Table of Contents

Introduction

1 Research goal and methodology

1.1 Research goal

In software development linters and tests are used to ensure all new added code follows the quality and best practice standards and does not break anything. A similar quality gate could be set up for ensuring conformance with accessibility standards. This would act as a gatekeeper to ensure that if there are any detectable problems in the added code then it can not be deployed until these have been fixed.

In this master thesis I intend to test if automated accessibility tests can potentially help improve the overall accessibility and awareness about it. Most testing tools also provide links and instructions about the issue that was detected. This could help improve the developers' knowledge about accessibility issues and how to fix them.

Many companies have design systems to help maintain consistency both in what the end users experience and how code is written. This seems like a good place to start with establishing a basic accessibility standard. Making changes there would have the widest impact. Most accessibility checkers are intended to be used for the whole page or website. I want to run these tests on the individual components in out component library.

There are definitely limitations to testing isolated components, but how I see it is similar to unit tests and end-to-end test in software development. Unit tests to ensure a small specific part of the code - unit works as intended. End-to-end tests are run when all these parts are combined to make up the end product. These tests are imitating what would happen when end users are using the product. I see these accessibility tests being run on isolated components being similar to unit tests - each one making sure that that particular element has no issues. Then when the components are used to make up a whole page there should already be fewer issues that need to be dealt with.

Many studies have been done that compare different accessibility testing tools (**Duran2017**; Alsaeedi, 2020; Ismailova & Inal, 2022; Rybin Koob et al., 2022; Sane, 2021; Vigo et al., 2013). I will take a look at the results of these studies and test out enough tools on my own to find one that would be usable for my purposes, but I am not intending to compare them as a part of this work.

I intend to test out and automated accessibility tool in my workplace on our React component library - Convention UI React (CUI). This will probably pose some limitations on the tool I can use. The selected tool should work with our current tooling and not disrupt the current development flow.

1.2 Research questions

RQ1: How good is the knowledge about accessibility standards, tools and best practices in the company currently?

RQ2: What kind of errors can be caught by running automated accessibility tests on a component library?

RQ3: To what extent can integrating automated testing to a component library's development pipeline help improve its compliance with WCAG standards?

RQ4: What are the biggest problems of integrating automated accessibility testing to a component libraries' development workflow?

1.3 Methodology

****TODO:** Write about methodology

- Literature review
- Survey
- Case study
- Statistical analysis of survey and case study results
- User centered research?

2 Background

2.1 Web accessibility

Accessibility is about providing equal access to goods and services to everyone regardless or their age, gender or disabilities. We all have different abilities, some run faster than others, some see more clearly and some might not be able to hear sounds. Everyone who designs the physical and virtual environment around us should consider these different limitations.

These principles can be applied to any various fields. For example in architecture it could be designing buildings in a way that that can be accessed by people who can walk on their own as well as the ones who need to use a wheelchair. In digital products it is more often related to the senses we use to consume content. Everything could be equally accessible whether I want to consume the information using my vision or ability to hear.

”The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect,” said Tim Berners-Lee, W3C Director and inventor of the World Wide Web(World Wide Web Consortium, 1997).

WHO estimates that 1.3 billion people – or about 16% of the global population – experience a significant disability and this number is growing. Disability is a part of being human and persons with disabilities are different from each other. (World Health Organization, 2022) We can also talk about temporary disability - this might be a mother who can only use one hand because she is holding a child in the other or someone who broke an arm and can't use it until it heals. The limited ability might be very similar to someone who has a permanent condition.

Web accessibility considers auditory, cognitive, neurological, physical, speech and visual disabilities - everything that might affect someone's ability to access the Web and people using different devices, who have reduces abilities because of aging, temporary disabilities, situational limitations or are using limited or slow internet. (Henry, 2022)

2.1.1 Standards

The standards for considering people with different abilities and making web content accessible for different ways of consuming are defined in Web Content Accessibility Guidelines (WCAG) (Kirkpatrick et al., 2018). It is that is developed by Web Accessibility Initiative (WAI) that is a part of World Wide Web Consortium (W3C).

WAI gives and overview of the laws and policies around the world. 25 out of 40 listed on the page are based on WCAG 2.0 or WCAG 2.0 derivative (Mueller et al., 2018). The most notable ones are the accessibility standard in European Union EN 301 549 V enforced by Web and Mobile Accessibility Directive (Mueller et al., 2017a) and Section 508 of the US Rehabilitation Act of 1973, as amended in United States (Mueller et al., 2017b).

These two affect a big part of the web, because they regulate how content should be presented in big parts of the world. Both these standards are based on WCAG 2.0 (Mueller et al., 2017a, 2017b). This makes WCAG 2.0 the most influential web accessibility standard.

The current version is WCAG 2.1 and the next version 2.2 is schedules to be finalized in April 2023 (Henry, 2023). In this thesis I will test against WCAG 2.1, because it is the latest published version of the most widely used standard.

The case study will be run in Pipedrive - a private company that develops a sales CRM (Customer relationship management) software. The biggest markets it operates in are us and Europe**TODO: Verify this and even tough there are now compulsory accessibility regulations that should be followed because it is a private sector company in EU and US the regulations that apply to public sector are based on WCAG 2.0. So following a widely used and reliable standard that is the same or very similar to regulations in the markets the company operates in makes sense.

