How to test your web app accessibility?

by Sylvain Hamann

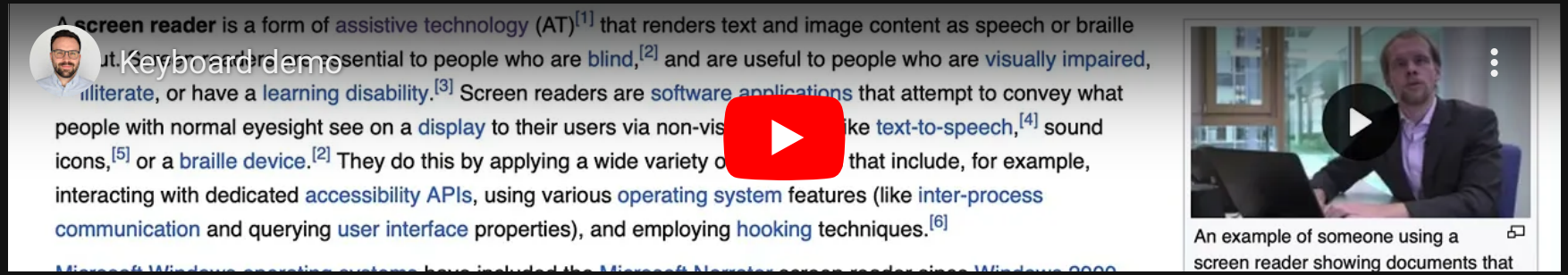
What’s Web Accessibility?

