Jun Young Jung

EVALUATING AND IMPROVING ACCESSIBILITY OF THE WEB APPLICATION FOR M-FILES OY

Faculty of Information Technology
and Communication Sciences
Master's thesis
June 2025

ABSTRACT

Jun Young Jung: Evaluating and Improving Accessibility of the Web Application for M-Files Oy
Master's thesis
Tampere University
Faculty of Information Technology and Communication Sciences
June 2025

This paper explores the importance of improving web accessibility in M-Files Oy's web application to support its expanding global user base. As digital transformation progresses, the company faces business challenges due to limited accessibility for users with various impairments. Understanding the significance of accessibility is important for developers to identify and resolve usability issues within the application.

Drawing on academic literature and case studies, the paper reviews key concepts of web accessibility and the real-world challenges due to insufficient accessibility. It analyses the Web Content Accessibility Guidelines (WCAG), outlining their structure and application, and examines M-Files Oy's internal accessibility standards. Through surveys and internal assessments, key accessibility issues within the application are identified. Based on these findings, the paper provides educational resources and practical guidelines for developers to apply accessibility improvements during the development process.

Keywords: accessibility, evaluation, M-Files, testing, WCAG, web accessibility.

The originality of this thesis has been checked using the Turnitin Originality Check service.

USE OF AI IN THESIS

I have utilised AI tools in my thesis:

☐ No

☒ Yes

The AI tools utilised in my thesis and their purposes are described below:

Names and versions of AI tools:

- Google Gemini, 1.5 Flash.
- Chat GPT, GPT-4o.

Purpose of using AI tools:

The purpose of using AI tools was to improve the written language by checking grammars and writing styles. Also, user survey questions were structured using AI tools.

Sections where AI tools were used:

- Chapter 1 – checking whether paragraphs are written logically and grammatically correct.
- Chapter 2 – summarising the book “Web Accessibility” to provide real-life examples of accessibility barriers and broader benefits of improving web accessibility.
- Chapter 5 – constructing user survey questions and reviewing the survey introduction.

I acknowledge that I am fully responsible for the entire content of my thesis, including the parts generated by AI, and accept accountability for any violations of ethical standards in publications.

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AIIM	Association for Intelligent Information Management
ARIA	Accessibility Rich Internet Applications
CSS	Cascading Style Sheets
HTML	Hyper Text Markup Language
TOV	Tone of Voice
UI	User Interface
UX	User Experience
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines

1. INTRODUCTION

As digital transformation progresses, businesses and organisations are moving their workflows to online environments. According to the 2023 State of the Intelligent Information Management Industry report by the Association for Intelligent Information Management (AIIM), 39% of 260 surveyed organisations have completed more than half of their digitalisation process, while 20% have reached over 75%. However, 25% have implemented only partial automation, and 6% are still in the early stages of digitalisation [1].

These results show that while digital transformation is progressing rapidly, not all organisations and users are adapting at the same speed. People who face accessibility challenges are more likely to experience difficulties in digital environments. For example, not only those with visual or hearing impairments but also older adults and individuals unfamiliar with digital devices often struggle with complex user interfaces or limited input options. Without addressing these challenges, those users continue having trouble to access important information and services as digitalisation expands. Therefore, making digital services accessible to everyone—regardless of physical ability or technical background—has become an essential part of service development and implementation.

M-Files Oy provides a Knowledge Automation platform that helps organisations process information more efficiently and improve decision-making through automated workflows. As a provider of this platform, it has a responsibility to ensure equal access for all users. However, in its early stages of implementing web-based services, accessibility was not fully considered. This made it difficult for some users to navigate and use certain functions.

To address these issues, M-Files Oy has worked internally to improve accessibility. As a result, changes in visual design have made in information easier to see and read. Adjustments such as higher colour contrast and clearer text presentation have improved visibility.

While visual accessibility has improved, functional accessibility still requires attention. Some features remain difficult to operate using only a keyboard, and assistive technologies like voice commands and screen readers are not yet fully supported. These issues cannot be solved with minor design changes alone, as they are tied to the software's overall structure and interaction methods.

The varying levels of accessibility awareness among developers also contribute to these challenges. As seen in the AIIM report, organisations adopt digital tools at different rates, and the same applies to accessibility considerations. Many developers do not prioritise accessibility from the user's perspective and may treat it as an optional feature rather than a necessary part of development. This approach often results in software that does not fully accommodate all users.

To address these challenges, this study aims to help M-Files Oy's developers understand the accessibility and integrate it into their development process. Through user surveys and accessibility testing, it identifies key accessibility issues in the web application named "M-Files Web", one of M-Files Oy's client-side applications, and provides useful examples for improvement.

This study sought to answer the following questions:

1. What information and learning materials do developers need to understand accessibility and apply it in their work?
2. What accessibility issues exist in M-Files Web, and what needs improvement?
3. What practical methods and tools can M-Files Oy's developers use to improve accessibility?

To answer these questions, the study carried out the following activities:

1. Reviewed accessibility concepts, disability types, and challenges.
2. Reviewed POUR principles (Perceivable, Operable, Understandable, and Robust) and the structure of WCAG 2.2 (Web Content Accessibility Guidelines).
3. Conducted user survey and accessibility testing to identify existing issues in M-Files Web and determine areas that require improvement.
4. Outlined key considerations for developing accessibility features with practical examples.

The rest of the thesis is organised as follows:

- Understanding Web Accessibility: This chapter will review existing literature on web accessibility and its different challenges, technical approaches and impact on social participation and business advantages.
- Web Accessibility Guidelines: This chapter will review the four fundamental principles and analyse the structure of WCAG.
- M-Files Accessibility Guidelines: This chapter will introduce M-Files Design System and Tone of Voice guidelines.

- Evaluating User Experience and Accessibility: This chapter will conduct user survey and accessibility testing and analyse their results.
- Implementing Accessibility Features: This chapter will review practical considerations and demonstrate practical examples of accessibility features.
- Conclusion: This chapter will summarise the key contributions of this thesis by answering the questions and outline potential directions for future development.

2. UNDERSTANDING WEB ACCESSIBILITY

Web developers must consider accessibility and acknowledge the challenges faced by individuals with disabilities when navigating digital environments. This helps developers realise that considering accessibility is not just an option but a necessary part of web design. This section reviews what accessibility is, the difficulties users with disabilities face online, technical approaches and different impacts of web accessibility.

2.1 What is Web Accessibility

Web accessibility means making websites and online content usable for everyone, including people with disabilities. This includes ensuring that people with visual, auditory, physical, speech, and neurological disabilities can access and interact with the web. [2, p. 2]

For instance, people who are blind can use screen readers to listen to text, while those with physical impairments can navigate websites using keyboard or other alternative devices. Web accessibility also benefits people without disabilities by allowing them to use websites in ways that suit their preferences, like customising text sizes or using keyboard shortcuts. Therefore, making the web accessible improves the experience for all users and brings practical and financial benefits to organisations.

2.2 Disabilities and Accessibility Barriers

Accessibility issues vary depending on the type and severity of a disability. Each disability creates specific barriers that make it harder for individuals to interact with websites. Understanding these challenges is important for designing websites usable for everyone.

Shawn Lawton Henry (2006) highlights the diverse challenges that individuals with disabilities face when accessing the web [2, pp. 2-3]. For instance, people with physical disabilities may have difficulty using traditional input devices such as a keyboard or mouse. Carl, who had polio and cannot move his arms, uses a mouth stick to interact with digital content. On the other hand, Sarah, who has cerebral palsy, has limited control over her arms and mouth.

Figure 1. Web accessibility tutorial participant using computer with head pointer [2, p. 3].

Figure 1 shows Sarah uses a head pointer to navigate the web. Similarly, Rechard, who has multiple sclerosis, is unable to use a mouse due to reduced motor precision and instead relies on a keyboard for web navigation.

Visual impairments also present significant barriers to web accessibility. Neal, who was born blind, and John, who lost his vision due to retinitis pigmentosa, depend on screen readers[1] to access digital content. However, screen readers are not only beneficial for individuals with blindness but also for those with cognitive processing difficulties. Tracy, for example, has normal vision but struggles with reading text, and therefore uses a screen reader for comprehension.

In addition to complete blindness, some individuals experience low vision or blurred vision, requiring adaptations to access digital content effectively. Shawn, who has a neurological condition, finds it difficult to focus on small text and enlarges text and images in the browser for better readability. Richard also experiences blurry vision and benefits from text size adjustments.

These real-life examples provide a clearer understanding of the specific disabilities people face, and the methods they utilise to navigate the challenges of digital environments. This will help developers recognise the importance of inclusive design.

[1] Screen readers are tools that assist visually impaired users by converting text on a screen into speech. When reading information on a website, screen readers capture it on web pages using Document Object Model (DOM). [2, pp. 104-106]

2.3 Technical Approaches to Web Accessibility

As discussed in the previous section, people with disabilities face various challenges when using digital platforms. To address these difficulties, web developers can implement several technical solutions [2, pp. 4-7]. One fundamental method is the use of alternative text (alt-text) for images.

Images are an essential part of many websites, but they are not accessible to users who are blind or in environments where images do not load. Alt-text provides a textual description of an image, allowing screen readers to convey the information aloud or text-based browsers to display it.

Different browsers and assistive technologies handle alt-text in various ways. Screen readers read it aloud, text browsers display it, and graphical browsers show it when images are turned off or fail to load. Without alt-text, users who rely on these technologies may miss important information. This is problematic when images are used for navigation, as the absence of descriptions can make a website unusable.

Alt-text does not alter the visual design of a website but improves its accessibility. It also benefits users on mobile devices, those with slow internet connections, and enhances search engine indexing.

In addition to alt-text, other features improve web accessibility by addressing different needs. Audio captions, for example, provide text for spoken content in videos and audio files. They are essential for users who are deaf or hard of hearing, but they also help in situations where sound cannot be played or is difficult to hear. Captions also make spoken content searchable, making it easier to find specific information.