WCAG is intended for developers of web content, authoring tools or web accessibility evaluation tools and others who need a standard for accessibility. The 4 principles of accessibility addressed in the guidelines are:

1. Perceivable - information and user interface can't be invisible to all the users senses,
2. Operable - user can interact with the interface,
3. Understandable - user should understand the information and how the user interface operates
4. Robust - content should be robust enough that it can be interpreted reliably by a wide variety of user agents.

Under each principle is a list of guidelines that address that principle. Under each guideline there are Success Criteria that describe specifically what conformance to it means. Each Success Criterion is a statement that will be either true or false for a specific Web content. (Accessibility Guidelines Working Group (AG WG) Participants, 2022)

2.1.2 Rules format

W3C Accessibility Guidelines Working Group has developed Accessibility Conformance Testing (ACT) Rules Format to provide developers of evaluation methodologies and testing tools a consistent interpretation of how to test for conformance with accessibility requirements like WCAG. The format describes both manual and automated tests. The aim of this is to make accessibility tests transparent and results reproducible. (Fiers et al., 2019)

Accessibility testing tools check in the HTML provided meets the requirements defined in WCAG. These requirements are different for each element, but they might be also combined

and include more criteria. Each element needs a specific set of requirements to be checked.

ACT Rules include atomic rules that define and element to be tested for a single condition and composite rules that can combine multiple atomic rules to determine if a single test subject satisfies an accessibility requirement. Each rule defines when it should be applied. For atomic rules this could be HTML tag name, computed role or distance between two elements for example and for composite rules it is a union of the applicability of the atomic elements is combines (Fiers et al., 2019)

2.2 Accessibility evaluation

Different evaluation methods are used to determine if a website is accessible. These include expert review, user testing, subjective assessment, screening techniques or barrier walkthrough. Each method has its pros and cons depending on resources, available experts and other factors.

Expert review, also called, conformance, standards or guidelines review or manual inspection is most widely used. It means checking if a page satisfies a checklist of criteria. The results depend on the evaluators opinions and depend on the chosen guidelines. For this method to be effective skillful evaluators are needed. (Brajnik, 2008)

Barrier walkthrough is a technique where the evaluator has to assess how severely a number of predefined barriers impact the user to achieve their goal on the website. Accessibility barrier can be any condition that makes it difficult for people to achieve a goal. Which method is suitable depends on a number of factors including availability of skilled auditors and resources. (Brajnik, 2008)

Screening techniques consist of evaluating a website by using an interface in a way that some sensory, motor or cognitive capabilities of the user are artificially reduced is called. Subjective assessment is based on a panel of users exploring a website themselves and giving feedback on what worked and what did not. User testing means conducting usability tests with disabled people while adopting think-aloud method to get feedback on their experience. (Brajnik, 2008)

Evaluation with experts are very dependent on the knowledge and experience of the evaluators. Research into the effect of expertise on web accessibility evaluation conducted by Brajnik et al. shows that when web pages are evaluated with non experts we see a drop in reliability and validity. Even with experts the result will vary a bit, but the results should even out when at least 3 experts are used. For the same level on reliability at least 14 evaluators that are not considered experts in the field on web accessibility are needed. (Brajnik et al., 2011)

2.2.1 Tools for accessibility testing

There are software programs and online services that help determine if web content meets accessibility guidelines. These tools may support various standards. They might be browser extensions, command line tools, desktop or mobile applications or online services. The feedback they give might be in the form of a report, inserting temporary icons and markup to the scanned page to highlight issues. (Abou-Zahra et al., 2017)

Browser extensions and online tools usually evaluate one page at a time. This might work well for small websites and one time audits, but can be quite ineffective to use as a continuous long term solution. Command-line interface (CLI) tools need more setup, but can potentially be seamlessly integrated to development workflow.

2.3 Automated accessibility testing

Accessibility testing done by software programs is called automated testing. By this I mean programs that can be set up to perform checks automatically with minimal manual effort. Mostly these are CLI tools that can be integrated to run together with other tests. The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web (MDN contributors, 2023). Program scans the rendered DOM of a website against accessibility standards, like WCAG to find errors. These errors are reported back with reference to code that produced the error.

Continuous integration (CI) is a software development practice where members of the team integrate their work frequently and each integration is verified with and automated build that includes test to detect and errors as quickly as possible. This is believed to help develop high quality software more rapidly. One of the practices in CI is making your code Self-Testing - adding a suite of automated tests that can check a large part of the code base for bugs. These tests need to be easy to trigger and indicate of any failures. (Fowler, 2006)

This last principle could be applied to accessibility. It goes well modern software development principles. Test will not catch everything, but they will catch enough to make it worthwhile. After the initial setup they should not need a lot of maintenance and will be run every time someone contributes to the codebase. This can act as a very effective gatekeeper for any potential accessibility issues.