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

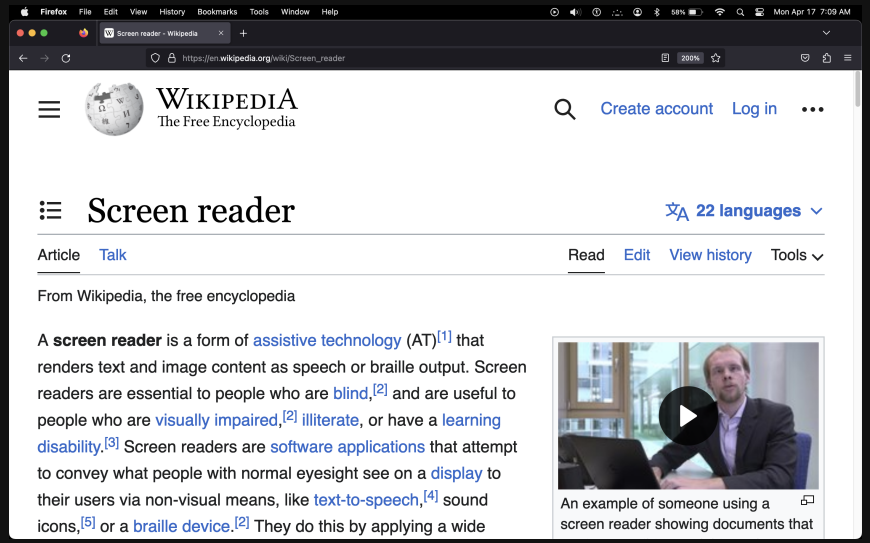
Via tab or shift+tab



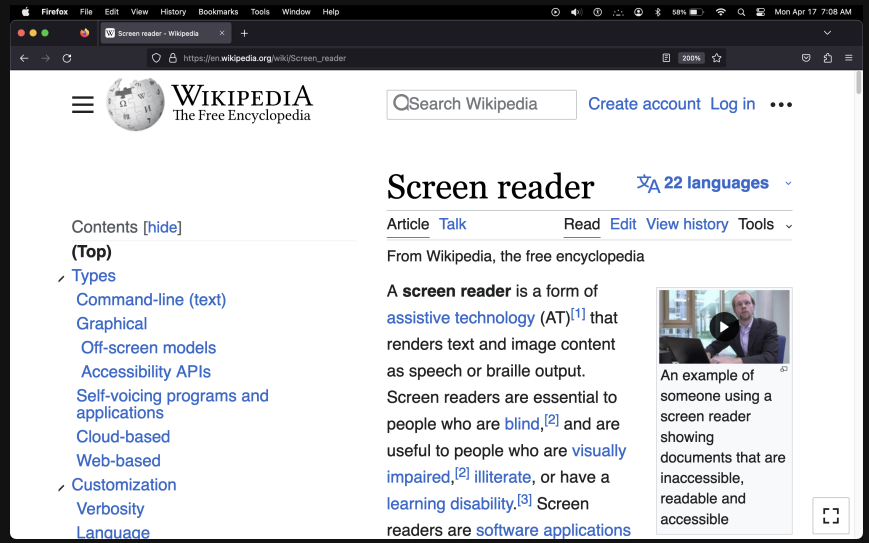
Zoom

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

Default zoom

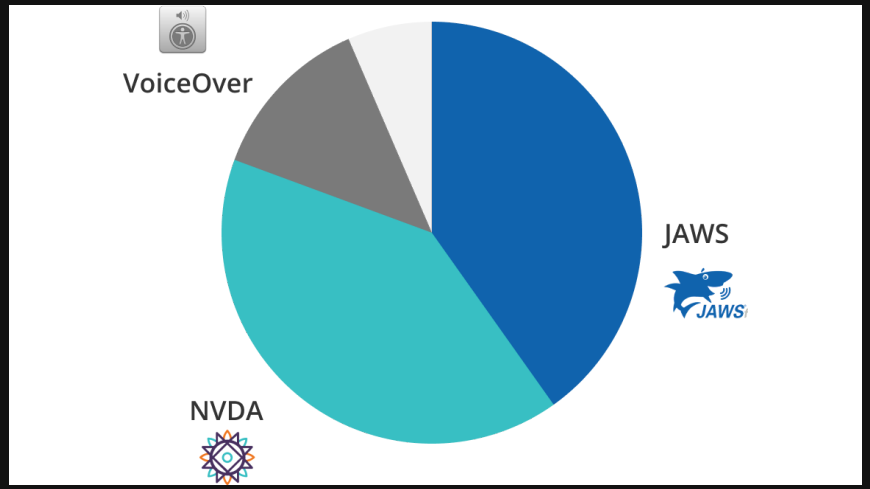


Text only zoom



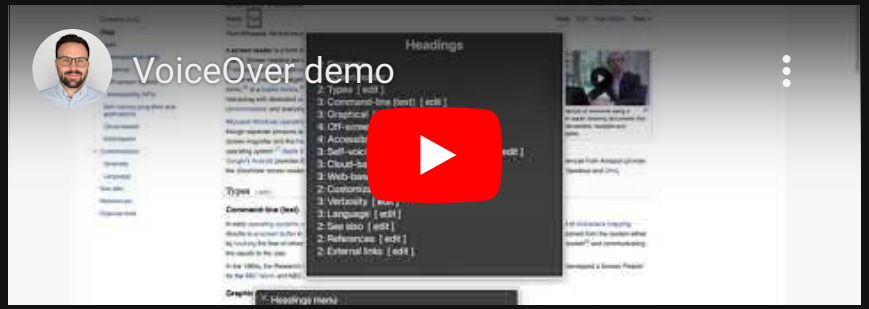
