Electronic Integrated Computer Algorithm Journal
Vol. 4, No. 1, October 2026, pp 33~40

ISSN: 3031-0350, DOI: 10.62123/enigma.v4i1.230

Multi-Browser Compatibility Evaluation of a CDN-Based Accessibility Plugin Following the WCAG 2.1 Standard

Jonathan Arya Wibowo[1*], Muharman Lubis[1], Rio Guntur Utomo[2], Aryo Pradipta Ramadhan[2], Amanda Bunga Gracia[3], Septiana Yustika Widyaningrum[4]

[1] *Program Studi S2 Sistem Infromasi, Fakultas Rekayasa Industri, Universitas Telkom, Bandung,* Indonesia.
[2] *Program Studi S1 Teknologi Informasi, Fakultas Informatika, Universitas Telkom, Bandung*, Indonesia.
[3] *Program Studi S1 Ilmu Konunikasi, Fakultas Komunikasi dan ilmu Sosial, Universitas Telkom, Bandung,* Indonesia.
[4] *Program Studi S1 Digital Public Relation, Fakultas Komunikasi dan ilmu Sosial, Universitas Telkom, Bandung,* Indonesia.

DOI : 10.62123/enigma.v4i1.230

Received : August 19, 2026
Revised : September 06, 2026
Accepted : September 29, 2026

Keywords:
Accessibility plugin, Cross-browser Compatibility, Text-to-speech, Voice command, WCAG 2.1

ABSTRACT

Website accessibility remains a barrier for users with visual and motor impairments, while modern sites keep adding interactive visual elements. This study develops a modular JavaScript accessibility plugin distributed through a Content Delivery Network, so that it can be attached to a site without altering the existing page structure. The plugin follows WCAG 2.1 and provides text-to-speech, voice command, display and text settings, magnifier, and keyboard navigation through a single control panel. A company profile website was used as the case study. Evaluation combined manual inspection using DevTools and a WCAG checklist with automated measurement using Lighthouse and DevTools Performance, followed by compatibility testing on Chrome, Firefox, Edge, and Brave. All features behaved as designed without JavaScript errors, and the performance score remained stable after activation, although features that change text spacing and page scale added rendering work. Display, text, and keyboard features ran consistently across the four browsers, whereas voice command failed on Firefox and Brave and behaved unstably on Edge because of differing speech recognition support. The results suggest that browser API dependency is the main limitation of overlay-based accessibility solutions. Findings are bounded by a single site, single run measurements, two rendering engines, and the absence of usability testing with users who have impairments.

1. INTRODUCTION

Websites no longer serve merely as one-way information channels, since most of them have grown into complex spaces for digital interaction. A web page should ideally be understood by anyone without demanding excessive cognitive effort [1]. Current development practice, however, tends to favour interactive visual elements and multimedia in order to deliver messages more effectively [2]. Such growing visual complexity appears to open a new type of barrier, particularly for users with visual or motor limitations.

Web accessibility addresses this concern by treating equal access to information as a design goal rather than an optional addition. The Web Content Accessibility Guidelines (WCAG) 2.1 issued by W3C summarise four principles known as POUR, namely perceivable, operable, understandable, and robust [5]. Semantic structure expressed through ARIA attributes further supports how assistive technology interprets a page [9]. Despite the availability of these guidelines, evaluations of educational and academic websites still report shortcomings in keyboard navigation, contrast ratio, and text alternatives [11][12]. Singh and colleagues, auditing thirteen education board portals with six W3C endorsed tools, found most of them satisfying less than half of the criteria examined [15], a pattern that seems remarkably persistent across regions and site categories.

Two broad responses to this situation can be distinguished in recent literature. One line of work aims at remediation during design, increasingly with automated or model driven support [17], while another intervenes at runtime by adjusting what the browser already renders, an approach Dash frames as real time accessibility enhancement [16]. The plugin proposed here belongs to the second family. A practical response frequently adopted by developers is the accessibility overlay or accessibility toolbar, which adds a supplementary interface layer to an existing site [8]. This approach shortens development time because the underlying structure remains untouched, yet it may increase DOM complexity and browser resource consumption [8]. A further concern arises from the execution environment, since differences in rendering engines and JavaScript implementations can produce inconsistent appearance and behaviour across browsers [3][4].