According to the survey conducted by Level Access about the state of digital accessibility 67% of organizations that practice continuous integration also include accessibility tests. This has gone up from 56% reported in 2021. Organizations who have an accessibility program that is in the early stages (2-6 years) or who have IAAP-certified personnel are more likely than average to have implemented CI testing process that includes accessibility. (Level Access et al., 2023)

These kinds of test can detect only a part of all the possible violations. UK Government Digital Service Accessibility team compared 13 automated tools performance in 2018 on a page that had 142 deliberately introduced accessibility issues and found that they found 13-40% of issues. Abbott compared two most popular accessibility tools that can be used in CI using the same deliberately inaccessible site in 2021 and reported that axe-core caught 27% and pa11y 20% (Abbott, 2021). Vigo et al. compared 6 different tools in 2013 and found that they found 23-50% of WCAG 2.0 Success Criteria. (Abbott, 2021; GDS Accessibility Team, 2018; Vigo et al., 2013) The tools and WCAG standards have changed during this time, but the common conclusion to most of similar comparisons is still that most tools will be able to detect errors on 20-30% or WCAG Success Criteria. **TODO: Find citation

Axe-core promises to find on average 57% issues, which is significantly higher than other tools on the market. (Deque Systems, 2023). They claim that the current statistics is founded on an inaccurate definition of accessibility coverage - percentage of individual WCAG Success Criteria. In reality some types of issues are found much more frequently and the issues found by automated tests form a higher percentage of all issues compared to those discovered by manual detection. They suggest that the coverage should be percentage of the number of issues found on a site. They conducted a study where they compared 2000 audit results from testing pages with axe-core and manual testing methodology. The average percentage of the number of issues detected by axe-core for each data set is 57.38%. They also say that looking at the results with current, in their opinion inaccurate, definition coverage would be in the range of 20-30%. (Deque Systems, 2021)

There definitely is a limit to what can be detected no matter how we define the coverage. More testing will always be required to ensure that the content is fully accessible. These kinds of tests can be set up to run automatically, and they provide measurable results. This means it a good way to monitor compliance with WCAG rules consistently without any extra effort. This can help avoid unwanted changes and show easy fixes in your code.

2.4 Component libraries

Component Driven Development (CDD) is a development methodology that anchors the build process around components (Coleman, 2017). In 2017 Coleman called CDD the biggest trend in user interface (UI) development.

A component is a well-defined and independent piece of UI, like a button, checkbox or card. This approach correlates well with other similar widespread principles that promote modularity in software development like atomic design and micro-frontends. Designing each component as a standalone unit improves maintenance, reusability, testing, shortens the learning curve and makes development faster. (Ella, 2019)

The key to success here is separation of concerns and isolating a logical piece so that it can be

worked without distractions. It promotes concentrating on details and refining the element as much as possible. These pieces can be combined into more complex component if needed and when combined they can make up views that become the whole site or webpage.

A component explorer is a tool that is often used in this approach. Working on a single component by manipulating the entire webpage or app to a certain state can be difficult. Component explorer is a separate application that showcases the components in various states. This allows developers to test a given component in all the states that have been defined in isolation and makes it easy to build one component at a time. It also makes it easier to go through all the possible states in one component and promotes reusability of these elements. (Coleman, 2017)

Bit, Storybook and Styleguidist are popular tools used in component library development. Bit lets you pack the bundled and encapsulated components and share them to the cloud where your team can visually explore them. Storybook supports multiple frameworks and provides a rapid component development and test environment. The environment also allows you to present and document your library for better reusability. Styleguidist is useful for documentation and demoing different components. (Ella, 2019)

A collection of multiple components that is shared to be reused is called a component library. Design systems include values and principles along, branding, guidelines and all the building blocks and design patterns to create a successful product or service. It governs the design process in the organization. A component library or UI kit, UI library, UI component library is one of the building blocks of a design system. (Ramotion, 2022)

Most big companies have a design system that includes a component library. Often these might be published with an open source code, making it possible to reuse and extend on them. Some of the more known ones include Material-UI (Material UI SAS, n.d.), Adobe Spectrum (Adobe, n.d.) and Bootstrap (Collings et al., n.d.). They help keep things consistent across multiple projects by providing small building blocks that can be used to create complex designs. I can be built with HTML or by using a framework like React or Vue.

Using a component library provides consistency and better quality. These components can be polished over time to provide the best user experience to end users when they are integrated to the final product. The effort that can be put into developing a component that will be used in multiple projects is bigger than what would be reasonable for a single use case.

3 Research

The research conducted for this thesis can be divided into 6 steps:

1. Researching automated testing tools
2. Pre case study questionnaire
3. Setting up automated testing tool in out companies component library
4. Manual accessibility audit
5. Comparing manual and automated testing results
6. Post case study questionnaire

The fist things was to look at different automated accessibility tools that are available, test them out and see what would fit our purpose the best. The tool needed to be easy to use for everyone and not block development. The next step was to send out a questionnaire to better understand the current approaches towards accessibility among the people who work on it the most. After that an automated accessibility tool - addon-a11y (Chromatic, n.d.-a) was added and announced in relevant channels to raise awareness about it. In parallel to observing how the tool is being used we preformed a manual accessibility audit in the same component library. These results where then compared with the automated testing results that we got form the same version on the library. As the last step I sent out a questionnaire to gather information about how the tool has been adopted. To get more details I also did some interviews with developers and designers I knew had used the tool during that time.