Screen Readers

Most popular Screen Readers



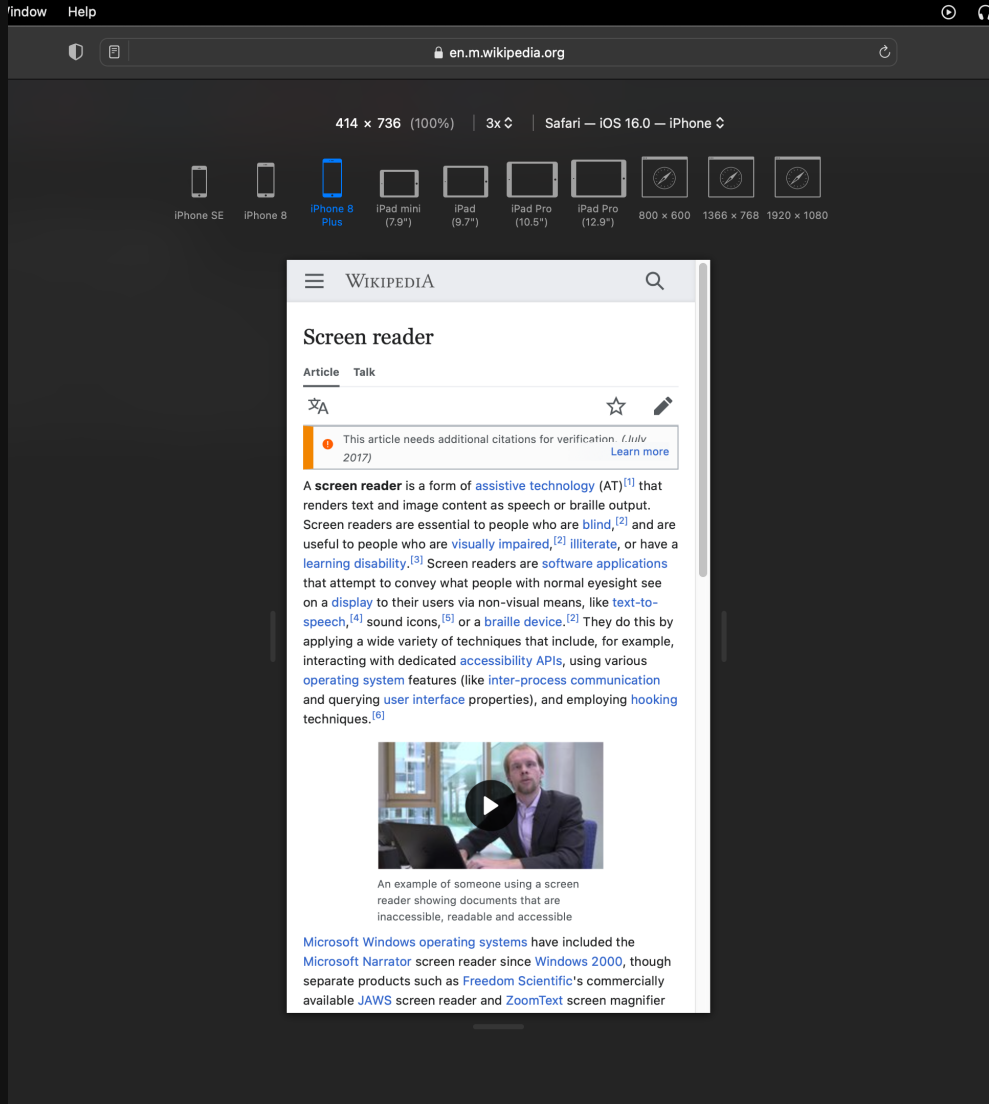
source: assistivlabs.com

VoiceOver on MacOS



Responsive Design

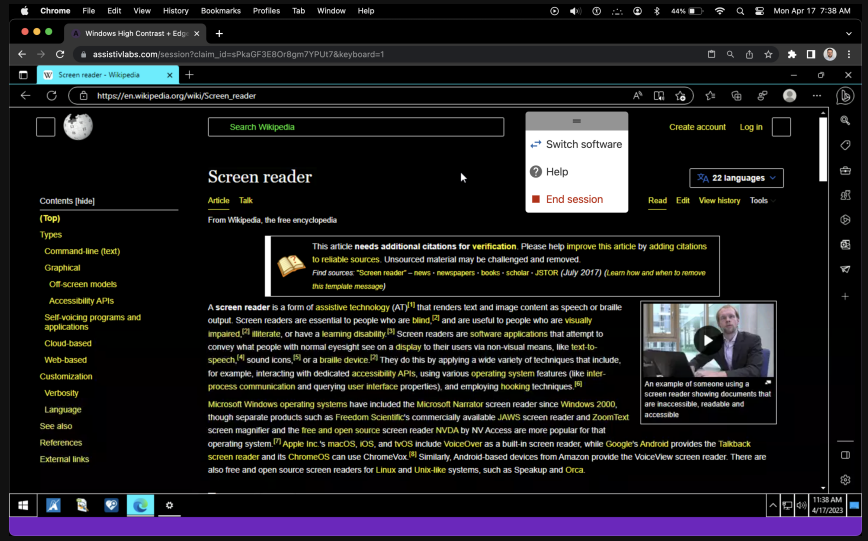
- Mobiles and Tablets also have screen readers!
- We saw that zooming might trigger "mobile" layout;
- Each major browser has a Responsive Design Mode.



