УДК 004.738.5+004.4’42 DOI https://doi.org/10.35546/kntu2078-4481.2025.2.2.21

Г. О. КОЗУБ
кандидат технічних наук, доцент,
доцент кафедри інформаційних технологій та систем
Навчально-науковий інститут математики та інформаційних технологій
ДЗ «Луганський національний університет імені Тараса Шевченка»
ORCID: 0000-0001-5387-050X

Ю. Ю. СУРМА
викладач фахового коледжу УКД,
керівник відділу автоматизації та електронного тестування
Університет Короля Данила
ORCID: 0009-0006-3308-7984

О. І. АРТЕМЕНКО
кандидат технічних наук, доцент,
завідувач кафедри комп’ютерних систем і технологій
Приватний вищий навчальний заклад «Буковинський університет»
ORCID: 0000-0002-4057-1217

РОЛЬ ВЕБКОМПОНЕНТІВ У ПОБУДОВІ СУЧАСНИХ ІНТЕРФЕЙСІВ: ПЕРЕВАГИ ТА ОБМЕЖЕННЯ

Мета статті. Метою статті є дослідження впливу вебкомпонентів на створення сучасних веб-інтерфейсів, аналіз їхньої ролі в забезпеченні модульності, сумісності, продуктивності, доступності та тестовності в контексті життєвого циклу програмного забезпечення (ПЗ). Стаття також спрямована на розробку стратегій подолання обмежень, пов’язаних із впровадженням вебкомпонентів, для підвищення якості веб-додатків.

Наукова новизна. Стаття поглиблює розуміння ролі вебкомпонентів, базуючись на стандартах W3C, у забезпеченні функціональних і нефункціональних вимог веб-розробки. Новизна полягає в систематизації підходів до інтеграції вебкомпонентів із сучасними фреймворками (React, Angular, Vue.js), аналізі їхнього впливу на всі етапи життєвого циклу ПЗ та розробці практичних рекомендацій для подолання викликів, таких як складність тестування Shadow DOM, оптимізація продуктивності через WebAssembly і забезпечення доступності за стандартами WCAG. Особливу увагу приділено недостатньо дослідженим аспектам, зокрема інтеграції з DevOps-практиками та автоматизованим тестуванням.

Результати. Дослідження показало, що вебкомпоненти, використовуючи Custom Elements, Shadow DOM, HTML Templates та ES Modules, сприяють створенню модульних і масштабованих інтерфейсів, зменшуючи складність проєктування та розробки. Вони забезпечують ізоляцію стилів і логіки, що запобігає конфліктам у великих проєктах, та сумісність із різними фреймворками, полегшуючи інтеграцію. На етапі тестування вебкомпоненти спрощують модульне тестування, але ускладнюють юніт-тестування через Shadow DOM, що вимагає спеціалізованих інструментів, таких як Playwright чи Testing Library. Продуктивність залежить від оптимізації рендерингу, де WebAssembly і модульний рендеринг зменшують затримки. Доступність потребує використання ARIA-атрибутів і семантичного HTML, а SEO – серверного рендерингу (SSR) чи статичної генерації (SSG). Порівняльний аналіз показав переваги вебкомпонентів у модульності та повторному використанні порівняно з традиційними підходами, але виявив необхідність додаткових зусиль для тестування та доступності.

Висновки. Вебкомпоненти є ефективним інструментом для побудови сучасних веб-інтерфейсів, забезпечуючи модульність, інкапсуляцію та сумісність. Вони спрощують розробку, розгортання та підтримку ПЗ, але потребують ретельного підходу до тестування, продуктивності та доступності. Запропоновані стратегії, такі як використання WebAssembly, автоматизованих інструментів тестування (Playwright, axe-core) і SSR для SEO, дозволяють долати обмеження. У 2025 році вебкомпоненти залишаються актуальними, інтегруючись із новими технологіями, такими як Web 3.0 та AI, що відкриває перспективи для їхнього подальшого розвитку.