Pipedrive has been developing a sales CRM using mostly typescript and React. Accessibility has never been high priority and at this point it is not very easy to get started. We have a design system and a React based component library to keep the look and UX consistent. This seems like a good place to start with solving accessibility issues. The library is used widely in the company and developer from different teams contribute to it. If a button in the reusable library gets fixed 95% of the buttons in the web app that our customers use should be improved.

This should not be taken as a way to solve all accessibility problems, but a good first step to get started. Making changes in a reusable UI library should have a wide impact on the products overall accessibility and without reaching some basic level there first it could be hard to start testing views in the final product.

3.1 Current state of awareness about accessibility in the company

First I sent out a survey in our company Slack channels (see figure 1) to understand what is the knowledge and general approach on web accessibility in the company. It was shared in

one front end developers channel with 212 members, designers channel with 73 members, accessibility channel with 31 members and in out component library channel with 124 members. Some people might also be in more than one of these channels. The aim was to reach people in the company who would be most likely to be using these tools and who would be likely contributors to the library. There were 7 questions and some of they also included a field for free text to give more details on the subject if they wanted.

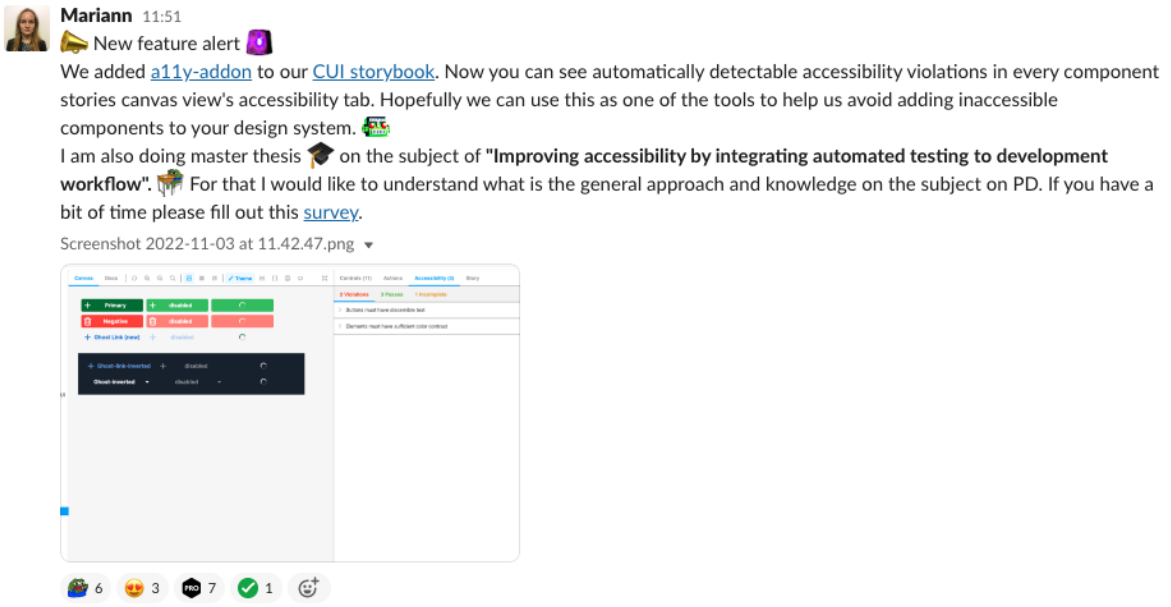

Figure 1: Message in company Slack announcing adding accessibility tool and inviting people to reply to survey.

In total 20 people replied to the survey - 6 designers and 14 developers, including one engineering manager. This does not give a full overview of the company, but it should give a good insight into what general opinions regarding accessibility might be. It is likely that developers and designers that are more involved with our component library and/or interested in accessibility where more likely to respond.

The results show that 10% of people who responded think their knowledge on accessibility is very good, while most think that knowledge level is average. 35% of responders know where do find resources about accessibility standards, 15% don't know and 50% know, but think they need more. They don't see that accessibility has high priority in the company currently, but at the same time 40% of responders think that following accessibility standards should be prioritized. There were several comments in the free text sections expressing saying they appreciate this subject being opened.

3.2 Adding automated accessibility tests to component library

The next step was adding some automated test to our component library development workflow and observing their usefulness. The aim was to find something that we can integrate to our development workflow without a need to learn a new tool or add unnecessary complexity.

We use Storybook for our UI development. It is an open source software for UI development tool that allows you to work on one component at a time (Chromatic, n.d.-b). It allows us to render isolated UI components without integrating them to the final product right away. Our developers use it to test out the components they are developing locally. We also have a version of storybook available for anyone in the company to see with all the components. If this is the first place where elements will be rendered it seems logical to try to find a way to start testing them there.

To show a component you need to write a story - this means use it the way you would use it in the real world, and it will be rendered in a browser inside Storybook UI. It also has a sidebar for navigating between different examples, controls for additional tools and documentation. This means using a browser extension for accessibility tests would also test the UI around the actual example.