Device independence is another important factor. Websites should not require a mouse to navigate. Keyboard access allows users with mobility impairments to browse without relying on a mouse and benefits those using mobile devices or preferring keyboard shortcuts.

A clear and consistent layout also improves accessibility. Users with cognitive impairments may struggle with complex designs, making navigation difficult. A predictable structure helps users find information more easily. It also benefits those who enlarge text for better readability and mobile users who view content on small screens.

2.4 Broader Impacts

Efforts to improve web accessibility expands opportunities for social participation [2, pp. 7-11]. For example, when people with disabilities can access education, healthcare, government services, and commerce through the web, their involvement in society increases. This is because the web is not only a space for information but also offers a platform for people to communicate and engage actively in social activities. Web accessibility provides equal opportunities for everyone, including those with disabilities, to participate in society, enabling more people to take part in social activities.

Moreover, web accessibility brings several business advantages [2, pp. 42-47]. One key reason is that it opens access to a broader market. Websites can reach a wider customer base, including older adults, people in low-bandwidth environments, and those with limited web experience. This leads to more users and potential customers, which can result in increased sales.

Next, web accessibility can lead to cost savings in terms of maintenance and operations. Websites designed with accessibility in mind often simplify maintenance tasks. For example, using standardised HTML and CSS reduces repetitive work when making design changes, and decreases the time and cost associated with translating and updating content. Additionally, accessible websites offer a consistent user experience across various devices and browsers, which eliminates the need for separate mobile versions and reduces development costs.

Web accessibility also reduces legal risks. Many countries have strengthened legal requirements related to web accessibility. Websites that fail to meet accessibility standards may face lawsuits and the associated legal costs.

Finally, companies that prioritise web accessibility often see improvements in their brand image and consumer trust. By showing a commitment to social responsibility, these organisations can build stronger customer loyalty, especially among diverse groups, including individuals with disabilities. This positive perception of the brand can contribute to a more inclusive public image, which supports better customer engagement and long-term business success.

By understanding the needs and impacts of web accessibility, M-Files Oy should integrate web accessibility into its products. This will help the company become globally accessible and trusted by a wide range of customers. M-Files Oy's developers should contribute by making the M-Files platform more inclusive, thus improving its position as a leading knowledge work automation platform.

3. WEB ACCESSIBILITY GUIDELINES

Chapter 2 reviewed web accessibility and real-life examples of the way how different people with impairments navigate digital environments. The chapter identified common design and technological solutions that can be implemented to enhance accessibility.

This chapter reviews the four fundamental principles of the Web Content Accessibility Guidelines (WCAG)—Perceivable, Operable, Understandable, and Robust (POUR)—form the inclusive digital experiences [3]. With understanding the POUR principles, this chapter analyses the structure of the WCAG to help developers how to integrate the guidelines into web development.

3.1 Web Accessibility Principles

The World Wide Web Consortium (W3C), one of the nongovernmental agencies who established guidelines for accessible online environments, introduced the POUR principles. Each letter in the acronym represents a set of guidelines in WCAG, indicating the core value for stakeholders to achieve by following the underlined guidelines. [2][4]

The first principle represented as P is **Perceivable**, which ensures users to interpret web content with varying sensory abilities. The same information presented in a website can be perceived by a certain modality, but not by others. For example, missing alternative text descriptions of visual cues, such as images and diagrams, makes them imperceivable for people with visual impairment who rely on screen readers. By ensuring the perceivable features, users can access the web content through their chosen modalities.

In addition to the perceivability, making a website fully interactive by different input devices is important for accessibility. This represents the **Operable** principle as O. Keyboard and mouse are the most common input devices. Combining both methods allows users to interact and navigate webpages without challenges. On the other hand, those who rely on only keyboard can face difficulties to access full functionalities and features. Navigating and interacting with web interface must be operable by users' chosen methods.

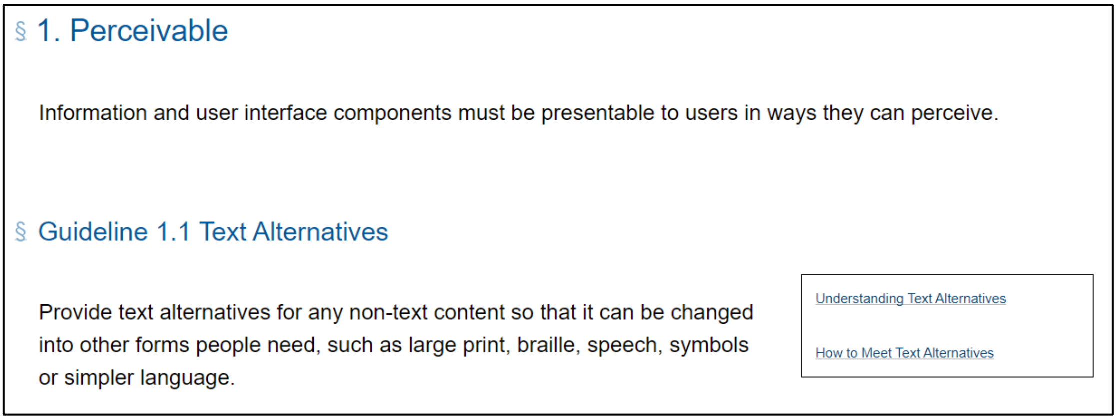
Perceivable and Operable principles of web accessibility are mainly related to the technical aspects, while the **Understandable** principle focuses on use of languages and design philosophy of web applications. Using complex terms or unclear descriptions can confuse users, making content harder to access and leading to unintended actions. Similarly, changing colours alone does not clearly indicate whether an element is interactive.

Maintaining consistency in design and colours helps users recognise functions and predict how to interact with a web application more easily.

Finally, the **Robust** principle ensures that web content functions properly across different user agents. Assistive technologies, such as screen readers, rely on the semantic information of web elements to provide context for users. Although this information is not visually apparent, user agents can interpret semantic HTML elements and attributes like “Name,” “Role,” and “Value” to determine the purpose and meaning of web content. For example, a button marked with proper semantic attributes allows a screen reader to announce it as a “button’ rather than plain text, helping users understand its function. Proper use of semantic elements improves accessibility and ensures compatibility across various devices and technologies.

3.2 Web Content Accessibility Guidelines

The Web Content Accessibility Guidelines (WCAG), developed by the W3C, provides a standardised framework to improve web accessibility for individuals with disabilities. Built on the POUR principles, WCAG helps create more inclusive digital environments. The guidelines offer clear methods to help web applications meet specific accessibility standards. Understanding WCAG’s structure allows developers and designers to find the correct guidelines and examples needed to follow technical accessibility standards effectively.


§ 1. Perceivable

Information and user interface components must be presentable to users in ways they can perceive.

§ Guideline 1.1 Text Alternatives

Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

Understanding Text Alternatives

How to Meet Text Alternatives

Figure 2. Guideline 1.1 Text Alternatives under 1. Perceivable [4].

The WCAG 2 contains 13 guidelines, and they are grouped by the POUR principles as shown in Figure 2. The brief explanation about the Perceivable principle represents the main goal of following four guidelines: Text Alternatives, Time-based Media, Adaptable, and Distinguishable. Similarly, relevant guidelines are listed for other principles. Acknowledging how they are categorised, developers can navigate appropriate guidelines easier for their use cases.

Each guideline contains two links to other pages; one helps understand the guideline in detail, and another shows sufficient technical solutions and failures for each success criterion under it [4]. For example, the description for the guideline 1.1 in Figure 2 is brief and it is hard to acquire full intention. By accessing the link “Understanding Text Alternatives”, developers can find the reason why the guideline is important to be considered and what they should follow to fulfil the standard.

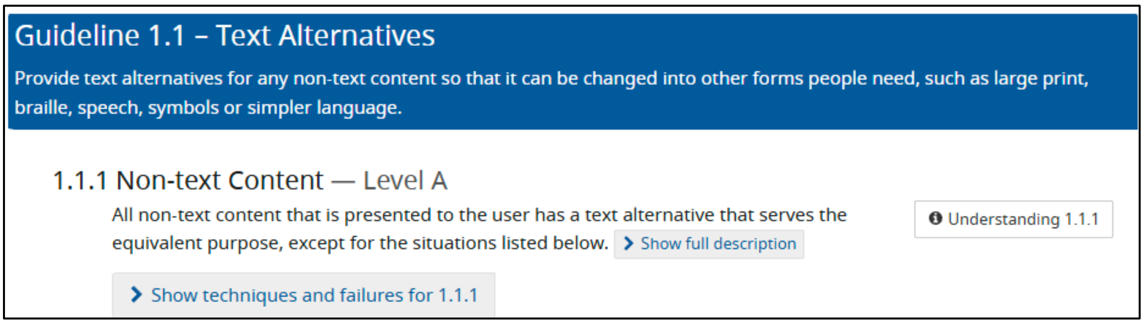


Figure 3. Success Criterion 1.1.1 Non-text Content in How to Meet WCAG [5].

In addition, the “How to Meet WCAG” page is useful to explore all relevant links that can be referenced for various situations, containing quick reference about descriptions, techniques and failures of each guideline [5]. Figure 3 shows the snapshot of the page after accessing the link “How to Meet Text Alternatives” from Figure 2. Expanding the section “Show techniques and failures for 1.1.1”, developers can check the common situations of using non-text content like images, chart or diagram. The criterion also indicates its conformance level, defined by WCAG in three distinct conformance levels: A (minimum), AA (recommended for most websites and applications), and AAA (strictest, addressing advanced accessibility needs) [4].

