DOI: https://doi.org/10.26565/2304-6201-2026-69-05

UDC 004.738.5:004.9-055.52

**Savchenko Mykhailo** *student*
*National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37 Beresteiskyi Ave., Kyiv-56, Ukraine, 03056*
**https://orcid.org/0009-0005-9441-3467**
*e-mail: its30316@gmail.com*

**Sulima Svitlana** *PhD, assistant professor;*
*assistant professor of the Department of Information technologies in telecommunications, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37 Beresteiskyi Ave., Kyiv-56, Ukraine, 03056*
*e-mail: itssulima@gmail.com;*
**https://orcid.org/0000-0002-6333-7693**

# Modular JavaScript library for ensuring web interface accessibility in accordance with WCAG 2.2

**Relevance**. Web accessibility has become a critical aspect of modern web development, considering the needs of more than 1.3 billion people with disabilities worldwide. Despite the existence of WCAG standards, the vast majority of websites remain inaccessible, highlighting the demand for comprehensive yet easy-to-integrate tools that address key accessibility challenges.
**Purpose**. The main goal is to develop a modular JavaScript library that provides comprehensive web interface accessibility enhancements in accordance with WCAG 2.2, while maintaining simplicity of integration and high performance.
**Research Methods**. The research applied a user-centered iterative development methodology with step-by-step validation of features through scripted evaluation, comparative testing with existing solutions, and the implementation of a browser extension for practical verification.
**Results**. A modular JavaScript library was developed consisting of seven independent components (dark mode, high contrast, keyboard navigation, text scaling, focus enhancement, dyslexia support, double-click protection), each addressing specific WCAG 2.2 success criteria. The effectiveness of the components was demonstrated through measurable improvements: enhanced contrast ratios (from 3.8:1 to 21:1), a 25% reduction in keystrokes for navigation, increased focus visibility (contrast improvement from 1 to 6.5), and full compliance with dyslexia readability parameters. Real-time interaction and dynamic content adaptation further improve user experience.
**Conclusions**. The proposed solution bridges the gap between fragmented accessibility tools by offering a unified approach with a high level of modularity. The library has demonstrated practical feasibility through a browser extension and is ready for integration into existing web projects. The proposed architecture provides a robust foundation for future research and development in the field of digital accessibility.
***Keywords:*** *web accessibility, WCAG 2.2, JavaScript library, browser extension, interface adaptation, support for users with disabilities, inclusivity.*

**Як цитувати:** Savchenko M. Sulima S. Modular JavaScript library for ensuring web interface accessibility in accordance with WCAG 2.2. *Вісник Харківського національного університету імені В. Н. Каразіна, серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2026. вип. 69. С.59-72. https://doi.org/10.26565/2304-6201-2026-69-05

**How to quote:** M. Savchenko S. Sulima,“ Modular JavaScript library for ensuring web interface accessibility in accordance with WCAG 2.2”, *Bulletin of V. N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems*, vol. 69, pp. 59-72, 2026. hhttps://doi.org/10.26565/2304-6201-2026-69-05

## Introduction

The publication of the Web Content Accessibility Guidelines (WCAG) introduced a formalized approach to web accessibility. With each iteration, the guidelines emphasized the need for developers to create accessible content and applications, paving the way for the development of specialized JavaScript libraries tailored for compliance with these standards [1]. The introduction of WCAG 2.2 further refined accessibility criteria, leading to the creation of modular JavaScript libraries that not only facilitate compliance but also enhance the overall user experience [1]. These libraries focus on reusability and modularity, allowing developers to easily integrate accessibility features into their projects while adhering to the latest guidelines. Consequently, the evolution of these libraries has become integral to modern web development practices, ensuring inclusivity for all users [2][3]. As the web continues to evolve, the

emphasis on accessibility remains critical, prompting ongoing innovation in the creation of tools and libraries that align with the latest standards and foster an inclusive digital environment for diverse user populations.

Web accessibility has become a fundamental requirement in modern web development, with over 1.3 billion people worldwide living with some form of disability according to the World Health Organization. Despite the existence of accessibility standards such as the Web Content Accessibility Guidelines [4] (WCAG), many websites remain inaccessible to users with disabilities. Recent studies indicate that 98% of websites have at least one WCAG failure, highlighting the urgent need for comprehensive accessibility solutions.

The challenge lies not only in awareness but also in the complexity of implementing accessibility features across diverse web interfaces. Traditional approaches often require extensive manual coding, specialized expertise, and significant development time, creating barriers to widespread adoption. Furthermore, existing solutions are frequently fragmented, addressing only specific accessibility needs rather than providing comprehensive support.

This research addresses these challenges by developing a modular JavaScript library that provides comprehensive accessibility enhancements while maintaining simplicity of integration and use. The library's modular design allows developers to implement specific accessibility features as needed, reducing complexity while ensuring compliance with international standards.

## 1. Research Objectives

The primary objectives of this research are:

- to develop a comprehensive, modular JavaScript library for web accessibility enhancement
- to ensure compliance with WCAG 2.2 standards across all implemented features
- to create an intuitive integration process that reduces implementation barriers
- to validate the library's effectiveness through practical testing and measurement
- to demonstrate the solution's applicability through a browser extension implementation

The scientific novelty of this work lies in several key areas:

1. Modular Architecture Innovation: Unlike existing accessibility solutions that often provide monolithic implementations, this research introduces a truly modular approach where each accessibility feature operates independently while maintaining seamless integration capabilities. This architecture allows for selective implementation based on specific user needs and project requirements.
2. Comprehensive WCAG 2.2 Mapping: The research provides a systematic mapping of each library component to specific WCAG 2.2 criteria, establishing a clear framework for compliance verification and effectiveness measurement.
3. Dynamic Adaptation System: The library implements dynamic adaptation mechanisms that respond to user interactions and preferences in real-time, providing personalized accessibility experiences without requiring page reloads or complex configuration.
4. Quantitative Effectiveness Measurement: The research introduces novel metrics and testing methodologies for quantifying accessibility improvements, including contrast enhancement ratios, navigation efficiency measurements, and focus visibility calculations.
5. Browser Extension Validation Framework: The development of a companion browser extension serves as both a practical implementation example and a validation tool, demonstrating real-world applicability and effectiveness.

## 2. Current State of Web Accessibility

Web accessibility research has evolved significantly since the introduction of the first WCAG guidelines in 1999. Current accessibility solutions can be categorized into several approaches: server-side implementations, client-side JavaScript libraries, browser extensions, and assistive technologies.

Server-side solutions, while comprehensive, require significant infrastructure changes and may not be feasible for all organizations. Client-side JavaScript libraries offer more flexibility but often lack comprehensive coverage of accessibility needs. Browser extensions provide user-controlled accessibility enhancements but typically operate independently of website design considerations.