Storybook has a wide ecosystem of add-ons and one of them is addon-a11y. It uses Deque's axe-core to audit the HTML rendered for an isolated component (Chromatic, n.d.-a). Axe is an accessibility testing engine for websites and other HTML-based user interfaces. It includes WCAG 2.0 and 2.1 on level A and AA and promises to catch 57% of WCAG issues on average. (Deque Systems, 2023) This should be taken with a grain of salt, because we are intending to test isolated element and not the whole webpage and other studies comparing accessibility testing tools effectiveness have found the coverage to be lower **TODO: needs citation.

This seemed like the best solution in our situation so addon-a11y was added to our component library's storybook. The tool is visible in the sidebar of every example. It shows all the checks that it has passed, all the violations that where found and any issues that could not be checked and might need manual testing. Every rule listed there also has reference to the HTML node that had the violation, explanation and links to Deque webpage with examples. This should be very useful in understanding and fixing these issues.

It does not generate a report of all the issues found across the whole library for this another tool was used that will go through all the examples and generate a summary of all the violations found.

The rules format that axe-core used is developed by Deque Systems and is an adoption of ACT rules format developed by WAI (Fiers, 2017). They have a set WCAG of that can be valuated in a fully automated way. They are divided by WCAG standards version (2.0, 2.1,

2.2) and Level (A & AA, AAA) and they also have some rules for industry accepted best practices that improve the user experience, but might not conform to WCAG success criterion (Fiers & Lambert, 2023).

Data gathered from automated testing report:

- How many occurrences in the accessibility violations report. This will show how many violations where detected from all the examples. Might contain the same issue multiple times.
- How many unique issues will only count different violations for each component.
- How many passed checks – this together with violations will show how many things where tested for each component – Most component have more than one story – the list will contain all different passed checks listed
- How many valid checks – are the passed checks relevant to the component – only count the ones that are related to the component that the example is about.

3.3 Manual accessibility audit

The second part is conducting a manual accessibility audit in the same library and comparing the results with the automatically generated test report results to determine what are its strengths and weaknesses and if testing isolated components poses any limitations.

3.3.1 Methodology for manual audit

In order to look at each component in isolation we used storybook here too. The accessibility add-on had already been installed, and we looked at the violations reported there too. We worked in a team of 4 people - 3 designers and 1 developer. We created a task for each component - 53 tasks in total.

We checked violations in accessibility add-on panel (see figure **TODO: add figure), tried using only a keyboard to navigate, tried using a screen reader to navigate. Each component had 1 or more examples. As many as was relevant to get the whole picture where looked at.

In the beginning of the audit we tested an add-on in storybook for mocking a screen reader (**TODO: Reference for this add-on and maybe and article about the difficulty of using a screen reader). It had the option to show the output a normal screen reader would play as audio as text. Initially it seemed like convenient solution with rather reliable results, but further investigation revealed that the output was very different from what an actual screen readers. For the rest on the audit we used Voice over - the MacBook built in screen reader, because it was available to us. There was and initial learning curve, but after that it went

quite smoothly. All the component that had already been tested with the faulty add-on where looked over again using Voice over.

The audit results where documented in a table. We approached the problems for the users perspective and looked at what issues different types of users might encounter. We separated them to 3 sections:

1. Mouse user issues
2. Keyboard user issues
3. Screen reader user issues

We see this separation as a good way to prioritize fixing the issues in the future. Mouse user is the user we are considering in all of our development currently. The issues they would encounter should be most critical. This category includes a lot of visual, color contrast and image text issues.

The second type of user would encounter all the issues from the first category plus everything that is unreachable to them by using a keyboard. We looked at what functionality is or should be working when you use a mouse and tried to do the same things by only using a keyboard.

The third user was imitated by using a screen reader. The prerequisite for this was keyboard usability - if it was not possible to navigate using a keyboard then most likely it would not be very usable by a screen reader.

In real life these users might not be so clearly separated and there were many issues that would affect all types of users, but as the intention was to come out of the audit with an actionable list we needed to prioritize the issues while we found them in order of severity and current customer impact. These categories also mostly depend on each other, so it would make sense in most cases to start from solving mouse user issues, then keyboard user issues and then screen reader user issues.

4 Results

4.1 Comparing results from manual audit and automated report

To get the whole list including all components a report was created that included the violations caught in each example of each component. This report was generated at the beginning of the manual audit, so the results obtained from both methods are based on the same source code.

