American University Kyiv

WEB ACCESSIBILITY OPTIMIZATION IN FRONT-END DEVELOPMENT
ОПТИМІЗАЦІЯ ВЕБ-ДОСТУПНОСТІ В ІНТЕРФЕЙСНІЙ РОЗРОБЦІ

By Starchenko Anastasiia

Presented in Partial Fulfillment of the Requirements for the Degree

Master of Software Engineering

APPROVED BY:

Serhii Lupenko, Ph.D.

2025

Abstract

This capstone project explores the critical importance of web accessibility in front-end development, emphasizing inclusivity for users with diverse abilities. With over 1 billion individuals globally experiencing some form of disability, this study focuses on creating a web accessibility analysis and optimization web app. The research identifies gaps in existing tools like Lighthouse, which often miss nuanced accessibility issues. Using WCAG standards, the app analyzes HTML code to detect violations, such as missing alt text, improper ARIA roles, and low contrast ratios. Featuring visualization, recommendations, and downloadable reports, the app aids developers in achieving WCAG AA compliance. The project underscores the value of inclusive digital design, offering practical implications for developers, businesses, and educators, while setting a foundation for future enhancements, including AI integration and internationalization.

Table of Contents

Introduction 5

Chapter 1. The Importance of Web Accessibility in Modern Development 6

1.1 The Curb Cut Effect 6

1.2 Multifaceted Value of Accessible Web Development 7

1.2.1 Legal Considerations 7

1.2.2 Ethical Considerations 8

1.2.3 Business Considerations 8

1.2.4 Accesibility and Profitability 9

1.3 Web Accessibility in the Development Process 9

1.3.1 Accessibility API 10

1.3.2 How Assistive Technologies Interact with a Web Page via the Accessibility API 11

1.4 Standards for Front-End Accessibility 12

1.4.1 Web Content Accessibility Guidelines (WCAG) 13

1.4.2 ARIA (Accessible Rich Internet Applications) 13

1.4.3 Semantic HTML 14

1.5 The Role of Assistive Technologies 14

1.5.1 How Assistive Technologies Interact With Accessible Web Content 14

1.5.2 Best Practices for Compatibility with Assistive Technologies 15

1.6 Challenges in Implementing Accessibility 17

1.6.1 Common Barriers to Accessibility Implementation 17

1.7 Case Studies of Accessible Websites 18

1.7.1 Apple 18

1.7.2 Ukrainian Government Portal 19

Chapter 2. Implementation of a Web App for Accessibility Analysis and Optimization 21

2.1 Key Functionalities 24

2.2System Modules 25

2.3 Workflow Description 25

2.4 Practical Relevance 26

2.5 Code Functionality 26

2.6 Key Advantages 32

2.7 Planned Enhancements and Extensions 32

Chapter 3. Conclusions 36

3.1 Summary of Contributions 36

3.2 Practical Implications 37

3.3 Future Directions 37
3.4 Final Conclusion 38
References 39

Introduction

Web accessibility is one of the most crucial aspects of modern web development, focusing on making digital platforms usable and inclusive for everyone, regardless of their physical, cognitive, or technological limitations. According to the World Health Organization (WHO), over 1 billion people, approximately 16% of the global population, live with some form of disability. [1] This number is expected to grow as populations age and as global health crises, such as conflicts and pandemics, increase the prevalence of temporary and permanent disabilities. Web accessibility ensures that digital platforms, including websites, applications, and online services, are designed to accommodate these diverse user needs, providing an equitable experience for all.

While web accessibility is a global issue relevant to all countries and societies, its importance is particularly pronounced in the context of Ukraine. The ongoing war has drastically reshaped the country's social and technological landscape. With millions of Ukrainians relying on digital platforms for essential services such as healthcare, education, e-commerce, and government communication, accessibility has become more critical than ever. The war has also led to an increase in disability rates due to injuries sustained in conflict, making it imperative to ensure that online resources are inclusive and accessible to individuals with varying needs. For instance, wounded veterans, displaced individuals with limited access to traditional support services, and others affected by the war require digital solutions that accommodate their circumstances.

Chapter 1. The Importance of Web Accessibility in Modern Development

1.1 The Curb Cut Effect

A significant barrier to achieving widespread web accessibility is the prevalence of myths and misconceptions that deter developers and organizations from prioritizing inclusive design. One common myth is that web accessibility is only relevant for a small percentage of users, while in reality, accessibility benefits a much broader audience [2]. Features like closed captions, originally developed for individuals with hearing impairments, are now widely used by non-native speakers, people in noisy environments, or those who prefer silent viewing. Similarly, keyboard navigation, designed for users with motor impairments, has become a favored tool for power users seeking efficiency and speed in navigating digital platforms.

Another misconception is that accessibility compromises the design or interactivity of a website. However, modern development practices and technologies show that inclusive design can seamlessly coexist with aesthetic and innovative user experiences. For example, high-contrast themes improve readability for users with visual impairments while also helping users in bright outdoor environments. Moreover, some believe that accessibility is too costly or complex to implement, but embedding accessibility from the start of the development process is often cost-effective and prevents the need for expensive retrofitting later.

These myths fail to account for the curb cut effect, which demonstrates how solutions originally designed for a specific group often bring widespread benefits to everyone. The concept of curb cut effect, rooted in urban design, originated in the mid-20th century when sloped ramps were introduced on street curbs to assist wheelchair users. However, over time, it became evident that these ramps were not only beneficial for individuals with mobility impairments but also for parents with strollers, delivery workers with carts, travelers with luggage, and even bicyclists. The curb cut effect has since become a metaphor in various fields, including web accessibility, to emphasize the universal benefits of inclusive design.

In the context of web development, the curb cut effect underscores how designing for accessibility improves user experience for all users, not just those with disabilities. Features initially introduced to accommodate specific needs, such as screen readers or keyboard navigation, often enhance usability and convenience for broader audiences [3]. Other examples include:

- Keyboard navigation - originally designed for users with motor impairments, keyboard navigation is widely used by power users who prefer shortcuts over mouse interaction.

- Closed captions and transcripts - primarily created for individuals who are deaf or hard of hearing, captions also benefit non-native speakers, users in noisy environments, and those who prefer to consume content silently.
- Responsive design - techniques developed to assist users with visual impairments, such as scalable text and high-contrast themes, improve readability for all users, especially on mobile devices or in bright environments.
- Voice assistants and speech recognition - initially developed for individuals with visual or motor impairments, technologies like Siri, Alexa, and Google Assistant are now mainstream tools used by millions worldwide for convenience and multitasking.

1.2 Multifaceted Value of Accessible Web Development

Web accessibility is not only a technical consideration but also a crucial aspect of legal compliance, ethical responsibility, and strategic business growth. By ensuring inclusivity in web design, organizations can avoid legal penalties, uphold ethical principles, expand their audience, and increase revenue.

1.2.1 Legal Considerations

Web accessibility is increasingly regulated by laws worldwide, holding organizations accountable for ensuring their digital platforms are inclusive. Non-compliance can lead to substantial fines and reputational damage.

- The Americans with Disabilities Act (ADA) in the United States establishes that websites must be accessible to individuals with disabilities, treating digital spaces as public accommodations. Lawsuits under the ADA have surged in recent years, highlighting the need for organizations to prioritize accessibility [4].
- In the European Union, the European Accessibility Act (EAA) sets binding requirements for digital accessibility across member states. Businesses offering digital services must adhere to these standards to operate legally in the EU [5].
- In Ukraine, the DSTU EN 301549:2019 standard aligns with the Web Content Accessibility Guidelines (WCAG) and regulates accessibility for information and communication technology. While relatively new, this standard reflects Ukraine's commitment to aligning with international accessibility norms, especially as digital transformation accelerates in the region [6].

These regulations not only protect the rights of individuals with disabilities but also drive organizations toward more inclusive and universally beneficial design practices.