Ключові слова: *життєвий цикл ПЗ, функціональні та нефункціональні вимоги, тестування.*

H. O. KOZUB
Candidate of Engineering Sciences, Associate Professor,
Associate Professor at the Department of Information Technology
and Systems
Educational and Scientific Institute of Mathematics and Information Technologies
of Luhansk Taras Shevchenko National University
ORCID: 0000-0001-5387-050X

YU. YU. SURMA
Lecturer at the Vocational College of Ukraine,
Head of the Automation and Electronic Testing Department
King Danylo University
ORCID: 0009-0006-3308-7984

O. I. ARTEMENKO
PhD in Technical Sciences, Associate Professor,
Head of the Computer Systems and Technologies Department
Private Higher Educational Institution "Bukovynian University"
ORCID: 0000-0002-4057-1217

THE ROLE OF WEB COMPONENTS IN BUILDING MODERN INTERFACES: ADVANTAGES AND LIMITATIONS

The purpose of the article. The purpose of the article is to study the impact of web components on the creation of modern web interfaces, to analyze their role in ensuring modularity, compatibility, performance, availability and testability in the context of the software life cycle. The article also aims to develop strategies for overcoming the limitations associated with the implementation of web components to improve the quality of web applications.

Scientific novelty. The article deepens the understanding of the role of web components based on W3C standards in meeting the functional and non-functional requirements of web development. The novelty lies in systematizing approaches to integrating web components with modern frameworks (React, Angular, Vue.js), analyzing their impact on all stages of the software life cycle, and developing practical recommendations for overcoming challenges such as the complexity of Shadow DOM testing, performance optimization through WebAssembly, and ensuring accessibility according to WCAG standards. Particular attention is paid to under-researched aspects, such as integration with DevOps practices and automated testing.

Results. The study found that web components using Custom Elements, Shadow DOM, HTML Templates, and ES Modules help create modular and scalable interfaces, reducing design and development complexity. They provide isolation of styles and logic to prevent conflicts in large projects and compatibility with different frameworks, making integration easier. At the testing stage, web components simplify unit testing, but complicate unit testing due to the Shadow DOM, which requires specialized tools such as Playwright or Testing Library. Performance depends on rendering optimization, where WebAssembly and modular rendering reduce latency. Accessibility requires the use of ARIA attributes and semantic HTML, and SEO requires server-side rendering (SSR) or static generation (SSG). The comparative analysis showed the advantages of web components in modularity and reusability compared to traditional approaches but revealed the need for additional efforts for testing and accessibility.

Conclusions. Web components are an effective tool for building modern web interfaces, providing modularity, encapsulation, and compatibility. They simplify software development, deployment, and support, but require a careful approach to testing, performance, and availability. The proposed strategies, such as the use of WebAssembly, automated testing tools (Playwright, axe-core), and SSR for SEO, allow overcoming the limitations. In 2025, web components remain relevant, integrating with new technologies such as Web 3.0 and AI, which opens prospects for their further development.

Key words: *software life cycle, functional and non-functional requirements, testing.*

Постановка проблеми

Сучасна веб-розробка зіштовхується із завданнями створення модульних, гнучких і масштабованих інтерфейсів, які відповідають зростаючій складності веб-додатків. Необхідність інтеграції з різними платформами, швидкого оновлення програмного забезпечення (ПЗ) та забезпечення високої якості користувацького досвіду підкреслюють важливість технологій, що сприяють повторному використанню коду, інкапсуляції та сумісності. Вебкомпоненти, засновані на стандартах W3C, таких як Custom Elements, Shadow DOM, HTML Templates і ES Modules, пропонують рішення для цих викликів, спрощуючи етапи життєвого циклу ПЗ, включаючи проєктування, розробку, тестування, розгортання та підтримку. Проте впровадження вебкомпонентів пов'язане з обмеженнями, такими як складність автоматизованого тестування, забезпечення продуктивності, відповідність стандартам доступності та підтримка застарілих браузерів. У контексті життєвого циклу ПЗ виникає потреба в детальному аналізі ролі вебкомпонентів у забезпеченні функціональних і нефункціональних вимог, зокрема модульності, продуктивності, доступності та тестовності, для створення ефективних і якісних веб-інтерфейсів.

