УДК 004.514
DOI https://doi.org/10.32782/2521-6643-2025-1-69.11

Трофименко О. Г., кандидат технічних наук, доцент, доцент кафедри інформаційних технологій
Національного університету «Одеська юридична академія»
ORCID: 0000-0001-7626-0886

Манаков С. Ю., кандидат технічних наук, доцент,
доцент кафедри інформаційних технологій
Національного університету «Одеська юридична академія»
ORCID: 0000-0001-5930-4592

Корнійчук М. М., студент факультету кібербезпеки
та інформаційних технологій
Національного університету «Одеська юридична академія»
ORCID: 0009-0009-4223-9291

Лобода Ю. Г., кандидат педагогічних наук, доцент, доцент кафедри інформаційних технологій
Національного університету «Одеська юридична академія»
ORCID: 0000-0001-7083-552X

Чикунов П. О., кандидат технічних наук, доцент,
доцент кафедри інформаційних технологій
Національного університету «Одеська юридична академія»
ORCID: 0000-0003-4959-7744

МОДАЛЬНІ ВІКНА В ІНТЕРФЕЙСІ КОРИСТУВАЧА ЗАСОБАМИ REACT/NEXT.JS

Модальні вікна стали повсюдним елементом цифрових інтерфейсів для відображення сповіщень, збору введених користувачем даних, керування навігацією. Метою статті є аналіз та систематизація сучасних підходів до реалізації модальних вікон у вебзастосунках, створених засобами React та Next.js. У статті проведено комплексний аналіз сучасних підходів до реалізації модальних вікон у frontend розробці. По-перше, досліджено роль модальних вікон у покращенні взаємодії користувачів у вебзастосунках. Попри їх широке використання є певна неоднозначність щодо того, коли використовувати модальні вікна, а коли альтернативні рішення – спливаючі діалогові вікна. Автори провели детальне порівняння цих компонентів інтерфейсу, різні варіанти їх використання, переваги та потенційні недоліки. По-друге, в статті, розглянуто різні проблеми при розробленні функціональних модальних вікон в інтерфейсі користувача. Зокрема це стосується проблем доступності, продуктивності та ефективності, інтеграції модальних вікон з маршрутизацією, проблем з мобільною версією, переривання поточної взаємодії, проблем з розміткою та стилями. По-третє, у роботі досліджено питання тестування модальних вікон у динамічних вебсередовищах. Традиційні засоби автоматизованого тестування часто мають проблеми з модальною взаємодією через їх тимчасовий характер і залежність від подій, ініційованих користувачем. У статті обговорюється те, як штучний інтелект (ШІ) здатен допомогти розв'язати ці складнощі, дозволяючи складний аналіз модальної поведінки, включно з динамічною адаптацією вмісту та реагування в режимі реального часу. Завдяки інтеграції ефективних методів доступності, оптимізації продуктивності і використанню розширеного тестування за допомогою ШІ, розробники можуть створювати інтуїтивно зрозумілі та ефективні інтерфейси, які покращують, а не перешкоджають взаємодії користувачів. Проведений аналіз може бути цінним як для новачків, так і для досвідчених розробників інтерфейсу, які прагнуть удосконалити свій підхід до модального дизайну в сучасних вебзастосунках.

Ключові слова: *модальне вікно, вебпрограмування, вебдизайн, інтерфейс, тестування, спливаюче вікно.*

Trofymenko O. G., Manakov S. Yu., Korniichuk M. M., Loboda Yu. V., Chykunov P. O. Modal Windows in the in UI with React/Next.js

Modal windows are a ubiquitous element of digital interfaces for displaying notifications, collecting user input, and controlling navigation. The purpose of the article is to analyze and systematize modern approaches to implementing modal windows in web applications created using React and Next.js. The article provides a comprehensive analysis of modern approaches to

implementing modal windows in frontend development. First, the role of modal windows in improving user interaction in web applications is investigated. Despite their widespread use, there is some ambiguity regarding when to use modal windows and when alternative solutions are pop-up dialog boxes. The authors conduct a detailed comparison of these interface components, their different uses, advantages, and potential disadvantages. Secondly, the article discusses various problems in developing functional modal windows in the user interface. This concerns accessibility, performance, and efficiency issues, integration of modal windows with routing, mobile issues, interruption of ongoing interaction, issues with markup and styles. Systematizing approaches to implementing these components allows you to ensure interface quality and increase the scalability and maintainability of web applications in the long term. Third, the paper explores the issue of testing modal windows in dynamic web environments. Traditional automated testing tools often have problems with modal interactions due to their temporal nature and dependence on user-initiated events. The paper discusses how artificial intelligence (AI) can help address these challenges by enabling sophisticated analysis of modal behavior, including dynamic adaptation of content and responsiveness in real time. AI-based testing frameworks can simulate complex user interactions, detect accessibility issues, and predict potential UX issues before deployment. By integrating effective accessibility practices, optimizing performance, and using advanced AI-powered testing, developers can create intuitive and effective interfaces that enhance, rather than hinder, user interactions. The analysis can be valuable for both novice and experienced interface developers looking to refine their approach to modal design in modern web applications.

