

How to test your web app accessibility?

by Sylvain Hamann

What's Web Accessibility?

It's the principle of designing and developing websites in a way that ensures they can be used by anyone, regardless of their abilities or disabilities, and regardless of the technology they use to access the web.

MANUAL AUDIT

Keyboard only navigation

Via tab or shift+tab

A screen reader is a form of assistive technology (AT)^[1] that renders text and image content as speech or braille output. Screen readers are essential to people who are blind,^[2] and are useful to people who are visually impaired, illiterate, or have a learning disability.^[3] Screen readers are software applications that attempt to convey what people with normal eyesight see on a display to their users via non-visual means like text-to-speech,^[4] sound icons,^[5] or a braille device.^[2] They do this by applying a wide variety of techniques that include, for example, interacting with dedicated accessibility APIs, using various operating system features (like inter-process communication and querying user interface properties), and employing hooking techniques.^[6]



Microsoft Windows operating systems have included the Microsoft Narrator screen reader since Windows 2000.

Zoom

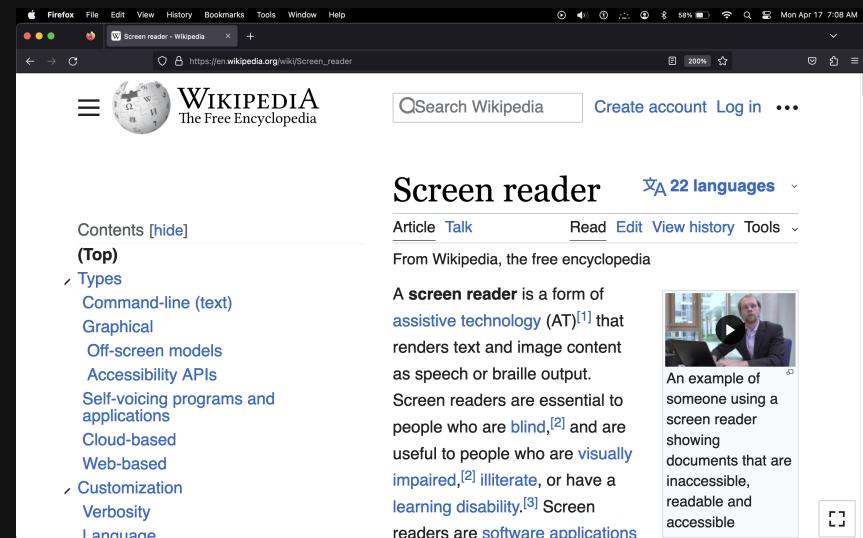
- Try different types of zoom and magnifiers;
- WCAG recommends to support 200% zoom.

Default zoom



A screenshot of a Firefox browser window displaying the Wikipedia article "Screen reader". The page is shown at 100% zoom. The interface includes the standard Firefox menu bar, a search bar, and a user login section. The main content area features the Wikipedia logo, a search bar, and the title "Screen reader". Below the title, there are sections for "Article" and "Talk", and a "Read" button. A video player is embedded in the page, showing a man using a screen reader. The overall layout is clean and typical of a web browser at standard zoom levels.