Аналіз останніх досліджень і публікацій

Дослідження у сфері веб-розробки підкреслюють значення вебкомпонентів для створення модульних і масштабованих інтерфейсів. Козуб Г. О., Козуб Ю. Г. [1] акцентують на декларативному підході до розробки мультиплатформних додатків, де вебкомпоненти сприяють модульності шляхом створення ізольованих елементів інтерфейсу, що спрощує проєктування та інтеграцію на етапі розробки. Gorai S. K. [2] аналізує бар'єри веб-розробки, зокрема складність інкапсуляції, і зазначає, що Shadow DOM у вебкомпонентах вирішує проблеми конфліктів стилів

і скриптів, але вимагає додаткових зусиль для SEO через динамічний контент. Maulana I. [3] досліджує доступність веб-інтерфейсів, наголошуючи, що вебкомпоненти потребують ретельного впровадження ARIA-атрибутів і семантичного HTML для відповідності стандартам WCAG, особливо для користувачів із порушеннями зору, що є ключовим нефункціональним вимогам. Tripathi M. та ін. [4] розглядають вебкомпоненти в контексті фреймворків, підкреслюючи їхню агностичність до технологій, що дозволяє інтеграцію з React, Angular чи Vue.js, сприяючи сумісності на етапі розробки та розгортання. Attri V. K. та ін. [5] вказують на складність освоєння вебкомпонентів розробниками, зокрема через необхідність розуміння стандартів W3C і тестування Shadow DOM, що ускладнює етап тестування життєвого циклу ПЗ. Shah H. [6] акцентує на ролі HTML5 Customized Built-in Elements у вдосконаленні компонентно-орієнтованої розробки, що сприяє повторному використанню коду та спрощує підтримку. Gu Y. [7] досліджує оптимізацію продуктивності фронтенду через модульний рендеринг і адаптивну гідратацію, які зменшують затримки рендерингу вебкомпонентів, що є критичним для нефункціональної вимоги продуктивності. Visconti E. та ін. [8] аналізують автоматизований моніторинг веб-інтерфейсів, пропонуючи інструменти для тестування вебкомпонентів, що підвищують надійність на етапі тестування. Monteiro M. та ін. [9] досліджують інтеграцію вебкомпонентів із no-code платформами, що спрощує розгортання та підтримку додатків, сприяючи автоматизації життєвого циклу ПЗ. Rao N. S. [10] підкреслює роль WebAssembly у підвищенні продуктивності вебкомпонентів шляхом прискорення ресурсоємних операцій, таких як рендеринг анімацій, що доповнює їхню модульність. Водночас питання ефективного тестування Shadow DOM, інтеграції з DevOps-практиками та забезпечення нефункціональних вимог, таких як продуктивність і доступність, залишаються недостатньо дослідженими, що визначає актуальність цього аналізу.

Формулювання мети дослідження

Метою дослідження є аналіз ролі вебкомпонентів у побудові сучасних веб-інтерфейсів з акцентом на їхній вплив на життєвий цикл ПЗ, виконання функціональних (модульність, сумісність) і нефункціональних вимог (продуктивність, доступність, тестованість) та розробка стратегій подолання обмежень для забезпечення якості веб-додатків.