Several JavaScript libraries address specific accessibility concerns. Libraries like "ally.js"[5] focus on focus management, while "a11y-dialog"[6] specializes in accessible modal dialogs. However, these solutions typically address single aspects of accessibility rather than providing comprehensive coverage.

The gap in existing solutions lies in the absence of a unified, modular approach that combines multiple accessibility features while maintaining independence between components. This research fills this gap by providing a comprehensive solution with a modular architecture.

WCAG 2.2 introduces additional success criteria that address mobile accessibility, cognitive disabilities, and low vision requirements. Implementing these standards requires detailed understanding of user needs and technical implementation strategies. This research addresses these challenges by providing pre-built, tested components that ensure compliance.

**3. Development Approach.**

The development methodology follows a user-centered design approach combined with iterative development and continuous testing. The process consists of several phases:

1. Requirements Analysis: Comprehensive analysis of WCAG 2.2 requirements and user needs.
2. Component Design: Modular architecture design ensuring independence and interoperability.
3. Implementation: Development of individual components with focus on performance and compatibility.
4. Testing: Automated and manual testing using industry-standard tools.
5. Validation: Browser extension development for real-world testing.

The library architecture is based on several key principles:

1. Modularity: Each component operates independently, allowing selective implementation and reducing code bloat.
2. Interoperability: Components can work together seamlessly when multiple accessibility features are needed.
3. Performance: Minimal impact on page load times and runtime performance through efficient code design and lazy loading.
4. Compatibility: Cross-browser compatibility ensuring functionality across modern web browsers.
5. Extensibility: Clear interfaces for adding new components and extending existing functionality.

**4. Library Architecture and Implementation**

The library employs a modular architecture where components are organized in a dedicated directory structure. The main architecture consists of:

```
accessibility-library/
├── components/
│   ├── highContrast.js
│   ├── focusMode.js
│   ├── darkMode.js
│   ├── keyboardNav.js
│   ├── doubleClick.js
│   ├── dyslexiaMode.js
│   └── increaseText.js
├── core/
│   └── loader.js
└── utils/
    └── helpers.js
```

Each component is designed with modularity in mind, enabling separate use. A folder structure supports extensibility, making it easy to add new components without making major changes to the existing code.

This clear project structure simplifies developers' work and provides a convenient entry mechanism for new team members. The architecture facilitates rapid deployment and engagement, making the library appealing to developers looking to swiftly and effectively enhance the user experience of their web applications.

**4.1. Component Specifications**

High Contrast Component (highContrast.js) addresses WCAG 2.2 criteria 1.4.3 (Contrast Minimum), 1.4.6 (Contrast Enhanced), and 1.4.1 (Use of Color). It implements dynamic style injection to achieve maximum contrast ratios between text and background elements.

Implementation features:

- Dynamic CSS injection for high contrast styles
- Preservation of original styles for reversibility
- Support for complex layouts and nested elements
- Real-time application without page refresh

Technical approach: the component creates a dedicated <style> element with high contrast CSS rules that override existing styles while maintaining layout integrity. The implementation uses CSS specificity and !important declarations strategically to ensure consistent application across diverse website designs.

Focus Enhancement Component (focusMode.js) enhances visual focus indicators to meet WCAG 2.2 criteria 1.4.13 (Content on Hover or Focus) and 2.4.7 (Focus Visible). It provides clear visual feedback for keyboard navigation and improves usability for users with motor impairments.

Implementation features:

- Enhanced focus outlines with customizable colors
- Shadow effects for improved visibility
- Support for all focusable elements
- Dynamic event handling for hover and focus states

Dark Mode Component (darkMode.js) addresses WCAG criteria 1.4.3, 1.4.11 (Non-text Contrast), and provides relief for users with light sensitivity or certain neurological conditions.

Implementation features:

- Intelligent color inversion algorithms
- Preservation of image and media content
- Site-specific optimizations for popular platforms
- Selective element targeting to maintain design coherence

Keyboard Navigation Component (keyboardNav.js) enhances keyboard navigation capabilities beyond standard browser implementations, addressing WCAG criteria 2.1.1 (Keyboard) and 2.1.2 (No Keyboard Trap).

Implementation features:

- Arrow key navigation between focusable elements
- Spatial awareness for logical navigation flow
- Visual feedback for current focus position
- Skip navigation for complex layouts

Novel algorithm: the component implements a spatial navigation algorithm that calculates element positions and determines the most logical navigation path based on geometric relationships rather than DOM order.

Double-Click Protection Component (doubleClick.js) provides protection against accidental activations, particularly beneficial for users with motor impairments or tremors. It addresses WCAG criteria 3.3.2 (Labels or Instructions) and 2.5.1 (Pointer Gestures).

Implementation features:

- Configurable delay between clicks
- Element-specific handling for different interaction types
- Visual feedback during delay periods
- Accessibility announcements for screen readers

Dyslexia Support Component (dyslexiaMode.js) implements typography and spacing modifications to improve readability for users with dyslexia, addressing WCAG criteria 1.4.5 (Images of Text) and 3.1.5 (Reading Level).

Implementation features:

- OpenDyslexic font integration
- Enhanced letter and word spacing
- Improved line height and paragraph spacing
- ARIA live announcements for mode changes

Text Scaling Component (increaseText.js) provides dynamic text size adjustment while maintaining layout integrity, addressing WCAG criteria 1.4.4 (Resize Text) and 1.4.10 (Reflow).

Implementation features:

- Proportional scaling across all text elements
- Layout preservation during scaling
- Original size restoration capability
- Accessibility announcements for changes

### 4.2. Integration and Usage

The library provides multiple integration methods to accommodate different development workflows (Fig. 1).

**Direct Integration:**

```javascript
// Load specific components
import { enableHighContrast } from './components/highContrast.js';
import { enableDarkMode } from './components/darkMode.js';

// Activate features
enableHighContrast(true);
enableDarkMode(true);
```

**Configuration-Based Integration:**

```javascript
// Configure multiple features
const accessibilityConfig = {
  highContrast: true,
  darkMode: false,
  textScale: 1.2,
  keyboardNav: true
};

AccessibilityLibrary.init(accessibilityConfig);
```

*Fig. 1 Integration*
*Рис. 1 Інтеграція*

### 4.3. Browser Extension Implementation

To validate the library's practical applicability, a browser extension was developed that demonstrates real-world usage scenarios.

The extension architecture includes following core files (Fig. 2):

- manifest.json: extension configuration and permissions
- popup.html: user interface template
- popup.js: user interaction handling
- content.js: content script for DOM manipulation
- styles.css: interface styling
- lib/components/ directory: this directory contains components that are responsible for specific accessibility features. Each component is a separate JavaScript file that implements a specific accessibility feature on web pages.