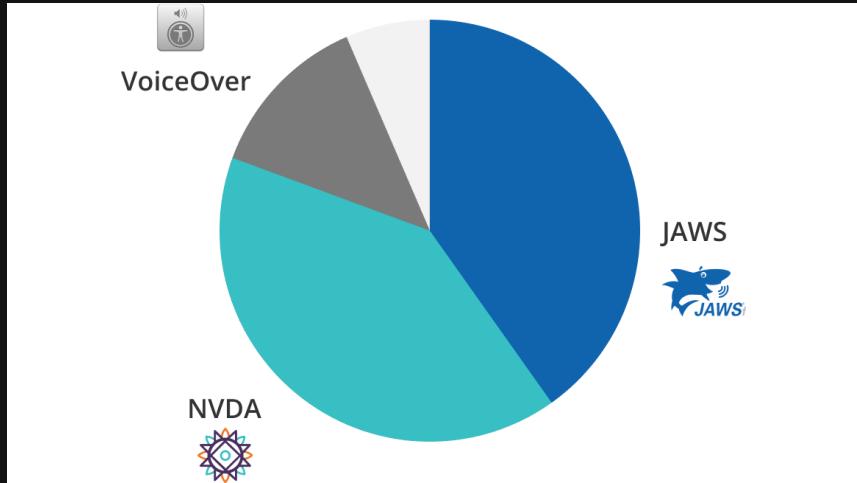
Text only zoom



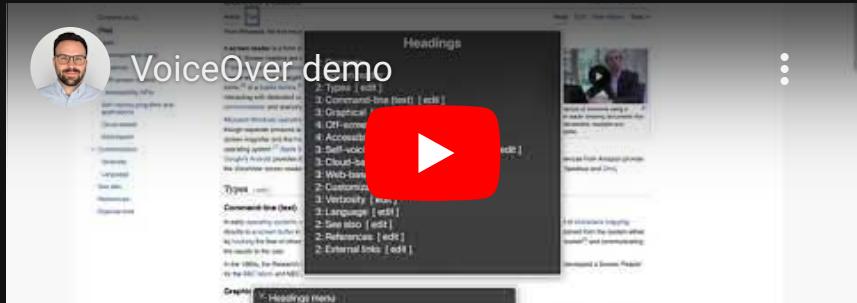
A screenshot of a Firefox browser window displaying the same Wikipedia article "Screen reader" but at 200% text-only zoom. The zoom setting is explicitly shown in the top right corner of the browser window. The text on the page is significantly larger and more spaced out compared to the default zoom. The sidebar on the left has expanded to show navigation links like "Contents [hide]", "Types", and "Customization". The main content area is also larger, and the video player is partially visible. The overall layout appears more spread out due to the increased text size.

Screen Readers

Most popular Screen Readers



VoiceOver on MacOS



source: assistivlabs.com

Responsive Design

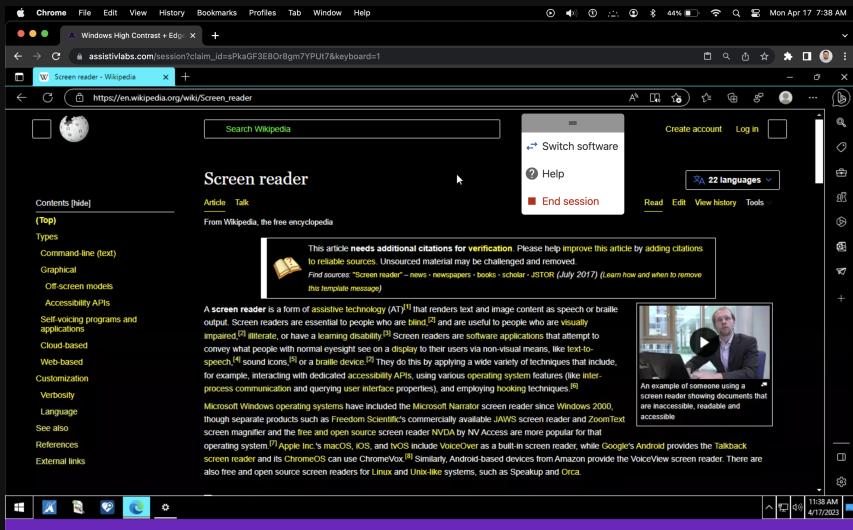
- Mobiles and Tablets also have screen readers!
- We saw that zooming might trigger "mobile" layout;
- Each major browser has a Responsive Design Mode.

The screenshot shows a Mac OS X desktop environment. A Safari window is open, displaying the Wikipedia article about Screen reader. The window title is "en.m.wikipedia.org". The status bar at the top of the window shows "414 x 736 (100%) | 3x | Safari — iOS 16.0 — iPhone". Below the status bar, there's a row of icons representing various Apple devices and screen resolutions: iPhone SE, iPhone 8, iPhone 8 Plus, iPad mini (7.9"), iPad (9.7"), iPad Pro (10.5"), iPad Pro (12.9"), 800 x 600, 1366 x 768, and 1920 x 1080. The main content area of the window shows the Wikipedia article titled "Screen reader". The article text discusses what screen readers are, their history, and how they work. It mentions assistive technology (AT), blind users, visually impaired users, illiterate users, learning disabilities, software applications, text-to-speech, sound icons, braille devices, accessibility APIs, operating system features, inter-process communication, user interface properties, and hooking techniques. A note at the top of the article page says "This article needs additional citations for verification. July 2017) Learn more". Below the text is a video player showing a man using a screen reader. The video caption reads: "An example of someone using a screen reader showing documents that are inaccessible, readable and accessible". At the bottom of the article, it says: "Microsoft Windows operating systems have included the Microsoft Narrator screen reader since Windows 2000, though separate products such as Freedom Scientific's commercially available JAWS screen reader and ZoomText screen magnifier".