Accessibility evaluation can hardly be delegated to automated audit tools alone. Lighthouse provides a fast measurement of performance and structural conformance [6], although its coverage is limited to static aspects so that interactive behaviour remains

*Corresponding Author Email: jonathanwibowo@student.telkomuniversity.ac.id

largely unrepresented [13]. Comparative work on ten evaluation tools reaches a similar verdict, reporting substantial disagreement between tools and incomplete criterion coverage in each of them [23]. Bibliometric mapping of the field points in the same direction, since automated auditing dominates the published record while user centred evidence remains comparatively scarce [25]. Combining manual inspection through DevTools [7] with automated auditing therefore seems more adequate for features such as text-to-speech, voice command, and magnifier.

Based on these conditions, this study develops an accessibility plugin distributed through a Content Delivery Network (CDN) and grounded in WCAG 2.1, then evaluates its compatibility across four modern browsers. The IDEA Hub company profile website is used as the implementation case study. The contribution lies in bringing together three concerns that are usually discussed separately, namely the implementation of reusable accessibility features, the measurement of their impact on page performance, and the examination of behavioural consistency across rendering engines.

2. LITERATURE REVIEW

2.1 Web Accessibility and WCAG 2.1

Web accessibility refers to the effort of ensuring that a website can be accessed by all users, including those with visual, motor, hearing, or cognitive limitations [5]. The success criteria adopted in this study cover text alternatives for non-text content (1.1.1), minimum contrast ratio (1.4.3), the ability to enlarge text and pages (1.4.4 and 1.4.10), text spacing adjustment (1.4.12), control over animation (2.2.2), and full operation through a keyboard (2.1.1). Keyboard operability deserves particular attention: Sarioglu and colleagues demonstrate, in the context of modelling tools, that a keyboard-only interaction path can be engineered deliberately rather than inherited by accident [18]. Magnification carries its own complications, since enlarging a page changes how text reflows and how long words are split across lines, with measurable consequences for reading [19]. Voice based interaction, meanwhile, is made possible by the Web Speech API, which provides both speech synthesis and speech recognition modules [10].

2.2 Accessibility Overlay and Multi-Browser Compatibility

Studies on accessibility toolbars indicate that features such as contrast mode, text resizing, and read aloud may improve readability and usability [8]. The feature scope discussed in those studies is generally limited and rarely includes voice command or magnifier. On the compatibility side, differences among rendering engines such as Blink, Gecko, and WebKit have been reported to cause inconsistent display and script execution [3], while a systematic review underlines cross-browser testing as an integral part of development [4]. More recent work classifies layout failures specific to responsive design and shows that such failures can be predicted from feature sets extracted per browser [20], which suggests that visual consistency and functional consistency are worth treating as separate questions. Evidence from the user side complements these technical findings: studies of university websites with visually impaired students [22] and of job application portals with screen reader users [24] repeatedly show barriers that no automated audit had flagged. The position of this study relative to earlier work is summarized in Table 1.

Table 1. Position of this study relative to earlier work

Study	Focus	Method	Browsers
Ismail and Kurniawan [11]	WCAG evaluation of education sites	Automated and manual	Not covered
Rahman and Hossain [12]	Academic website accessibility	WCAG evaluation	Not covered
Choudhary et al. [3]	Cross-browser testing	Browser testing	Covered
Santos and Mello [4]	Browser testing	Systematic review	Covered
Rajapakse and Brereton [13]	Limits of automated tools	Tool comparison	Not covered
Watanabe et al. [20]	Responsive layout failures	Classification models	Covered
Dash [16]	Runtime adaptation	Prototype and evaluation	Not covered
This study	CDN plugin based on WCAG 2.1	Manual, Lighthouse, DevTools	Four browsers

3. RESEARH METHODS

This study applies an applied research approach combined with evaluative testing, so that the outcome consists of a directly usable artefact together with evaluation data on that artefact [14]. The research workflow comprises seven stages, namely identification of accessibility problems in the case study, analysis of the relevant WCAG 2.1 criteria, plugin development, integration through a CDN, feature implementation, system testing, and evaluation of results.