The extension dynamically loads library components based on user selections, demonstrating the modular architecture's flexibility and performance benefits.

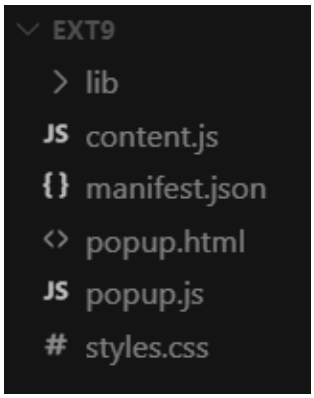


*Fig. 2 Extension structure*
*Рис. 2 Структура розширення*

This structure allows for flexibility and ease of expansion, as each individual part of the extension performs a clearly defined role, and centralized management of components and settings allows the interface to be quickly adapted to user needs.

The extension uses a multi-level architecture, where the main component is the content script (content.js). This script is automatically loaded on every page the user visits and acts as a coordinator, linking the user interface with accessibility features.

The accessibility components, located in the lib/components/ directory, are standalone modules responsible for specific functions (e.g., dark mode or text enlargement). They are loaded dynamically depending on the user's choice, which optimizes performance. When the user activates a specific feature via the popup interface, content.js receives a signal and initiates the loading of the corresponding component. After loading, the component makes changes to the DOM of the web page to improve accessibility without the need to reload the page.

In addition, the system saves user settings, and these settings are automatically restored the next time the page is visited. This architecture ensures high performance and allows you to easily add new features by creating new modules in the lib/components/ directory and registering them in the system.

The interface is implemented via a popup window (popup.html and popup.js) – Fig. 3.

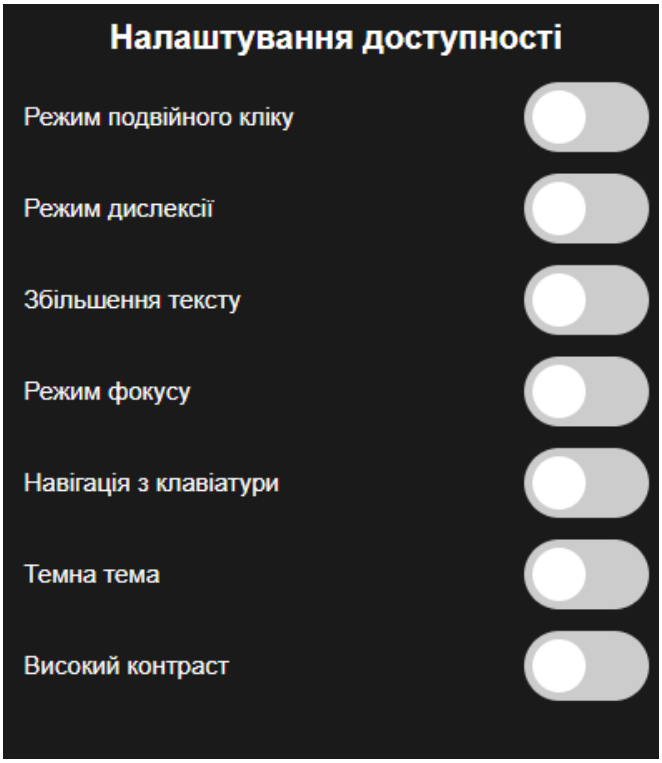


*Fig. 3 User interface of the extension*
*Рис. 3 Користувацький інтерфейс розширення*

### 4.4. Browser Extension Implementation

Let's take a few functions as an example and evaluate their effectiveness and consistency with the purpose.

First, we can evaluate the effectiveness of the high contrast mode function.

Using the WAVE Evaluation Tool, we will visit a website, for example, KPI Campus:

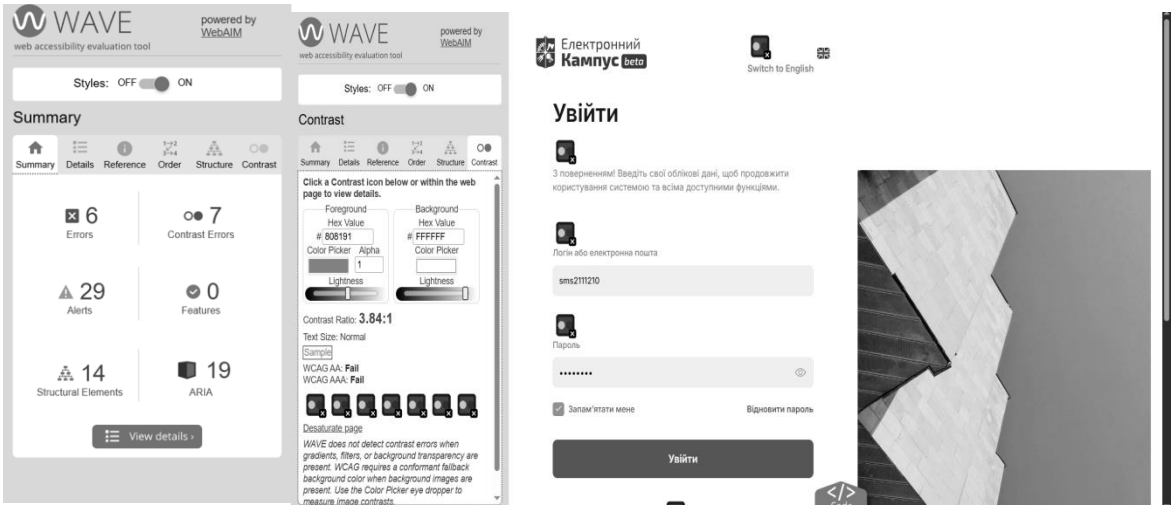


*Fig. 4 Contrast evaluation with the WAVE tool without using the extension*

*Рис. 4 Оцінка контрасту за допомогою інструменту WAVE без використання розширення*

As we can see, there are elements (7) on the website that do not meet the WCAG contrast standards and have a value of ~3.8:1, which is less than the standard.