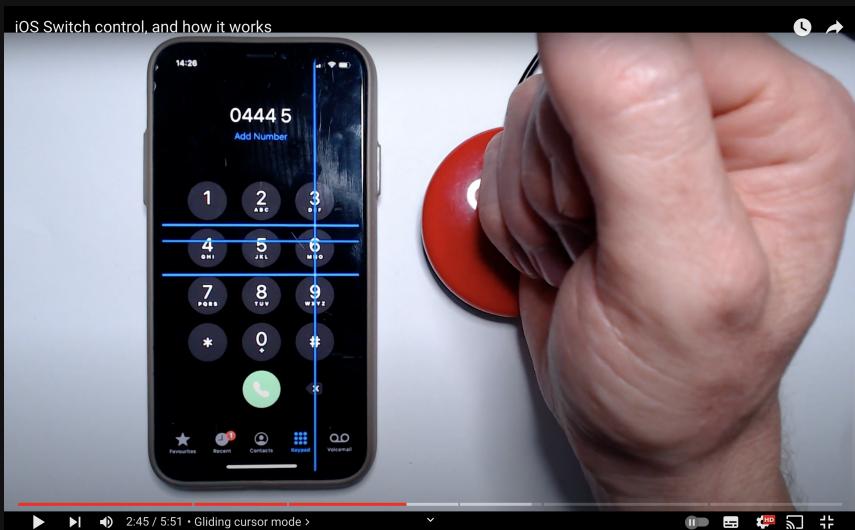
There are many others assistive technologies

For example:

High Contrast Mode on Windows



Switch device



Another one: Reduce screen motion on Mac OS.

Source: Assistive Tech Stuff on YouTube

AUTOMATIC TESTS

Static testing

- ESLint (see `eslint-plugin-jsx-a11y`) can help you to detect a11y issues while coding;

A screenshot of a code editor showing a tooltip over an `` element. The tooltip contains the following text:

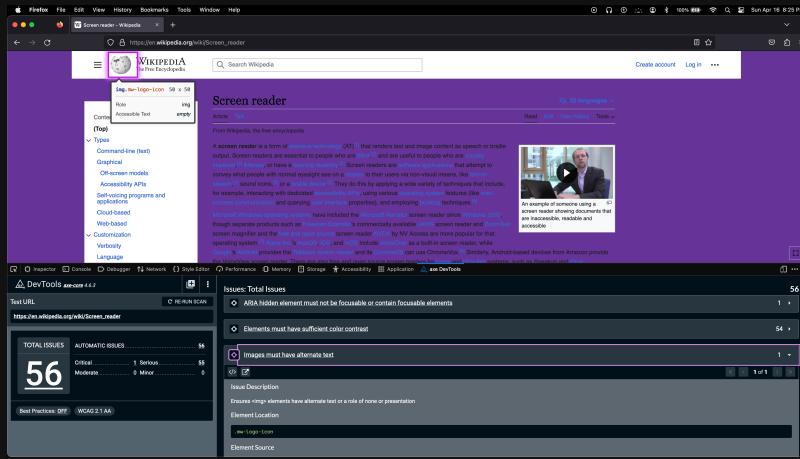
```
tu img elements must have an alt prop, either with
km meaningful text, or an empty string for decorative
im images. (jsx-a11y/alt-text) eslint
```

Below the tooltip, there are "Quick Fix..." and "Peek Problem" buttons.

- You can create custom rules for your Design System;
- TypeScript plus JSDoc could be used to achieve similar purpose;
- This does not test the DOM.

axe by Deque Systems

Via a browser extension (axe DevTools)



- Lighthouse has a similar feature;
- This is still kind of manual.