Key words: *modal window, web programming, web design, interface, testing, pop-up window.*

Постановка проблеми. Модальне вікно є важливим елементом більшості вебзастосунків, оскільки його призначення полягає у фокусуванні уваги користувача на певному об'єкті, що містить важливу інформацію або спонукає до дії, яку необхідно виконати перед тим, як продовжити роботу з основним контентом вебресурсу.

Актуальність глибоко аналізу підходів до реалізації модальних вікон у вебзастосунках на React і Next.js зумовлена поширеним використанням цих технологій у сучасній фронтенд-розробці. React є однією з найпопулярніших бібліотек для створення інтерфейсів користувача, а Next.js – потужним фреймворком, який забезпечує серверний рендеринг, генерацію статичних сторінок та оптимізацію продуктивності.

При розробленні складних, динамічних вебінтерфейсів модальні вікна є одним з ключових елементів взаємодії з користувачем, тому правильна їх реалізація має велике значення. Сучасні вебзастосунки часто потребують гнучких, ефективних способів створення модальних вікон. Такі способи можуть включати використання порталів, управління станом через хуки або контекст, а також інтеграцію з UI-бібліотеками. Крім того, реалізація таких вікон вимагає дотримання принципів доступності, зокрема правильного управління фокусом, використання ARIA-атрибутів і забезпечення навігації з клавіатури. Усе це безпосередньо впливає на користувацький досвід.

Ще одним важливим аспектом є вплив модальних вікон на продуктивність і рендеринг. Враховуючи специфіку підходів SSR (Server Side Rendering) і CSR (Client Side Rendering) у Next.js, неправильна реалізація може спричинити проблеми з продуктивністю, завантаженням сторінки або гідратацією компонентів. Особливої уваги потребує інтеграція компонентів модальних вікон у системи маршрутизації (вбудовані, як у Next.js або кастомні). Зокрема передбачення способів керування історією та поточним станом URL у випадках, коли відкриття і закриття модального вікна прив'язане до навігації.

Аналіз останніх досліджень та публікацій. Оскільки з потребою розробки модальних та спливаючих вікон розробники інтерфейсу вебзастосунків стикаються доволі часто, то й актуальність добору ефективної реалізації відповідних елементів взаємодії з користувачем підтверджується різнобічними дослідженнями у цій сфері. Багато дизайнерів інтерфейсу користувача (User Interface, UI) обговорюють принципи, яких слід дотримуватися при розробленні функціональних аспектів інтерфейсу користувача [1]. Збір даних через вебформи та діалогові вікна є одним з основних способів взаємодії з користувачами на вебсайті. Щоб користувачі довіряли діалоговим вікнам та вебформам, вони мають відповідати функціональним потребам, нормам конфіденційності та юридичними зобов'язанням [2]. У роботі розглянуто чотири методи створення модальних та спливаючих вікон як елементів інтерфейсу користувача [3]. Дослідження [4] спрямоване на виявлення відмінностей між спливаючими сповіщеннями та повідомленнями у вигляді форм щодо часу їх виконання. Суб'єктивні оцінки обох методів не виявили суттєвої переваги жодного з методів. У роботі [5] звертають увагу на складнощі тестування спливаючих вікон в інтерфейсі вебзастосунків засобами Selenium. При тестуванні вебелементів виконуються численні перевірки візуальних і структурних частин інтерфейсу користувача на відповідність вимогам зручності використання, доступності, продуктивності та стандартам, що створює проблеми для автоматизованого тестування [6]. Суттєвим є те, що скрипти для тестування графічного інтерфейсу (GUI) є малопридатними для перевірки юзабіліті через проблеми налагодження тестового сценарію і не можуть бути пакетними або об'єднаними в систему [7]. Крім того, автоматизовані функціональні тести не завжди є надійними, оскільки тестові скрипти можуть не виявляти помилки, які не прописані в сценарії перевірки. До таких недоліків належать, зокрема, неточне позиціювання модальних та спливаючих вікон, помилки у тексті повідомлень, дефекти форм, з якими скрипт не взаємодіє під час виконання [8]. Тому подальші дослідження підходів до розробки та тестування модальних вікон засобами

React і Next.js дозволять знайти ефективні рішення, які забезпечать якісну взаємодію з користувачем, відповідатимуть сучасним вимогам до продуктивності й доступності, та є гнучкими для подальшої підтримки й масштабування вебзастосунків.

Мета статті – аналіз та систематизація сучасних підходів до реалізації модальних вікон у вебзастосунках, створених засобами React та Next.js.

Виклад основного матеріалу. Розробка застосунків для сучасного Інтернету охоплює цілу низку складних інженерних процесів, серед яких важливу роль відіграє проєктування інтерфейсів користувача, які поєднують взаємодію інтерактивних, динамічних компонентів. Одним із важливих елементів інтерфейсів є модальні вікна, які активно використовуються для взаємодії з користувачем вебсторінки.