1.2.2 Ethical Considerations

At its core, web accessibility is an ethical imperative. The internet has become an essential aspect of modern life, facilitating education, employment, healthcare, and communication. Excluding individuals with disabilities from accessing online content is both unjust and discriminatory.

Ethical web development is grounded in the principle of equal access for all. Everyone, regardless of ability, should have the opportunity to navigate websites, access information, and participate in digital spaces. By prioritizing accessibility, developers and organizations demonstrate a commitment to inclusivity and equity, contributing to a more just digital society.

1.2.3 Business Considerations

In addition to being a legal and ethical requirement, web accessibility offers significant business benefits.

- Expanding the audience. Accessible websites cater to individuals with disabilities, who represent a substantial portion of the global population—over 1 billion people, or 16% of the world's population, according to the World Health Organization. Furthermore, accessibility features often appeal to broader user groups, such as non-native speakers or older adults, who benefit from improved usability.
- Improved SEO. Many accessibility practices, such as providing alternative text for images and ensuring proper heading structure, enhance search engine optimization (SEO). This increases a website’s visibility in search results, attracting more users and driving organic traffic [7].
- Increased revenue. Accessibility directly impacts a company’s bottom line by expanding its customer base. More users mean more potential customers, and inclusive design ensures no one is excluded from engaging with a company’s products or services. For instance, research "The Click-Away Pound Report 2019" found that inaccessible websites led to lost revenue, as many users with disabilities abandoned purchases due to barriers. The report also shows that accessible e-commerce platforms outperform their non-accessible counterparts, as they accommodate a more diverse range of users [8].

While it is challenging to collect precise statistics on the financial benefits of accessibility, the curb cut effect suggests that investments in inclusive design often yield returns far beyond their initial scope.

1.2.4 Accesibility and Profitability

The material benefits of web accessibility are clear: inclusive websites attract more users, and more users result in greater revenue potential. This is particularly true in sectors like e-commerce, where user experience directly impacts sales. For example:

- An accessible website can reach users who rely on assistive technologies, ensuring they can navigate the purchasing process without barriers.
- Enhanced usability features, such as simplified navigation and responsive design, improve the experience for all users, increasing conversion rates.
- Companies with accessible websites often experience enhanced brand loyalty, as inclusivity resonates positively with socially conscious consumers.

Accessibility is not merely an obligation; it is a strategic advantage. Businesses that prioritize inclusivity position themselves as leaders in their industry, attracting diverse audiences and fostering long-term growth.

1.3 Web Accessibility in the Development Process

Web accessibility is a comprehensive practice that encompasses every stage of the development lifecycle, including planning, design, coding, and testing. Each phase plays a crucial role in ensuring that digital platforms are inclusive and usable for all users, regardless of their abilities.

- Design phase. Accessibility begins with inclusive design principles. Designers must consider color contrast, typography, layout flexibility, and interactive element visibility to accommodate users with diverse needs. Prototypes should include accessible navigation and prioritize content clarity.

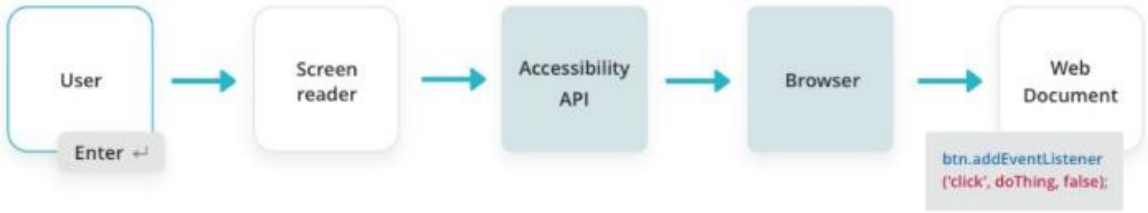
- Development phase. Developers are responsible for implementing accessibility features in the codebase. This includes semantic HTML, ARIA (Accessible Rich Internet Applications) roles, and responsive design techniques. Adhering to accessibility guidelines during development ensures compatibility with assistive technologies like screen readers.

- Testing and QA. Testing for accessibility is a vital part of the quality assurance process. Developers and testers use automated tools, manual checks, and user testing with assistive technologies to identify and fix barriers.

Front-end development is pivotal in delivering accessible experiences, as it directly shapes how users interact with a website or application. By adhering to established accessibility standards and leveraging specialized tools, front-end developers can ensure that their code supports usability for all.

1.3.1 Accessibility API

As technical professionals, it is crucial to understand how assistive technologies operate, as they rely on the Accessibility API integrated into operating systems, simplifying the testing process. This ensures that if a feature functions correctly on a standard screen reader, it will typically work across other devices, such as Braille displays, since both utilize the same foundational API.



This diagram illustrates the flow of interaction between a user and a web document when accessibility features are utilized, emphasizing the role of assistive technologies and APIs. Here's a detailed breakdown:

- User interaction:

The user initiates an action, such as pressing the "Enter" key, to interact with a web application or website. This step represents the starting point of user input.

- Screen reader:

The input is processed by a screen reader, an assistive technology tool designed to interpret and verbalize on-screen content for users with visual impairments. The screen reader communicates this action to the system.

- Accessibility API:

The screen reader leverages an Accessibility API (Application Programming Interface). This API acts as a bridge between the assistive technology and the browser, ensuring that the user’s actions and content on the page are accurately interpreted and conveyed.

- Browser:

The browser receives input from the Accessibility API and processes it. The browser interprets the intended action and facilitates its execution within the context of the web page.

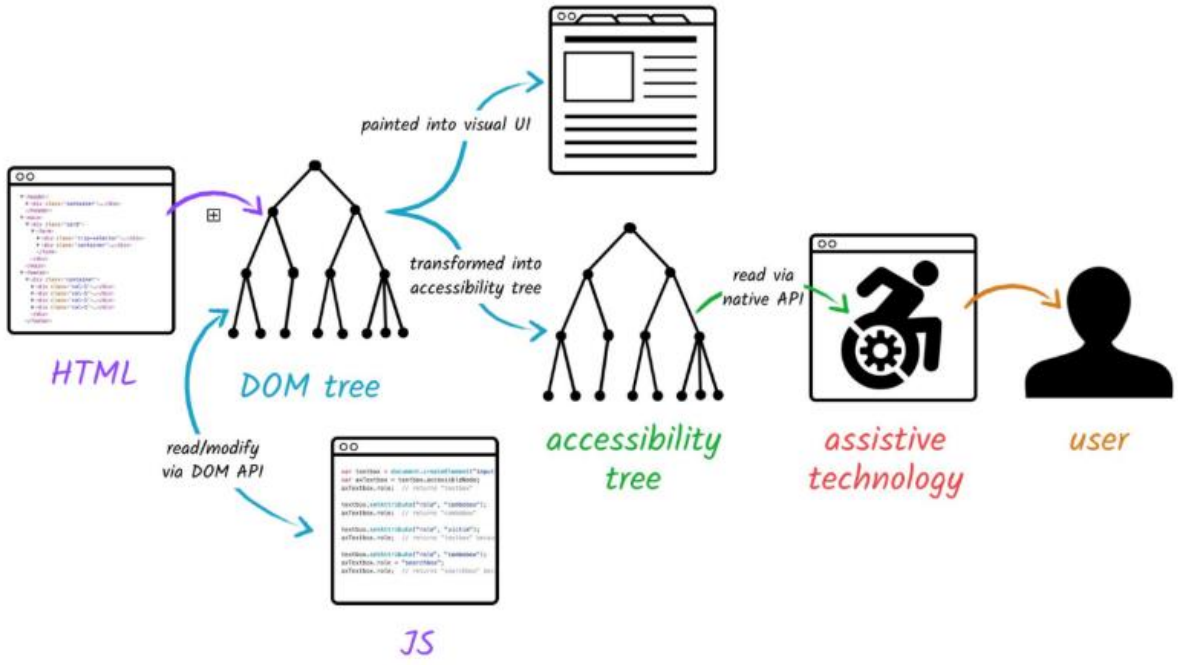
- Web Document:

The browser interacts with the web document, where the core HTML, CSS, and JavaScript code reside. For instance, in the diagram, there is an example of JavaScript code:

```
btn.addEventListener('click', doThing, false);
```

This code snippet demonstrates a typical event listener in JavaScript, where a click event triggers a function called doThing. This ensures interactive components are responsive to user actions.

