e.g. images) and do any interactive controls (e.g. icon buttons) have alternative texts or accessible names?
- Do form fields have labels? Are they described well and close to the related input? Are error messages understandable, clear and programmatically tied to “their” input field?
- Especially important for JavaScript-driven projects: Is there success and failure feedback?
- Is the whole app operable with keyboard only? Are there parts of your app that are only reachable on hover?
- Regarding touch input: are the targets big enough; are there alternatives to touch gestures?
- Could animation or parallax effect present in your app cause seizures for people with photosensitivity?
- Do you prevent auto-playing media, and do you offer a pause function for auto-scrolling, auto-playing, auto-updating, moving or blinking content?

- Do you use semantic HTML when outlining your document and describing its content?
- Do you supply information on the web app's language in the document (human language, like English, French, German or Spanish, in this case)?
- In case of audio and video content, do you offer captioning, transcripts or audio description?

As you can see from the list of questions above and the looming blocks of introductory tutorials below – web accessibility is a large and diverse topic. It is also one that is covered in many articles, courses and books. Since it does not make sense to try to explain what others have explained before (and in a better way, I'm sure) – and since that book is about Vue.js accessibility specifically, here are references and links to great starting points:

Recommended free materials

<https://accessibility-for-teams.com/accessibility-for-ux-designers>⁷

<https://marcysutton.com/web-accessibility-resources>⁸

<https://www.w3.org/WAI/fundamentals/foundations-course/>⁹

<https://www.microsoft.com/design/inclusive/>¹⁰

<https://www.w3.org/WAI/fundamentals/accessibility-intro/>¹¹

<https://webaim.org/intro/>¹²

<https://developer.mozilla.org/en-US/docs/Learn/Accessibility>¹³

<https://scotch.io/tutorials/web-accessibility-for-beginners>¹⁴

⁷<https://avue.link/a11yteams>

⁸<https://avue.link/marcyresources>

⁹<https://avue.link/w3foundations>

¹⁰<https://avue.link/msinclusive>

¹¹<https://avue.link/w3a11yintro>

¹²<https://avue.link/webaimintro>

¹³<https://avue.link/mnda11y>

¹⁴<https://avue.link/scotcha11y>

<https://web.dev/accessible/>¹⁵

<https://www.w3.org/WAI/tutorials/>¹⁶

<https://www.coursera.org/learn/accessibility>¹⁷

<https://www.w3.org/blog/2019/12/free-online-course-introduction-to-web-accessibility/>¹⁸

Hidde de Vries' talk "It's the markup that matters" on YouTube¹⁹

<https://www.udacity.com/course/web-accessibility-ud891#>²⁰

<https://pressbooks.library.ryerson.ca/iwacc/>²¹

<https://pressbooks.library.ryerson.ca/wafd/>²²

<https://dev.opera.com/articles/introduction-to-wai-aria/>²³

Recommended paid materials

Derek Featherstone's video courses on Lynda.com²⁴

Jon Kuperman's video course on frontendmasters.com²⁵

"Accessibility for Everyone" by Laura Kalbag²⁶

"A Web for Everyone: Designing Accessible User Experiences" by Regine M. Gilbert²⁷

"Practical Web Inclusion and Accessibility" by Ashley Firth²⁸

"Inclusive Design Patterns" by Heydon Pickering (eBook)²⁹

"The Bootcampers Guide to Web Accessibility" by Lindsey Kopacz (eBook)³⁰

¹⁵<https://avue.link/webdeva11y>

¹⁶<https://avue.link/waituts>

¹⁷<https://avue.link/coursera>

¹⁸<https://avue.link/w3course>

¹⁹<https://avue.link/hiddemarkup>

²⁰<https://avue.link/udacity>

²¹<https://avue.link/pressbook1>

²²<https://avue.link/pressbook2>

²³<https://avue.link/introaria>

²⁴<https://avue.link/feather>

²⁵<https://avue.link/jonkup>

²⁶<https://avue.link/kalbag>

²⁷<https://avue.link/idbook>

²⁸<https://avue.link/practical>

²⁹<https://avue.link/ifedp>

³⁰<https://avue.link/bootcamper>

Ask users, do User Testing

If you worked through the material above or knew your fair share of accessibility before opening this book, you know: On the one hand, humans and disabilities are diverse. On the other hand, people building websites, organizations building user agents (browsers) and governments making laws facilitating the participation of all people have to agree on standards.

Standards are the results of a discussion, an agreement between parties and, for the most parts, a compromise. Since humans and their abilities are diverse, accessibility is a spectrum, and so are accessibility standards. *The* central document and standard for digital accessibility are the Web Content Accessibility Guidelines (WCAG). Consider: There are but a bare minimum to aim for, not the end of the process of building your project inclusively. Don't fall into the misconception that with WCAG compliance, everything is perfectly accessible.

The matter becomes even more complex that there are many materials about accessibility out there that cannot be considered "actual standards", even if they are coming from the W3C. When dealing with the accessibility of JavaScript-driven interfaces and web apps, one often stumbles upon the WAI-ARIA Authoring Practices. Beware: these are not standards! I will go into the details in the next chapter, but it is important not to put the Authoring Practices in the same "standards bucket" as WCAG's success criteria.

If you are truly caring for an inclusive web product, you pair conformance to (real or misunderstood) web accessibility standards with running web accessibility user testing. Ideally at regular intervals, ask your users about their experiences, expectations and preferences. The creators of the web app "Invision" have written a great introductory blog post on this:

With accessibility user testing, the main goal is to verify whether your assumptions concerning accessibility features were right. The objective is to discover

potential problems and opportunity areas based on the already working product or the prototype.

It is worth studying the [article](#)³¹ in-depth to learn about the importance of goal setting, methodology and recruitment in such a process,

Note that mention of “assumptions” in the quote above? On [uxdesign.cc](#)³² (with is at the time of writing, sadly hosted on the rather inaccessible platform Medium.com), Daniel Pidcock shares an eye-opening story about an assumption he as a Head of Accessibility held:

An online food ordering service should be awesome for deaf people, right? I thought so. Find out why I was wrong and how I learned the importance of user testing with disabled people.

Only after he had the opportunity to check his assumptions – after having had contact with affected people on social media – Pidcock learned about points of failure that makes the ordering of food on the service’s website impossible for deaf customers: [“Accessibility user testing: a cautionary tale”](#)³³.

(End of sample file. Thank you for your interest in "Accessible Vue"! To purchase it, head over to <https://accessible-vue.com>)

³¹<https://avue.link/usertest>

³²<http://uxdesign.cc>

³³<https://avue.link/ordering>