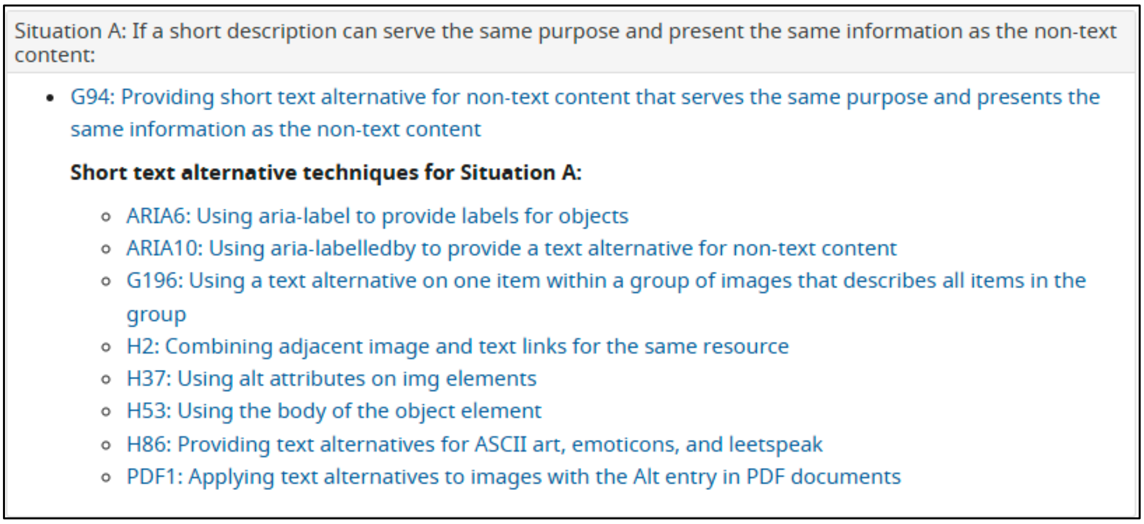


Figure 4. Situation A of 1.1.1 Non-text Content [5].

Each situation contains a list of accessibility techniques to fulfil the criterion. For example, Figure 4 shows techniques for resolving the situation A of the success criterion 1.1.1. The situation describes that a non-text content requires a short description to carry the same information as the content so that it can be accessed using assistive technologies.

The technique “G94” provides general explanations of providing alternative text for non-text content. It helps understand the technique and its applications. The following techniques after G94 demonstrate different examples of providing alternative texts. The details and examples are available through the corresponding link to each technique.

Example 3: Providing a label for Math

Below is an example of a MathML function, using the math role, appropriate label, and MathML rendering:

```
<div role="math" aria-label="6 divided by 4 equals 1.5">
  <math xmlns="https://www.w3.org/1998/Math/MathML">
    <mfrac>
      <mn>6</mn>
      <mn>4</mn>
    </mfrac>
      <mo>=</mo>
      <mn>1.5</mn>
  </math>
</div>
```

Figure 5. Example 3 of Technique ARIA6 [6].

Technique “ARIA6” explains the purpose of using a label for non-text content, providing examples how to apply the technique [6]. The example in Figure 5 demonstrates the use of the “aria-label” attribute to describe the math equation “6/4=1.5.” With the descriptive label, screen readers can deliver the full information of the non-text content to users who rely on the assistive technology.

Example 1

In the following example, part of the content is contained in an image that is presented by CSS alone. In this example, the image TopRate.png is a 180 by 200 pixel image that contains the text, "19.3% APR Typical Variable."

```
p#bestinterest {
   padding-left: 200px;
   background: transparent url(/images/TopRate.png) no-repeat top left;
 }
```

It is used in this excerpt:

```
<p id="bestinterest">Where else would you find a better interest rate?</p>
```

Figure 6. Example 1 of Technique F3 [7].

Addition to the examples like Figure 5, it is important to study the opposite techniques of failing the success criterion. Technique F3 illustrates those cases where web content does not meet the criterion 1.1.1 [7]. Attaching the image “TopRate.png” to the HTML element through CSS as background, the example in Figure 6 failed to deliver the important information “19.3% APR Typical Variable.” This is because those images attached though CSS properties cannot be associated with alternative texts in the HTML code. Thus, it is impossible for screen readers to detect and analyse the information.

There are 83 success criteria and 424 techniques in WCAG 2.2 [4][8]. It is not an easy task for developers to remember all guidelines, success criteria, and techniques. Instead, by understanding the POUR principles and the structure of WCAG, they can effectively navigate which accessibility techniques to apply to their specific situations. This approach will help M-Files Oy’s developers to identify the relevant strategies to improve accessibility and create more inclusive web applications, while also making the development process smoother and more efficient.

4. M-FILES ACCESSIBILITY GUIDELINES

M-Files Oy has established itself as a leading company of Intelligent Information Management (IIM), offering a unique document management system (DMS) [9]. Its DMS changed the approach from traditional folder-based structures to metadata-driven system. This allows users to locate and manage information efficiently, building a more productive knowledge work environment. As M-Files Oy encourages its customers to bring their information into digital environments, ensuring its products accessible for everyone has become one of the important goals.

This chapter introduces the M-Files Oy's accessibility guidelines currently employed across different products: **M-Files Design System**, and **M-Files Tone of Voice**. The guidelines mainly focus on colours, layouts and use of language, providing guidance of visual structures and user-centric UI texts.

4.1 M-Files Design System

The M-Files Design System establishes a central repository of design principles, guidelines, and tools [10]. This includes colour palettes, typographic styles, spacing specifications, and guidelines for shadows and borders. Additionally, the system provides an accessible library of pre-designed UI components, along with documentation and sample UI layouts.

One key aspect of the M-Files Design System is colour choice. The system offers a range of colour palettes, including main themes, system colours, and icon colours. Following WCAG guidelines such as 1.4.3 Contrast (Minimum), 1.4.6 Contrast (Enhanced), and 1.4.11 Non-text Contrast ensures the colour combinations within M-Files Oy' products to provide appropriate contrast ratios [10].

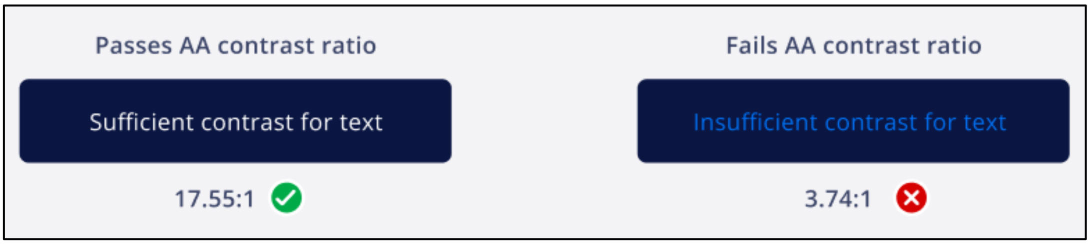


Figure 7. Contrast ratio examples illustrating compliance and non-compliance with WCAG AA minimum requirements (captured from M-Files Design System [10]).

When choosing colour combinations, designers should use web-based tools to ensure the contrast ratio meets the minimum requirement of 4.5:1, as defined by WCAG AA guidelines. **Error! Reference source not found.** shows one combination that meets this s tandard and one that does not. While some designers may find both examples legible due to their sensitivity to colour differences, this perception is not reliable. Users with visual impairments may not be able to read text with insufficient contrast. Therefore, contrast ratios should be checked using tools such as the WebAIM Contrast Checker[2].

In addition to the colour choices, layouts and spacing are fundamental elements within the M-Files Design System, establishing visual consistency, hierarchy, and balance across the user interface [10]. Layouts define the arrangement of visual components on the screen. This includes the positioning of text and other UI elements. With a multiples-of-4 spacing system, the system maintains consistent spacing between elements.

These design principles serve to guide user attention, making information within the M-Files interface easier to understand and navigate. Layouts with sufficient spacing between interactive elements ensure that users with motor control limitations can interact with the interface effectively.

4.2 M-Files Tone of Voice

The M-Files Tone of Voice (TOV) establishes consistent and user-centric texts within M-Files Oy's products [10]. This section outlines the three core values of the M-Files TOV, which directly influence user experience: approachability, trustworthiness, and informativeness.

First, UI text must be written in an approachable tone, demonstrating empathy towards the user throughout their interaction with M-Files. Users may prosses limited contextual knowledge about the software. The text should convey a sense of understanding an attentiveness to user needs. For example, when information is not available, simply stating "No data available" will not guide users what to do next. Instead, the text should address them what they can act differently by saying like "Try it later," or "Search different keywords."

Next, UI text must be trustworthy. It should create a sense of security and reliability regarding user data, ensuring consistency in messaging across different UI elements and maintaining alignment with the broader context of the application. For example, when displaying an error message related to data submission failure, the text should clearly explain the issue, avoid technical jargon, and reassure the user that their data remains safe.

---

[2] WebAIM Contrast Checker: https://webaim.org/resources/contrastchecker

Last, UI text must provide users with the precise information they need to complete their tasks. The message should be concise and avoid introducing extraneous details that may overwhelm the user. For example, instead of displaying a generic message like "Something went wrong," a more effective message would state, "Failed to upload file. Please check your internet connection and try again."

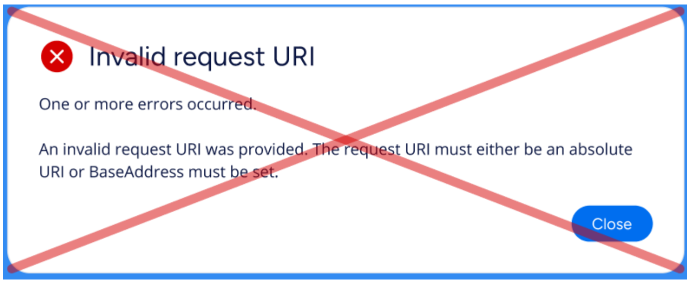


Figure 8. UI text before applying M-Files TOV principles [10].

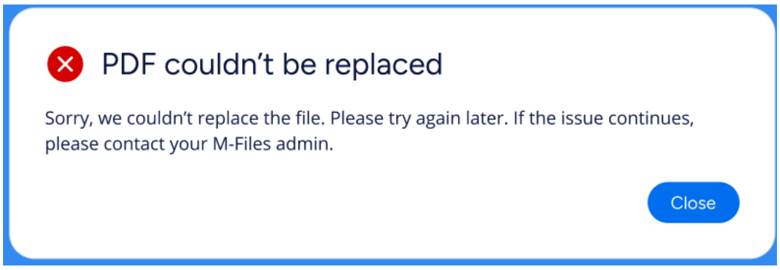


Figure 9. UI text after applying M-Files TOV principles [10].