1. Відмінності модальних вікон (modals) від спливаючих (pop-ups)

Модальні вікна дещо схожі на спливаючі діалогові вікна у вебзастосунках, але вони виконують різні функції і мають суттєві відмінності в тому, як вони поводяться і як реалізуються (табл. 1).

Таблиця 1

Основні відмінності між модальними і спливаючими діалоговими вікнами

Характеристика	Модальне вікно (modal)	Спливаюче вікно (pop-up)
Взаємодія з інтерфейсом	Блокує доступ до основного контенту, поки користувач не взаємодіє з вікном	Не блокує основний контент, користувач може продовжувати роботу
Місце відображення	Відображається поверх основного контенту вебсайту чи застосунку	Може відкрити нову вкладку або з'явитися поверх контенту
Тип контенту	Використовуються для важливих повідомлень або форм, які потребують взаємодії	Може містити повідомлення, рекламу або інші менш важливі елементи
Приклад використання	Повідомлення про помилки, підтвердження дій, форми для введення даних	Легкі повідомлення або сповіщення про оновлення, реклама
Застосування в React	Зазвичай використовуються компоненти з умовним рендерингом для контролю видимості	Може бути реалізоване через прості компоненти або використання вікон браузера

Голова відмінність між спливаючими діалоговими і модальними вікнами полягає в тому, як саме відповідне вікно дозволяє користувачеві взаємодіяти з документом, коли воно відкрите. Так спливаюче вікно (pop-up) при появі на екрані дозволяє користувачеві взаємодіяти з іншим контентом сторінки, коли воно відкрите (тобто натискати кнопки та посилання в зоні доступності навколо відповідного діалогового вікна. Модальне вікно (modal) блокує сторінку, доки користувач не виконає запропоновану дію (рис. 1). Тим самим воно зменшує кількість інформації, яку користувачі мають обробити за раз, і зосереджує його увагу на найважливішому. Це може бути підтвердження покупки або підписка на розсилку – у будь-якому разі ці елементи призводять до конкретних дій.

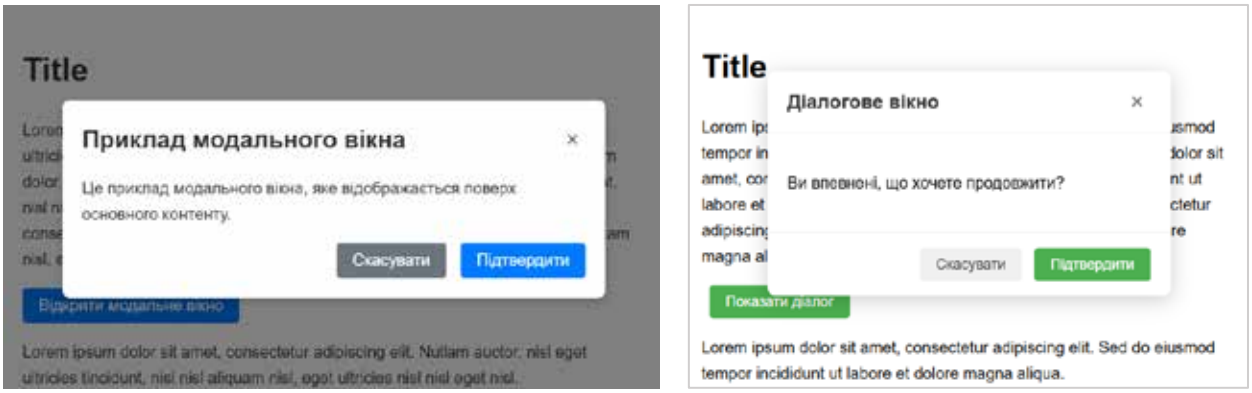

Рис. 1. Різниця між відображенням на вебсторінці модального (ліворуч) та спливаючого (праворуч) діалогових вікон

Хоча подекуди інструменти pop-ups і modals вважають схожими, але по суті вони зовсім різні. Спливаючі вікна – це окремі вікна, які відкриваються в браузері. Завдяки вбудованим таймерам вони можуть автоматично зникати через кілька секунд, якщо користувач не взаємодіє, мінімізуючи збої. Серед різних типів деякі спливаючі вікна можна відфільтрувати за допомогою спеціальних блокувальників реклами. Водночас системні спливаючі вікна є ефективними для поінформування користувачів про успішність чи

невдачу виконаних дій. На відміну від pop-up, модальні вікна є частиною того самого вікна браузера, затемнюючи і блокуючи фоновий вміст вебсторінки на кшталт певної накладки. Наприклад, за допомогою модальних вікон можна запропонувати ввести певну реєстраційну інформацію, прийняти умови або повідомити про новий функціонал застосунку. Часто такі вікна використовують для реєстрації нових користувачів, проводячи їх через процес реєстрації, при цьому залишаючись по суті на головній сторінці [9]. Наприклад, LinkedIn використовує modals, щоб спонукати людей заповнювати свої профілі або підтримувати зв'язки [10].