Викладення основного матеріалу дослідження

Вебкомпоненти, як технологія, заснована на стандартах W3C, включають Custom Elements для створення власних HTML-тегів із кастомною логікою, Shadow DOM для інкапсуляції стилів і поведінки, HTML Templates для визначення повторно використовуваних фрагментів розмітки та ES Modules для стандартизації імпорту JavaScript-коду [1, 4]. Ці технології дозволяють створювати ізольовані, модульні елементи інтерфейсу, які спрощують життєвий цикл ПЗ. На етапі проєктування вебкомпоненти сприяють формуванню компонентно-орієнтованої архітектури, де кожен елемент інтерфейсу, наприклад кнопка чи форма, чітко відповідає визначеним функціональним вимогам, що зменшує складність проєктування великих систем [6]. У процесі розробки модульність і повторне використання скорочують обсяг коду, оскільки компоненти, такі як віджети чи навігаційні панелі, можуть бути використані в різних частинах додатка чи навіть у різних проєктах [1, 4]. Інкапсуляція через Shadow DOM ізолює стилі та скрипти компонента від глобального контексту, запобігаючи конфліктам у CSS чи JavaScript, що особливо важливо в проєктах із розподіленими командами [2, 6]. Сумісність вебкомпонентів із фреймворками, такими як React, Angular чи Vue.js, забезпечує гнучкість вибору технологій, дозволяючи інтегрувати їх у гетерогенні системи без значних модифікацій [4, 7]. Наприклад, вебкомпонент може бути імпортований у React-додаток через обгортку, що використовує ref для доступу до методів компонента, забезпечуючи безшовну інтеграцію [7].

На етапі тестування вебкомпоненти створюють як переваги, так і виклики. Ізольованість компонентів спрощує модульне тестування, оскільки кожен компонент можна тестувати незалежно від інших частин додатка [8]. Проте Shadow DOM ускладнює доступ до внутрішньої структури компонента під час юніт-тестування, що вимагає використання спеціалізованих інструментів, таких як Testing Library або Playwright, які дозволяють проникати в Shadow DOM через API, наприклад, shadowRoot.querySelector [5, 8]. Автоматизоване тестування в CI/CD-пайплайнах, наприклад із використанням GitHub Actions чи Jenkins, може включати Puppeteer для симуляції взаємодії користувача з компонентами, що підвищує надійність тестів, але збільшує витрати часу на їх налаштування [8]. Для функціонального тестування критично важливо перевіряти відповідність компонентів специфікаціям, наприклад, чи коректно відображається форма з валідатором у різних браузерах [4]. Нефункціональні вимоги, такі як продуктивність, також потребують уваги. Неефективна організація вебкомпонентів, наприклад надмірне вкладення Shadow DOM, може призвести до затримок під час клієнтської гідратації, особливо в додатках із великою кількістю динамічного контенту [7]. Для оптимізації продуктивності пропонується використовувати модульний рендеринг, який розбиває рендеринг на менші асинхронні завдання, або інтеграцію з WebAssembly для прискорення обчислень у браузері [10]. Наприклад, WebAssembly може бути використаний для обробки складних операцій у компонентах, таких як рендеринг графіків чи анімацій, що зменшує навантаження на JavaScript-движок [10].

Доступність є ще однією нефункціональною вимогою, яка потребує ретельного підходу. Вебкомпоненти повинні відповідати стандартам WCAG, включаючи використання ARIA-атрибутів (наприклад, aria-label, aria-hidden) і семантичного HTML для забезпечення сумісності з допоміжними технологіями, такими як екранні читачі [3].

Наприклад, компонент навігаційної панелі має містити атрибути role=»navigation» і aria-label для чіткого позначення його призначення. Недостатня увага до доступності може призвести до виключення користувачів із порушеннями зору чи моторики [3]. Щодо SEO, динамічний контент, створений вебкомпонентами, може ускладнювати індексацію пошуковими системами, якщо не використовується серверний рендеринг (SSR) або статична генерація сайту (SSG) [2]. Наприклад, фреймворки, такі як Next.js, дозволяють генерувати статичний HTML для вебкомпонентів, що покращує SEO [7]. Підтримка застарілих браузерів, таких як Internet Explorer, вимагає використання поліфілів, наприклад @webcomponents/webcomponentsjs, що додає накладні витрати на продуктивність і ускладнює розгортання [5]. На етапі розгортання стандартизація вебкомпонентів полегшує інтеграцію в CI/CD-пайплайни, оскільки компоненти є автономними модулями, які можна окремо оновлювати чи масштабувати [9]. На етапі підтримки модульність вебкомпонентів спрощує внесення змін, наприклад оновлення стилів чи логіки компонента без впливу на інші частини додатка [4].