3.1 Plugin Design and Architecture

The plugin is designed in a modular manner so that each feature can operate independently while remaining controlled through a single panel. The file accessibility.js acts as the main controller that handles initialisation, event management, and the mapping of user interaction to the relevant module. The module tampilan.bundle.js manages visual features such as high contrast, monochrome, hide images, big cursor, and pause animation, whereas teks.bundle.js manages text and audio-based features such as

text-to-speech, voice command, zoom, and magnifier. All visual manipulation is carried out through aksesibilitas.css as an accessibility layer, leaving the main page structure unchanged. The plugin architecture is presented in Figure 1.

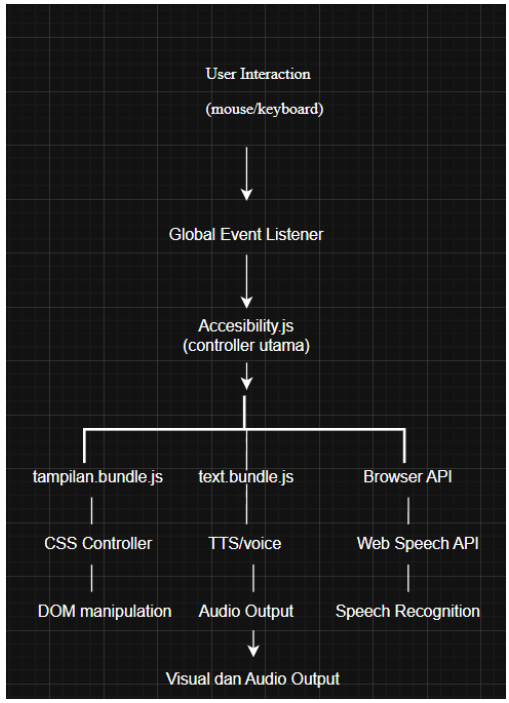


Figure 1. Architecture of the accessibility plugin

Integration is performed by adding a single external script before the closing body tag, which loads accessibility.js from the jsDelivr service. This mechanism makes the plugin reusable, because it can be attached to other pages without structural adjustment. The mapping of each feature to the WCAG 2.1 criteria is presented in Table 2.

Table 2. Mapping of features to WCAG 2.1 criteria

Feature	Function	Criterion
Hide images	Hides decorative images	1.1.1
High contrast	Improves text readability	1.4.3
Monochrome	Reduces reliance on colour	1.4.1
Big cursor	Improves pointer visibility	Supporting
Pause animation	Reduces moving elements	2.2.2
Text spacing	Adjusts line and word spacing	1.4.12
Font family	Provides a dyslexia friendly font	1.4.12
Text alignment	Adjusts paragraph alignment	1.4.8
Zoom	Enlarges text and page	1.4.4
Magnifier	Enlarges a selected area	1.4.10
Text-to-speech	Reads page content aloud	Supporting
Voice command	Enables voice based navigation	2.1.1
Keyboard navigation	Operation without a mouse	2.1.1

Two entries in Table 2 require justification. Text-to-speech and cursor enlargement are not addressed by any specific success criterion of WCAG 2.1, since the guidelines describe properties that content must satisfy rather than assistive functions that a page may offer. Attaching them to criteria 1.1.1 and 1.4.4 would be difficult to defend, because reading aloud does not by itself supply a text alternative and cursor size is unrelated to text resizing. Both are therefore labelled as supporting features, meaning that they serve the Perceivable principle without claiming conformance with a numbered criterion. The remaining entries map directly, and the same criterion numbers are reused in the conformance results so that the two tables can be read together.

3.2 Testing Scenario

Testing was carried out on a device with an AMD Ryzen 5 5600H processor, 16 GB of RAM, Windows 11, and a screen resolution of 1920 x 1080 pixels. Cache was disabled for every measurement session and device emulation was not activated. Four types of testing were performed in sequence. Manual testing used Chrome DevTools to observe DOM changes, applied CSS classes, and JavaScript errors [7]. Conformance testing matched every feature against the WCAG 2.1 checklist, complemented by keyboard navigation tracing and focus indicator inspection [5]. Performance measurement used Lighthouse in desktop mode before and after the plugin was activated [6], accompanied by main thread recording through DevTools Performance. Each Lighthouse audit was executed after a full page reload with the cache cleared, and each scenario was recorded once, so the values reported below are single measurements rather than averages of repeated runs. The recording window was not fixed at a single value because the features differ in interaction length. Toggle based features such as contrast or hide images were recorded for about five seconds, which is sufficient to capture the class change and the subsequent repaint, whereas continuous features such as the magnifier and text-to-speech were recorded for up to twelve seconds so that cursor tracking and speech playback were fully contained in the trace. This variation, together with the single run design, limits the reproducibility of the exact figures and is revisited in Subsection 4.5. The final stage assessed compatibility on Google Chrome, Mozilla Firefox, Microsoft Edge, and Brave by checking rendering consistency, script execution, keyboard navigation, and Web Speech API support. All four browsers were run on the same Windows machine, so Blink and Gecko are represented while WebKit is not covered.