Figure 8 and Figure 9 illustrate two different approaches to error messages displayed when replacing a PDF file fails. The first message (Figure 8) focusses on technical details, using jargon that may be confusing for users without a technical background. It simply explains what went wrong without offering any practical solutions. This type of message is not user-friendly and does not provide actionable steps for resolving the issue.

In contrast, the second message (Figure 9) follows the M-Files TOV principles. It provides clear context regarding the error and suggests a specific action users can take if the problem persists. This user-centric approach avoids technical jargon and helps users to potentially fix the issue themselves.

M-Files Oy has been utilising the M-Files Design System and the M-Files Tone of Voice during development process, referring how UI components look and behave based on user interactions. This provides easier and clearer development process. However, the guidelines do not cover every accessibility concern. It is important that developers become familiar with WCAG and check whether the specifications fulfil the standards or not, instead of following them implicitly. Continuous user research and feedback are also essential to gain a deeper understanding of evolving accessibility needs.

5. EVALUATING USER EXPERIENCE AND ACCESSIBILITY

This chapter conducts a user survey and accessibility testing to evaluate M-Files Oy's products regarding user experience and accessibility. By evaluating user experience and accessibility, we understand accessibility needs of users and identify areas for improvement.

5.1 User Survey

User surveys are a valuable research tool that allows researchers to gather quantitative data from a target population. This data can be used to understand user demographics, opinions, experiences, and behaviours. The goal of conducting the user survey is to collect user feedback on accessibility features and challenges when using M-Files Oy's products, especially those client-side applications (M-Files clients) where end-user interacts.

To gather quantitative data on user experiences and accessibility needs regarding M-Files clients (web, desktop, and others), we chose a web-based survey platform, Microsoft Forms, due to its user-friendly interface and secure data management capabilities.

The survey began with a brief introduction explaining accessibility and the purpose of the survey, which was to evaluate the accessibility of M-Files clients. Participants were informed that their responses would be recorded anonymously. After the introduction, four parts of survey questions are placed for participants to provide answers and feedback.

To increate participant engagement, a variety of question types were utilised (Appendix 1):

- **Multiple-choice**: These questions assessed M-Files client usage patterns, such as "How often do you use M-Files clients?" with options for daily, weekly, etc.
- **Dichotomous (Yes/No)**: These questions were useful for gathering information on specific usability aspects that can impact accessibility such as "Did you find the layouts of M-Files clients clear and organised?"

- **Likert scale**: These questions evaluated user satisfaction levels with specific aspects of the clients.
- **Open-ended**: These questions allowed participants to describe their experiences and accessibility challenges using their own words, providing rich quantitative data.

The initial plan anticipated a broader audience reach, including external users, through diverse distribution channels (email, social media posts, and embedded website links). Despite the initial plan, a pilot survey was conducted with a small target user group within M-Files Oy's employees during a limited period (nine days, March 27th - April 4th, 2024).

5.2 User Survey Result and Analysis

Eighteen responses were received from internal users, all of whom reported frequent M-Files client usage. While this sample size provides initial insights, a larger and more diverse participant poll would be necessary for generalisability.

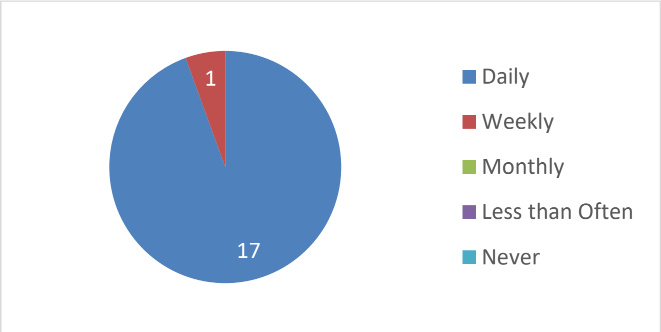


Figure 10. How often do you use M-Files clients?

In Figure 10, seventeen respondents indicated daily use of one or more clients, while one reported weekly usage. This data suggests a high reliance on M-Files clients within the internal users.

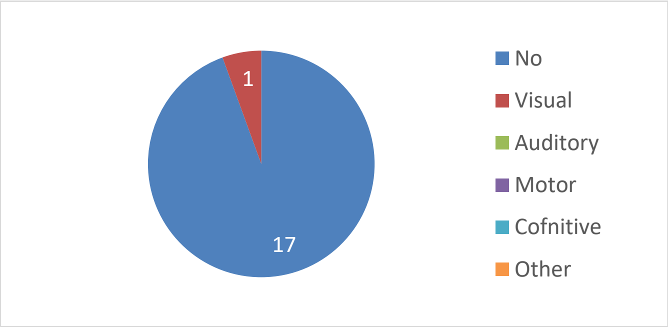


Figure 11. Do you identify yourself as having any impairments, if yes which type of impairment(s) you experience?

Figure 11 shows only one participant disclosed experiencing a visual impairment, while the remaining respondents indicated no experience with any impairments. The limited number of users with reported impairments in this pilot survey restricts the ability to draw definitive conclusions about accessibility barriers for users with disabilities.

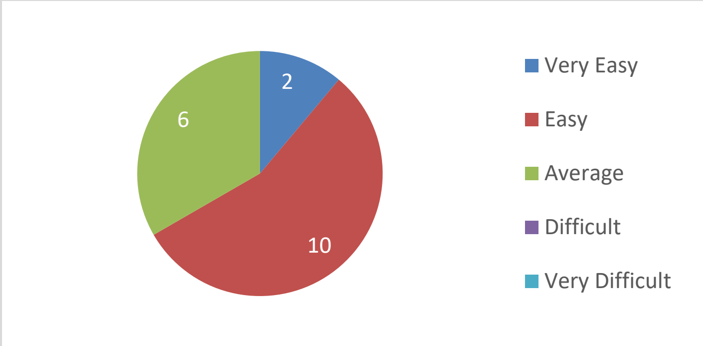


Figure 12. Rate the ease of navigating M-Files clients on a scale of 1 (Very Difficult) to 5 (Very Easy).

Approximately 67% of respondents found navigating M-Files clients to be easy in Figure 12. However, a subsequent question revealed a potential accessibility gap. Less experienced users expressed difficulty due to a perceived lack of intuitive user guides for navigation and task completion. This finding highlights the need for improved user onboarding materials and in-app guidance for users with varying levels of experience.

Only three respondents reported utilising built-in accessibility features, primarily keyboard navigation, and their usage was perceived as limited due to predominant use of the mouse for content navigation. This suggests that built-in accessibility features may not be well-known or well-understood by users. Further investigation into user awareness of accessibility features and potential usability issues with these features is warranted.

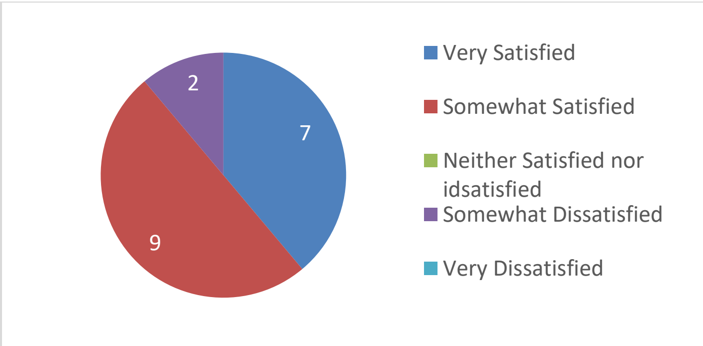


Figure 13. Overall, how satisfied are you with the accessibility of M-Files clients?

In Figure 13, nearly 90% of respondents expressed satisfaction with the accessibility of M-Files clients. However, one open-ended response highlighted a professional concern about the lack of best practices in accessibility, potentially impacting customer discussions. This response indicates the importance of accessibility not only for user experience but also for brand reputation and customer perception.

The pilot survey's limited scope, including internal users and a short timeframe, resulted in a smaller and less diverse participant pool. Despite this limitation, the open-ended questions provided valuable insights into areas for improvement, informing efforts to enhance accessibility for a broader user base with varying abilities. Future research should aim to recruit a larger and more diverse sample of participants, including users with disabilities, to gain a more comprehensive understanding of accessibility needs and challenges.

5.3 Accessibility Evaluation

Evaluating accessibility is an important process that ensures websites and web applications are usable by everyone, regardless of ability [11]. This evaluation aims to identify and remove barriers limiting user interaction for individuals with disabilities. Among the M-Files Oy’s client-side applications, the main web application, M-Files Web, was evaluated using a combined approach utilising both automated and manual testing methods.

Automated testing tools help developers to identify potential accessibility barriers within the web application. These tools perform a programmatic analysis of an application's code and content, scanning large volumes of web content and flagging potential WCAG violations across various success criteria.

Specific examples of automated testing tools include:

- **WAVE (Web Accessibility Evaluation Tool):** A web-based tool developed by Web AIM (Web Accessibility in Mind) that accesses conformance with WCAG success criteria at a basic level. It provides a quick overview of potential accessibility issues across various aspects such as alternative text for images, colour contrast, and keyboard navigation. [12]
- **Axe (Automated Accessibility Checker):** A more comprehensive automated testing tool that offers detailed reports outlining potential WCAG violations. It allows for customisation of the testing scope and provides specific guidance for remediation efforts. [13]

These automated tools are available as extensions for several web browsers like Chrome, Edge, and Firefox. WAVE tool was chosen for the evaluation to identify any accessibility violation in M-Files Web.

