How we made our products accessible

Challenges and Testing Approaches

SEMRUSH

Introduction

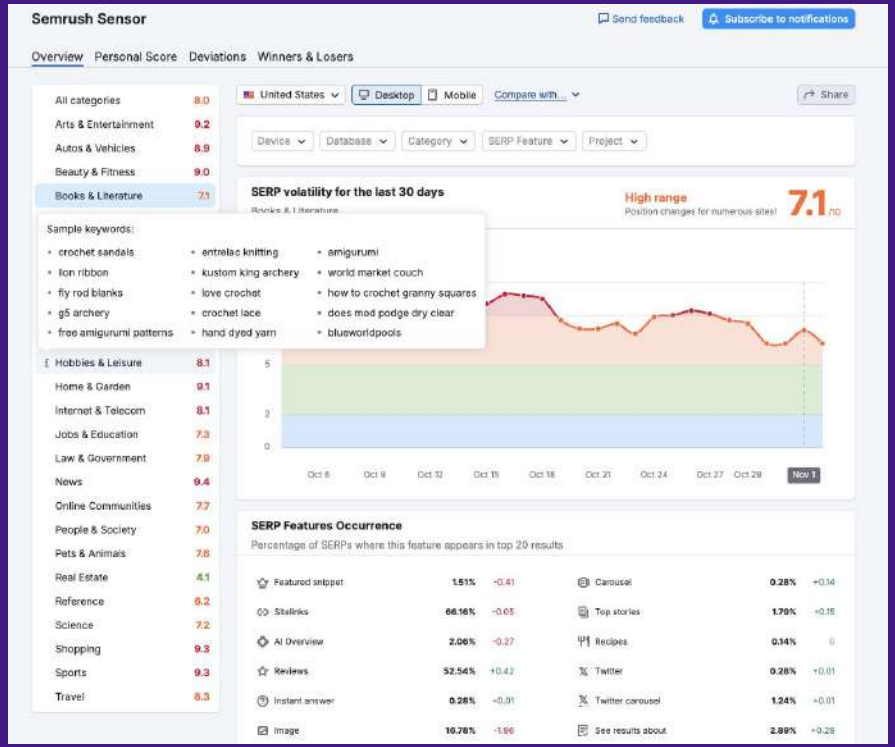
Daria Naidikova

Frontend Developer, Semrush

@daria-naidikova

What is Semrush?

We are a leading online visibility management software-as-a-service platform.