4. RESULT AND DISCUSSIONS

4.1 Plugin Implementation and Accessibility Panel

The plugin was successfully integrated into the case study website without altering the main page structure. Once the script is loaded, the system performs initialization automatically, loads the CSS file and the JavaScript modules, and then injects the accessibility panel together with a skip link into the DOM. The panel appears as a floating panel that can be opened either by mouse click or by keyboard, as shown in Figure 2.

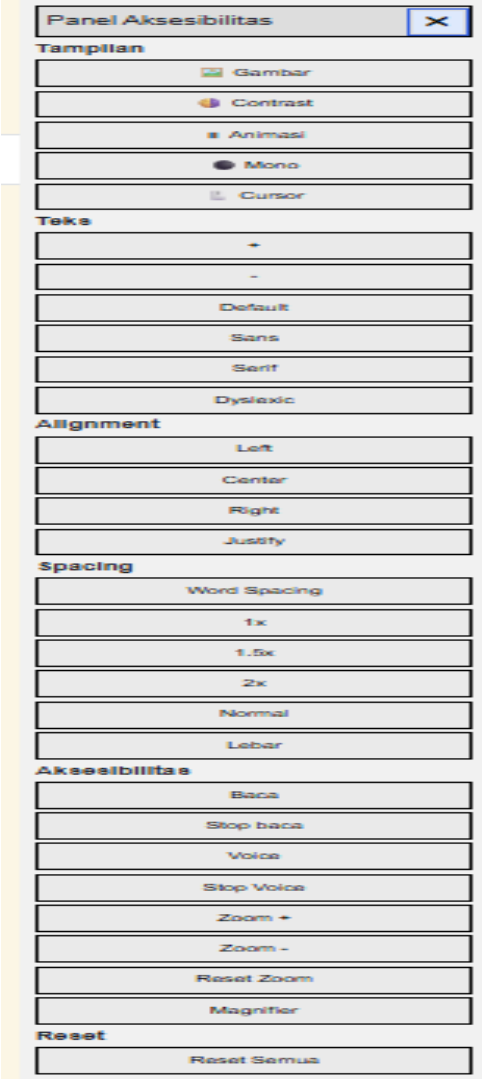


Figure 2. Accessibility panel produced by the implementation

The panel groups the features into display, text, alignment, spacing, accessibility, and reset categories. Keyboard navigation is supported through the Esc key to close the panel, Enter or space to activate a button, and the arrow keys to move between elements. The focus indicator is rendered as a coloured outline on the active element, and the initial focus is placed on the accessibility tab once the page finishes loading. Text-to-speech was implemented using the speech synthesis module, while voice command relies on webkitSpeechRecognition with commands for reading, stopping, searching, and scrolling up or down [10].

4.2 Functional Testing and WCAG Conformance

Observation through DevTools showed that activating each feature triggered the expected changes in CSS classes and computed style, without raising JavaScript errors in the console. The results of matching the features against the WCAG 2.1 checklist are summarised in Table 3.

Table 3. Results of testing against the WCAG 2.1 checklist

Feature tested	Criterion	Observed result
Hide images	1.1.1	Image elements hidden
High contrast	1.4.3	Text contrast increased
Monochrome	1.4.1	Grayscale applied
Big cursor	Supporting	Pointer enlarged
Pause animation	2.2.2	Animation stopped
Spacing control	1.4.12	Line and word spacing changed
Font family	1.4.12	Font replaced as selected
Text alignment	1.4.8	Alignment changed as selected
Zoom	1.4.4	Layout remained stable
Magnifier	1.4.10	Selected area enlarged
Text-to-speech	Supporting	Content read and stoppable
Voice command	2.1.1	Voice commands responded to
Keyboard navigation	2.1.1	All buttons reachable

Table 3 uses the same criterion numbers as Table 2, so each feature can be traced from design intent to observed behaviour. Every feature behaved as specified. Two qualifications apply to this result. First, the conformance demonstrated here is functional, meaning that the mechanism required by a criterion is present and operable, which is weaker than a full conformance claim for the site as a whole. Second, the two supporting features were judged on whether they worked, not on whether they satisfy a criterion, since no criterion applies to them.