While automated tools offer valuable insights, manual testing remains essential for a thorough accessibility evaluation. Involving real users with diverse disabilities during manual testing allows to identify the accessibility barriers limiting users to navigate or operate through their preferred user agents. However, the researcher conducted the manual testing without real users, instead simulated the following methods:

- **Keyboard Navigation Testing:** The application was thoroughly navigated using only the keyboard to ensure all functionalities were accessible to users who rely on this method.
- **Screen Reader Testing:** Compatibility with assistive technologies such as screen reader was evaluated to ensure users with visual impairments could access and interact with the application effectively.
- **User Testing with Assistive Technologies (Future Investigation):** While not employed in this specific evaluation, user testing sessions with participants who rely on assistive technologies, such as screen readers or screen magnification software, represent a valuable approach for further investigation. This can provide deep insights into real-world user experiences and challenges faced while interacting with the application.

5.4 Accessibility Evaluation Results

The accessibility evaluation of M-Files Web identified a range of barriers that limit access and interaction for users with disabilities. This section presents key findings from the evaluation.

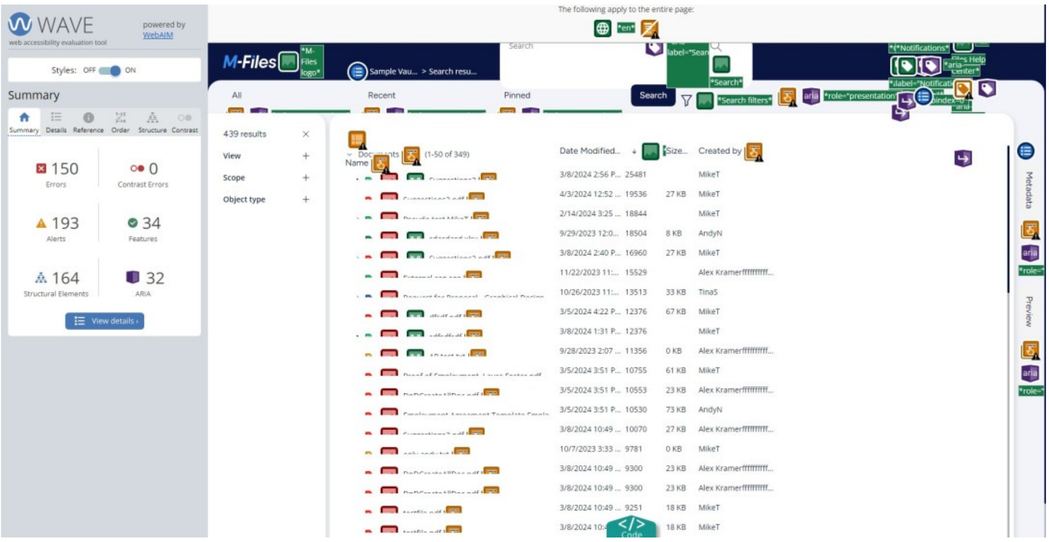


Figure 14. Accessibility analysis from WAVE tool.

While the M-Files Design System covers sufficient colour contrast and content spacing within the web application, automated testing revealed violations against the Perceivable principle. In Figure 14, WAVE web extension identified numerous WCAG violations. Notably, the application lacked descriptive alternative text for images and icons, essential for users relying on screen readers. During screen reader compatibility testing, it was unable to interpret the content of these elements due to the missing alternative text. Additionally, the tool identified empty form labels with no text inside the application. This shows the label tag was used incorrectly.

Although WCAG does not mandate title attributes for HTML elements, the WAVE report in Figure 14 highlighted instances where title attributes mirrored element value content. Titles should only provide supplementary information, such as displaying omitted text or additional descriptions.

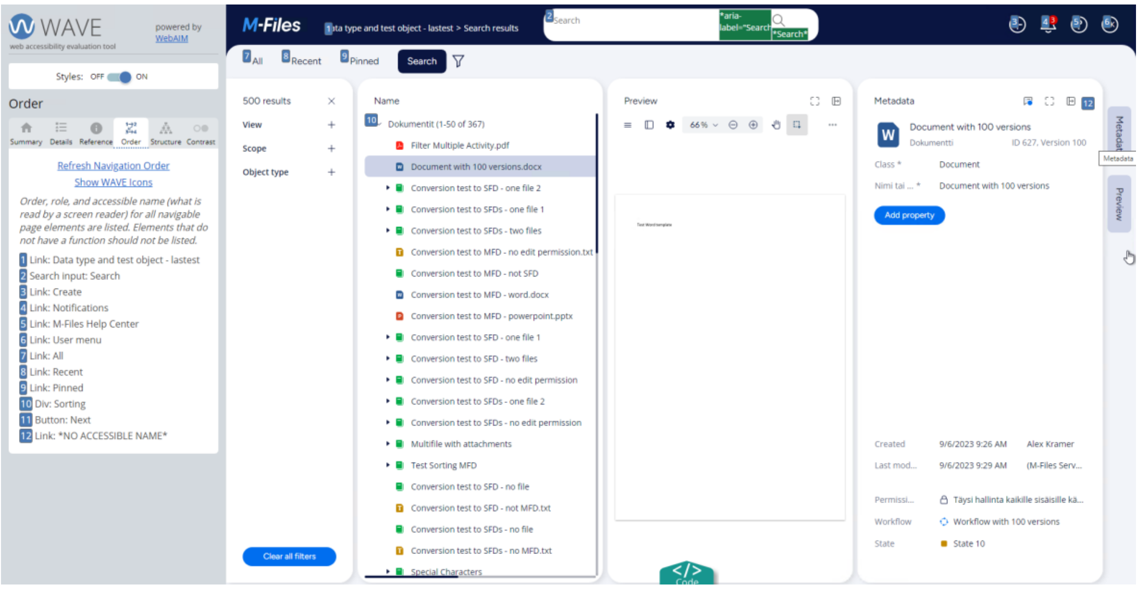


Figure 15. Navigation order analysis from WAVE tool.

In addition to automated testing, manual keyboard navigation testing revealed that numerous elements within M-Files Web were not reachable or interactive using keyboard controls. As identified by WAVE tool in Figure 15, there are many interactive elements that are not included in the navigation order. For example, the buttons ("X" and "+" icons) of the filter section remained inaccessible via keyboard controls during manual testing. This prevented users from opening and closing filter options. Additionally, list items in second left section were not keyboard navigable until a user selected an item with the mouse.

Keyboard operability limitations extended to functionalities like top menu items. Pressing "Enter" on these items did not trigger interaction to open associated sub-menus, and even after manually opening the "Create" menu sub-items using the mouse click on the "Create" menu (represented by "+" icon), navigating these sub-items with arrow keys proved impossible.

Manual testing also identified a focus trap with the metadata section (Figure 15, right section). Pressing the "Tab" key resulted in being trapped within the section without allowing users to escape via keyboard controls.

Screen reader compatibility testing revealed a lack of semantic markup for specific user interface elements. Web accessibility best practices, as outlined in WCAG and specifications like **WAI-ARIA** (WAI Accessible Rich Internet Applications), emphasise the importance of semantic markup. Semantic elements convey the meaning and functionality of interface elements to assistive technologies. In the absence of semantic markup, screen reader users are unable to obtain the purpose of the elements such as input fields and combo boxes.

For example, certain combo boxes in the web application displayed compatibility issues with Narrator when navigating and interacting items using arrow keys. This arises from the absence of a properly defined relationship between the combo box and its individual items. The screen reader relies on semantic markup to establish these relationships and present the information to users in a meaningful way. Without such markup, the screen reader cannot determine how the combo box functions or how its items are connected.

5.5 Discussion and Recommendations

The user survey and accessibility evaluation of M-Files Web revealed valuable insights into the current state of accessibility and user needs. While the survey indicated a generally positive perception of accessibility among internal users, the evaluation identified several areas for improvement: missing alternative text, misuse of labels and title attributes, keyboard navigation issues, and lack of semantic markup.

Based on these findings, the following recommendations are proposed for enhancing the accessibility of the web application:

- **Implement Descriptive Alternative Text**: All images and icons should be accompanied by concise and descriptive alternative text that accurately conveys their meaning and context.
- **Correct Label and Title Attributes Usage**: Ensure that all form labels are properly associated with their corresponding input fields. Title attributes should

only be used when providing additional information that is not already conveyed by the element itself.

- **Improve Keyboard Navigation**: Fix all identified keyboard navigation issues to ensure that all functionalities and elements can be accessed and interacted with using the keyboard alone. This includes addressing the filter close button, menu items, list items, and the focus trap within the metadata section.
- **Utilise Semantic Markup**: Implement semantic markup for user interface elements to provide assistive technologies with the necessary information to accurately represent the purpose and functionality of each element to users.
- **Continued Accessibility Testing**: Regularly conduct accessibility testing throughout the development lifecycle to identify and address any new accessibility barriers that may be introduced. This can involve using automated testing tools, manual testing with assistive technologies, and user testing with participants with disabilities.

These recommendations highlight the importance of ongoing accessibility testing and user feedback to ensure a truly inclusive user experience for individuals with disabilities.

6. IMPLEMENTING ACCESSIBILITY FEATURES

Chapter 5 identified the fundamental reasons for accessibility issues in M-Files Web. Elements are inaccessible or inoperable using only keyboard in the web application due to lack of keyboard accessibility support. This chapter covers practical considerations and demonstrates the implementation of an accessible web component for keyboard and screen reader users. Additionally, it includes the implementation of keyboard shortcuts for power users.