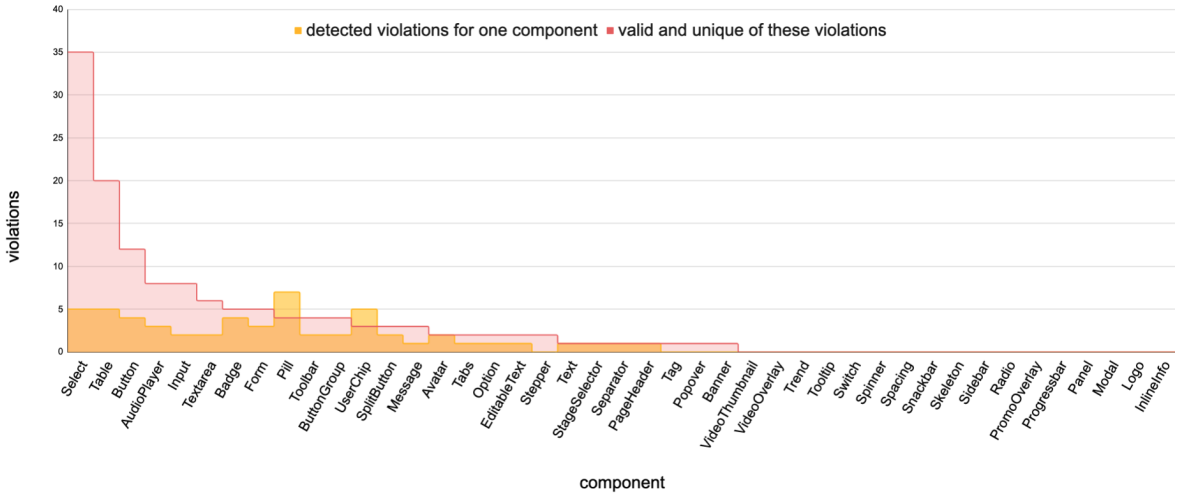

Figure 2: All violation found by addon-a11y and how many of them are valid.

I prepared a comparison table form both results. For automated accessibility tests I recorded the number of examples that included violations, the number of different violations and the number of passed checks for each component. In most cases there were more examples with violations because the same thing was reported in more than one example (see figure 2). Addon-a11y did not report any issues for 27 components out of 53 components.

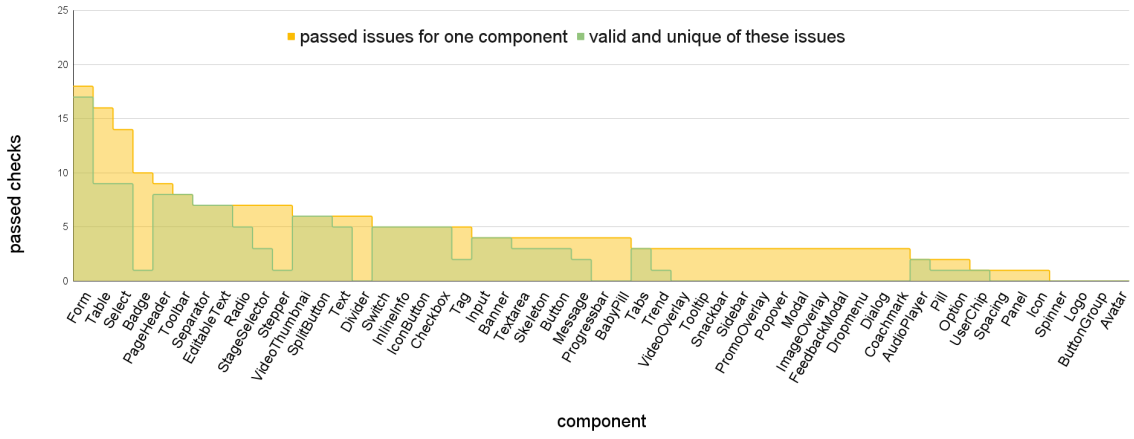

Figure 3: All passed checks reported by addon-a11y and how many of them are valid.

Passed issues where looked over to determine how many where valid (see figure 3). 4

components did not have any passed issues and 22 did not have any valid passed issues.

Looking at all the fails and passes gives an overview of what was checked for each component. Out of 53 components only 2 did not get checked by addon-a11y at all. In addition, components that become visible only when triggered by another element, like modals and popups that currently in our library displayed with a button as a trigger, don't get tested. These components are seen on figure 3 starting from *VideoOverlay* and ending with *Coachmark* - 27 over all. This means 29 components where not tested but this tool and rest of the components had passed or failed issues, but often not both.

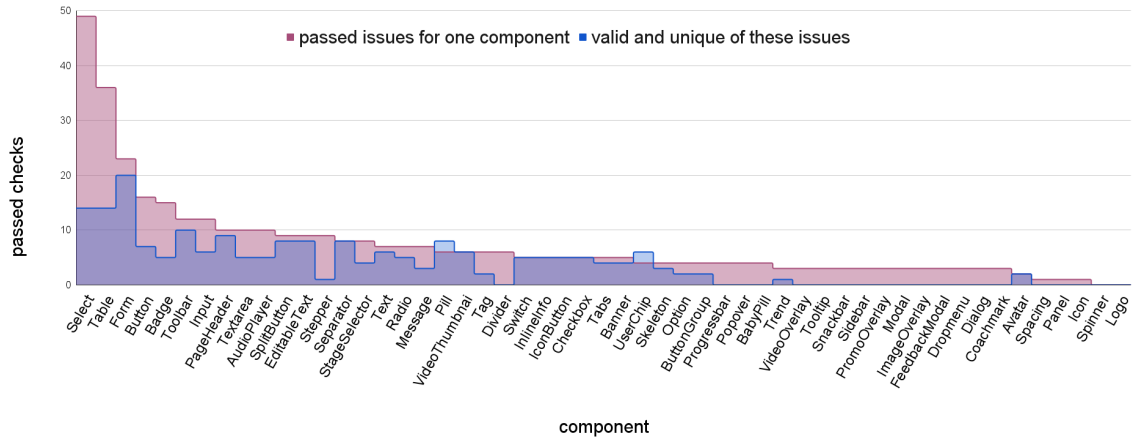

Figure 4: All issues that where tested by addon-a11y. This means both passed and failed issues combined.