1.3.2 How Assistive Technologies Interact with a Web Page via the Accessibility API



This image illustrates the interaction between HTML, the DOM tree, the Accessibility Tree, assistive technologies, and the user, highlighting how accessibility is integrated into the web development process:

- HTML as the starting point:

The process begins with the HTML of a web page, which defines the structure and content of the page using semantic elements like <button> or <section>. This provides the foundation for building accessible web content.

- DOM tree construction:

When the browser loads the page, it parses the HTML to create the DOM (Document Object Model) tree, a hierarchical representation of all elements on the page. The DOM tree can be read and modified using JavaScript via the DOM API, enabling dynamic updates to the page content and structure.

- Accessibility tree creation:

From the DOM tree, the browser generates an Accessibility Tree, a simplified representation that focuses only on the semantic and meaningful elements needed for assistive technologies. For example, elements like or without roles or ARIA attributes are omitted, while roles, states, and text content of other elements are included.

- Rendering into the visual UI:

The DOM tree is also used to render the visual user interface (UI) of the web page, ensuring that sighted users can see and interact with the content.

- Interaction with assistive technologies:

The Accessibility Tree is read by assistive technologies (e.g., screen readers, Braille displays) through the native Accessibility API provided by the operating system. This API acts as a bridge between the browser and assistive tools, ensuring that users can perceive and interact with the page.

- User interaction:

The user, often using assistive tools, interacts with the web page. For example, they might use a screen reader to navigate through headings or activate a button.

- JavaScript for interactivity:

Interactions triggered by the user, such as clicking a button or submitting a form, are handled by JavaScript. JavaScript uses event listeners (e.g., btn.addEventListener('click', doThing, false)) to capture user actions and execute associated functionality.

1.4 Standards for Front-End Accessibility

Front-end accessibility standards provide a structured framework for developers to create web interfaces that are inclusive and functional for everyone. Adherence to these standards ensures compatibility with assistive technologies and compliance with legal requirements. Below is a deeper dive into the key standards guiding front-end accessibility.

1.4.1 Web Content Accessibility Guidelines (WCAG)

The WCAG, developed by the World Wide Web Consortium (W3C), is the most widely recognized standard for web accessibility. It is organized around four main principles:

- Perceivable. Information and UI components must be presented in ways users can perceive. For instance:
    - Use text alternatives (e.g., alt text) for non-text content like images.
    - Ensure color contrast ratios of at least 4.5:1 for normal text and 3:1 for large text.
    - Provide captions for audio and video content.
- Operable. Navigation and UI must be usable by all users, including those relying on keyboards or assistive devices. Key considerations include:
    - Ensuring that all interactive elements are keyboard-accessible.
    - Providing sufficient time for users to interact with content.
    - Avoiding elements like flashing animations that could trigger seizures.
- Understandable. Information and operations must be clear and easy to understand:
    - Use readable fonts and simple language.
    - Offer instructions and error messages that guide users.
- Robust. Content must be accessible across a range of current and future technologies:
    - Use semantic HTML to ensure compatibility with assistive technologies.

WCAG 2.1, the most recent version, includes additional criteria to address mobile accessibility, low-vision users, and those with cognitive or learning disabilities [9].

1.4.2 ARIA (Accessible Rich Internet Applications)

The ARIA specification is essential for dynamic, interactive web applications. It provides additional attributes to HTML, enabling developers to describe the roles, properties, and states of UI elements.

Key ARIA practices include:

- Assigning roles like role="button" or role="navigation" to enhance the semantic meaning of elements.
- Using aria-label and aria-labelledby to define clear labels for interactive elements.
- Applying aria-live regions to alert screen readers of dynamically updated content.

Although ARIA is powerful, developers are encouraged to use native HTML elements whenever possible, as they are inherently accessible [10].

1.4.3 Semantic HTML

Semantic HTML is a cornerstone of front-end accessibility. It involves using HTML elements that convey the purpose of content and structure, allowing assistive technologies to interpret pages correctly.

Examples of semantic elements include:

- Structural tags: <header>, <footer>, <article>, <section> for clear content hierarchy.
- Interactive tags: <button>, <input>, <a> for elements users can interact with.
- Table tags: <table>, <thead>, <tbody>, <th> for accessible data representation.

Proper semantic markup reduces the need for additional ARIA attributes and improves usability for all users.

1.5 The Role of Assistive Technologies

Assistive technologies are vital tools that empower individuals with disabilities to access and interact with digital content. These technologies bridge the gap between the inherent limitations of users and the web's complexities, ensuring inclusivity and usability. For developers, understanding how these tools function and adopting best practices for compatibility is essential to creating genuinely accessible websites.

1.5.1 How Assistive Technologies Interact With Accessible Web Content

1. Screen readers.

   Screen readers are software applications that convert text and elements on a screen into synthesized speech or braille output. Popular screen readers include NVDA (NonVisual Desktop Access), JAWS (Job Access With Speech), and VoiceOver (for macOS and iOS). They rely on semantic HTML and ARIA roles to interpret and navigate web content effectively. For example, <h1> tags help users understand the structure of a page, while ARIA landmarks like role="navigation" assist in jumping to specific sections.

   Common challenges include poorly labeled buttons, non-descriptive links (e.g., "click here"), or improperly implemented ARIA roles can confuse users [11].

2. Voice recognition software.

Voice recognition tools, like Dragon NaturallySpeaking or Google Assistant, enable users to interact with web interfaces using spoken commands. These tools interpret text input fields, clickable buttons, and hyperlinks through semantic HTML and accessible design patterns. For instance, a properly labeled <button> element is easier to identify and activate using voice commands.

Common challenges include dynamic or non-standard controls (e.g., custom dropdowns) can be difficult for voice recognition software to interpret unless coded with accessibility in mind [12].

3. Braille displays.

   Refreshable braille displays convert text content into braille, allowing visually impaired users to read web content tactually. These devices integrate seamlessly with screen readers to provide a dual-mode experience. Braille displays depend heavily on clean, structured HTML and semantic markup. They require precise labeling and logical navigation to avoid confusion.

   Common challenges include overcomplicated layouts or missing alt text for images can disrupt the tactile reading experience.

4. Screen magnifiers.

   These tools enlarge portions of the screen, often requiring developers to maintain reflowable layouts and ensure proper zoom functionality.

5. Color filters.
   Used by individuals with color blindness to differentiate between similar hues. Developers must ensure proper color contrast ratios.

1.5.2 Best Practices for Compatibility with Assistive Technologies

Ensuring compatibility with assistive technologies requires a focus on inclusive design principles and adherence to accessibility standards. Emphasizing clear structure and meaningful organization allows assistive tools to interpret and present content effectively. It is essential to design interfaces that are operable, perceivable, and understandable by a diverse range of users, considering varying needs and abilities. Incorporating accessibility as a core part of the design and development process, including regular testing for compatibility with assistive technologies, helps identify potential barriers. This approach fosters equitable access and usability, ensuring that digital experiences are inclusive for all users.

1. Prioritize semantic HTML

- Use semantic elements (<header>, <main>, <article>) to provide meaningful structure.
- Avoid using and as substitutes for buttons or links.