6.1 Keyboard Accessibility

Keyboard accessibility ensures that users can navigate and interact with web components solely using their keyboard. This is important for users who experience motor limitations or who prefer keyboard navigation. Key considerations for ensuring robust keyboard accessibility include **focus management**, **tab order**, and **keyboard events.** [14]

- **Focus Management**: Employing JavaScript, developers can manage focus within a component. This ensures that the appropriate element receives focus when the component load or upon user interaction by keyboard (e.g., focusing the search input field within a search bar component upon activation). For keyboard users, the "Tab" key navigates through interactive elements. While semantic HTML elements like **<button>** and **<input>** are accessible via "Tab" key, elements like **<div>** and **<span>** require the **tabindex** attribute for focus activation. If the "Tab" key fails to focus the desired element, developers can programmatically manage focus using JavaScript. It is also important to provide visual indicators, such as outlines, for keyboard users to recognise which element is currently focused. On the other hand, non-interactive elements should not be keyboard focusable, as this can cause confusion.
- **Logical Tab Order**: A logical tab order represents the sequence in which elements receive focus when a user presses "Tab." This order should align seamlessly with the functionality of components, typically mirroring the visual flow of the page (left to right, top to bottom). The source code of a web page defines this navigation and reading order (for screen readers). Assistive technologies like screen readers rely on the structure of the HTML code to determine the order in which elements are presented to the user. For optimal results, the source code should accurately reflect the visual layout of the page to ensure a consistent and

intuitive reading/navigation experience for both keyboard users and screen reader users. Developers should avoid using **tabindex** values greater than 1 to alter the default keyboard navigation order, as this can disrupt the logical flow established by the HTML structure.

- **Keyboard Events**: Implementation of keyboard events allows users to interact with web components using the keyboard. This functionality is important for users who rely on keyboard navigation or who have motor limitations that prevent them from using a mouse. Common keyboard events include "Enter," "Space," and "Arow" keys. These keystrokes enable users to perform actions such as submitting forms, activating buttons, and navigating through lists and menus. It is recommended to test these common interactions with standard keystrokes to ensure they function as expected. For instance, the standard keystrokes to activate buttons are "Enter" or "Space." While the semantic HTML **<button>** element activates natively without additional keyboard events, elements with ARIA **role="button"** must contain keyboard events manually configured by developers to activate both key commands.

6.2 Screen Reader Compatibility

Screen reader compatibility allows users who rely on assistive technologies to comprehend and interact with web components. Key considerations for achieving this compatibility include following [15][16]:

- **Semantic HTML**: Utilising semantic HTML elements establishes a clear and meaningful structure for the component. Screen readers rely on this structure to convey information to users. Semantic elements like headings, paragraphs, lists, and buttons inherently communicate their purpose to assistive technologies, delivering information for users. For example, a **<button>** element inherently signifies an interactive button to a screen reader, while a **<span>** element might require additional context by adding a button role.
- **ARIA Attributes**: ARIA (Accessible Rich Internet Applications) attributes act as an additional layer of information for screen readers. ARIA defines roles and attributes that enhance the accessibility of web content. By utilising ARIA attributes, developers can provide supplementary information to screen readers. Common ARIA attributes within web components include **aria-label**, **aria-labelledby**, and **role**. These attributes allow developers to provide labels, roles, and other contextual information for the component, ensuring clear communication with screen

reader users. For example, an icon representing "delete" might not be understood by a screen reader on its own. Adding the **aria-label="Delete"** attribute would provide a clear description for the user.

6.3 Implementing Accessible Web Components

Based on the key considerations for keyboard accessibility and screen reader compatibility discussed in sections 6.1 and 6.2, this section demonstrates the implementation of a web component using React, making it interactive by both keyboards and screen readers. React is a free and open-source JavaScript library for building user interfaces or UI components [17].

```
import { useState } from "react";

const Checkbox = ( props ) => {

  // Destruct ARIA attributes and props.
  const {
    "aria-label": ariaLabel,
    "aria-labelledby": ariaLabelledBy,
    disabled,
    onChange,
    checked
  } = props;

  // Checkbox state.
  const [ isChecked, setIsChecked ] = useState( false );

  // Update checkbox state on click/keydown.
  const handleClick = ( event ) => {
    if( disabled ) {
      return;
    }
    if( onChange ) {
      onChange( event );
    } else {
      setIsChecked( !isChecked );
    }
  }
  const handleKeyDown = ( event ) => {
    event.preventDefault();
    if( disabled ) {
      return;
    }
    if( event.key === " " ) {
      if( onChange ) {
        onChange( event );
      } else {
        setIsChecked( !isChecked );
      }
    }
  }

  return <span
    tabIndex={ disabled ? -1 : 0 }
```

```
            role="checkbox"
44          aria-checked={ checked || isChecked }
            aria-label={ ariaLabel }
46          aria-labelledby={ ariaLabelledBy }
            aria-disabled={ disabled }
48          onClick={ handleClick }
            onKeyDown={ handleKeyDown }
50        />;
        }
```

Program 1. *Example of **Checkbox** component using **<span>** with **role** and ARIA attributes.*

Program 1 demonstrates how to create an accessible checkbox component using a **<span>** element in React instead of the semantic checkbox element (**<input type="checkbox">**). While semantically correct HTML elements are the recommended approach for web component development, there are situations where developers may choose to use non-semantic elements for visual styling or other purposes. In such cases, it becomes a big challenge to cover all additional accessibility considerations, so that the component functions as same as the corresponding HTML element.

Making elements focusable is the first step of keyboard accessibility. By default, a **<span>** element is not focusable using the "Tab" key. Line 42 addresses this by setting the **tabIndex** attribute to **0** when the component is enabled, allowing it to receive focus during keyboard navigation. This makes users access the component and interact with it. On the other hand, when disabled, the **tabIndex** is set to **-1** to prevent it from receiving focus.

Lines 18-39 define functions (**handleClick** and **handleKeyDown**) to manage the component's internal state (**isChecked**) based on user interaction (mouse clicks or keyboard presses). By following the common keyboard interaction for checkboxes, pressing the "Space" key triggers to update the state [13]. Moreover, these functions consider the disabled state of the component before updating the state.

To ensure compatibility with screen readers, line 43 sets the **role** attribute to **"checkbox"**. This instructs them to recognise the element as a checkbox control and announce it accordingly. Lines 44-47 utilise ARIA attributes to provide essential information for screen readers. For example, the **aria-checked** attribute conveys the checked state of the component (checked or unchecked). This allows screen readers to announce the current state of the checkbox to users. The **aria-label** attribute provides a textual description of the checkbox's purpose when no associated label element exists. This ensures screen reader users understand the function of the checkbox even without a visual representation. The **aria-labelledby** attribute references a corresponding label element using its ID, allowing the screen reader to announce the label's text for describing the checkbox. This is the preferred approach when a visible label element is associated with the checkbox.

This example showcased that accessible checkbox components can be effectively built using React and ARIA attributes, even with non-semantic elements like **<span>**. It highlighted the significance of exploring alternative accessibility solutions when standard HTML elements are insufficient and emphasised how ARIA attributes are important when delivering information to assistive technologies. Overusing non-semantic elements without knowing side effects will cause web applications less robust excluding users who rely on assistive technologies. M-Files Oy’s developers must be aware of the consequences when using incorrect elements while implementing web component.

6.4 Keyboard Shortcuts

Ensuring all web elements accessible allows users to interact with M-Files Web no matter which input method they use. However, navigating through the application often causes repeatedly pressing the “Tab” key for those users relying on a keyboard to access a certain function or an element.

For example, M-Files Web consists of multiple panels; the “Listing” panel displays the list of file or non-file objects in the M-Files file management system, selecting a file in the “Listing” panel opens in the “Preview” panel, and the detailed information describing the selected file is presented in the “Metadata” panel. To navigate across those panels, users must press the “Tab” key through all elements. Implementing keyboard shortcuts allows frequently used actions and elements faster for not only those users who rely on keyboard or experience motor limitations, but also those who use M-Files Web in a more advanced, efficient, or intensive way than average.

On the other hand, the design of effective keyboard shortcuts is one of the most important steps during the implementation. As an example, Sam Seely (2021) writes in his blog about emphasising the importance of creating a positive user experience with keyboard shortcuts by focusing on memorability and avoiding conflicts [18].

Key binding selection should prioritise user learnability and memorisation. A single key should reflect the action of a key combination. For example, the combination of “Control” and “S” keys is widely used for the saving action in software like web browsers because “S” represents the word “Save”. Similarly, the opening action often uses “Control” and “O” keys. It allows users to remember such keyboard shortcuts intuitively.

When implementing key combinations for keyboard shortcuts in M-Files Web, it is also important to avoid them from overriding existing browser or operating system commands.

As mentioned in the previous paragraph, the combination of “Control” and “S” keys is reserved for a built-in shortcut. Having the same keys for another action will block the existing feature unexpectedly. Thus, developers should be aware of the built-in shortcuts that may be affected by their implementation so that users can benefit from those system commands while seamlessly interacting with the web application.

6.5 Implementing Keyboard Shortcuts

This section demonstrates examples of design decisions of key combination and implementation of keyboard shortcuts for M-Files Web. The focus of this section is to access frequently used elements and sections faster using mnemonic key combinations. This includes moving focus to tab menus and panels within the application.

The keyboard shortcut functionality is implemented using JavaScript’s event handling capabilities. Event listeners are attached to relevant elements within the application to detect specific key combinations. For example, pressing a designated key combination triggers a function that programmatically focuses a target element.

For keyboard navigation to function effectively, the target element must be focusable. Semantic HTML elements such as buttons and inputs are inherently focusable by default in web browsers like Chrome, Edge, and Firefox. However, certain elements like **<div>** require additional configuration to be focusable.

The following code snippets (written in React) demonstrate the focus management of the example elements.

```
const HeaderTabs = () => {
  return (
    <ul className="header">
      <li className="header-tab">
        <a className="header-tab-link" href="/all">
          All
        </a>
      </li>
      {/* Additional tab elements omitted for brevity */}
    </ul>
  );
}
```

Program 2. *Example of tab menus of application header.*

```
const MetadataCardPanel = () => {
  return (
    <div tabIndex={ 0 } id="metadataCardPanel">
      {/* ... Content of the panel ...*/}
    </div>
```

```
  );
}
```

Program 3. *Example of metadata card panel component.*

These code examples in Program 2 and Program 3 illustrate the distinction between elements that are focusable by default and those requiring manual configuration. The link elements (“**header-tab-link**”) utilise the semantic **<a>** tag. Consequently, they are focusable using keyboard controls like “Tab” key navigation or programmatically calling **focus** method, without the need for setting the value of the **tabindex** attribute manually. However, it is important to note that these link elements still require a value for the **href** attribute to be recognised as focusable links.