In some cases there were 0 violations detected, and 0 valid checks passed – this means that the automated testing was not effective (**TODO: Make chart or table for this).

4.2 Limitations of using Storybook's addon-a11y

The accessibility add-on in Storybook analyzes the examples that have been made for the component and unsuitable example can cause false results. Like Components triggered by a button described before. This is because the initial HTML that the accessibility tests are being run on only has the button and the tests are not being run again after triggering the element. This could potentially be remedied with better examples.

The biggest limitation of this tool currently is that it can only be view-d in storybook. To see the number of passes and fails you need to open the accessibility tab for each component. I looked into ways of automating this so that the same checks could be run on every change to the library and added to continuous integration (CI) workflow. In the current version of Storybook there is no easy way to do this, but it will become much easier in the next major version.

Upgrading out component library to that version need some extra work to make it compatible, but I have tested out this solution on a test library, and it seems like it would definitely be an improvement. Running the tests in CI would ensure that they are run every time someone makes a change and not only when we choose to. We could also block changes that don't pass the required accessibility checks.

TODO: Reasons for automated check not being effective

5 Discussion

5.1 Limitations of the study

The accessibility add-on in Storybook analyzes the examples that have been made for the component and unsuitable example can cause false results. Like Components triggered by a button described before. This is because the initial HTML that the accessibility tests are being run on only has the button and the tests are not being run again after triggering the element. This could potentially be remedied with better examples.

The biggest limitation of this tool currently is that it can only be view-d in storybook. To see the number of passes and fails you need to open the accessibility tab for each component. I looked into ways of automating this so that the same checks could be run on every change to the library and added to continuous integration (CI) workflow. In the current version of Storybook there is no easy way to do this, but it will become much easier in the next major version.

Upgrading out component library to that version need some extra work to make it compatible, but I have tested out this solution on a test library, and it seems like it would definitely be an improvement. Running the tests in CI would ensure that they are run every time someone makes a change and not only when we choose to. We could also block changes that don't pass the required accessibility checks.

**TODO: Reasons for automated check not being effective

5.2 Future research

Conclusion

References

Abbott, C. (2021). Axe-core vs PA11Y. Retrieved 03/12/2023, from https://craigabbott.co.uk/blog/axe-core-vs-pa11y/

Abou-Zahra, S., Steenhout, N., & Keen, L. (Eds.). (2017). *Selecting Web Accessibility Evaluation Tools*. Web Accessibility Initiative (WAI). https://doi.org/10.1145/1061811.1061830

Accessibility Guidelines Working Group (AG WG) Participants (Ed.). (2022). *Wcag 2.1 understanding docs: Introduction to understanding wcag*. Web Accessibility Initiative (WAI). https://www.w3.org/WAI/WCAG21/Understanding/intro#understanding-the-four-principles-of-accessibility

Adobe. (N.d.). *React Spectrum: A react implementation of spectrum, adobe's design system.* Retrieved 03/19/2023, from https://react-spectrum.adobe.com/react-spectrum/index.html

Alsaeedi, A. (2020). Comparing Web Accessibility Evaluation Tools and Evaluating the Accessibility of Webpages: Proposed Frameworks. *Information*, *11*(1), 40. https://doi.org/10.3390/info11010040

Brajnik, G. (2008). A comparative test of web accessibility evaluation methods. *Proceedings of the 10th International ACM SIGACCESS Conference on Computers and Accessibility*, 113–120. https://doi.org/10.1145/1414471.1414494

Brajnik, G., Yesilada, Y., & Harper, S. (2011). The expertise effect on web accessibility evaluation methods. *Human-Computer Interaction*, *26*(3), 246–283. http://ezproxy.tlu.ee/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=65184804&site=eds-live

Chromatic. (N.d.-a). *Accessibility tests*. maintained by Chromatic. Retrieved 03/02/2023, from https://storybook.js.org/docs/react/writing-tests/accessibility-testing/

Chromatic. (N.d.-b). *Introduction to storybook for react*. maintained by Chromatic. Retrieved 03/01/2023, from https://storybook.js.org/docs/react/get-started/introduction

Coleman, T. (2017). *Component-Driven Development*. Chromatic. Retrieved 03/20/2023, from https://www.chromatic.com/blog/component-driven-development/

Collings, S. J., Honnibal, M., & Vanderwerff, P. (N.d.). *React-Bootstrap*. Retrieved 03/19/2023, from https://react-bootstrap.github.io/