There are many others assistive technologies

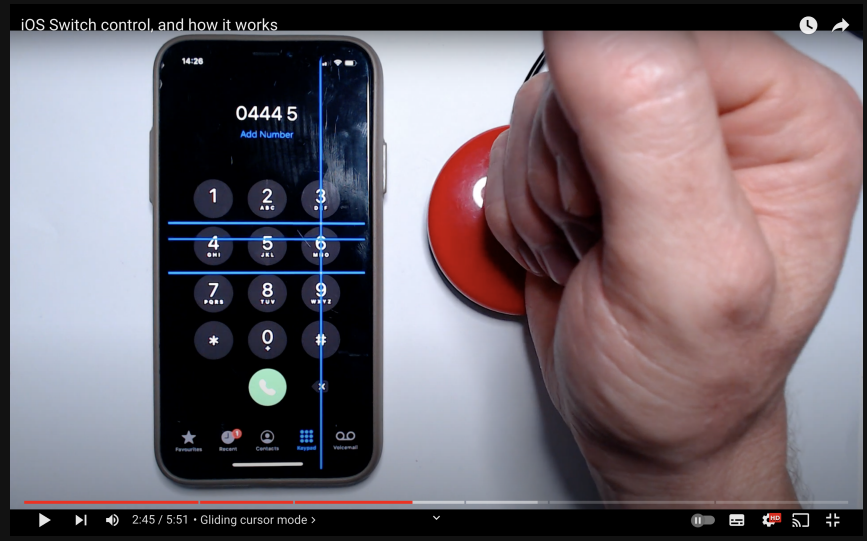
For example:

High Contrast Mode on Windows



Another one: Reduce screen motion on Mac OS.

Switch device

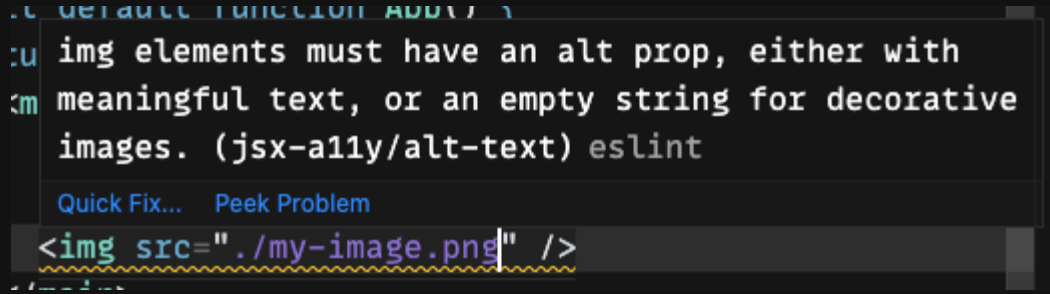


Source: Assistive Tech Stuff on YouTube

AUTOMATIC TESTS

Static testing

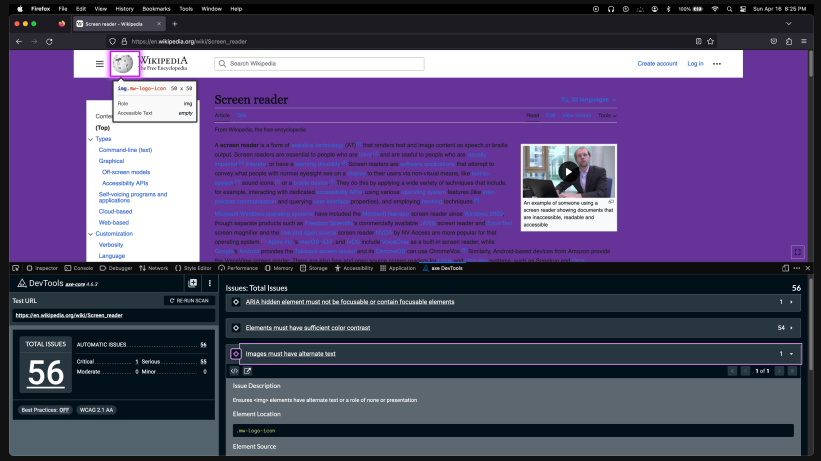
- ESLint (see `eslint-plugin-jsx-a11y`) can help you to detect a11y issues while coding;