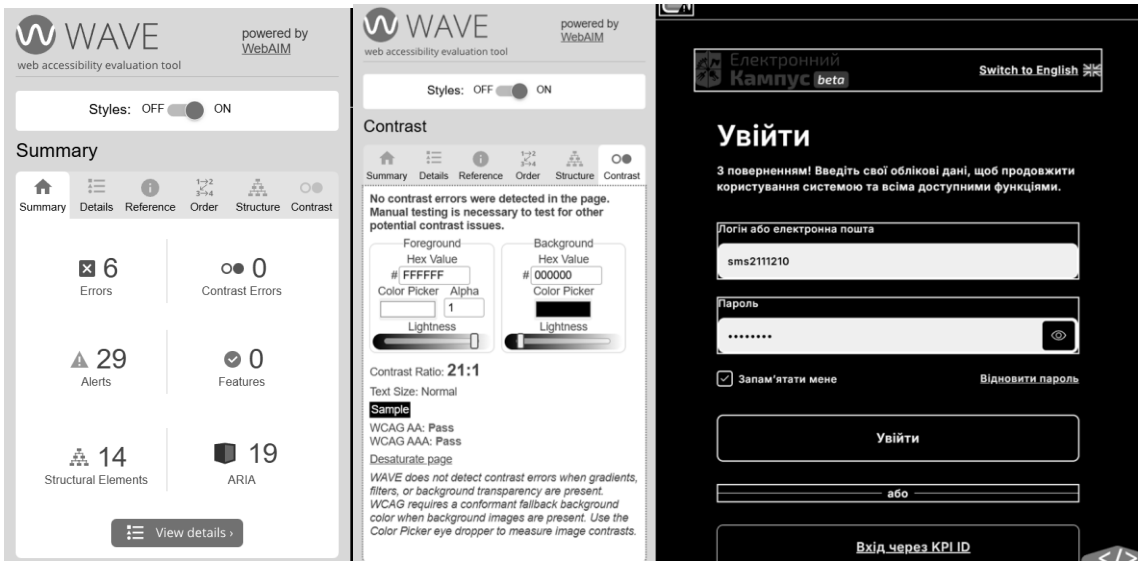


*Fig. 5 Contrast evaluation with the WAVE tool without using the extension*

*Рис. 5 Оцінка контрасту за допомогою інструменту WAVE без використання розширення*

Here is the path you need to follow from start to finish with standard keyboard navigation – Fig. 6.

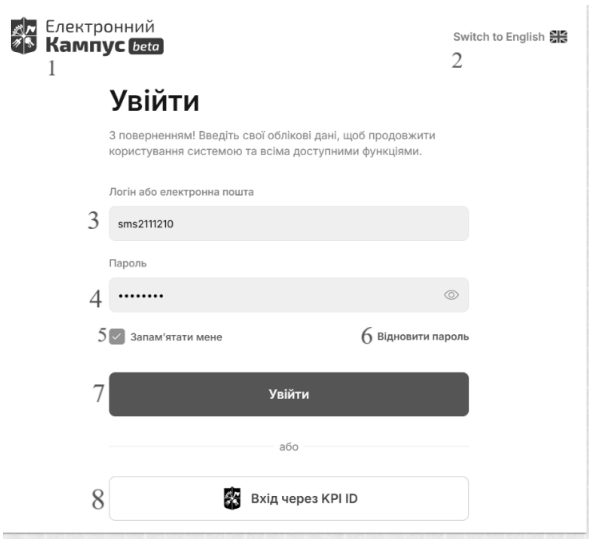


*Fig. 6 Path from start to finish with standard navigation*
*Рис. 6 Шлях від початку до кінця зі стандартною навігацією*

And here is the path with the developed navigation – Fig. 7.

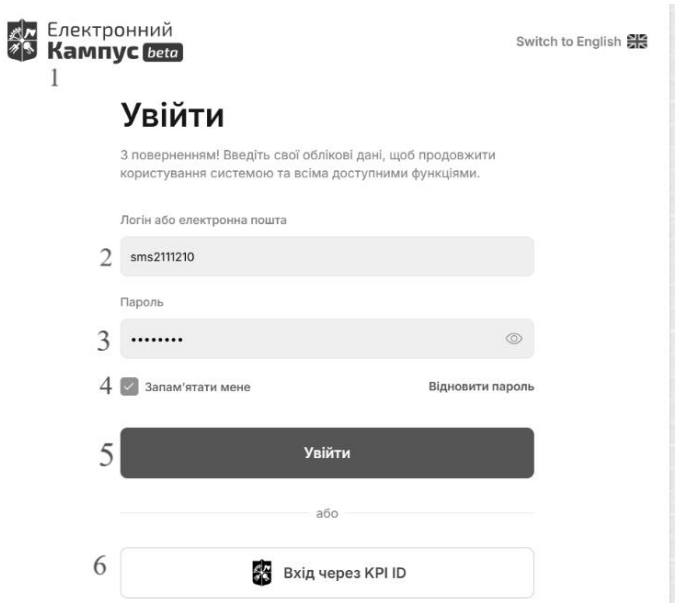


*Fig. 7 Path from start to finish using extension navigation*
*Рис. 7 Шлях від початку до кінця за допомогою навігації з розширенням*

As we can see, even in such a simple example, where there are not many elements and they are all quite consistent, in the first case you need to go through 8 elements, in the second 6. This means that the number of actions is reduced by 25%, which significantly increases the speed and convenience of navigation. And this is the minimum value, because on more complex web interfaces with a larger number of elements, the result will be much longer for standard navigation, which cannot be said about navigation with this extension.

To evaluate the focus mode and its compliance with the specified goal, you can develop a small script that would check the contrast of elements. For example, the following script allows you to evaluate how noticeable the focus is on interactive elements: when you hover the cursor or focus (via the keyboard), it analyzes how the element stands out — using an outline or box shadow — and calculates the contrast between these styles and the background color. The results are displayed in the console and help verify that the focus meets accessibility requirements.

```
(() => {
  const focusableSelector = 'a, button, input, select, textarea, [tabindex]:not([tabindex="-1"])';

  function getRGB(str) {
    const match = str.match(/rgba?\((\d+),\s*(\d+),\s*(\d+)/);
    return match ? match.slice(1, 4).map(Number) : null;
  }

  function luminance(rgb) {
    if (!rgb) return 0;
    const [r, g, b] = rgb.map(v => {
      v /= 255;
      return v <= 0.03928 ? v / 12.92 : Math.pow((v + 0.055) / 1.055, 2.4);
    });
    return 0.2126 * r + 0.7152 * g + 0.0722 * b;
  }

  function contrast(rgb1, rgb2) {
    if (!rgb1 || !rgb2) return 1;
    const l1 = luminance(rgb1);
    const l2 = luminance(rgb2);
    return ((Math.max(l1, l2) + 0.05) / (Math.min(l1, l2) + 0.05)).toFixed(2);
  }

  function analyze(el) {
    const cs = getComputedStyle(el);
    const bg = getRGB(cs.backgroundColor);
    const outlineColor = cs.outlineStyle !== 'none' ? getRGB(cs.outlineColor) : null;
    const shadowMatch = cs.boxShadow.match(/rgba?\([^)]+\)/g);
    const shadowColor = shadowMatch ? getRGB(shadowMatch.at(-1)) : null;

    const contrastOutline = contrast(outlineColor, bg);
    const contrastShadow = contrast(shadowColor, bg);

    console.clear();
    console.log('🔍 Элемент:', el);
    console.log('Outline:', cs.outline);
    console.log('Box-shadow:', cs.boxShadow);
    console.log('Контраст outline:', contrastOutline);
    console.log('Контраст shadow:', contrastShadow);
  }

  function handler(e) {
    const el = e.target.closest(focusableSelector);
    if (el) analyze(el);
  }

  document.addEventListener('mousemove', handler);
  document.addEventListener('focusin', handler);
```

*Fig.8 Script code for evaluating focus mode*
*Рис. 8 Код скрипта для визначення режиму фокусування*

For example, here is the result without focus mode enabled – Fig. 9.