Via React Testing Library

```
import { render } from "@testing-library/react";
import { axe } from "jest-axe";

import { Button } from "./Button";

test("Button a11y", async () => {
  const { container } = render(<Button>Click me</Button>);
  const results = await axe(container);

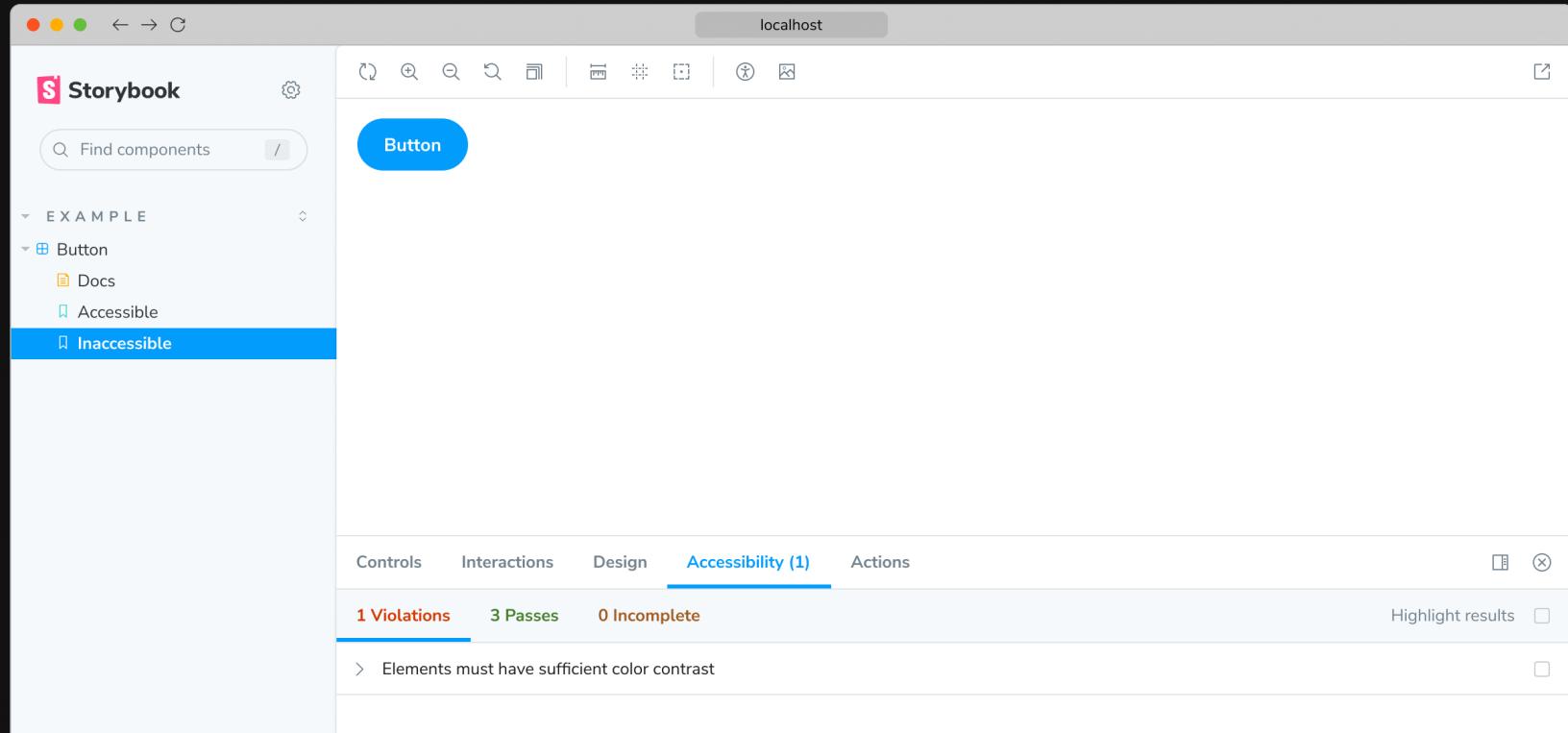
  expect(results).toHaveNoViolations();
});
```

- ⚡ jsdom won't catch contrast issues;
- Also available for E2E testing libraries such as Cypress or Playwright;
- It feels redundant.

axe can "only" catch up to 57% of a11y issues.

A11y tests with Storybook

- Storybook is a tool for developing, documenting, and testing UI components in isolation;
- Use `storybook-addon-a11y` to check the a11y of each story.



Test keyboard interactions

```
test('Menu keyboard interaction', async () => {
  render(<Menu />);
  const button = screen.getByRole("button", { name: "Dashboard" });

  expect(button).toHaveAttribute("aria-expanded", "false");

  await userEvent.keyboard("{tab}");
  await userEvent.keyboard("{enter}");

  expect(button).toHaveAttribute("aria-expanded", "true");
  expect(screen.getByRole("menuitem", { name: "Profile" })).toHaveFocus();

  await userEvent.keyboard("{arrowdown}");

  expect(screen.getByRole("menuitem", { name: "My account" })).toHaveFocus();

  await userEvent.keyboard("{Escape}");

  expect(button).toHaveFocus();
  expect(button).toHaveAttribute("aria-expanded", "false");
});
```

DASHBOARD

Resources

- Accessibility Myths
- Learn Accessibility
- A11ycasts with Rob Dodson
- Testing Accessibility by Marcy Sutton
- Practical Accessibility with Sara Soueidan

Conclusion

- Acknowledge your personal biases;
- Include a11y audits during design phase, code reviews, bug hunts...
- Consider all possible interactions in your automatic tests;
- Seek feedback from individuals with diverse abilities via companies such as Fable or by hiring them!