4.3 Performance Measurement

Lighthouse measurement was conducted before and after the plugin was activated, with representative results presented in Table 4. Total Blocking Time and Cumulative Layout Shift were zero in all scenarios and are therefore omitted from the table.

Table 4. Lighthouse measurement results

Test condition	Perf.	Acc.	FCP (s)	LCP (s)	SI (s)
Before plugin	97	90	0.8	1.1	0.8
After plugin	98	90	0.7	1.1	0.7
Plugin and contrast	98	90	0.7	1.1	0.7
Plugin and dyslexic font	98	90	0.6	1.1	0.6
Plugin and spacing 2x	98	90	0.6	1.1	0.6
Plugin and magnifier	98	90	0.6	1.1	0.6
Plugin and reset	98	90	0.6	1.1	0.6

The performance score did not decline after the plugin was activated and remained stable at 98. The accessibility score stayed at 90 both before and after implementation. This finding should not be read as an absence of benefit, but rather as a consequence of the way Lighthouse assesses accessibility statically from the HTML structure [6], whereas the plugin operates dynamically by adjusting the presentation at runtime. A similar limitation has been reported in earlier comparisons of accessibility evaluation tools [13], [23], and it cautions against reading a stable score as evidence that nothing has changed for the user. DevTools Performance recording provides a more detailed picture of the processing load, as shown in Table 5.

Table 5. DevTools Performance measurement results (ms)

Test condition	Script	Render	Paint	Main thread
Before plugin	0	0	0	54.1
After plugin	2	24	16	120.4
Plugin and contrast	2	38	33	116.9
Plugin and larger text	2	50	45	185.4
Plugin and spacing 2x	2	53	52	162.4
Plugin and text-to-speech	2	5	12	57.1
Plugin and magnifier	16	44	39	141.2
Plugin and reset	3	10	18	106.7

Main thread activity increased once the plugin was activated, particularly for features that affect many elements at once. Doubling the line spacing recorded the highest load in rendering and painting, which may be explained by layout recalculation across nearly all text elements on the page. The magnifier showed the largest scripting load because the lens position is updated as the cursor moves. Taken together, the recorded figures remain within a reasonable range and did not cause blocking rendering.

4.4 Multi-Browser Testing

The consistency of plugin behaviour was examined on four browsers, with the results summarized in Table 6.

Table 6. Multi-browser compatibility testing results

Aspect tested	Chrome	Firefox	Edge	Brave
Plugin loading	Success	Success	Success	Success
Display rendering	Clear	Clear	Clear	Clear
JavaScript execution	Success	Success	Success	Success
Keyboard navigation	Success	Success	Success	Success
Focus indicator	Clear	Clear	Clear	Clear
Voice command	Stable	Failed	Unstable	Failed

Display, text, and keyboard features behaved consistently across the four browsers, so the larger part of the plugin can be regarded as satisfying the robust principle. The differences are concentrated in voice command. Firefox and Brave do not yet support the speech recognition interface of the Web Speech API, which caused the feature to fail, while Edge produced unstable responses even though recognition was available. This pattern reinforces the observation that differing implementations across rendering engines remain a source of inconsistency in modern web features [3][4], and it indicates that dependence on built-in browser APIs is the principal weakness of the proposed design. The engine coverage of this test is itself partial. Chrome, Edge, and Brave are all Blink based, so the four browsers in fact represent only two engines, Blink and Gecko. WebKit based browsers such as Safari were not examined, and their handling of speech recognition, CSS filters, and focus outlines may differ from what is reported here. Work on responsive layout failures indicates that such divergences tend to surface in layout rather than in script execution [20], so the display features that appeared stable here are precisely the ones that would need re-examination on WebKit.