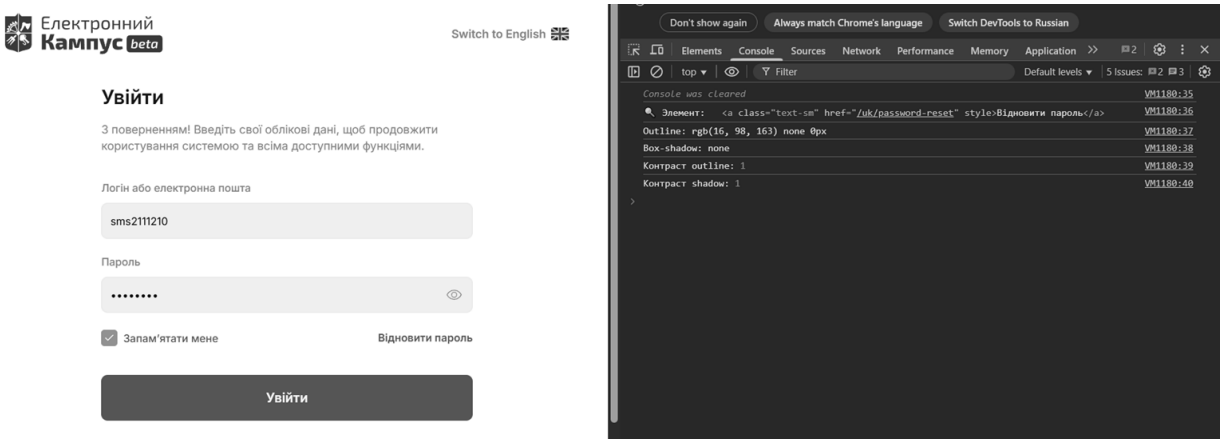


*Fig. 9 Script result without focus mode enabled*
*Рис. 9 Результат виконання скрипта без увімкненого режиму фокусування*

And here is the result with focus mode enabled - Fig. 10.

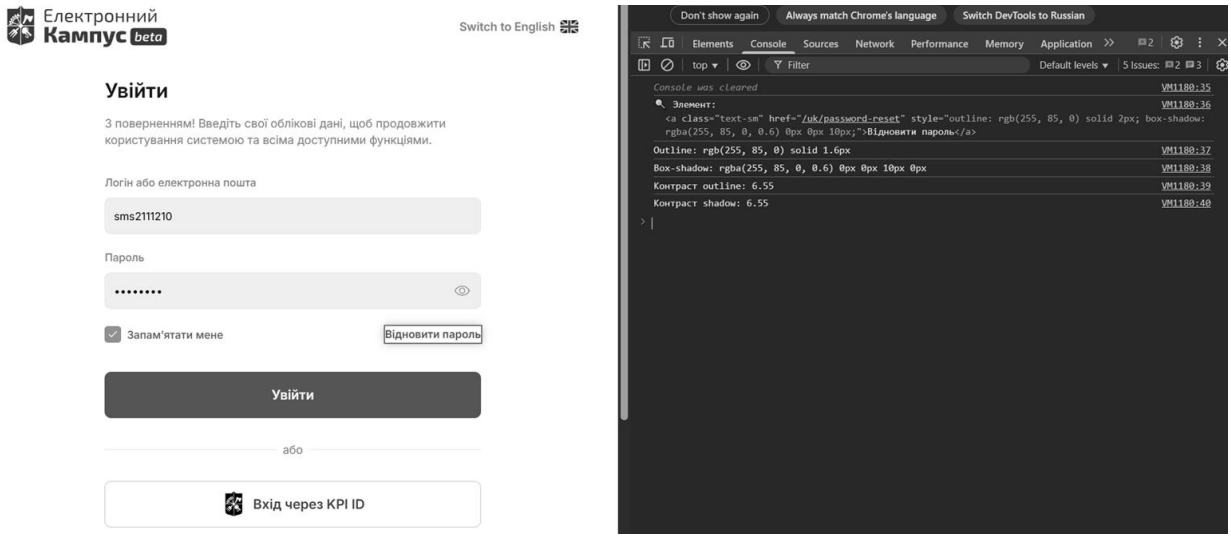


*Fig. 10 Script result with focus mode enabled*
*Рис. 10 Результат виконання скрипта при увімкненому режимі фокусування*

As we can see, without active focus mode, the element has no visible outline or shadow, and the contrast level is 1 — the focus is visually invisible. However, with the specified style (outline and box-shadow with a bright color), the contrast increases to 6.55, which is sufficient to ensure a noticeable focus on the element according to WCAG standards (minimum 3:1 for visible focus).