2. Provide descriptive labels
    - Use the aria-label or aria-labelledby attributes to clarify the purpose of UI elements.
    - Avoid empty or vague link texts like "Read more"; instead, use descriptive labels like "Read more about our accessibility features."
3. Ensure keyboard accessibility
    - Make all interactive elements, such as buttons, links, and forms, operable via keyboard alone.
    - Use logical tab orders and visible focus states to guide navigation.
4. Test with assistive technologies
    - Regularly test web pages using screen readers (e.g., NVDA, VoiceOver) and voice recognition software.
    - Simulate tactile navigation with braille displays where possible.
5. Implement ARIA thoughtfully
    - Use ARIA roles and attributes sparingly and only to enhance native HTML capabilities. Overusing ARIA can lead to confusion.
    - Ensure dynamic content changes are announced to screen readers using aria-live attributes.
6. Design for error handling
    - Provide clear, concise error messages for forms. For example, use aria-describedby to link errors to their corresponding fields.
    - Avoid relying on color alone to indicate errors; include text descriptions.
7. Focus on responsive and mobile accessibility
    - Design interfaces that adapt seamlessly to screen readers and voice controls on mobile devices.

- Use WCAG's mobile-specific guidelines, such as maintaining large touch targets and avoiding hover-dependent actions.

1.6 Challenges in Implementing Accessibility

Web accessibility is a crucial element of modern web development, ensuring that digital platforms are usable by people with various disabilities. However, many developers face significant challenges when trying to implement accessibility features. These barriers can stem from a lack of awareness, limited budgets, or outdated legacy codebases. In this chapter, we will explore these common obstacles and propose strategies for overcoming them.

1.6.1 Common Barriers to Accessibility Implementation

1. Lack of awareness and knowledge

   A significant challenge in the implementation of web accessibility is the lack of awareness and understanding among developers. Many developers are unaware of the legal, ethical, and business implications of accessibility. They may not be familiar with accessibility guidelines like WCAG (Web Content Accessibility Guidelines) or the ARIA (Accessible Rich Internet Applications) standards. This lack of knowledge can lead to the neglect of accessibility during the development process.

   Impact: websites often miss critical accessibility features, such as proper alt text for images, keyboard navigation, or accessible form fields. This exclusion leaves users with disabilities unable to fully engage with the website.

2. Limited budgets and resources

   Implementing accessibility often requires additional time, resources, and expertise, all of which may not be readily available, particularly in small teams or organizations with limited budgets. Accessibility may be viewed as a non-essential luxury, especially when compared to other business priorities, such as speed, design aesthetics, or functionality.

   Impact: accessibility features might be deprioritized or cut from the final product, resulting in a website that is not inclusive for all users, especially those relying on assistive technologies.

3. Legacy codebases and technical debt

   Older websites or applications often suffer from technical debt - outdated code that was written before accessibility considerations were widely recognized. These legacy systems can be challenging to modify or update, especially when integrating accessibility features into a complex or rigid architecture.

Impact: legacy systems may lack proper semantic HTML or accessible forms, and making these systems compatible with assistive technologies can require significant redevelopment efforts.

4. Time constraints

   Developers are often under pressure to meet tight deadlines. In such environments, accessibility can be overlooked in favor of speedier delivery. Adding accessibility checks to the development cycle may be seen as an extra step that could delay the launch.

   Impact: quick fixes or incomplete accessibility features can be implemented, leading to subpar user experiences for those with disabilities.

5. Resistance to change

   There can be resistance within organizations, particularly from management or stakeholders who do not see the immediate return on investment (ROI) from implementing accessibility. Accessibility may be viewed as a costly, time-consuming effort without clear benefits.

   Impact: this resistance can lead to poor prioritization of accessibility initiatives or a lack of buy-in from key decision-makers.

1.7 Case Studies of Accessible Websites

The following case studies will illustrate the transformative impact of prioritizing accessibility on user experience, business growth, and legal compliance. By analyzing real-world examples, we can better understand the challenges faced by organizations and how they overcame these barriers to create more inclusive web experiences.

1.7.1 Apple

Apple [13] is often regarded as one of the leaders in making technology accessible to a wide range of users, including those with disabilities. The company has integrated accessibility features across its entire ecosystem, from websites to mobile apps to hardware, and its website reflects this commitment.

Key changes made:

- VoiceOver and screen reader integration. Apple's website is fully compatible with its VoiceOver screen reader, which reads aloud the content on the screen for blind users.
- Accessibility on all devices. Apple ensures that its website is compatible with various assistive technologies, including braille displays, switch control, and magnification tools.

- Semantic HTML and ARIA. Apple’s developers use semantic HTML tags and ARIA (Accessible Rich Internet Applications) roles to ensure that the website structure is correctly understood by assistive technologies.

Impact:

- Usability. Apple’s site is intuitive and usable for individuals with a range of disabilities, providing a seamless experience across its devices and platforms.
- Business growth. Apple’s commitment to accessibility contributes to a more diverse customer base, with people of all abilities able to use their products, thereby expanding their reach.
- Compliance. Apple's website complies with global accessibility standards, including WCAG and ADA, ensuring that it meets the requirements of both international and local regulations.

1.7.2 Ukrainian Government Portal

The Ukrainian Government Portal [14], a key platform for public services, has taken steps toward improving digital accessibility in recent years. Given the significance of this portal for citizens, including those with disabilities, the government has prioritized making the website accessible to a broader audience.

Key changes made:

- WCAG compliance. The website underwent a major overhaul to align with WCAG 2.1 guidelines. It now features accessible text sizes, color contrast adjustments, and options for enlarging or reducing content to support users with visual impairments.
- Keyboard navigation. The portal ensures that all sections are navigable via keyboard alone, helping users with motor impairments or those who rely on assistive technology.
- Screen reader support. The site was optimized for screen reader compatibility to make it easier for visually impaired users to access government services and information.

Impact:

- Usability. Citizens with disabilities now have improved access to essential government services, leading to more inclusive participation in public life.
- Business growth. As the Ukrainian government continues to digitize services, accessibility enhancements on the portal promote greater trust and engagement with the platform, benefiting the government’s reputation and accessibility goals.

- Compliance. The portal complies with Ukrainian laws, including the DSTU EN 301549 standard for accessibility, aligning with EU directives and improving transparency and accountability.

Chapter 2. Implementation of a Web App for Accessibility Analysis and Optimization

For my capstone project, I am developing a Web Accessibility Analysis and Optimization Web App aimed at addressing a significant gap in the tools currently available for accessibility evaluation. The problem I am solving is the lack of a comprehensive tool that can analyze all aspects of HTML code to fully detect web accessibility issues. Existing tools like Lighthouse and other automated accessibility checkers often only provide superficial analysis, missing many of the nuanced issues that are critical to meeting the Web Content Accessibility Guidelines (WCAG).

These existing tools primarily rely on automated tests, which, while useful, cannot cover the full breadth of WCAG success criteria. They fail to detect more complex issues, particularly those related to dynamic content, user interactions, and accessibility challenges that can only be identified through manual testing. As a result, websites may pass automated checks but still be inaccessible to certain users, leaving developers unaware of critical issues.

To solve this, my Web Accessibility Analysis and Optimization Web App will offer a more thorough and comprehensive solution. It will analyze HTML files to identify accessibility issues that go beyond the scope of automated tools.

This chapter focuses on the development and implementation of a web application designed to analyze HTML files for adherence to web accessibility standards. The application serves as a practical tool for identifying, visualizing, and resolving accessibility issues, promoting the creation of inclusive and user-friendly web content. Its primary goal is to empower developers by identifying accessibility issues in web content, providing actionable recommendations for resolving them, and automating parts of the optimization process. By integrating analysis and visualization capabilities into a single interface, the dashboard simplifies the process of making websites accessible to all users, including those with disabilities.