4.5 Limitations of the Study

Several boundaries should be kept in view when reading the results above. The evaluation rests on a single company profile website with a fairly regular structure, so the figures reported in Tables 4 and 5 describe how the plugin behaves on that page rather than how it would behave in general. Sites with heavier media, dynamic content injected after load, or component frameworks that manage their own DOM may produce different rendering costs, and the claim of reusability therefore remains a design property that has not yet been tested empirically on other sites.

The evaluation is also entirely technical. Conformance was assessed by matching features against criteria and by observing whether they operated, without any direct involvement of users with visual or motor impairments. No task completion, error rate, or satisfaction data were collected, so the extent to which the panel actually improves the browsing experience of its intended users remains an open question. Automated auditing does not fill this gap either, given the known inability of such tools to evaluate interactive behaviour [13][23]. Reuschel and colleagues illustrate the distance between the two kinds of evidence: screen reader users failed to complete roughly half of the job applications they attempted, on sites whose technical markup gave little warning of the difficulty [24]. Whether the panel proposed here would fare better is, at this stage, an untested assumption rather than a finding.

Two further constraints concern measurement and coverage. Each performance scenario was measured once with a recording window that varied between five and twelve seconds according to the feature under test, so small differences between rows should not be over-interpreted, and repeated runs with a fixed window would be needed before the values are treated as stable benchmarks. Browser coverage was limited to two rendering engines, as noted in Subsection 4.4. The magnifier and zoom results warrant one

additional caveat, since reflow behaviour under magnification varies with the page content itself [19] and the single case study offers no basis for generalizing it.

5. CONCLUSSION

A modular JavaScript accessibility plugin delivered through a CDN can be attached to the case study website without meaningful changes to the existing system structure. All developed features, ranging from display and text settings to text-to-speech, voice command, magnifier, and keyboard navigation, functioned as designed. Nine of them map to a specific WCAG 2.1 success criterion, while cursor enlargement and read aloud are better described as supporting functions that serve the Perceivable principle without corresponding to a numbered criterion. Results from Lighthouse and DevTools Performance indicate that the plugin does not degrade page performance, although additional rendering activity appears in features that modify spacing and display scale.

Testing on four browsers shows that most features run consistently, with the exception of voice command, which depends on the speech recognition support built into each browser. These conclusions hold within clear limits. They rest on one website, on measurements taken once per scenario, and on two rendering engines, and they say nothing about how the plugin performs in the hands of the users it is meant to serve.

Four directions follow from those limits. Testing should be extended to WebKit based browsers so that all three major engines are represented. Evaluation should cover a broader set of sites, including pages built with component frameworks, before reusability is claimed as an established property. Performance should be re-measured with repeated runs and a fixed recording window so that the figures become comparable across features. Most importantly, usability testing with users who have visual or motor impairments should be conducted, since task based evidence from those users is the only way to establish whether the plugin delivers the benefit it was designed for. Established protocols for such studies exist, and the choice among local, remote, and prototype based sessions has itself been examined for blind participants [21]. Replacing the recognition mechanism with a more universal library and adding a build process with bundling and minification would also strengthen the implementation itself. Coupling runtime adaptation of this kind with remediation at design time, as recent surveys of automated approaches recommend [17], may prove more durable than either strategy pursued alone.

REFERENCES

[1] S. Krug, Don't Make Me Think: A Common Sense Approach to Web Usability, 3rd ed. Berkeley, CA, USA: New Riders, 2014.

[2] J. Nielsen, Usability Engineering. San Francisco, CA, USA: Morgan Kaufmann, 1994.

[3] S. Choudhary, H. Versee, and A. Orso, "Automated cross-browser compatibility testing," in Proc. IEEE Int. Conf. Software Testing, Verification and Validation, Paris, France, 2010.

[4] F. A. Santos and R. P. Mello, "A systematic literature review on cross-browser testing," Journal of Software Engineering Research and Development, vol. 9, pp. 1–14, 2021.

[5] World Wide Web Consortium (W3C), "Web Content Accessibility Guidelines (WCAG) 2.1," Jun. 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/

[6] Google Developers, "Lighthouse documentation," 2024. [Online]. Available: https://developer.chrome.com/docs/lighthouse/

[7] Google Chrome Developers, "Chrome DevTools documentation," 2024. [Online]. Available: https://developer.chrome.com/docs/devtools/

[8] M. Silva and T. Oliveira, "Digital accessibility in virtual learning environments," EaD em Foco, vol. 13, pp. 1–15, 2023.