Deque Systems. (2021). *The automated accessibility coverage report: Why we need to change how we viewaccessibility testing coverage.* (research rep.). Deque Systems. https://accessibility.deque.com/hubfs/Accessibility-Coverage-Report.pdf

Deque Systems. (2023). *Axe-core*. Retrieved 03/02/2023, from https://github.com/dequelabs/axe-core

Ella, E. (2019). A Guide to Component Driven Development (CDD). Retrieved 03/20/2023, from https://dev.to/giteden/a-guide-to-component-driven-development-cdd-1fo1

Fiers, W. (2017). *W3c standardized rules*. Deque Systems. Retrieved 04/02/2023, from https://github.com/dequelabs/axe-core/blob/develop/doc/act-rules-format.md

Fiers, W., Kraft, M., Mueller, M. J., & Abou-Zahra, S. (Eds.). (2019). *Accessibility Conformance Testing (ACT) Rules Format 1.0*. World Wide Web Consortium (W3C). Retrieved 03/12/2023, from https://www.w3.org/TR/act-rules-format/

Fiers, W., & Lambert, S. (2023). *Rule descriptions*. Deque Systems. Retrieved 04/02/2023, from https://github.com/dequelabs/axe-core/blob/develop/doc/rule-descriptions.md

Fowler, M. (2006). Continuous Integration. *martinfowler.com*. Retrieved 03/05/2023, from https://martinfowler.com/articles/continuousIntegration.html

GDS Accessibility Team. (2018). *How do automated accessibility checkers compare?* Government Digital Service. Retrieved 03/19/2023, from https://alphagov.github.io/accessibility-tool-audit/

Henry, S. L. (Ed.). (2022). *Introduction to Web Accessibility*. Web Accessibility Initiative (WAI). Retrieved 12/26/2022, from https://www.w3.org/WAI/fundamentals/accessibility-intro/

Henry, S. L. (Ed.). (2023). *WCAG 2 Overview*. Web Accessibility Initiative (WAI). Retrieved 03/18/2023, from https://www.w3.org/WAI/standards-guidelines/wcag/

Ismailova, R., & Inal, Y. (2022). Comparison of Online Accessibility Evaluation Tools: An Analysis of Tool Effectiveness. *IEEE Access, Access, IEEE*, *10*, 58233–58239. https://doi.org/10.1109/ACCESS.2022.3179375

Kirkpatrick, A., O'Connor, J., Campbell, A., & Cooper, M. (Eds.). (2018). *Web content accessibility guidelines (wcag) 2.1* (2.1). Web Accessibility Initiative (WAI). Retrieved 03/18/2023, from https://www.w3.org/TR/WCAG21/

Level Access, eSSENTIAL Accessibility, The Global Initiative for Inclusive, & International Association of Accessibility Professionals. (2023). *2022 state of digital accessibility*. Retrieved 03/04/2023, from https://www.levelaccess.com/earesources/state-of-digital-accessibility-report-2022/

Material UI SAS. (N.d.). *MUI: The React component library you always wanted*. Retrieved 03/19/2023, from https://mui.com/

MDN contributors. (2023). *Introduction to the DOM - Web APIs — MDN*. MDN Web Docs. Retrieved 03/19/2023, from https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Mueller, M. J., Jolly, R., & Eggert, E. (Eds.). (2017a). *European Union*. Web Accessibility Initiative (WAI). Retrieved 04/02/2023, from https://www.w3.org/WAI/policies/european-union/

Mueller, M. J., Jolly, R., & Eggert, E. (Eds.). (2017b). *United States*. Web Accessibility Initiative (WAI). Retrieved 04/02/2023, from https://www.w3.org/WAI/policies/united-states/

Mueller, M. J., Jolly, R., & Eggert, E. (Eds.). (2018). *Web Accessibility Laws & Policies*. Web Accessibility Initiative (WAI). Retrieved 04/02/2023, from https://www.w3.org/WAI/policies/

Ramotion. (2022). *Design System vs Component Library: Key Differences*. Retrieved 03/19/2023, from https://www.ramotion.com/blog/design-system-vs-component-library/

Rybin Koob, A., Ibacache Oliva, K. S., Williamson, M., Lamont-Manfre, M., Hugen, A., & Dickerson, A. (2022). Tech Tools in Pandemic-Transformed Information Literacy Instruction: Pushing for Digital Accessibility. *Information Technology & Libraries*, *41*(4), 1–32. https://doi.org/10.6017/ital.v41i4.15383

Sane, P. (2021). A brief survey of current software engineering practices in continuous integration and automated accessibility testing. *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. https://doi.org/10.1109/wispnet51692.2021.9419464

Vigo, M., Brown, J., & Conway, V. (2013). Benchmarking web accessibility evaluation tools. *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*. https://doi.org/10.1145/2461121.2461124

World Health Organization. (2022). *Disability*. World Health Organisation. Retrieved 12/26/2022, from https://www.who.int/news-room/fact-sheets/detail/disability-and-health

World Wide Web Consortium. (1997). *World Wide Web Consortium Launches International Program Office for Web Accessibility Initiative*. Retrieved 12/26/2022, from https://www.w3.org/Press/IPO-announce

** All references are listed remove before submitting

Appendices

List of Figures