- 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

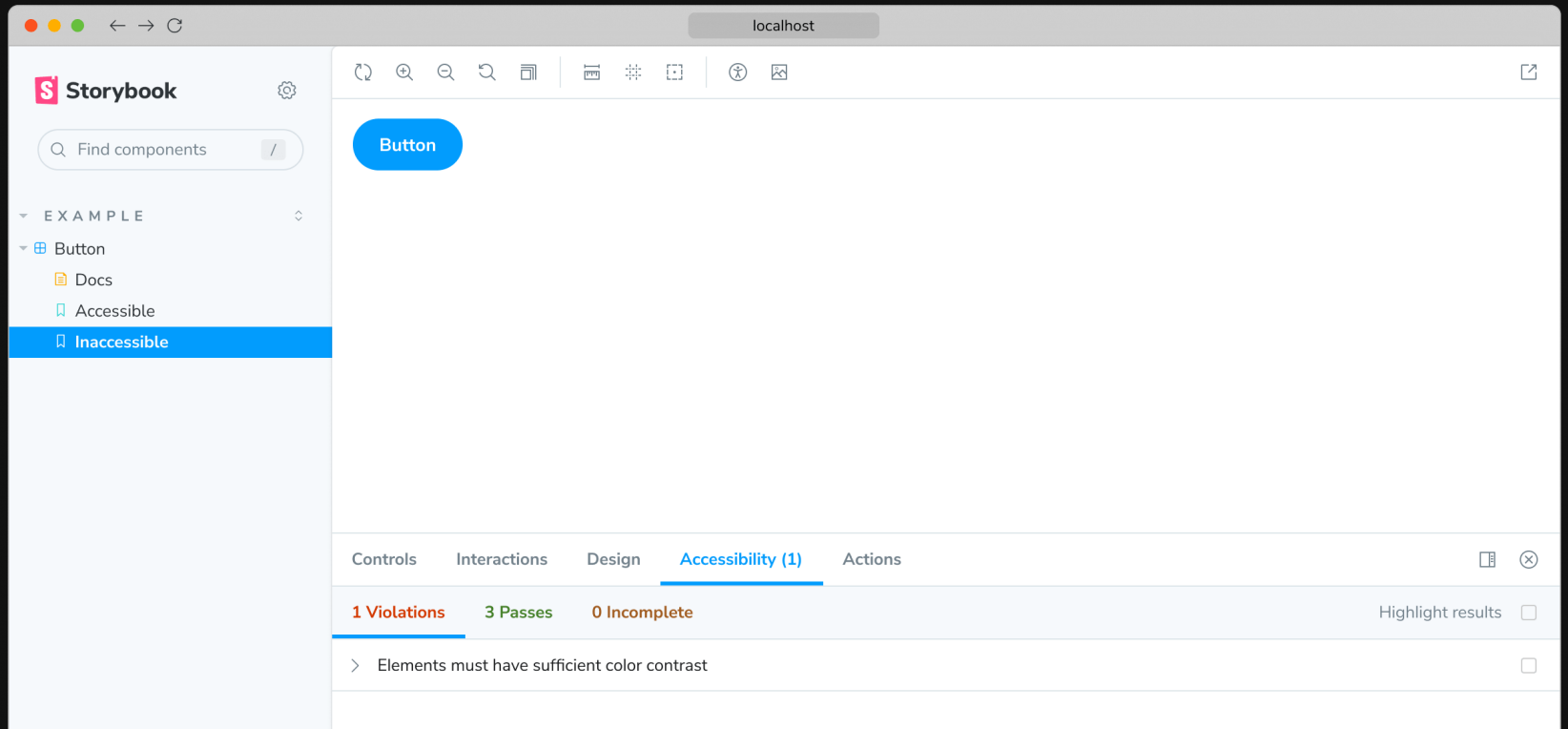
```
import { render } from "@testing-library/react";
import { axe } from "jest-axe";

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");
});
```

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!