Next, we can evaluate the effectiveness in terms of compliance with the purpose of dyslexia mode. The script analyzes key parameters that affect readability for people with dyslexia: the font used, the spacing between letters, words, and lines, and the presence of dynamic support through the aria-live attribute. The result is a summary score that helps to quickly determine how well the text meets the basic accessibility criteria for this mode.

```
// Функція для отримання кольору у форматі RGB
function getRGB(color) {
    const div = document.createElement('div');
    div.style.backgroundColor = color;
    document.body.appendChild(div);
    const computed = window.getComputedStyle(di
    document.body.removeChild(div);
    const rgb = computed.match(/\d+/g).map(Numb
    return rgb.length >= 3 ? rgb : [255, 255, 2
}

// Функція для обчислення яскравості (залишена
function getLuminance(rgb) {
    const [r, g, b] = rgb.map(v => v / 255);
    const a = [r, g, b].map(v => {
        return v <= 0.03928 ? v / 12.92 : Math.
    });
    return 0.2126 * a[0] + 0.7152 * a[1] + 0.07
}

// Основна функція для оцінки параметрів режиму
function evaluateDyslexiaMode() {
    const paragraphs = document.querySelectorAl
    const results = {
        font: { score: 0, value: '' },
        letterSpacing: { score: 0, value: 0 },
        wordSpacing: { score: 0, value: 0 },
        lineHeight: { score: 0, value: 0 },
        ariaLive: { score: 0, value: 'Не знайде
    };

    if (paragraphs.length > 0) {
        const p = paragraphs[0];
        const style = window.getComputedStyle(p
        const fontFamily = style.fontFamily.toL
        results.font.value = fontFamily;

        // Оцінка шрифту
        if (fontFamily.includes('opendyslexic')
            results.font.score = 2;
        } else if (!fontFamily.includes('serif'
            results.font.score = 1;
        } else {
            results.font.score = 0;
        }

        // Інтервал між літерами
        const letterSpacing = parseFloat(style.
        const fontSize = parseFloat(style.fontS
        results.letterSpacing.value = letterSpa
        if (letterSpacing >= 0.12 * fontSize) {
            results.letterSpacing.score = 1;
        }

        // Інтервал між словами
        const wordSpacing = parseFloat(style.wordSpacing);
        results.wordSpacing.value = wordSpacing;
        if (wordSpacing >= 0.16 * fontSize) {
            results.wordSpacing.score = 1;
        }

        // Міжрядковий інтервал
        const lineHeight = parseFloat(style.lineHeight) / fontSize;
        results.lineHeight.value = lineHeight;
        if (lineHeight >= 1.5) {
            results.lineHeight.score = 1;
        }
    }

    // Перевірка наявності aria-live
    const ariaLiveElement = document.querySelector('[aria-live]');
    if (ariaLiveElement) {
        results.ariaLive.value = ariaLiveElement.getAttribute('aria-live');
        if (['polite', 'assertive'].includes(results.ariaLive.value)) {
            results.ariaLive.score = 1;
        }
    }

    // Виведення результатів у консоль
    console.log('📊 Оцінка параметрів для режиму дислексії:');
    console.log(1. Шрифт: ${results.font.value} → Бал: ${results.font.score});
    console.log(2. Інтервал між літерами: ${results.letterSpacing.value}px → Бал: ${results.letterSpaci
    console.log(3. Інтервал між словами: ${results.wordSpacing.value}px → Бал: ${results.wordSpacing.sc
    console.log(4. Міжрядковий інтервал: ${results.lineHeight.value} → Бал: ${results.lineHeight.score}
    console.log(5. aria-live: ${results.ariaLive.value} → Бал: ${results.ariaLive.score});

    const totalScore = Object.values(results).reduce((sum, r) => sum + r.score, 0);
    const maxScore = 6; // 2 бали за шрифт, по 1 за інші 4 критерії

    console.log(\n🔹 Загальна підсумкова оцінка: ${totalScore}/${maxScore} балів);
}

// Виклик функції
evaluateDyslexiaMode();
```

*Fig. 11 Script code for evaluating dyslexia mode*
*Рис. 11 Код скрипта для активації режиму дислексії*

Next, we evaluate the effectiveness of dyslexia mode based on key parameters such as font, spacing, and aria-live support. The script calculates the score for each criterion.

Below are examples of results from the console:

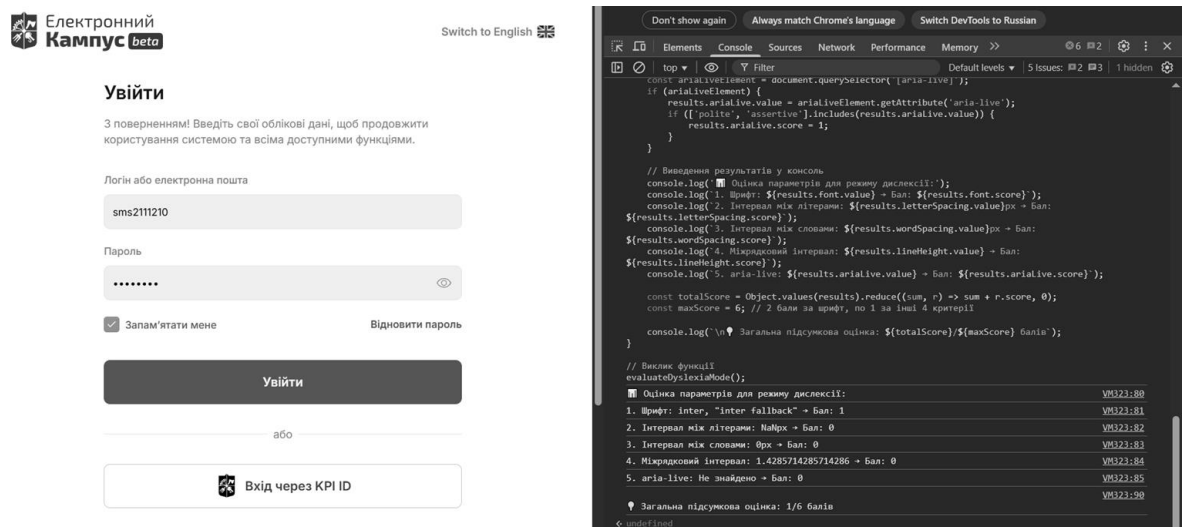


*Fig. 12 Script result without dyslexia mode enabled*

*Рис. 12 Результат виконання скрипта без увімкненого режиму дислексії*

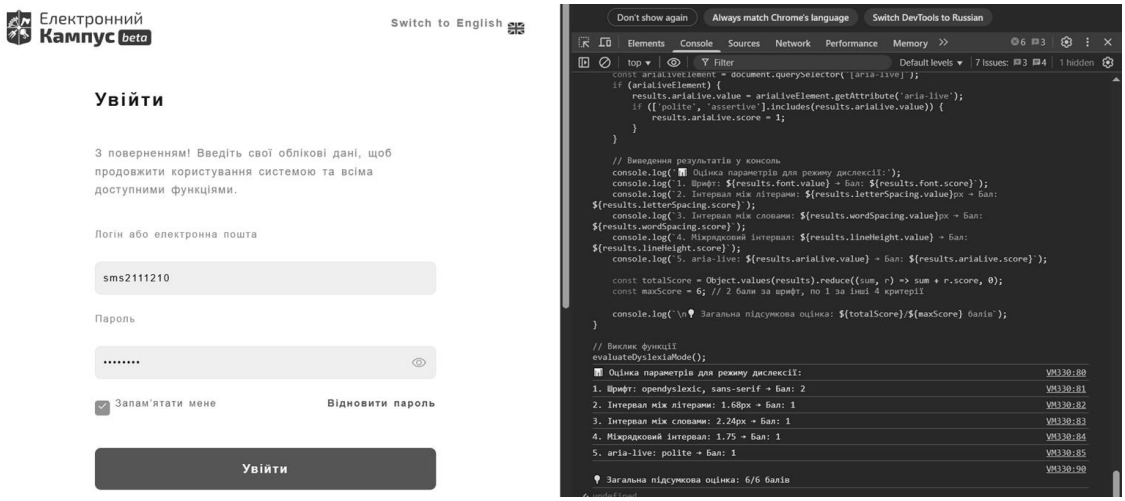


*Fig. 13 Script result with dyslexia mode enabled*

*Рис. 13 Результат виконання скрипта з увімкненим режимом для людей з дислексією*

Initially, the dyslexia mode score was low – only 1 out of 6 points, indicating that the web resource was not sufficiently inclusive. After the changes were made, the result improved to a maximum of 6 out of 6, which means full compliance with the basic requirements for comfortable reading by users with dyslexia.