[9] World Wide Web Consortium (W3C), "Accessible Rich Internet Applications (WAI-ARIA) 1.2," Jun. 2023. [Online]. Available: https://www.w3.org/TR/wai-aria-1.2/

[10] Mozilla Developer Network (MDN), "Web Speech API," 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API

[11] A. R. Ismail and R. Kurniawan, "Accessibility evaluation of university websites using WCAG 2.1 guidelines," Journal of Information Systems and Technology, vol. 7, no. 2, pp. 101–110, 2022.

[12] M. A. Rahman and S. Hossain, "Evaluation of academic website accessibility for users with disabilities," International Journal of Computer Science, vol. 18, no. 4, pp. 233–241, 2021.

[13] D. Rajapakse and P. Brereton, "Comparing WCAG tools for accessibility evaluation of websites," International Journal of Human Computer Studies, vol. 152, pp. 102–118, 2021.

[14] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach, 9th ed. New York, NY, USA: McGraw-Hill Education, 2019.

[15] U. Singh, J. Divya Venkatesh, A. Muraleedharan, K. S. Saluja, A. J. H., and P. Biswas, "Accessibility analysis of educational websites using WCAG 2.0," Digital Government: Research and Practice, vol. 5, no. 3, art. 32, pp. 1–28, 2024, doi: 10.1145/3696318.

[16] S. Dash, "AI-powered real-time accessibility enhancement: a solution for web content accessibility issues," Jurnal Online Informatika, vol. 9, no. 1, pp. 80–88, 2024, doi: 10.15575/join.v9i1.1310.

[17] G. Vera-Amaro and J. R. Rojano-Cáceres, "Towards accessible website design through artificial intelligence: a systematic literature review," Information and Software Technology, vol. 186, art. 107821, 2025, doi: 10.1016/j.infsof.2025.107821.

[18] A. Sarioğlu, H. Metin, and D. Bork, "Accessibility in conceptual modeling: a systematic literature review, a keyboard-only UML modeling tool, and a research roadmap," Data and Knowledge Engineering, vol. 158, art. 102423, 2025, doi: 10.1016/j.datak.2025.102423.

[19] F. E. Sandnes, "To wrap or not to wrap? A study of how long words are split when reflowed on magnified web pages," Universal Access in the Information Society, vol. 24, no. 1, pp. 435–447, 2023, doi: 10.1007/s10209-023-01066-y.

[20] W. M. Watanabe, D. dos Santos, and C. de Oliveira, "Layout cross-browser failure classification for mobile responsive design web

applications: combining classification models using feature selection," ACM Transactions on the Web, vol. 17, no. 4, pp. 1–34, 2023, doi: 10.1145/3580518.

[21] M. Miao, H. A. Pham, J. Friebe, and G. Weber, "Contrasting usability evaluation methods with blind users," Universal Access in the Information Society, vol. 15, no. 1, pp. 63–76, 2016, doi: 10.1007/s10209-014-0378-8.

[22] N. Menzi-Çetin, E. Alemdağ, H. Tüzün, and M. Yıldız, "Evaluation of a university website's usability for visually impaired students," Universal Access in the Information Society, vol. 16, no. 1, pp. 151–160, 2017, doi: 10.1007/s10209-015-0430-3.

[23] J. Shree DV and P. Biswas, "Comparing ten WCAG tools for accessibility evaluation of websites," Technology and Disability, vol. 33, no. 3, pp. 163–185, 2021, doi: 10.3233/TAD-210329.

[24] W. Reuschel, M. McDonnall, and D. Burton, "The accessibility and usability of online job applications for screen reader users," Journal of Visual Impairment and Blindness, vol. 117, no. 6, pp. 479–490, 2023, doi: 10.1177/0145482X231216757.

[25] K. Chemnad and A. Othman, "Digital accessibility in the era of artificial intelligence: bibliometric analysis and systematic review," Frontiers in Artificial Intelligence, vol. 7, art. 1349668, 2024, doi: 10.3389/frai.2024.1349668.