Діалоги та модальні вікна стали звичайною частиною Інтернету, але подекуди розробникам, особливо початківцям, не зрозуміло, який із них підходить для певної ситуації краще. Розуміння відмінностей між pop-up та modal, дозволяє зробити правильний вибір найкращого варіанта для певної ситуації та уникнути негараздів в інтерфейсі. Позаяк модальні вікна з'являються поверх основного вмісту вебсайту або застосунку, блокуючи доступ до решти інтерфейсу на час виконання певної дії користувачем (наприклад, натискання кнопки «Закрити», «Підтвердити» чи то іншої дії), то використовувати їх варто для важливих взаємодій, які потребують негайної уваги користувача, наприклад, для підтвердження важливої дії при реєстрації, видаленні файлів, виходу із системи тощо. Тобто модальне вікно зазвичай вимагає активної взаємодії з користувачем перед тим, як він повернеться до основного контенту. Спливаючі вікна доцільні для надання додаткової інформації або ненав'язливих сповіщень, наприклад: для повідомлень про оновлення, підказок, реклами. Дотримання кращих практик UX-дизайну компонентів робить вебзастосунок більш зручним і привабливим для користувачів. В будь-якому варіанті організації модального або спливаючого вікна не варто перевантажувати користувача занадто великою кількістю варіантів в одному діалоговому вікні.

2. Проблеми програмної реалізації модальних вікон у вебзастосунках

Модальні вікна у React та Next.js потребують ретельно продуманого підходу до реалізації. Одним із базових аспектів є використання механізмів умовного рендерингу та управління станом, що дозволяє динамічно керувати їх відображенням. У React це досягається за допомогою хуків, зокрема useState або useReducer, а в складніших випадках – через глобальні стейт-менеджери. У Next.js важливо враховувати специфіку серверного рендерингу, що може вплинути на рендеринг модального компонента, зокрема на етапі гідратації.

2.1. Проблеми з доступністю

Важливою частиною сучасної реалізації модальних вікон є доступність (accessibility, також відома як a11y). Створюючи подібні компоненти, розробник має забезпечити відповідність стандартам WCAG (Web Content Accessibility Guidelines, доступність вебконтенту для людей із зоровими та іншими обмеженнями), правильне управління фокусом, використання ARIA-атрибутів і підтримку клавіатурної навігації [11]. Якщо модальне вікно реалізоване без врахування стандартів доступності, це може спричинити проблеми для користувачів, які користуються спеціальними пристроями, наприклад, екранними читачами (screen readers). Тобто такі користувачі не зможуть взаємодіяти з модальним вікном або з основним контентом після його закриття.

Можливими варіантами рішень проблем доступності є:

– забезпечення фокусування (focus trapping, фокус-трапінг) всередині модального вікна, коли воно відкрите, й автоматичне відновлення фокуса на попередньому елементі після його закриття;

– блокування інших елементів сторінки від доступу через screen reader, наприклад, через атрибут aria-hidden;

– підтримка коректної навігації через клавіатуру (стрілки, Escape, Tab, Tab/Shift+Tab, Enter тощо);

– використання атрибутів ARIA (Accessible Rich Internet Applications) для полегшення навігації для людей з обмеженими можливостями, наприклад: aria-modal=»true» (вказує, що вікно є модальним і взаємодія має бути зосереджена лише на ньому), aria-labelledby (зв'язування заголовка з модальним вікном), aria-describedby (пояснення змісту вікна).

Приклад використання ARIA-атрибутів:

```
div className="modal" role="dialog" aria-modal="true" aria-labelledby="modal-title">
  <h2 id="modal-title">Модальне вікно</h2>
  <p>Контент модального вікна</p>
  <button onClick={closeModal}>Закрити</button>
</div>
```

У середовищі React ці задачі можуть реалізовуватись як вручну, так і за допомогою спеціалізованих бібліотек, як-от: Headless UI, Radix UI чи Reach UI, які забезпечують семантичну точність та високий рівень доступності.

В таблиці 2 наведено результати проведеного порівняння реалізацій доступності (a11y) модальних вікон при ручній реалізації та у трьох вищезгаданих бібліотеках.

Таблиця 2

Порівняння ручної реалізації доступності модальних вікон та засобами бібліотек Headless UI, Radix UI, Reach UI