Where we started

What we had

- Projects
- Storybook

What we hadn't

- Experience
- A list of issues to fix
- No approach to testing accessibility

Key objectives

1. Creating a list of priority tasks
2. Develop an approach for accessibility testing.

Creating a list of tasks

Methods for identifying accessibility issues

- Browser Extensions
- Accessibility checklists
- Lighthouse

HeadingsMap

THE A11Y PROJECT

WAVE
web accessibility evaluation tool

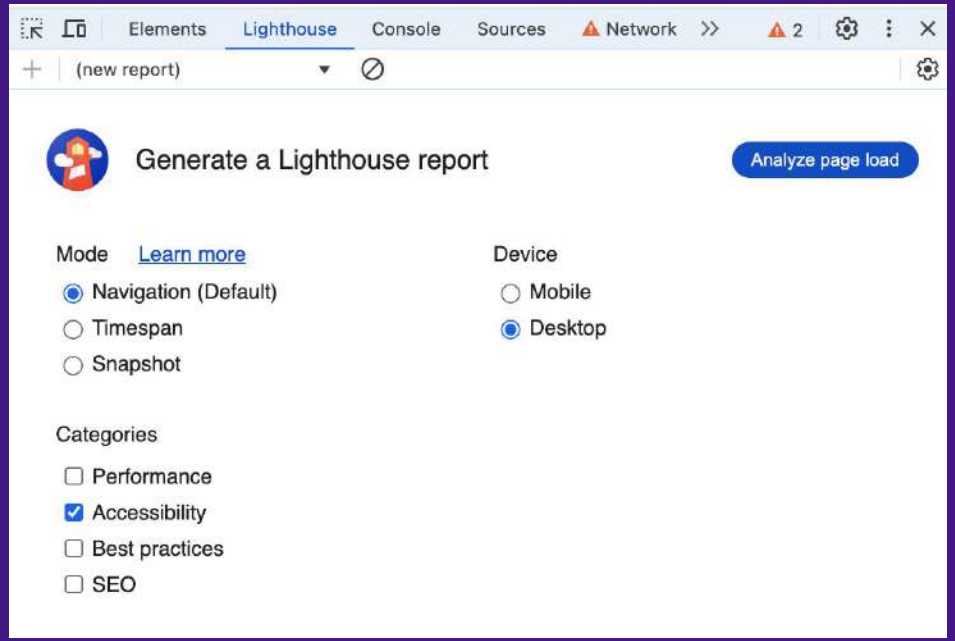
Accessibility Styles
Accessibility Tree

Lighthouse

ARIA Authoring Practices Guide (APG)

Lighthouse. Advantages

- No installation and setup needed.
- Useful report.
- Axe under the hood.



Lighthouse. Report example

93

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

NAMES AND LABELS

▲ Buttons do not have an accessible name

When a button doesn't have an accessible name, screen readers announce it as "button", making it unusable for users who rely on screen readers. Learn how to make buttons more accessible.

Failing Elements

button.___SButton_ornwh_gg_._size_m_ornwh_gg_._theme_primary-info_ornwh_gg_

Deque University

Buttons must have discernible text

Rule ID: button-name Ruleset: axe-core 4.10 User Impact: Critical Guidelines: WCAG 2.1 (A), WCAG 2.0 (A), WCAG 2.2 (A), Section 508.

How to Fix the Problem

Correct markup solutions

The `button-name` rule has five markup patterns that pass test criteria:

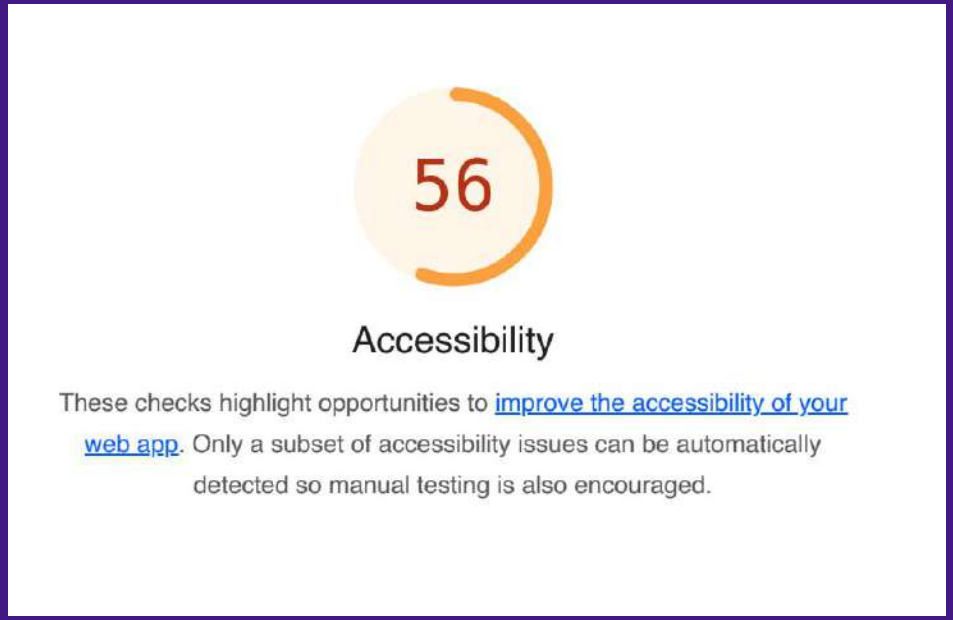
```
<button id="text">Name</button>

<button id="al" aria-label="Name"></button>

<button id="alb" aria-labelledby="labeldiv"></button>
<div id="labeldiv">Button label</div>

<button id="combo" aria-label="Aria Name">Name</button>

<button id="buttonTitle" title="Title"></button>
```

Lighthouse. Limitations

- Testing different states is challenging (Snapshot mode).
- No keyboard navigation checks.
- Difficult for automation in CI/CD.