The dashboard operates in two main phases:

1. Analysis of web content and
2. Visualization of issues found

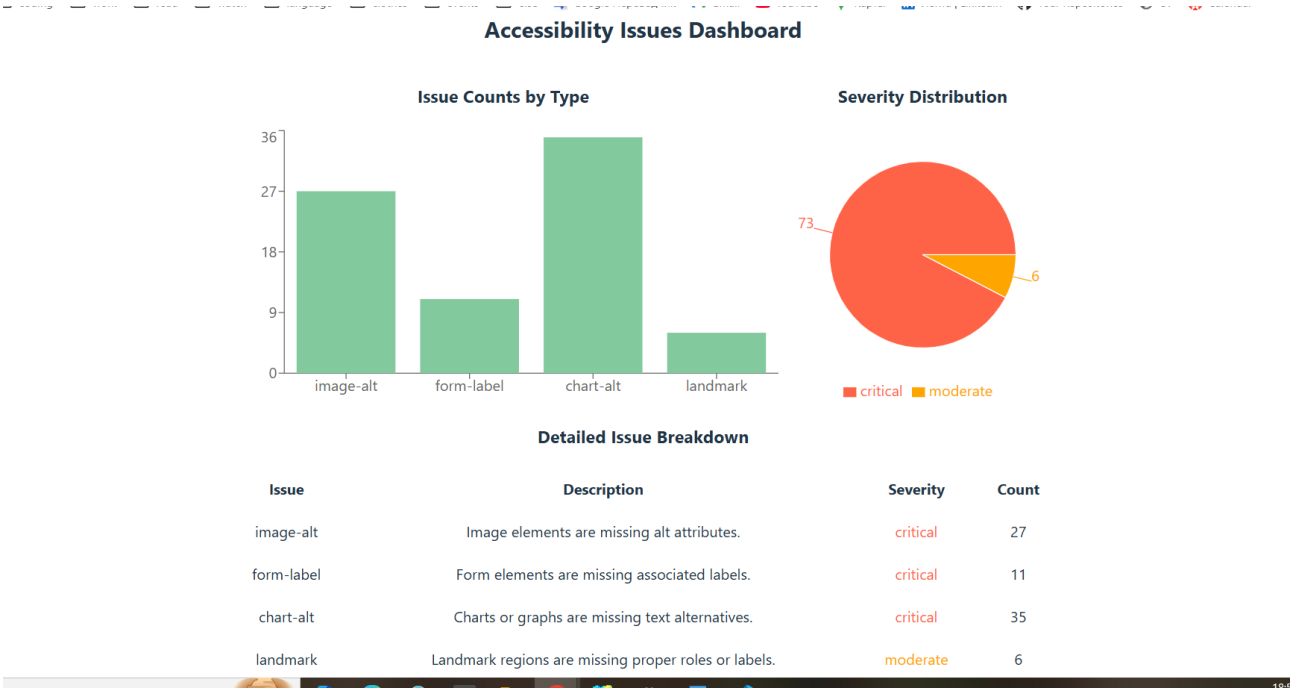
Users can upload an HTML file to initiate the process.

Accessibility Dashboard

Choose File No file chosen

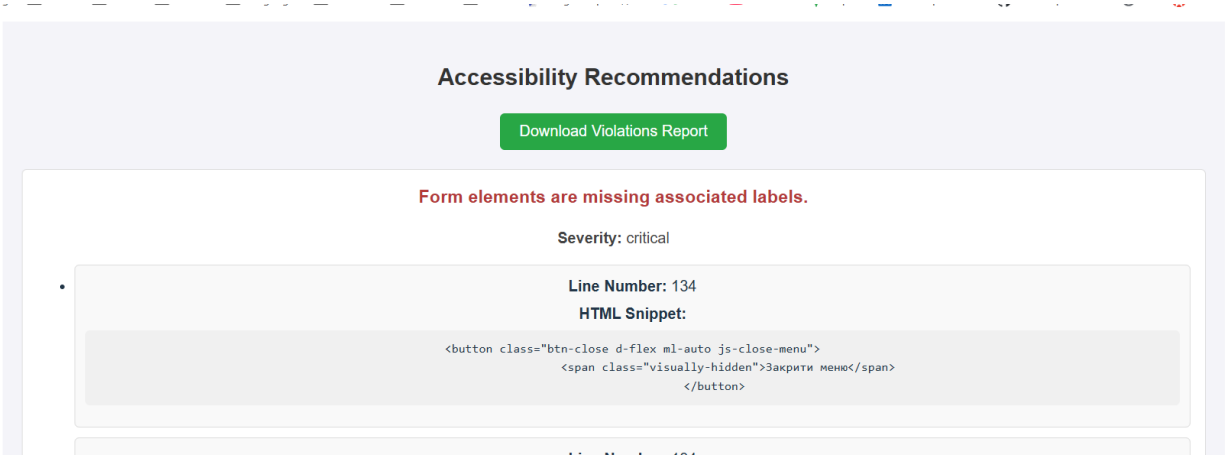
Once the file is uploaded, the dashboard performs a comprehensive analysis of the web content, checking for common accessibility issues based on established guidelines. It evaluates elements such as images, forms, links, text contrast, headings, media content, and more, ensuring that they adhere to accessibility standards.

Based on the analysis results, the dashboard identifies violations of accessibility standards and offers actionable suggestions for improving the content. The violations are categorized by severity, ranging from critical issues (e.g., missing alt attributes for images) to moderate ones (e.g., improper heading nesting), allowing users to prioritize fixes accordingly.

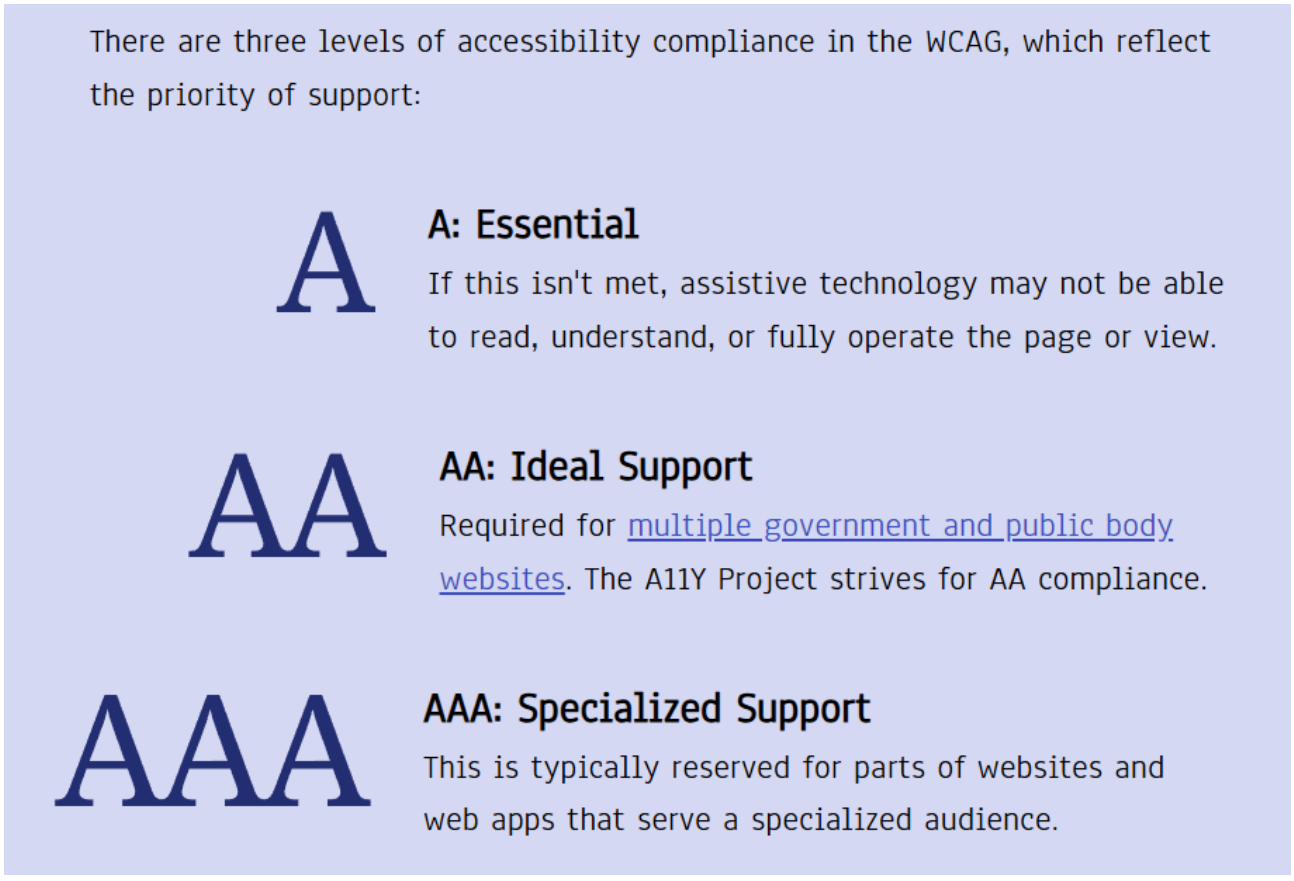


To make the process more user-friendly, the dashboard provides clear descriptions of each issue along with affected code snippets and the exact line numbers where the problem occurs. This helps users quickly understand the problem and where to make changes in the code.