Для наочної оцінки переваг і обмежень вебкомпонентів порівняно з традиційними підходами до розробки інтерфейсів, такими як монолітні фреймворки чи бібліотеки без стандартизації, нижче наведено таблицю, яка аналізує ключові критерії. Таблиця 1 ілюструє, як вебкомпоненти відповідають функціональним і нефункціональним вимогам, включаючи модульність, тестованість і продуктивність.

Таблиця 1

Порівняння вебкомпонентів і традиційних підходів до розробки інтерфейсів

Критерій	Вебкомпоненти	Традиційні підходи
Модульність	Висока (інкапсуляція через Shadow DOM)	Середня (залежить від фреймворку)
Повторне використання	Високе (незалежність від фреймворків)	Обмежене (залежить від архітектури)
Тестування	Складне (Shadow DOM вимагає спеціалізованих інструментів)	Простіше (стандартні інструменти)
Продуктивність	Залежить від оптимізації (можливі затримки через гідратацію)	Залежить від фреймворку
Доступність	Вимагає додаткових зусиль (ARIA), семантичного HTML	Залежить від реалізації

Таблиця 1 демонструє, що вебкомпоненти переважають за модульністю та повторним використанням завдяки стандартам W3C, але потребують додаткових зусиль для тестування та забезпечення доступності. Традиційні підходи, такі як використання React без вебкомпонентів, можуть бути простішими в тестуванні, але менш гнучкими в інтеграції з іншими технологіями.