Критерій	Headless UI	Radix UI	Reach UI	Ручна реалізація
Фокус трапінг	Є, вбудований	Є, вбудований	Є, вбудований	Треба реалізовувати самому або з focus-trap
ARIA ролі / атрибути	Напівавтоматично (деякі потрібно додати вручну)	Автоматично	Автоматично	Треба вручну задати role, aria-labelledby, aria-describedby
Закриття по натисканню клавіші Esc	Так	Так	Так	Треба додати обробник подій вручну
Фокус після закриття	Повертає фокус на тригер	Повертає фокус	Повертає фокус	Треба запам'ятати активний елемент і повернути фокус вручну
aria-hidden для фону	Треба реалізувати вручну	Вбудовано	Вбудовано	Треба самому приховати інші елементи для screen reader
Семантика / відповідність (WAI-ARIA)	Залежить від реалізації користувача	Максимально семантична	Висока, орієнтована на WCAG	Можлива помилка, велика відповідальність на розробнику
Гнучкість / кастомізація	Висока, повна свобода стилізації та логіки	Висока	Обмежена, порівняно з іншими	Повний контроль розробником
Документація з a11y	Мінімальна	Докладна	Добра	Відсутня – залежність від сторонніх статей/гайдів
Розмір бібліотеки	Середній (разом з Tailwind UI)	Дуже легкий	Легкий	–
Час на розробку	Середній	Малий	Малий	Значний, багато нюансів доступності
Надійність	Висока	Висока	Висока	Варіативна, залежить від досвіду розробника

Проведений порівняльний аналіз не виявив універсального інструмента для реалізації доступності модальних вікон у вебінтерфейсі. Так, потужна бібліотека Radix UI поєднує гнучкість низькорівневих компонентів з повною реалізацією доступних модальних вікон, а тому є найкращим вибором для семантики та доступності, позаяк в цій бібліотеці компонент Dialog повністю відповідає стандарту доступності WAI-ARIA (Web Accessibility Initiative – Accessible Rich Internet Applications є технологічним стандартом від W3C для надання можливості повноцінного використання Інтернету людьми з фізичними обмеженнями органів зору та опорно-рухового апарату). Усі компоненти модального вікна в Radix вже містять усе необхідне для відповідності WAI-ARIA практикам: фокус-трапінг, закриття по Escape, aria-атрибути, автоматичне приховування фону для screen reader'ів. На відміну від Reach UI, яка також фокусується на доступності, але надає більш високорівневі компоненти з обмеженою кастомізацією, Radix дає більше контролю над DOM-структурою, стилізацією і поведінкою. У порівнянні з Headless UI, яка вимагає ручного налаштування більшості атрибутів доступності та поведінки, Radix значно полегшує впровадження правильної a11y-логіки з мінімальними зусиллями з боку розробника. Це дозволяє розробнику зосередитися на дизайні та бізнес-логіці, а не витрачати час на реалізацію дрібних, але критичних деталей доступності. Портальна структура (Dialog. Portal) розв'язує типові проблеми з layering та scroll-lock, які в Headless UI потребують ручної реалізації або додаткових бібліотек. Також Radix UI вигідно виділяє високий рівень деталізації та хороша документація.

Втім, бібліотека Radix UI має і недоліки. Так, вона не має стилів (unstyled), що добре для повного контролю, але потребує більше часу на створення візуального подання. Для новачків або швидкого прототипування це є певним бар'єром, особливо порівняно з готовими рішеннями в Reach UI. Ще один нюансом є те, що Radix орієнтована саме на React і не підтримує інші фреймворки, що дещо обмежує її використання в мультифреймворкових проєктах. Також вона активно оновлюється, а тому деякі API можуть часто змінюватися, що потребує уважності від розробника. Тому Radix UI – це вибір для тих, хто хоче мати повний контроль над UI, при цьому не турбуючись про правильність реалізації доступності. Але вона вимагає певного рівня підготовки, часу на стилізацію та розуміння структури.

Reach UI є зручною бібліотекою для тих, хто хоче швидко отримати стабільне, але менш кастомне рішення, тоді як Headless UI краще підійде тим, хто вже використовує Tailwind і готовий додавати a11y-логіку вручну, наприклад: aria-hidden, aria-labelledby, aria-describedby.

На відміну від Radix UI, яка надає гнучкі, низькорівневі примітиви без стилів, Reach UI є вужчим за функціональністю, але простішим у впровадженні інструментом. Ця бібліотека надає менш модульну структуру: один компонент Dialog інкапсулює усю логіку, що обмежує можливості його кастомізації, але водночас робить його зручним у використанні в стандартних випадках. Порівняно з Headless UI, яка є повністю unstyled і вимагає від розробника реалізувати доступність вручну або за допомогою окремих компонентів, Reach UI виглядає як золота середина – вона дозволяє швидко інтегрувати доступні діалоги без значних зусиль, зберігаючи при цьому певний контроль над стилізацією. Використання Reach UI доречне в проєктах, де важливі швидкість впровадження, стабільна підтримка доступності та простота – наприклад, у внутрішніх панелях керування, адміністративних інтерфейсах або невеликих вебзастосунках, де кастомізація UI не є пріоритетною.

Ключова відмінність ручної реалізації доступності модальних вікон полягає у тому, що вона має переваги, зумовлені можливостями повного контролю над структурою, стилями, логікою, можливостями оптимізації під свої потреби, без зайвого коду та відсутністю залежності від сторонніх бібліотек. Проте така реалізація має високий ризик пропустити важливі моменти доступності, вимагає більше часу на тестування з клавіатурою та screen reader'ом часто потребує сторонніх утиліт (focus-trap, scroll-lock, tabbable тощо).