Additionally, users can download a detailed report of all the violations found in the analysis. The report can be saved as a .txt file, summarizing the issues along with line numbers and descriptions, making it easier to track and address accessibility concerns across the site.



The Web Accessibility Analysis and Optimization Web App emphasizes achieving WCAG A and AA compliance. There are three levels of accessibility compliance as defined by the Web Content Accessibility Guidelines (WCAG). It explains the priority of support for each level (15):



The Web Accessibility Analysis and Optimization Web App emphasizes achieving WCAG AA compliance. This level is chosen as the primary focus due to its balance between accessibility requirements and practicality for most websites, particularly those serving government entities, public organizations, and businesses aiming to provide inclusive experiences.

Why WCAG AA Compliance?

1. Universal Applicability: Level AA compliance is often a legal or best-practice standard for public-facing websites. It ensures that content is accessible to a wide range of users, including those with moderate disabilities.
2. Enhanced User Experience: While Level A compliance focuses on essential accessibility, Level AA addresses more advanced barriers, such as color contrast, navigation, and adaptability, making websites usable and navigable for diverse audiences.
3. Feasibility: Level AA strikes a balance between accessibility and effort, as achieving Level AAA compliance may not be practical for most general-purpose websites due to its stringent requirements.

2.1 Key Functionalities

1. Input handling

The web app allows users to upload an HTML file. This ensures that developers can analyze static files.

2. Accessibility analysis

The app conducts a detailed analysis of the web content. It detects violations related to accessibility guidelines, such as missing alt attributes, improper ARIA roles, and inadequate color contrast.

3. Visualization of issues

Identified issues are presented in both tabular and graphical formats. The tabular view lists each issue along with its description, severity, and affected elements. The graphical representation (e.g., bar charts) provides an overview of the most common problems, helping users prioritize their efforts.

4. Recommendations

For each detected issue, the dashboard provides clear, actionable recommendations aligned with web accessibility standards such as WCAG 2.1. These recommendations guide developers in manually addressing the issues, offering practical examples where applicable.

2.2 System Modules

The accessibility dashboard is structured around modular components using React, each responsible for a specific aspect of the process. The architecture comprises:

- Input module

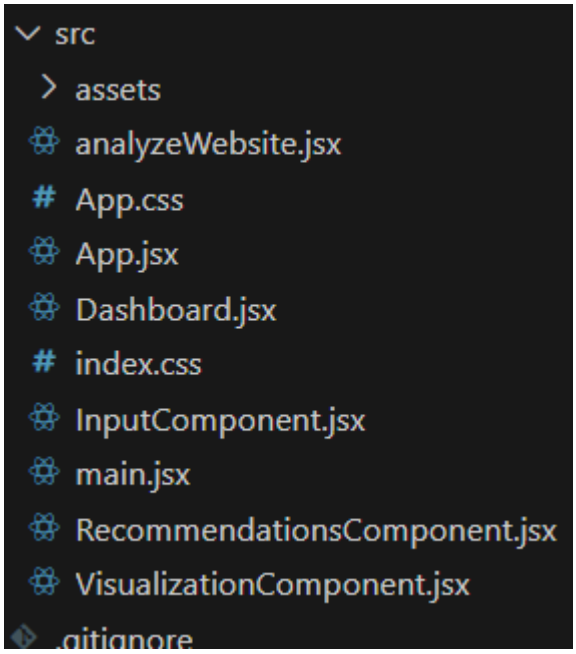
Handles HTML file uploads.

- Analysis module

Scans the input file for accessibility violations, producing a detailed report.

- Visualization module

Processes analysis results and generates visual and tabular representations for easier interpretation using Recharts.js.



2.3 Workflow Description

The workflow of the accessibility dashboard is designed to ensure a seamless user experience:

1. Input phase:
    - The user uploads an HTML file via the dashboard interface.
    - The input is validated to ensure it is in the correct format for analysis.

2. Analysis phase:
    - The code analyzes the input for accessibility violations.
    - A detailed report of the issues is generated, including the type, severity, and affected elements.
3. Visualization phase:
    - The results are presented in a table and a bar chart for clear and concise communication.
    - Users can interact with the visualization to explore specific issues in depth.
4. Recommendation phase:
    - For each detected issue, the dashboard displays textual recommendations, guiding users on how to fix the problem manually if desired.

2.4 Practical Relevance

The Mini Accessibility Dashboard addresses a critical need for efficient and accessible web development practices. By combining analysis, visualization, and automated optimization, it simplifies the often complex process of meeting accessibility standards. The tool is particularly valuable for:

- Developers seeking to improve the accessibility of their websites without extensive manual effort.
- Organizations aiming to ensure compliance with accessibility regulations.
- Educators and students exploring the practical applications of web accessibility in real-world scenarios.

This dashboard demonstrates a practical implementation of front-end development principles, emphasizing the importance of accessibility as an integral aspect of modern web design.

The functionality focuses on identifying common accessibility issues and grouping them into violation categories for better organization and usability. Below is a detailed breakdown of the code, its components, and potential use cases.

2.5 Code Functionality

1. Parsing the HTML document

The code uses the DOMParser API to parse a given HTML string into a Document object, enabling detailed analysis of the DOM structure.

2. Accessibility checks

The script performs key accessibility checks:

- Image elements missing alt attributes

  The script scans all <img> tags to ensure that each image has an associated alt (alternative text) attribute. This is crucial for users who rely on screen readers to understand the content of images. If an image lacks a descriptive alt attribute or has an empty value, it is flagged as a violation.

  Why it matters: screen readers use the alt attribute to describe images to users with visual impairments. Missing or poorly written alt text can leave users without essential context.

- Form elements lacking labels

  The script checks for <input>, <textarea>, <select>, and <button> elements to verify that they are properly associated with a label, either through the <label> element or ARIA attributes such as aria-label or aria-labelledby. This ensures that users with disabilities can correctly identify and interact with form controls.

  Why it matters: accessible form labels are critical for users who rely on assistive technologies to understand the function of form elements. Without labels, users may struggle to interact with the forms.

- Low text contrast

  The script analyzes the color contrast between text and its background to ensure it meets WCAG (Web Content Accessibility Guidelines) standards. Text elements are flagged if the contrast ratio is below the recommended minimum of 4.5:1 for normal text and 3:1 for large text.

  Why it matters: users with low vision or color blindness may have difficulty reading text with insufficient contrast. Proper contrast ensures readability and accessibility for a broader audience.

- Missing text alternatives for charts and graphs

  The script scans for SVGs, <canvas>, and elements with the role img that might represent charts or graphs. These elements are flagged if they do not have a text alternative via the aria-label or aria-labelledby attributes.

Why it matters: charts and graphs often contain critical data, and without text descriptions, users with visual impairments cannot access this information. Text alternatives (captions or descriptions) provide context for all users.

- Data tables missing header cells

The script checks if <table> elements are missing <th> header cells, which are essential for providing structure to data tables. If a table does not contain header cells, it is flagged for review.