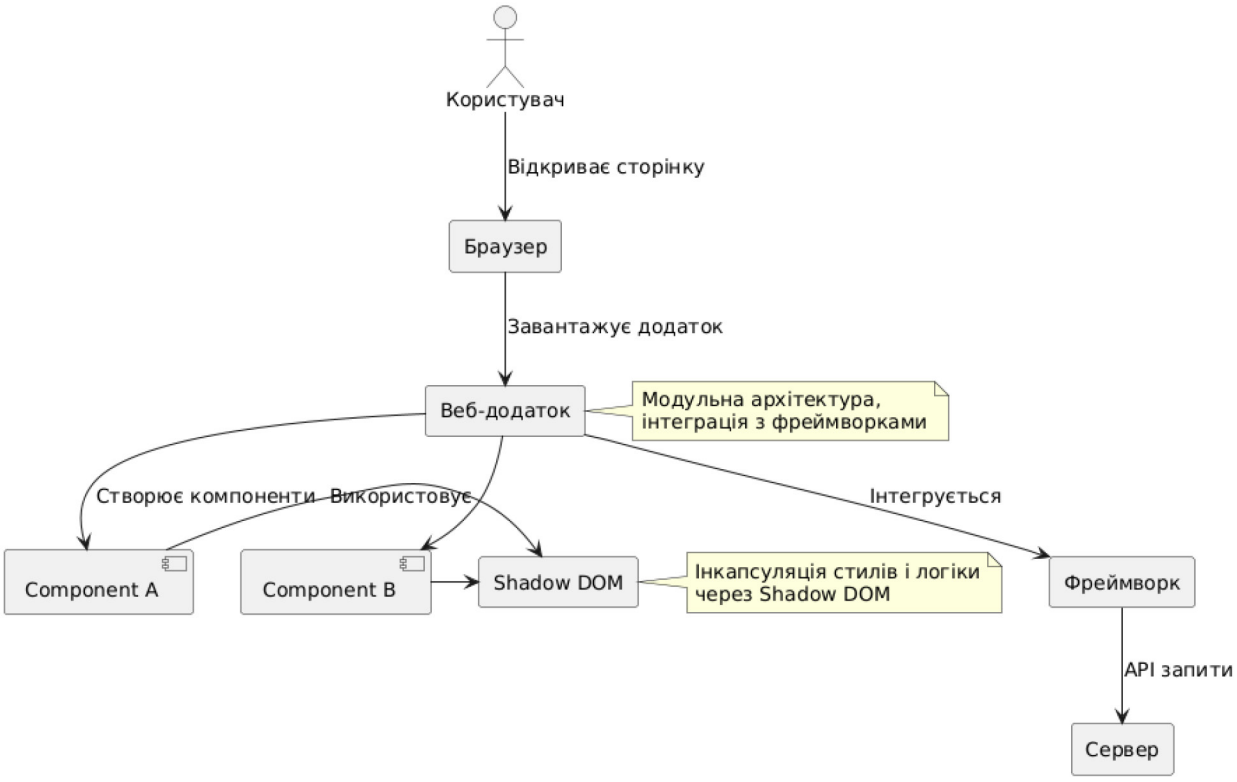

Рис. 1. Архітектура веб-додатка з використанням вебкомпонентів

Джерело: авторська розробка в PlantUML online editor

Для ілюстрації взаємодії вебкомпонентів у структурі веб-додатка нижче наведено діаграму, створену за допомогою PlantUML. Діаграма відображає, як користувач взаємодіє з браузером, який обробляє веб-додаток, що містить вебкомпоненти з ізольованими Shadow DOM. Компоненти інтегруються з фреймворком, який взаємодіє з сервером для рендерингу чи отримання даних, підкреслюючи модульність і інкапсуляцію.

Рисунок 1 підкреслює ізольованість вебкомпонентів через Shadow DOM, що забезпечує незалежність стилів і логіки, а також їхню інтеграцію з фреймворками та сервером. Наприклад, ComponentA може бути кнопкою з власним стилем у Shadow DOM, яка викликає API-запит через фреймворк, не впливаючи на ComponentB, що демонструє модульність.

Стратегії подолання обмежень вебкомпонентів включають технічні підходи до забезпечення якості. Для тестування рекомендується використовувати інструменти, такі як Playwright або Testing Library, які дозволяють взаємодіяти з Shadow DOM через методи, як shadowRoot.querySelector, а також інтегрувати тести в CI/CD-пайплайни для автоматизації [8]. Для підвищення продуктивності пропонується застосовувати модульний рендеринг, який розбиває рендеринг на асинхронні завдання, або WebAssembly для обробки ресурсоємних операцій, таких як рендеринг анімацій [7, 10]. Доступність забезпечується впровадженням ARIA-атрибутів і семантичного HTML на етапі проєктування, а також використанням автоматизованих інструментів, таких як axe-core, для перевірки відповідності WCAG [3]. Для SEO рекомендується застосовувати SSR або SSG, наприклад через фреймворк Next.js, що генерує статичний HTML для компонентів [7]. Підтримка застарілих браузерів досягається за допомогою поліфілів, таких як @webcomponents/webcomponentsjs, хоча це може знизити продуктивність [5].