2.2. Інтеграція модальних вікон з маршрутизацією

Ще одним аспектом, що потребує аналізу, є інтеграція модальних вікон з маршрутизацією. У Next.js із його файловою структурою роутів виникає потреба враховувати URL-стан при відкритті модального вікна. Це дозволяє не лише покращити користувацький досвід через підтримку історії переходів, а й надає змогу ділитися посиланням, яке одразу відкриває модальне вікно. Така реалізація використовує query-параметри або динамічні маршрути у поєднанні з умовним рендерингом модальних компонентів.

Особливу роль у сучасних підходах до створення модальних вікон відіграє використання React Portal, що дозволяє рендерити модальні вікна поза основним DOM-деревом компонента [12]. Це забезпечує кращу керованість накладанням елементів, ізоляцію стилів та уникнення конфліктів з батьківськими контейнерами.

NextJS з повноцінним впровадженням у версії 14 нового роутера App Router пропонує власний специфічний підхід до реалізації модальних вікон. Для цього мають використовуватися так звані паралельні та перехоплені маршрути (parallel and intercepted routes) [13]. Призначення parallel routes полягає у композиції сторінки з окремих блоків, які існують паралельно і рендеринг яких може бути оптимізовано. Intercepted routes дозволяють за допомогою внутрішніх механізмів маршрутизації відображати контент спеціального компонента, накладаючи його на контент основної сторінки, на якій відбулося перехоплення. При цьому видимий URL замінюється на URL спеціального компонента. Швидкість роботи такого підходу залежить від того, чи необхідне для рендерингу такого спеціального компонента завантаження додаткових даних, адже звертання до сервера у таких випадках може впливати на швидкодію, що однак можна подолати використанням механізмів кешування, якщо відповідні дані треба зберігати. При цьому лише перше відкриття такого модального вікна буде повільнішим, а всі наступні – швидкими. Стан модального вікна контролюється маршрутизатором, а не React-хуками типу useState – замість цього використовуються навігаційні події, зокрема натискання на спеціальний компонент посилання Link (який розширює функціональність стандартного елемента anchor) чи виклик функції router.push() (що додає новий запис у історію URL) [12].

2.3. Проблеми продуктивності та ефективності

Проблема продуктивності та ефективності при реалізації модальних вікон у front-end зазвичай виникає в контексті рендерингу, керування станом, ізоляції фокусу та блокування фону. Неправильне використання або надмірне використання модальних вікон може спричинити погіршення продуктивності, особливо на старих або технічно слабких пристроях. Щоб забезпечити плавну роботу інтерфейсу без затримок, розробники вдаються до кількох стратегій оптимізації. Для покращення продуктивності важливо оптимізувати рендеринг модальних вікон, особливо якщо вони мають складний контент (наприклад, форми з кількома елементами).

Одним із найважливіших рішень є використання порталів (React Portal), які дозволяють рендерити модальне вікно безпосередньо в DOM-дереві під <body>, тим самим уникати проблем із z-index або overflow. Це знижує навантаження на дерево компонентів і зменшує ризик повторного рендерингу батьківських вузлів.

Ще одним ефективним підходом є динамічне завантаження контенту модального вікна, тобто lazy loading або code-splitting. Замість того, щоб тримати важкий контент модального вікна постійно в DOM, його підвантажують лише при відкритті, що значно зменшує обсяг початкового рендерингу сторінки. Це корисно в проєктах, де модальне вікно містить складні форми, таблиці або інтерактивну графіку. У такому разі важливо реалізувати збереження стану або кешування, щоби уникнути повторного завантаження при кожному відкритті вікна.

Щодо анімацій, то тут важливо уникати важких CSS-анімацій, які блокують основний потік (main thread). Найкращим варіантом є використання бібліотек, які використовують GPU-оптимізовані трансформації, наприклад, Framer Motion. У складних інтерфейсах, де модальне вікно відкривається поверх великої

кількості динамічного контенту, продуктивність можна покращити шляхом попереднього приховування неактивного контенту за допомогою aria-hidden, inert або навіть від'єднання DOM-вузлів з основного потоку.

У випадках, де модальне вікно взаємодіє з великими обсягами даних, продуктивність також залежить від ефективного менеджменту стану. Використання локального стану замість глобального (наприклад, через useState замість Redux) зменшує кількість непотрібних рендерів та ізолює модальне вікно від змін у зовнішньому інтерфейсі. Ще один підхід – мемоізація та використання React.memo або useMemo для обчислень всередині модального вікна.

У складних інтерфейсах із великою кількістю одночасних модальних елементів (наприклад, підтвердження дій, багаторівневі діалоги), можна реалізувати менеджер модальних вікон – централізовану систему, яка рендерить лише активну модальну компоненту, зберігаючи інші в «сплячому» стані. Це дозволяє зменшити кількість одночасно присутніх у DOM елементів, оптимізувати обсяг пам'яті та прискорити відповідь інтерфейсу на взаємодію користувача.