Why it matters: screen readers rely on header cells (<th>) to announce the meaning of table columns and rows. Without headers, data tables become difficult to navigate and interpret for users with disabilities.

- Fieldsets without legends

The script checks for <fieldset> elements and verifies if they are missing a <legend>. The <legend> provides a description of the fieldset, which is essential for grouping form elements logically.

Why it matters: without a <legend>, users may not understand the context of grouped form elements, making it harder to complete forms accurately, especially for users with cognitive disabilities.

- Incorrect landmark usage

The script examines the usage of landmark roles such as header, footer, main, nav, and aside. It checks if these elements are correctly marked up with the appropriate role or ARIA attributes. Incorrect or missing landmark roles are flagged for review.

Why it matters: landmarks help users with assistive technologies quickly navigate to key sections of a webpage. Incorrect or missing landmark roles can make it difficult for users to find and access content, particularly for those using screen readers or keyboard navigation.

- Improper heading nesting

The script analyzes the structure of headings (<h1>, <h2>, <h3>, etc.) to ensure they are properly nested. Improperly nested headings (e.g., skipping heading levels) are flagged as violations.

Why it matters: headings provide a hierarchical structure to the content of a webpage. Properly nested headings help users, especially those using screen readers, to understand the organization and flow of content.

- Links without discernible text

The script scans all <a> (link) elements and checks if they contain discernible text or an aria-label. Links with no text content, empty hrefs, or inaccessible link text are flagged.

Why it matters: links need to have descriptive text so users can understand their purpose. Empty or unclear links are problematic for users relying on screen readers, as they won't know where the link leads or its function.

- Focusable elements without accessible names

The script checks if elements that are focusable (e.g., <a>, <button>, <input>, etc.) have accessible names. An element is considered focusable if it can receive keyboard input or can be navigated to using the Tab key. If a focusable element lacks an accessible name (through text or ARIA attributes), it is flagged.

Why it matters: focusable elements, such as buttons and links, must have clear labels so users can understand their purpose. Missing accessible names make it difficult for users with visual impairments to interact with the page effectively.

- Document title missing or empty

The script checks if the document has a valid <title> element. The <title> provides a brief description of the webpage and is important for both usability and search engine optimization.

Why it matters: a missing or empty title can confuse users, especially those using screen readers, as they won't have a clear understanding of the page's content.

- Missing document language attribute

The script verifies if the <html> element contains the lang attribute, which specifies the language of the document. This attribute is crucial for users with screen readers or other assistive technologies.

Why it matters: the lang attribute helps assistive technologies properly interpret and read the content aloud in the correct language, ensuring a better user experience for people with disabilities.

- Tabindex values greater than 0

The script checks for the use of tabindex values greater than 0. While tabindex allows custom tab order, values greater than 0 can break the expected flow and cause accessibility issues.

Why it matters: using tabindex values greater than 0 can disrupt the natural tab order of a page, which is problematic for users navigating with a keyboard. It can lead to confusion and poor navigation.

- Duplicate ARIA roles

The script checks for the use of duplicate ARIA roles, particularly those that should be unique across a webpage, such as banner, main, and navigation. Multiple elements with the same ARIA role are flagged for review.

Why it matters: duplicate ARIA roles can confuse assistive technologies, making it difficult for users to identify and interact with key areas of the page. ARIA roles should be unique to provide clarity in navigation.

- Iframes without titles

The script checks all <iframe> elements to ensure they contain a title attribute. The title attribute provides a description of the embedded content, which is essential for accessibility.

Why it matters: iframes are often used to embed content from external sources. Without a title, users may not understand the purpose of the iframe, which could lead to confusion or missed content.

- Empty links or buttons

The script checks if links (<a>) and buttons (<button>) are empty or lack descriptive text or ARIA labels. These elements are flagged as violations if they do not provide accessible information.

Why it matters: empty links and buttons are completely unusable for users with assistive technologies, as they provide no context or interactivity.

- Duplicate IDs

The script scans the document for elements with duplicate id attributes. Each id in an HTML document must be unique to prevent confusion and ensure proper functionality of JavaScript and accessibility tools.

Why it matters: duplicate IDs can cause issues with navigation, styling, and accessibility, as they may confuse assistive technologies that rely on unique identifiers for elements.

- Audio/Video missing captions or transcripts

The script checks <audio> and <video> elements to ensure they have either captions (via <track> elements) or accessible transcripts (via aria-describedby). These are flagged if missing.

Why it matters: captions and transcripts provide essential information for users with hearing impairments or those unable to play audio or video content.

- Unlabeled landmarks

The script checks if landmark regions (such as header, footer, nav, main, etc.) have appropriate ARIA labels or roles. Unlabeled landmarks are flagged for review.

Why it matters: landmarks are essential for navigating a page. Labeled landmarks make it easier for users with assistive technologies to identify and skip to important content sections.

- Viewport meta tag restricting zoom

The script checks for the presence of the <meta name="viewport"> tag and verifies that it does not restrict user zoom or resizing by checking properties like maximum-scale=1 or user-scalable=no.

Why it matters: restricting zoom or text resizing can make a webpage unusable for users with visual impairments or those who need larger text.

- CSS or JavaScript restrictions on Zoom/Resizing

The script analyzes the webpage for any CSS or JavaScript settings that restrict zooming or text resizing, which are flagged for review.

Why it matters: users should be able to adjust text size and zoom to meet their needs. Restricting these features can make it difficult for users with vision impairments to access the content effectively.

3. Violation grouping

Detected issues are grouped by type (e.g., image-alt, form-label, contrast). Each group is represented as an object with the following structure:

- id: A unique identifier for the violation type (e.g., image-alt).
- violation-group: A description of the violation category.
- nodes: An array of affected elements, including:
    - target: The affected DOM element(s).

- html: The HTML string representation of the element.

4. Output Structure

The final result is an array of grouped violations, where each object represents a specific violation category. This structure facilitates easy understanding and debugging.

2.6 Key Advantages

1. No external dependencies

Unlike solutions like axe-core, this script performs checks manually, allowing for fine-grained control over the analysis process.

2. Customizability

The modular structure makes it easy to add more accessibility checks as needed.

3. Organized output

Grouping violations by type improves clarity, making it easier to focus on specific categories of issues.

4. Focus on core accessibility needs

The code addresses fundamental WCAG principles, such as:

- Perceivable (e.g., alt attributes for images).
- Operable (e.g., labels for form elements).
- Understandable (e.g., sufficient contrast for readability).

2.7 Planned Enhancements and Extensions

The Web Accessibility Analysis and Optimization Web App, developed by me as a single front-end developer, has laid the foundation for a tool that addresses critical gaps in accessibility optimization. While the current version offers robust features for analyzing HTML files and providing recommendations, there is immense potential for future enhancements. These improvements could significantly broaden the app's scope, usability, and impact, particularly with the collaboration of a multidisciplinary team that includes designers and backend developers. Below is a detailed exploration of the planned enhancements and extensions:

1. Expand accessibility checks

The app currently focuses on key WCAG A and AA compliance checks. However, the following additional checks can further improve its comprehensiveness:

- Keyboard navigation validation.

  Ensure that all interactive elements (e.g., buttons, links, forms) are accessible via keyboard-only navigation, catering to users who rely on assistive technologies.

- Heading structure analysis.

  Validate that headings follow a logical hierarchy (e.g., H1 > H2 > H3), which is crucial for both accessibility and search engine optimization (SEO).

- Dynamic content handling.

  Add checks for accessibility in dynamic content (e.g., modals, sliders, pop-ups), which often pose challenges for users relying on assistive technologies.

2. Internationalization

To ensure the app meets the needs of global users, the following internationalization features can be introduced:

- Language attribute validation.

  Check for the correct usage of lang attributes in HTML elements to ensure content is properly identified for screen readers.

- Localized recommendations.

  Provide accessibility recommendations tailored to specific languages and regional WCAG interpretations, ensuring cultural and linguistic relevance.