In contrast, the **MetadataCardPanel** component utilises a **<div>** tag, which is not inherently focusable. Therefore, the **tabindex** attribute is explicitly set to **0** within the code to enable keyboard focus. When making elements focusable manually, it is important to ensure they have appropriate visual indicators to notify users of their focused state. More details about keyboard accessibility area illustrated in the section 6.1.

The following example demonstrates the implementation details of capturing and processing key combinations for shortcut functions. The example utilises the “Control” key as the initial key that initiates key combinations.

```
// Store key combinations.
const keys = [];

// Event handler for shortcut functionality.
const handleKeyDown = ( event ) => {
  // Get the current key pressed (lowercase).
  const eventKey = event.key.toLowerCase();
  const isCtrlPressed = event.ctrlKey;

  // Skip if control key is not pressed.
  if( !isCtrlPressed ) {
    return;
  }

  // Avoid registering same keys from repeated events.
  if( event.repeat ) {
    return;
  }

  // Skip "shift" and "alt" keys for potential layout variations.
  if( eventKey === "shift" || eventKey === "alt" ) {
    return;
  }

  // Start new key combination if control key is pressed by itself.
  if( eventKey === "control" ) {
    keys = [ "ctrl" ];
    return;
  }
```

```
30
        // Registers key in the combination.
32      keys.push( eventKey );

34      // ... Execute keyboard shortcut operations
        // based on the combination ...
36    }
```

Program 4. *Example of keydown event handler for capturing and processing key combinations.*

The **handleKeyDown** function servers as the event handler for capturing key presses. It first retrieves the lowercase representation of the pressed key and checks if the "Control" key is held down concurrently. The followings are explanations of the code example in Program 4.

- Lines 11-13: If the "Control" key is not held concurrently, the function exits as this is not the intended starting point for a keyboard shortcut.
- Lines 16-18: The function avoids registering the same key repeatedly due to potential event bouncing.
- Lines 21-13: "Shift" and "Alt" keys are excluded from being registered as part of the combination to prevent conflicts with keyboard layouts in different languages. It is worth noting that this way of registering key combinations has limitations with engaging "Shift" and "Alt" keys.
- Lines 26-29: If the "Control" key is pressed by itself, it is treated as the start of a new key combination, and the **keys** array is reset.
- Line 32: The pressed key is then pushed onto the **keys** array, effectively building the key combination.

Following the key combination registration process, well-defined functions are important to execute the desired actions triggered by these shortcuts. This section showcases two practical examples: moving focus to the metadata card panel (defined in Program 3) and navigating between the tab menus (defined in Program 2) within the application's header bar.

```
const moveFocusToMetadtaCardPanel = () => {
   // Get the metadata card panel element.
  const metadataCardPanelElement =
    document.getElementById( "metadataCardPanel" );

  // Focus the element only exists.
  if( metadataCardPanelElement ) {
    metadataCardPanelElement.focus();
  } else {
    // ... Handle the case where the element is
    // not found (optional) ...
```

```
    }
  }
```

Program 5. *Example of function to move focus to the metadata card panel.*

This function in Program 5 provides a programmatic approach to focus the metadata card panel element within the web application. It utilises the **document.getElementById** method to retrieve the element using its unique identifier ("**metadataCardPanel**"). The function checks the availability of the element using the DOM (Document Object Model) before attempting to focus the element. If the element is found, the **focus** method is invoked on the element, bringing it into focus for keyboard interaction. An optional section is included to address potential scenarios where the element may not be present in the DOM.

```
const moveFocusToHeaderTabs = () => {
   // Get all tab elements from the header bar.
  const headerTabElements = Array.from(
    document.getElementsByClassName( "header-tab-link" )
  );

  // Skip if no tab element exists.
  if( headerTabElements.length === 0 ) {
    return;
  }

  // Find the currently focused tab element (if any).
  const activeTabElement = headerTabElements.find( ( tab ) =>
    tab === document.activeElement );

  // Determine the next tab index based on current focus.
  const nextTabIndex = ( activeTabElement
    ? headerTabElements.indexOf( activeTabElement ) + 1
    : 0 ) % headerTabElements.length;

  // Focus the next tab element or the first one
  // if none are focused or at the end.
  headerTabElements[ nextTabIndex ].focus();
}
```

Program 6. *Example of function to move focus to the tab elements.*

The function **moveFocusToHeaderTabs** in Program 6 facilitates efficient navigation between the tab menus located within the application's header bar. It retrieves all elements with the class name "**header-tab-link**" using **document.getElementsByClassName**. The **Array.from** method is employed to convert the returned HTMLCollection object into a JavaScript array for easier manipulation. A preliminary check ensures at least one tab menu element exists before proceeding.

Next, the function identifies the currently focused tab element within the array using the **find** method. If a focused tab element is located, its index within **headerTabElements** array is retrieved. The **nextTabIndex** variable is subsequently calculated by either adding 1 to the current index (to move to the next tab) or by setting it to 0 (to move to the first tab) if no element is currently focused.

Finally, the function utilises the calculated **nextTabIndex** to access the corresponding element within the **headerTabElements** array and calls the **focus** method on that element, effectively shifting focus to the desired tab.

Establishing a mechanism to associate these functions with their corresponding keyboard shortcut combinations is an important step in the implementation of keyboard shortcut functionality.

```
const shortcutFunctions = {
  "ctrl+m+m": moveFocusToMetadataPanel,
  "ctrl+m+h": moveFocusToHeaderTabs,
  // ... Other shortcut functions ...
}
```

Program 7. *Example of object with pairs of a key combination and its corresponding function.*

The **shortcutFunctions** object in Program 7 serves as a central repository for mapping keyboard shortcut combinations to their corresponding action functions. It utilises a JavaScript object literal to define key-value pairs. Each key within the object represents a unique keyboard shortcut combination, constructed using a string format that reflects the "Control" key, and the actual keystrokes involved. The corresponding value for each key is a reference to the function that should be executed when that shortcut combination is pressed. It is worth noting that the object should be defined after the function declarations.

This mapping approach offers several advantages, including enhanced clarity as the association between keyboard shortcuts and their functionalities is readily apparent within the code, promoting code readability and maintainability. Secondly, adding, removing, or modifying shortcut-function pairings becomes a straightforward process of editing the object literal. This simplifies future updates and modifications. Finally, the scalable design ensures the system can adapt to evolving user needs. This structure readily accommodates a growing number of keyboard shortcuts within the application.

Key combinations for moving focus to different elements and sections were designed to start with the “Control” and “M” keys. This approach aims to help users infer the key bindings and their corresponding operations more easily due to the association of “M” with “Move Focus.” Building upon this base combination, frequently used elements and sections can be navigated with a following key. For example, the combination “ctrl+m+m” in Program 7 moves the focus to metadata card panel, while “ctrl+m+h” focuses on the tab menus within the application’s header bar. This consistent structure, where the initial combination implies a general action (“Move Focus”), improves user learning and fulfils expectation around keyboard shortcut functionality.

Having established the mechanism for mapping keyboard shortcuts to their corresponding functions, we can now integrate this functionality into the **handleKeyDown** event handler (previously presented in Program 4). The following example elaborates on how the registered key combinations are compared against the defined shortcuts and the corresponding actions are triggered.

```
    const handleKeyDown = ( event ) => {
2     // ... Key registration process above ...

4     // Convert key combination to a string.
      const keyCombination = keys.join( "+" );
6
      // Check if the key combination is registered as a shortcut.
8     if( Object.keys( shortcutFunctions )
          .includes( keyCombination ) ) {
10      // Execute the corresponding shortcut function.
        shortcutFunctions[ keyCombination ]();
12      // Prevent any default actions afterward.
        event.preventDefault();
14    }

16    // For moving focus shortcuts (starting with "ctrl+m"),
      // reassign the key combination after current navigation,
18    // so that other navigations can be executed consecutively
      // without pressing "control" and "m" keys again.
20    if( keyCombination.startsWith( "ctrl+m" ) &&
          keys.length > 2 ) {
22      keys = [ "ctrl", "m" ];
      }
24  }
```

Program 8. *Example of applying shortcut functions in* ***handleKeyDown*** *(continued from Program 4).*

Following the key registration process within the **handleKeyDown** function, the registered keys are concatenated into a single string using the "join" method, separated by plus signs ("+"). This string representation of the key combination (e.g., "ctrl+m+m") is then compared against the keys defined within the **shortcutFunctions** object.

- Lines 8-9: The **Object.keys(shortcutFunctions).includes(keyCombination)** method determines if the constructed key combination string exists as a key within the **shortcutFunctions** object.
- Line 11: Upon a successful match, the corresponding function reference associated with that key in the **shortcutFunctions** object is retrieved and invoked using bracket notation ("[]"). This effectively executes the desired action mapped to the keyboard shortcut combination.

- Line 13: The **event.preventDefault()** method is called to prevent any default browser behaviour associated with the pressed keys, ensuring the custom functionality takes precedence.
- Lines 20-23: This block incorporates a specific handling mechanism for keyboard shortcuts that involve moving focus within the application (identified by starting with "ctrl+m"). If the key combination begins with the string "ctrl+m" and the **keys** array has more than two elements, the **keys** array is deliberately reset to contain only "ctrl" and "m." This facilitates a more streamlined user experience for consecutive focus navigation actions by allowing users to navigate different elements without releasing and pressing the "Control" and "M" keys again. In result, it simplifies the interaction flow.