Висновки

Вебкомпоненти є ключовою технологією для побудови сучасних веб-інтерфейсів, забезпечуючи модульність, інкапсуляцію та сумісність із різними фреймворками. Вони спрощують життєвий цикл ПЗ, зменшуючи обсяг коду на етапі розробки, полегшуючи оновлення на етапі підтримки та сприяючи стандартизації на етапі розгортання. Основні переваги включають повторне використання компонентів, ізоляцію стилів і логіки через Shadow DOM, а також агностичність до технологій. Обмеження, такі як складність тестування Shadow DOM, проблеми з продуктивністю через неефективну гідратацію, вимоги до доступності та підтримка застарілих браузерів, потребують ретельного планування. Стратегії подолання цих обмежень включають використання спеціалізованих інструментів тестування, оптимізацію продуктивності через модульний рендеринг і WebAssembly, впровадження стандартів WCAG для доступності та SSR для SEO. У 2025 році вебкомпоненти продовжують еволюціонувати, інтегруючись із WebAssembly і автоматизованими інструментами моніторингу, що забезпечує їхню адаптивність до нових вимог веб-розробки, включаючи інтеграцію з Web 3.0 і AI-технологіями.

Список використаної літератури

1. Козуб Г. О., Козуб Ю. Г. Декларативний підхід при створенні мультиплатформних додатків. *Вісник Східноукраїнського національного університету імені Володимира Даля*. № 5(275). 2022. С. 10–15. DOI: https://doi.org/10.33216/1998-7927-2022-275-5-10-15

2. Gorai S. K. Web Development Barriers and Challenges // International journal of scientific research in engineering and management. 2025. Vol. 09, iss. 01. P. 1–9. DOI: https://doi.org/10.55041/ijsrem40734 URL: https://ijsrem.com/download/web-development-barriers-and-challenges/

3. Maulana I. Web Accessibility: Designing User-Friendly Websites for Individuals with Visual Impairments // Golden Ratio of Data in Summary. 2025. Vol. 5, iss. 1. P. 14–19. DOI: https://doi.org/10.52970/grdis.v5i1.896 URL: https://goldenratio.id/index.php/grdis/article/view/896

4. Tripathi M., Jain D., Sharma K., Anuradha, Daulatram. Web Development Framework // International Journal of Advanced Research in Science, Communication and Technology. 2024. P. 97–100. DOI: https://doi.org/10.48175/ijarsct-22615 URL: https://ijarsct.co.in/Paper22615.pdf

5. Attri V. K., Pathania P., Simran. Challenges of web development // World Journal of Advanced Engineering Technology and Sciences. 2025. Vol. 14, iss. 1. P. 199–201. DOI: https://doi.org/10.30574/wjaets.2025.14.1.0029 URL: https://journalwjaets.com/node/286

6. Shah H. Advancing Web Development – Enhancing Component-Based Software Engineering and Design Systems Through HTML5 Customized Built-in Elements // International journal of Web & Semantic Technology. 2024. Vol. 15, iss. 1. P. 15–26. DOI: 10.5121/ijwest.2024.15102. URL: https://www.researchgate.net/publication/377564087_Advancing_Web_Development_-_Enhancing_Component-Based_Software_Engineering_and_Design_Systems_through_HTML5_Customized_Built-in_Elements

7. Gu Y. Practical Approaches to Developing High-performance Web Applications Based on React // Frontiers in Science and Engineering. 2025. Vol. 5, iss. 2. P. 99–105. DOI: https://doi.org/10.54691/5ccss512 URL: https://fse-journal.org/index.php/ojs/article/view/22

8. Visconti E., Tsigkanos C., Nenzi L. Automated Monitoring of Web User Interfaces // ACM Transactions on the Web. 2025. Vol. 19, iss. 2. P. 1–27. DOI: https://doi.org/10.1145/3708512 URL: https://dl.acm.org/doi/10.1145/3708512

9. Monteiro M., Branco B. C., Silvestre S., Avelino G., Valente M. T. NoCodeGPT: A No-Code Interface for Building Web Apps With Language Models // Software: Practice and Experience. 2025. DOI: https://doi.org/10.1002/spe.3432 URL: https://onlinelibrary.wiley.com/doi/10.1002/spe.3432

10. Rao N. S. WebAssembly: Revolutionizing Web User Interface Development through Performance and Cross-Language Integration // International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2024. Vol. 10, iss. 6. P. 1973–1981. DOI: https://doi.org/10.32628/cseit241061235 URL: https://ijsrcseit.com/index.php/home/article/view/CSEIT241061235