Now let's check the effectiveness of dark mode. This script collects the background colors of the main blocks of the page (body, header, main, section), converts them to RGB numerical values, and calculates their brightness (lux) using a formula that takes into account the perception of color by the human eye: $0.2126 \times R + 0.7152 \times G + 0.0722 \times B$, where R, G, B are the values of the red, green, and blue channels, respectively.

The script then calculates the average brightness of the background to estimate the overall brightness of the page.

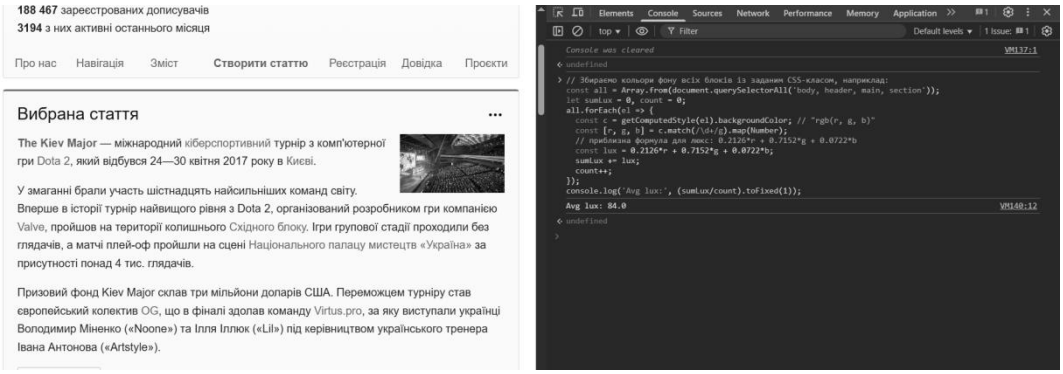


*Fig. 14 Result of the script without dark mode enabled*
*Рис. 14 Результат виконання скрипта без увімкненого темного режиму*

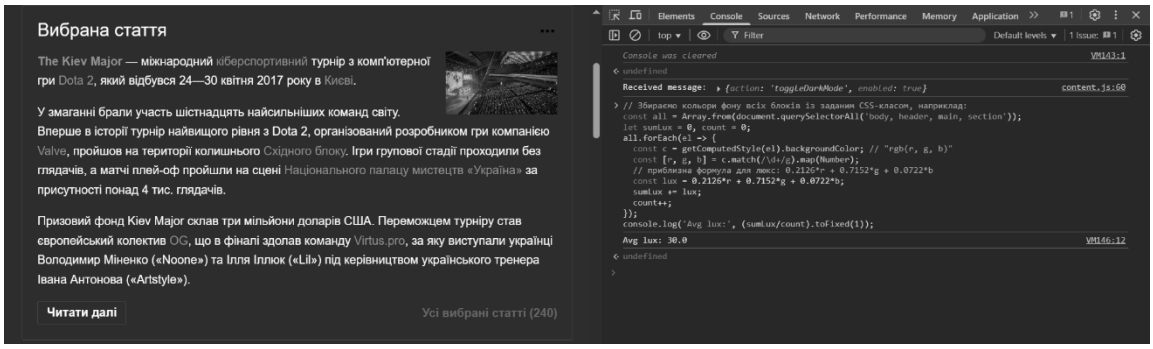


*Fig. 15 Result of the script with dark mode enabled*
*Рис. 15 Результат виконання скрипта з увімкненим темним режимом*

We tested the brightness of the main page blocks. Before enabling dark mode, the average lux value was 84.0, which corresponds to a fairly bright background. After activating dark mode, this indicator decreased to 30.0, which indicates a significant darkening of the interface.

According to WCAG recommendations and accessibility practices, optimizing brightness levels helps reduce eye strain, especially for users with photosensitivity or visual impairments. Dark mode with reduced lux levels increases viewing comfort and makes the interface more accessible to a wider range of users.

Let's move on to evaluating the effectiveness of double-click implementation. To do this, we used a script that analyzes all interactive elements on the page and calculates what percentage of them belong to the types supported by the doubleClick component. These are considered to be those that can correctly process a second click — in particular, links, buttons, checkboxes, and text blocks.

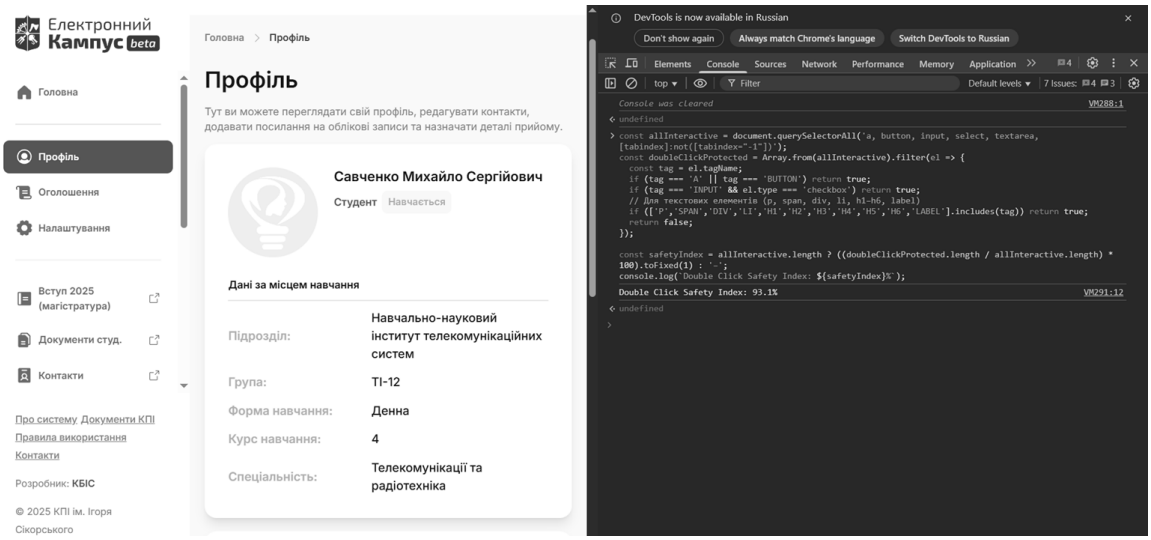


*Fig. 16 The result of the script that checks the effectiveness of the double-click function*
*Рис. 16 Результат виконання скрипта, що перевіряє працездатність функції подвійного клацання*

This means that 93.1% of all interactive elements on the page (links, buttons, checkboxes, text blocks, etc.) are protected by double-click logic. In other words, these elements have a built-in delay between clicks, which prevents accidental or premature activation. This implementation significantly reduces the risk of accidental interactions — for example, accidentally following a link or clicking a button. This is especially important for users with motor impairments, tremors, reaction delays, or those who use alternative input devices. The introduction of double-clicking as an action filter improves the accuracy of interaction with the interface, increases the level of control for the user, and contributes to a more inclusive and accessible experience.