Lighthouse. Our use cases

- Issues with accessible names (critical)
- Issues with roles and ARIA attributes (critical)



Manual testing for keyboard users

Step 1.

Took our test cases for the product.

Step 2.

Run tests using the keyboard.

Manual testing for keyboard users. Main issues.

- Buttons and links don't work when pressing Enter.
- Focus does not return to the trigger.
- Incorrect focus sequence.

Critical scope of a11y issues

- Tasks identified by Lighthouse
- Keyboard navigation issues identified during testing

What did we not include in the critical scope of tasks?

Screen reader testing

Screen Readers User Survey	
Screen Readers	% of period completion
NVDA	**65.6%**
JAWS	**60.5%**
VoiceOver	**43.9%**
Narrator	**37.3%**

https://webaim.org/projects/screenreadersurvey10

Screen reader testing

If a component is accessible via the keyboard and has no accessibility errors (correct roles, attributes, etc.), then it will be supported by the screen reader.

Our team

A11y testing

Before we start. Mysterious a11y

A C C E S S I B I L I T Y

A11Y

Before we start. Incredible axe.

- Accessibility engine for automated testing
- Can find 57% of WCAG issues automatically

Our a11y testing friends

- eslint-plugin-jsx-a11y
- storybook-addon-a11y
- axe-core + vitest-axe (jest-axe)
- @testing-library/react

eslint-plugin-jsx-a11y

Static checker for accessibility rules on JSX elements.

https://www.npmjs.com/package/eslint-plugin-jsx-a11y

eslint-plugin-jsx-a11y

Advantages

- Easy installation and configuration

Limitations

- Only catches errors in static code

https://www.npmjs.com/package/eslint-plugin-jsx-a11y

storybook-addon-a11y

This Storybook addon can be helpful to make your UI components more accessible.

https://storybook.js.org/addons/@storybook/addon-a11y

storybook-addon-a11y. Advantages

- Easy installation and configuration
- Convenient error highlighting
- Axe under the hood
- Test runner