Отже, ефективність реалізації модальних вікон значною мірою залежить від поєднання рішень: архітектурних, поведінкових і технічних. Ключем до високої продуктивності є поетапне завантаження, ефективне керування рендерингом, правильна ізоляція модального вікна від решти DOM, а також делегування складних завдань перевіреним бібліотекам, які враховують специфіку браузерного оточення.

2.4. Переривання поточної взаємодії

Проблема переривання поточної взаємодії при реалізації модальних вікон у front-end виникає тоді, коли користувач виконує дію (наприклад, вводить дані у формі, читає вміст, прокручує сторінку), і раптове відкриття модального вікна призводить до втрати контексту або фокусу. У середовищі React та Next.js така ситуація часто пов'язана з некоректним або занадто агресивним керуванням станом, зокрема коли модальне вікно відкривається через глобальний стан, який синхронізується з маршрутом (router.push, useSearchParams) або приходить через пропси (props) від батьківських компонентів.

Щоб уникнути такого переривання, необхідно чітко контролювати умови відкриття модального вікна і зберігати поточний стан взаємодії. У React це досягається, завдяки локальному стану, наприклад, з useState, у поєднанні з useRef, щоб зберігати фокус або дані, які користувач уже ввів [14]. У Next.js важливо уникати повного перерендерингу сторінки під час відкриття модального вікна. Доцільний спосіб реалізації модальних вікон у цьому випадку – це інтеграція модальних вікон у маршрутизацію без перезавантаження контенту сторінки.

Запобігання перериванню взаємодії – це, в першу чергу, архітектурне питання, яке вирішується за рахунок локального або частково глобального стану, router-based modals, оптимізованого рендерингу та грамотного використання ефектів для збереження контексту користувача. Це забезпечує як технічну ефективність, так і відчуття неперервного досвіду.

2.5. Проблеми з розміткою та стилями

Коли йдеться про реалізацію модальних вікон у front-end засобами React та Next.js, однією з основних проблем є правильне керування розміткою і стилями, зокрема щодо накладання елементів, ізоляції фокусу, керування фоном, а також адаптивності модальних вікон для різних типів пристроїв. Складність полягає в тому, що модальне вікно часто накладається на інші елементи інтерфейсу, і неправильно налаштовані стилі можуть спричинити численні помилки: конфлікти z-index, некоректну поведінку при прокручуванні сторінки, проблеми з доступністю.

Одним з основних підходів до розв'язання цих проблем є використання React Portal зі вставкою модального вікна безпосередньо в тег <body>, що дозволить уникнути проблем із накладанням елементів та збереженням належних стилів.

З точки зору стилів, найпоширеніший підхід полягає в тому, щоб зробити модальне вікно фіксованим відносно вікна браузера, використовуючи властивості CSS на кшталт position: fixed або position: absolute, і встановити відповідні значення для top, left, right, і bottom. Це дає змогу модальному вікну залишатися поверх іншого контенту на сторінці, навіть при прокручуванні. Важливо заборонити прокручування основної сторінки, коли модальне вікно відкрите, наприклад так:

```
useEffect(() => {
  if (isOpen) {
    document.body.style.overflow = 'hidden'; // Забороняє прокручування
  } else {
    document.body.style.overflow = ''; // Відновлює прокручування
  }
}, [isOpen]);
```

Важливо також забезпечити, щоб фон за модальним вікном був затемнений або напівпрозорий для фокусу користувача, зазвичай це досягається через застосування покращення стилів у CSS:

```
.modal-overlay {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background: rgba(0, 0, 0, 0.5); /* Темний фон */
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 1000; /* Важливий для перекриття основного контенту */
}
.modal-content {
  background: white;
  padding: 20px;
  border-radius: 8px;
  min-width: 300px;
  max-width: 500px;
}
```

Для адаптивного коригування розмірів та стилів модальних вікон на різних пристроях з різними розмірами екранів: від мобільних пристроїв до десктопів доцільно використати медіа-запити (CSS media queries):

```
@media (max-width: 768px) {
  .modal-content {
    width: 90%;
  }
}
@media (min-width: 769px) {
  .modal-content {
    width: 50%;
  }
}
```

Використовуючи розглянуті стилі, модальне вікно буде центроване, матиме темний фон для покращення видимості, відповідно адаптуватиметься до різних розмірів екрану, матиме керування прокручуванням фону та фокусом, для забезпечення кращого користувацького досвіду та доступності.

2.6. Проблеми з мобільною версією

Модальні вікна можуть некоректно відображатися на мобільних пристроях або не працювати належним чином на різних розмірах екрану. Модальне вікно може перекривати важливі елементи інтерфейсу або бути незручним для взаємодії на мобільних пристроях.