**Conclusions**

This research successfully developed and validated a comprehensive, modular JavaScript library for enhancing web interface accessibility. The library addresses critical gaps in existing accessibility solutions by providing a unified, easy-to-integrate approach that maintains compliance with WCAG 2.2 standards while offering significant flexibility and performance benefits.

The scientific significance of this work extends beyond the immediate technical contributions:

1. Methodological Advancement: The research establishes new approaches for quantifying accessibility improvements and provides frameworks for systematic accessibility enhancement.
2. Standardization Contribution: The comprehensive mapping to WCAG 2.2 criteria and systematic implementation approach contributes to standardization efforts in accessibility technology.
3. Interdisciplinary Integration: The work bridges computer science, human-computer interaction, and disability studies, providing insights valuable across multiple disciplines.

The practical impact of this research is demonstrated through:

1. Immediate Applicability: The library can be immediately integrated into existing web projects with minimal modification.
2. Developer Accessibility: The modular approach reduces barriers to accessibility implementation for developers with varying expertise levels.
3. User Empowerment: The browser extension demonstrates how accessibility tools can be made directly available to users.
4. Cost Reduction: The library reduces the cost and complexity of implementing comprehensive accessibility features.

This research establishes a foundation for future developments in accessibility technology. The modular architecture pattern can be applied to other accessibility domains. The evaluation methodologies can be used to assess other accessibility solutions. The component library can serve as a foundation for community-driven accessibility improvements.

The successful development and validation of this library demonstrates that comprehensive web accessibility can be achieved through thoughtful design, systematic implementation, and rigorous testing. The modular approach provides a sustainable path forward for improving web accessibility while maintaining the flexibility and performance requirements of modern web development.

As web technologies continue to evolve, this research provides both immediate practical benefits and a foundation for future accessibility innovations. The combination of technical excellence, standards compliance, and practical applicability makes this work a significant contribution to the ongoing effort to create a more inclusive digital world.

REFERENCES

1. Pixel Free Studio, “The Impact of Client-Side Rendering on Accessibility,” Pixel Free Studio Blog. [Online]. Available: https://blog.pixelfreestudio.com/the-impact-of-client-side-rendering-on-accessibility/. [Accessed: Jul. 20, 2025].
2. Pixel Free Studio, “Building Accessible Web Applications with JavaScript Frameworks,” Pixel Free Studio Blog. [Online]. Available: https://blog.pixelfreestudio.com/building-accessible-web-applications-with-javascript-frameworks/. [Accessed: Jul. 20, 2025].
3. A. Smith, “Accessibility in User Interfaces: Confronting Common Challenges,” Online Scientific Research, [Online]. Available: https://www.onlinescientificresearch.com/articles/accessibility-in-user-interfaces-confronting-common-challenges.pdf. [Accessed: Jul. 20, 2025].

4. W3C, "Web Content Accessibility Guidelines (WCAG) 2.2," W3C Recommendation, Oct. 5, 2023. [Online]. Available: https://www.w3.org/TR/WCAG22/. [Accessed: Jul. 19, 2025].
5. R. Ritter, "ally.js," 2015. [Online]. Available: https://allyjs.io/. [Accessed: Jul. 19, 2025].
6. H. Giraudel, "A11y Dialog," 2014. [Online]. Available: https://a11y-dialog.netlify.app/. [Accessed: Jul. 19, 2025].

**Савченко Михайло Сергійович** *студент*
*Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», проспект Берестейський, 37, Київ, Україна, 03056*

**Суліма Світлана Валеріївна** *к.т.н., доцент, доцент кафедри Інформаційних технологій в телекомунікаціях*
*Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», проспект Берестейський, 37, Київ, Україна, 03056*
*e-mail:* itssulima@gmail.com*;*
**https://orcid.org/0000-0002-6333-7693**

# Модульна JavaScript-бібліотека для забезпечення доступності вебінтерфейсів згідно з WCAG 2.2

**Актуальність**. Вебдоступність є критично важливим аспектом сучасної веброзробки, зважаючи на потреби понад 1,3 мільярда людей з інвалідністю у світі. Попри існування стандартів WCAG, переважна більшість сайтів лишається недоступною, що свідчить про потребу у комплексних, але водночас простих у реалізації інструментах.
**Мета**. Розробити модульну JavaScript-бібліотеку, яка забезпечує всебічне покращення доступності вебінтерфейсів згідно з WCAG 2.2, зберігаючи простоту інтеграції та високу продуктивність.
**Методи дослідження**. Застосовано методологію ітеративної розробки, орієнтованої на користувача, із поетапною валідацією функцій через скриптові перевірки, порівняльні тести з існуючими рішеннями та впровадженням браузерного розширення.
**Результати**. Створено бібліотеку з семи незалежних компонентів (темний режим, високий контраст, навігація клавіатурою, масштабування тексту, фокусування, підтримка людей з дислексією, захист від подвійного кліку), кожен з яких відповідає конкретним критеріям WCAG 2.2. Ефективність продемонстровано через вимірювані покращення у контрастності, зменшення кількості дій для навігації, підвищення доступності фокусу та адаптації інтерфейсу під потреби користувачів.
**Висновки**. Розроблене рішення заповнює наявну прогалину між розрізненими інструментами доступності, пропонуючи уніфікований підхід із високим рівнем гнучкості. Бібліотека демонструє практичну доцільність завдяки браузерному розширенню та може бути легко інтегрована у наявні вебпроекти. Запропонована архітектура створює основу для подальших досліджень і розвитку технологій доступності.

***Ключові слова:*** *вебдоступність, WCAG 2.2, JavaScript-бібліотека, модульність, браузерне розширення, адаптація інтерфейсу, підтримка користувачів з інвалідністю, інклюзивність.*

Надійшла у першій редакції 16.01.2026; рекомендована до друку 15.03.2026; опублікована 03.04.2026.
The first version has been received on 16.01.2026; was recommended for printing on 15.03.2026; published 03.04.2026.