main.ts

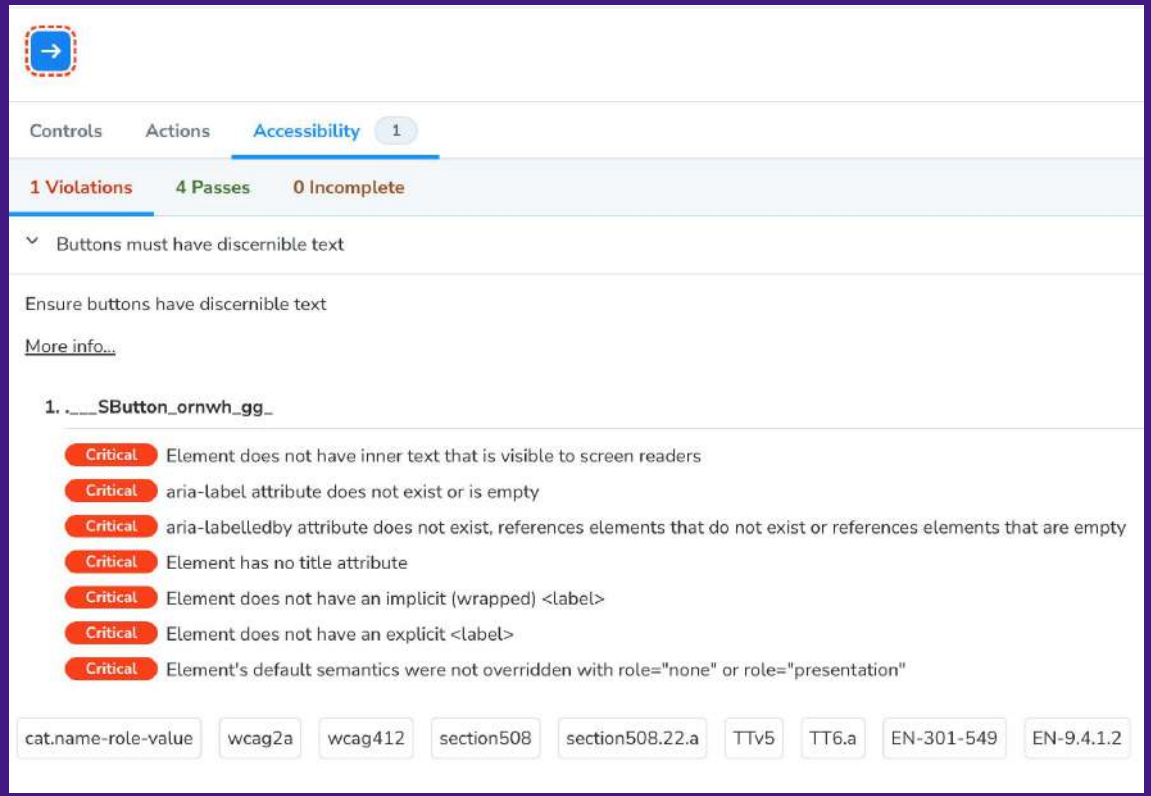
```
addons: [
  '@storybook/addon-essentials',
  '@storybook/addon-a11y'
],
```

preview.ts

```
parameters: {
  a11y: {
    element: '#storybook-root',
    config: {
      rules: [{ id: 'color-contrast', enabled: false }]
    }
  },
```

https://storybook.js.org/addons/@storybook/addon-a11y

storybook-addon-a11y. Advantages



storybook-addon-a11y. Advantages

```
import type { TestRunnerConfig } from '@storybook/test-runner';
import { injectAxe, checkA11y } from 'axe-playwright';

const config: TestRunnerConfig = {
  async preVisit(page : Page ) {
    await injectAxe(page);
  },
  async postVisit(page : Page ) {
    await checkA11y(page,  context: '#storybook-root',  axeOptions: {
      detailedReport: true,
      detailedReportOptions: {
        html: true,
      },
      axeOptions: {
        rules: {
          ['color-contrast']: {
            enabled: false,
          },
        },
      },
    });
  },
};

export default config;
```

https://storybook.js.org/docs/writing-tests/accessibility-testing#setup

storybook-addon-a11y. Limitations

- Storybook is needed

Axe-core + vitest-axe (jest-axe)

- **Axe-core** - accessibility engine for automated Web UI testing
- **Jest-axe, vitest-axe** - matchers for axe for testing accessibility

https://github.com/dequelabs/axe-core

Axe-core + vitest-axe (jest-axe). Advantages

- Easy installation and configuration
- Integration into the main testing process

```
import { configureAxe } from 'vitest-axe';

export const axe = configureAxe( options: {
  rules: {
    ['color-contrast']: {
      enabled: false,
    },
  },
  runOnly: [
    'wcag2a',
    'wcag2aa',
    'wcag21a',
    'wcag21aa',
    'best-practice'
  ],
});
```

https://www.npmjs.com/package/vitest-axe

Axe-core + vitest-axe (jest-axe). Advantages

shouldBeA11y.ts

```
import { axe } from "./configureAxe";
import { render } from "@testing-library/react";
import { test, expect } from "vitest";

export const shouldBeAccessible = (
  testName: string,
  component: React.ReactElement,
) => {
  test(`@axe ${testName}`, async () => {
    const { container } = render(component);
    const result = await axe(container);
    expect(result).toHaveNoViolations();
  });
};
```