The final step involves attaching the event listener for the **handleKeyDown** function. This enables the web application to listen for key presses and trigger the appropriate actions based on the captured key combinations. There are two primary approaches to achieve this. One is using **document.addEventListener** and the other is using the React event handler **onKeyDown**.

```
// Use addEventListener.
document.addEventListener( "keydown", handleKeyDown );

// Or use React event handler.
<div onKeyDown={ handleKeyDown }>
  {/* ... Application content ... */}
</div>
```

Program 9. *Example of adding the event listener for the **handleKeyDown** function.*

As shown in Program 9, the **handleKeyDown** function is effectively integrated into the application, ensuring it captures key combinations, compares them to defined shortcuts, and executes the corresponding actions, ultimately enhancing the user experience through efficient keyboard navigation.

6.6 Further Implementation

Chapter 6 presented accessibility considerations and code example when implementing accessible web components and keyboard shortcuts. Although the examples do not cover all cases in M-Files Web, understanding the key considerations and those accessibility techniques will help M-Files Oy’s developers resolve issues regarding those components inaccessible or inoperable by keyboard and screen reader users.

In addition to ensuring keyboard and screen reader, emerging technologies like voice input and control offer users alternative options, allowing users to interact with M-Files

Web more dynamically. Future iterations of web application could include the integration of voice control functionalities, making the application navigable, and interactive by voice commands.

Telemetry analysis offers a valuable approach to understanding user behaviour within the web application. By collecting and analysing telemetry data, developers can gain insights into how frequently users uses accessibility features such as alternative navigation techniques. This data can contain an important metric for evaluating the effectiveness of implemented accessibility features and identifying areas for further optimisation. By monitoring user behaviour patterns, developers can prioritise accessibility features and enhance the user experience for individuals with disabilities.

7. CONCLUSION

This thesis aimed to support the developers at the company, M-Files Oy, in evaluating the accessibility of its web application, M-Files Web, and to provide practical improvement methods for areas that fail to meet WCAG standards. Prior to the evaluation, an academic text was reviewed to understand the concept of accessibility and the real-world challenges faced by individuals with various disabilities. The WCAG guidelines were also reviewed to study their structure and recommended techniques. Based on this foundation, the application's accessibility was evaluated using both automated tools and manual testing methods.

Although the M-Files Design System guidelines helped M-Files Web offer a basic level of accessibility for users with vision and those using both a mouse and keyboard, the evaluation revealed major limitations for users who depend on assistive technologies to access all features and content. For example, the top-pane menus in the application were neither accessible nor operable via keyboard navigation, creating a serious obstacle for users who do not use a mouse.

The most important area for improvement in M-Files Web was converting interactive elements into accessible components. This study collected information on the use of semantic HTML elements, ARIA attributes, focus management techniques, and strategies to ensure compatibility with assistive technologies such as screen readers. In addition, it provided practical code examples that developers at M-Files Oy can apply directly to enhance the accessibility of the application.

This study addressed the research questions as follows. First, to support developer learning, it emphasised the importance of accessibility and provided guidance on established standards such as WCAG, the M-Files Design System, and the M-Files Tone of Voice. Second, the evaluation identified specific accessibility issues and areas needing improvement within M-Files Web. Third, the study proposed technical solutions—semantic structure, ARIA implementation, and proper focus handling—supported by actionable code example.

While this study outlined core accessibility issues and proposed essential improvements within M-Files Web, long-term compliance will depend on ongoing integration of accessibility practices into the development cycle. As the application evolves, regular audits, user testing—especially with assistive technology users—and updates aligned with changes in WCAG, and platform guidelines will be necessary to ensure sustained accessibility.

REFERENCES

[1] Association for Intelligent Information management (AIIM). (2023). AIIM 2023 State of the Intelligent Information Management Industry. Accessed: 10 May 2024. Available: https://info.aiim.org/state-of-the-intelligent-information-management-industry-2023

[2] Jim T, Michael B, Christian H, Shawn H, Andrew K, Patrick L, at al. (2006). Web Accessibility: Web standards and Regulatory Compliance. Apress Berkeley, CA. Accessed: 18 November 2024. Available: https://link.springer.com/book/10.1007/978-1-4302-0188-5

[3] W3C Web Accessibility Initiative (WAI). (2023). Accessibility Principles. Accessed: 11 May 2024. Available: https://www.w3.org/WAI/fundamentals/accessibility-principles

[4] World Wide Web Consortium (W3C). (2023). Web Content Accessibility Guidelines (WCAG) 2.2. Accessed: 11 May 2024. Available: https://www.w3.org/TR/WCAG22

[5] World Wide Web Consortium (W3C). (2024). How to Meet WCAG (Quick Reference). Accessed: 15 Mar 2025. Available: https://www.w3.org/WAI/WCAG22/quickref

[6] World Wide Web Consortium (W3C). (2024). ARIA6: Using aria-label to provide labels for objects. Accessed: 15 Mar 2025. Available: https://www.w3.org/WAI/WCAG22/Techniques/aria/ARIA6.html

[7] World Wide Web Consortium (W3C). (2024). F3: Failure of Success Criterion 1.1.1 due to using CSS to include images that convey important information. Accessed: 15 Mar 2025. Available: https://www.w3.org/WAI/WCAG22/Techniques/failures/F3.html

[8] World Wide Web Consortium (W3C). (2024). All WCAG 2.2 Techniques. Accessed: 15 Mar 2025. Available: https://www.w3.org/WAI/WCAG22/Techniques/

[9] M-Files. (2024). Accessed: 11 May 2024. Available: https://www.m-files.com

[10] M-Files Design System. (2024). M-Files. Unpublished internal company document.

[11] W3C Web Accessibility Initiative (WAI). (2023). Evaluating Web Accessibility Overview. Accessed: 12 May 2024. Available: https://www.w3.org/WAI/test-evaluate/

[12] Web AIM. (2024). WAVE Web Accessibility Evaluation tool. Accessed: 12 May 2024. Available: https://wave.webaim.org/

[13] Deque Systems, Inc. (2024). Axe: Accessibility Testing Tools and Software. Accessed: 12 May 2024. Available: https://www.deque.com/axe/

[14] Web AIM. (2022). Keyboard Accessibility. Accessed: 12 May 2024. Available: https://webaim.org/techniques/keyboard/

[15] MDN Web Docs. (2023). Semantics. Accessed: 12 May 2024. Available: https://developer.mozilla.org/en-US/docs/Glossary/Semantics

[16] MDN Web Docs. (2024). ARIA. Accessed: 12 May 2024. Available: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA

[17] React. (2024). Accessed: 12 May 2024. Available: https://react.dev/

[18] Sam Seely. (2021). How to design great keyboard shortcuts. Accessed: 12 May 2024. Available: https://knock.app/blog/how-to-design-great-keyboard-shortcuts

APPENDIX 1: USER SURVEY: M-FILES CLIENTS ACCESSIBILITY

User Survey: M-Files Clients Accessibility

Thank you for your time and willingness to participate in this user survey. This survey is being conducted as part of a master's thesis aimed at evaluating and improving the accessibility of M-Files clients for all users.

Accessibility is all about creating an inclusive experience for everyone using M-Files products. This means ensuring the platform is usable regardless of a person's abilities.

This survey aims to understand your experience using M-Files clients such as M-Files Desktop, M-Files Web, M-Files Admin tool, M-Files Ment, and so on. Your feedback will be crucial in identifying areas where M-Files can enhance its accessibility offerings and ensure a more inclusive user experience for everyone.

This survey should take approximately 10-30 minutes to complete. All your responses will be kept confidential and anonymous.

Thank you again for your participation.

* Required

Part 1: General Information

1. How often do you use M-Files clients? *

- ◯ Daily
- ◯ Weekly
- ◯ Monthly
- ◯ Less than Often
- ◯ Never

2. Do you identify yourself as having any visual, auditory, motor, or cognitive impairments? *

- ◯ Yes
- ◯ No

3. If yes to question 2, please specify the type of impairment(s) you experience.

- ☐ Visual
- ☐ Auditory
- ☐ Motor
- ☐ Cognitive
- ☐ Other

Part 2: Navigation and Use

4. Rate the ease of navigating M-Files clients on a scale of 1 (Very Difficult) to 5 (Very Easy). *

	Very Difficult	Difficult	Average	Easy	Very Easy
Rate	○	○	○	○	○

5. Please describe your reasons of the answer you gave in question 4.

6. Were you able to locate the information or functionalities you needed easily? *

○ Yes

○ No

7. If no to question 6, what challenges did you encounter in finding the information or functionalities?

8. Did you find the layouts of M-Files clients clear and organized? *

○ Yes

○ No

9. If no to question 8, which clients and what aspects of the layouts you found clear or unclear?

10. Were you able to use all the features of M-Files clients without difficulty? *

- ◯ Yes
- ◯ No

11. If no to question 10, which features posed challenges and why?

Part 3: Accessibility Features

12. Did you use any built-in accessibility features that are currently existing in M-Files clients (e.g., screen reader compatibility, keyboard navigation)? *

- ◯ Yes
- ◯ No

13. If yes to question 12, which features did you find helpful or difficult? Please also describe your experiences with keyboard navigation or screen reader compatibility.

14. Are there any additional accessibility features you would find beneficial?

Part 4: Screen Design and Readability

15. Rate the clarity and readability of the text on M-Files clients on a scale of 1 (Very Difficult) to 5 (Very Easy). *

	Very Difficult	Difficult	Average	Easy	Very Easy
Rate	○	○	○	○	○

16. Please describe your reasons of the answer you gave in question 15.

17. Were the colors used on M-Files clients easy on the eyes? *

○ Yes

○ No

18. If no to question 17, which clients and how the colors impacted your experience?

19. Did you encounter any issues with image or icon clarity? *

○ Yes

○ No

20. If yes to question 19, which images or icons were unclear and what made them unclear? (e.g., lack of detail, poor contrast, missing context, lack of alternative text descriptions)

Part 5: Overall Experience

21. Overall, how satisfied are you with the accessibility of M-Files clients? *

	Very dissatisfied	Somewhat dissatisfied	Neither satisfied nor dissatisfied	Somewhat satisfied	Very satisfied
Rate	◯	◯	◯	◯	◯

22. Do you have any additional comments or suggestions for improving the accessibility of M-Files clients?