- RTL (Right-to-Left) language support.

  Extend checks to ensure proper handling of layouts for languages such as Arabic and Hebrew, which use right-to-left scripts.

3. Integration with CI/CD pipelines

Automating accessibility checks as part of the software development lifecycle can ensure continuous compliance:

- Pipeline automation.

  Develop integrations with popular CI/CD tools (e.g., Jenkins, GitHub Actions, GitLab CI) to run accessibility tests during deployment processes. This would ensure that every production release meets WCAG standards.

- Reporting features.

Generate automated reports summarizing accessibility compliance for development teams, enabling them to track progress and address recurring issues.

4. Collaborative functionality with a multidisciplinary team

As a solo developer, my focus has been on front-end development. However, with a larger team, the app's scope can be expanded to include the following:

- Designer contributions:
    - User-centered dashboards

      Collaborate with UX/UI designers to enhance the app's interface, making it more intuitive for users of all technical levels.

    - Color accessibility simulations

      Implement features that simulate how users with color blindness or other visual impairments perceive the website.

- Backend developer contributions:
    - Data storage and history

      Enable users to save analysis results and track the evolution of their projects over time.

    - Multi-file analysis

      Support batch uploads to analyze multiple HTML files simultaneously.

    - User authentication

      Add user accounts with personalized dashboards for managing projects and tracking accessibility scores.

5. Artificial intelligence integration

Integrating AI features can enhance the app's analytical capabilities and user support:

- Automated fix suggestions

Use machine learning to provide context-aware suggestions for resolving accessibility issues, such as recommending appropriate ARIA attributes or color combinations.

- Accessibility chatbot

Include an AI-powered chatbot that answers questions about WCAG compliance and guides users in resolving complex accessibility problems.

- Image recognition for alt text

Implement AI algorithms to analyze images and suggest appropriate alternative text descriptions, reducing manual effort.

6. Expanded ompliance levels

While the app currently focuses on WCAG AA compliance, future versions can include:

- AAA compliance

Extend checks to cover the highest level of accessibility, catering to specialized audiences with stricter requirements.

- Custom compliance levels

Allow users to define their own accessibility standards, enabling flexibility for unique project needs.

Chapter 3. Conclusions

The development and implementation of the web accessibility analysis and optimization web app represents a forward-thinking approach to addressing the critical challenges of web accessibility. This project not only bridges gaps in existing tools but also introduces innovative functionalities to enhance web development practices. Through its comprehensive and user-friendly features, the app serves as a significant tool in promoting inclusivity and ensuring compliance with accessibility standards.

3.1 Summary of Contributions

- Addressing accessibility challenges:

Existing tools, such as Lighthouse, often fall short of identifying nuanced accessibility issues. This project bridges the gap by providing a comprehensive analysis that extends beyond automated checks to cover user interactions and dynamic content.

The app's ability to identify issues such as missing alternative text, improper ARIA roles, and color contrast violations equips developers with actionable insights to create more inclusive web platforms.

- Emphasis on WCAG AA compliance:

The project prioritizes WCAG AA compliance, striking a balance between rigorous accessibility standards and practical feasibility for public-facing websites. This focus ensures that the app's recommendations are both impactful and achievable for most organizations.

By addressing core principles of perceivability, operability, and understandability, the app empowers developers to meet accessibility needs effectively.

- Innovative functionalities:

The dashboard integrates features such as tabular and graphical visualizations of accessibility issues, enabling users to interpret results efficiently.

Actionable recommendations tailored to specific violations ensure that developers are equipped with clear guidance for optimization.

A modular architecture using React provides scalability and a solid foundation for future enhancements.

- Downloadable reports:

To further enhance the user experience, the app allows users to download a detailed file of the identified violations. This feature enables developers and organizations to keep a record of the issues

found and track their resolution over time, making the process of addressing accessibility issues more efficient and structured.

3.2 Practical Implications

The app's design and functionalities are directly applicable to real-world scenarios, benefiting developers, organizations, and educational institutions. It can:

- Streamline accessibility evaluation:

The app automates much of the manual testing required to identify accessibility issues, saving developers time and effort while improving the accuracy of evaluations.

- Ensure compliance with legal and regulatory standards:

With the increasing emphasis on digital accessibility laws and guidelines, the app helps organizations avoid non-compliance risks by ensuring that their websites meet accessibility requirements.

- Enhance user experience for individuals with disabilities:

By identifying and suggesting improvements for accessibility issues, the app promotes inclusivity, making digital platforms more accessible to users with various disabilities.

3.3 Future Directions

Building upon the project's foundational work, several enhancements can extend its scope and impact:

- Advanced accessibility checks

Incorporate validation for keyboard navigation, heading structures, and dynamic content to address broader accessibility challenges.

- AI integration

Use machine learning for automated suggestions and alternative text generation, further reducing manual effort.

- Global reac

Support internationalization features, including RTL layout validation and localized recommendations, to cater to diverse linguistic and cultural needs.

- CI/CD integration

Embed accessibility checks within continuous integration pipelines to ensure ongoing compliance during development cycles.

3.4 Final Conclusion

The Web Accessibility Analysis and Optimization Web App serves as a testament to the importance of combining technical innovation with an unwavering commitment to inclusivity. By addressing critical gaps in existing solutions and offering actionable, user-friendly features, this project underscores the vital role of accessibility in modern web development. It highlights how a systematic approach to identifying and resolving accessibility barriers can lead to more inclusive and equitable digital experiences for all users. Beyond its immediate functionality, the app's design exemplifies the potential of integrating accessibility into the core principles of development, ensuring that digital platforms are built to empower users regardless of their abilities.

The future potential of this tool is vast, with opportunities to expand its capabilities through the integration of advanced features, collaborative input from multidisciplinary teams, and broader global applicability. As technology evolves, the app can serve as a dynamic resource that adapts to emerging accessibility challenges, reaffirming the commitment of developers to foster inclusivity and compliance. Ultimately, this project lays a promising foundation for advancing the state of web accessibility optimization, making a meaningful contribution to the creation of a more inclusive digital world.

References

1. World Health Organization (WHO) Disability Statistics. URL: https://www.who.int/news-room/fact-sheets/detail/disability-and-health
2. a11yMyths: Common Misconceptions About Web Accessibility. URL: https://a11ymyths.com/
3. Web Accessibility Perspectives Videos: Explore the Impact and Benefits for Everyone. URL: https://www.w3.org/WAI/perspective-videos/
4. Americans with Disabilities Act (ADA) U.S. Department of Justice. URL: https://www.ada.gov/
5. European Accessibility Act (EAA) European Union Official Documents. URL: https://ec.europa.eu/social/main.jsp?catId=1202
6. DSTU EN 301549:2019 (Ukraine) National Standards Documentation (available in Ukrainian). URL: https://uas.org.ua/standards/dstu-en-301549-2019/
7. Search Engine Journal: Accessibility and SEO. URL: https://www.searchenginejournal.com/intersection-of-seo-and-accessibility-optimizing-for-all-users/510254/
8. The Click-Away Pound Report 2019: https://www.clickawaypound.com/
9. Web Content Accessibility Guidelines (WCAG) - W3C Web Accessibility Initiative (WAI). URL: https://www.w3.org/WAI/standards-guidelines/wcag/
10. Accessible Rich Internet Applications (WAI-ARIA): W3C. URL: https://www.w3.org/TR/wai-aria/
11. WebAIM: Screen Reader User Survey. URL: https://webaim.org/projects/screenreadersurvey10/
12. WebAIM: Assistive Technology Experiment: Dragon NaturallySpeaking. URL: https://webaim.org/blog/at-experiment-dragon/
13. Apple website: https://www.apple.com/
14. The Ukrainian Government Portal: www.kmu.gov.ua
15. The Web Content Accessibility Guidelines (WCAG) checklist. URL: https://www.a11yproject.com/checklist/
16. GitHub repository with code: https://github.com/anastasiia-starchenko/accessibility-dashboard