References

1. Kozub H. O., Kozub Yu. H. (2022). Deklaratyvnyi pidkhid pry stvorenni multyplatformnykh dodatkiv [Declarative approach in creating multiplatform applications]. *Visnyk Skhidnoukrainskoho natsionalnoho universytetu imeni Volodymyra Dalia – Bulletin of Volodymyr Dahl East Ukrainian National University*, no. 5 (275), pp. 10–15. DOI: https://doi.org/10.33216/1998-7927-2022-275-5-10-15. [in Ukrainian].

2. Gorai S. K. (2025). Web Development Barriers and Challenges. *International journal of scientific research in engineering and management*, no. 09(01), pp. 1–9. DOI: https://doi.org/10.55041/ijsrem40734 Retrieved from https://ijsrem.com/download/web-development-barriers-and-challenges/ [in English].

3. Maulana I. (2025). Web Accessibility: Designing User-Friendly Websites for Individuals with Visual Impairments. *Golden Ratio of Data in Summary*, no. 5(1), pp. 14–19. DOI: https://doi.org/10.52970/grdis.v5i1.896 Retrieved from https://goldenratio.id/index.php/grdis/article/view/896 [in English].

4. Ms. Monika Tripathi, Ms. Dimpal Jain, Ms. Khushboo Sharma, Ms. Anuradha, & Mr. Daulatram. (2024). Web Development Framework. *International Journal of Advanced Research in Science, Communication and Technology*, pp. 97–100. DOI: https://doi.org/10.48175/ijarsct-22615 Retrieved from https://ijarsct.co.in/Paper22615.pdf [in English].

5. Varinder Kaur Attri, Purva Pathania, & Simran. (2025). Challenges of web development. *World Journal of Advanced Engineering Technology and Sciences*, no. 14(1), pp. 199–201. DOI: https://doi.org/10.30574/wjaets.2025.14.1.0029 Retrieved from https://journalwjaets.com/node/286 [in English].

6. Shah, Hardik. (2024). Advancing Web Development – Enhancing Component-Based Software Engineering and Design Systems Through HTML5 Customized Built-in Elements. International journal of Web & Semantic Technology, no. 15. DOI: 10.5121/ijwest.2024.15102. Retrieved from https://www.researchgate.net/publication/377564087_Advancing_Web_Development_-_Enhancing_Component-Based_Software_Engineering_and_Design_Systems_through_HTML5_Customized_Built-in_Elements [in English].

7. Gu Y. (2025). Practical Approaches to DevelopingHigh-performance Web Applications Based on React. *Frontiers in Science and Engineering*, no. 5(2), pp. 99–105. DOI: https://doi.org/10.54691/5ccss512 Retrieved from https://fse-journal.org/index.php/ojs/article/view/22 [in English].

8. Visconti E., Tsigkanos C., & Nenzi L. (2025). Automated Monitoring of Web User Interfaces. *ACM Transactions on the Web*, no. 19(2), pp. 1–27. DOI: https://doi.org/10.1145/3708512 Retrieved from https://dl.acm.org/doi/10.1145/3708512 [in English].

9. Monteiro M., Branco B. C., Silvestre S., Avelino G., & Valente M. T. (2025). NoCodeGPT: A No-Code Interface for Building Web Apps With Language Models. *Software: Practice and Experience*. DOI: https://doi.org/10.1002/spe.3432 Retrieved from https://onlinelibrary.wiley.com/doi/10.1002/spe.3432 [in English].

10. Nikhil Sripathi Rao. (2024). WebAssembly: Revolutionizing Web User Interface Development through Performance and Cross-Language Integration. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, no. 10(6), pp. 1973–1981. DOI: https://doi.org/10.32628/cseit241061235 Retrieved from https://ijsrcseit.com/index.php/home/article/view/CSEIT241061235 [in English].