https://github.com/dequelabs/axe-core
https://www.npmjs.com/package/vitest-axe

Axe-core + vitest-axe (jest-axe). Advantages

Button.test.tsx

```
import React from 'react';
import { describe } from "vitest";
import { shouldBeAccessible } from "../tests/shouldBeA11y";
import { IconButton } from "./Button";

describe("IconButton", () => {
  shouldBeAccessible( testName: "IconButton", <IconButton />);
});
```

https://github.com/dequelabs/axe-core
https://www.npmjs.com/package/vitest-axe

Axe-core + vitest-axe (jest-axe). Limitations

- Report lacks interactivity.

```
FAIL src/components/Button.test.tsx > IconButton > @axe IconButton
Error: expect(received).toHaveNoViolations(expected)

Expected the HTML found at $('button') to have no violations:

<button class="___SButton_ornwh_gg_ _size_m_ornwh_gg_ _theme_primary-info_ornwh_gg_" style="margin-right: 8px;" data-ui-name="Button" type="button" tabindex="0">

Received:

"Buttons must have discernible text (button-name)"

Fix any of the following:
  Element does not have inner text that is visible to screen readers
  aria-label attribute does not exist or is empty
  aria-labelledby attribute does not exist, references elements that do not exist or references elements that are empty
  Element has no title attribute
  Element does not have an implicit (wrapped) <label>
  Element does not have an explicit <label>
  Element's default semantics were not overridden with role="none" or role="presentation"

You can find more information on this issue here:
https://dequeuniversity.com/rules/axe/4.10/button-name?application=axeAPI
 ❯ src/tests/shouldBeAlly.ts:12:20
     10|     const { container } = render(component);
     11|     const result = await axe(container);
     12|     expect(result).toHaveNoViolations();
       |                   ^
     13|   });
     14| };
```

https://github.com/dequelabs/axe-core

@testing-library/react for keyboard testing

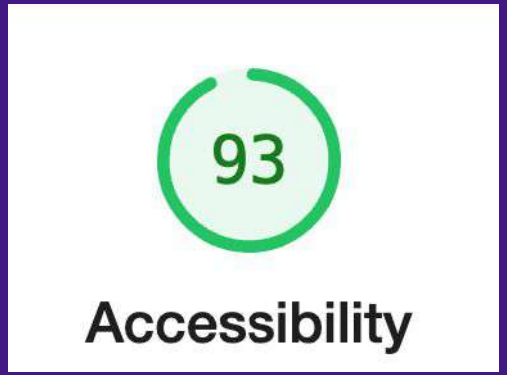
```
import React from 'react';
import { describe, test, vi, expect } from 'vitest';
import userEvent from '@testing-library/user-event';
import { render } from "@testing-library/react";
import { shouldBeAccessible } from '../tests/shouldBeA11y';
import { IconButton } from './Button';

describe('IconButton', () => {
  shouldBeAccessible( testName: 'IconButton', <IconButton />);

  test('onClick fn should be called via keyboard', async () => {
    const onClick = vi.fn();
    const user = userEvent.setup();

    render(<IconButton onClick={onClick} />);
    await user.keyboard( text: '[Tab]');
    await user.keyboard( text: '[Enter]');
    expect(onClick).toHaveBeenCalled();
  });
});
```

https://testing-library.com/docs/react-testing-library/intro/

Conclusion: What has been achieved?

- Covered the main accessibility issues
- Monitor regression after library updates
- Lighthouse a11y score increased

Conclusion: What has been achieved?



Conclusions: what's next?

- Testing by accessibility testers

Thank you and follow us!

@semrush

@semrush

@semrush

@semrush

@semrush

@semrush