Проблема адаптації модальних вікон до мобільної версії зазвичай зводиться до обмеженого простору екрана, особливостей сенсорного управління та поведінки мобільного браузера (наприклад, появи віртуальної клавіатури або небажаного скролінгу фону). Щоб уникнути таких проблем, модальні вікна мають бути максимально гнучкими, займати весь екран або його велику частину, уникати фіксованих розмірів і коректно працювати з touch-подіями. Також важливо забезпечити, щоб після відкриття модального вікна інтерфейс не «стрибав» і не виникали конфлікти з виведенням екранної клавіатури. Такий підхід дозволяє уникати типових помилок – часткового виходу модального вікна за межі екрана, порушення фокусування на полях введення або перекривання елементів при зменшенні viewport. Він також забезпечує відповідність сучасним очікуванням користувача щодо мобільного UI.

Загалом для ефективної реалізації модальних вікон у вебзастосунках важливо враховувати різні фактори, серед яких: доступність, адаптивність, продуктивність, користувацький досвід, інтерактивність тощо. За допомогою різних підходів, серед яких використання ARIA-атрибутів, оптимізація рендерингу, управління фокусом та застосування адаптивного дизайну, можна значно покращити взаємодію користувача і забезпечити коректну роботу модальних вікон на всіх пристроях та в різних браузерах.

3. Специфіка тестування модальних вікон

Під час тестування UI/UX варто ретельно перевіряти коректність роботи модальних вікон, оскільки їхня специфіка може спричиняти проблеми, які впливають на зручність користування та цілісність взаємодії з інтерфейсом. Однією з головних проблем, що виникають під час тестування, є перевірка коректності фокусування елементів. Модальні вікна мають автоматично фокусувати курсор або навігацію на перший інтерактивний елемент всередині себе, не дозволяючи користувачеві взаємодіяти з фоновими елементами.

У разі порушення цієї логіки користувач може опинитися в ситуації, коли навігація клавіатурою або читання екраном стає заплутаним і неочікуваним [15]. Окрім того, недоопрацьована логіка закриття вікна – зокрема, відсутність можливості закриття через клавішу Esc або натискання за межами вікна – може значно погіршити користувацький досвід і викликати фрустрацію.

Ще одним аспектом є тестування адаптивності та доступності модальних вікон. У разі використання на мобільних пристроях або при зміненні розміру екрану модальне вікно має залишатися в центрі, масштабуватися відповідно до розміру екрана та зберігати логічну структуру. Тестування має враховувати різні сценарії: відкриття з клавіатури, підтримку технологій читання з екрана, коректне відображення на темній темі, а також уникнення перекриття важливих елементів інтерфейсу.

Також потребує перевірки ситуація, коли модальне вікно відкривається з помилками валідації форм або в момент критичних змін стану сторінки – це дозволяє виявити потенційні помилки синхронізації станів або недоступність певних елементів DOM.

З точки зору користувацького досвіду (User Experience, UX) модальні вікна можуть або покращити взаємодію, або суттєво її ускладнити, якщо не дотримано принципу мінімального втручання. Якщо модальне вікно не працює як органічна частина userflow: має надмірну кількість опцій або неочевидні способи закриття, це негативно впливає на загальне враження від продукту. Тому під час тестування важливо оцінити не лише технічну справність, а й логічність розміщення елементів, послідовність дій, наявність зрозумілих інструкцій, а також час, який користувач витрачає на завершення взаємодії з вікном.

Тому тестування модальних вікон у контексті UI/UX потребує комплексного підходу, який включає в себе не лише перевірку функціональності, а й глибоке розуміння логіки взаємодії користувача з інтерфейсом. Лише всебічне тестування дозволяє гарантувати, що модальні вікна не порушують загальну навігацію, не створюють когнітивного перевантаження і залишаються ефективним засобом комунікації в межах цифрового продукту.

Автоматизація тестування модальних вікон, попри переваги, має низку специфічних проблем, які можуть ускладнювати процес реалізації тестів і впливати на надійність результатів. Однією з ключових складностей є динамічність появи модальних вікон, адже вони зазвичай створюються в DOM під час виконання скриптів, а не завантажуються разом зі сторінкою. Це означає, що автоматизовані тести мають точно «впіймати момент» появи модального вікна, і якщо час очікування недостатній або неправильно налаштований, тест може завершитися невдачею. Крім того, модальні вікна можуть мати анімаційні ефекти із затримкою появи, що ще більше ускладнює синхронізацію в тестах.

Ще однією поширеною проблемою є конфлікти позиціювання на екрані (z-index) та фокусу: якщо модальне вікно не перекриває коректно всі елементи фону, або якщо елементи поза модальним вікном залишаються активними, це може ввести тестові фреймворки в оману й змусити їх взаємодіяти не з тими елементами. В таких випадках автоматизовані інструменти можуть втратити фокус або неправильно інтерпретувати структуру DOM, що призводить до хибнопозитивних або хибнонегативних результатів.

Крім технічних складностей, проблеми виникають і при тестуванні з точки зору UX. Автоматизовані фреймворки переважно орієнтуються на перевірку функціональності, але не завжди здатні оцінити логічність або зручність інтерфейсу. Наприклад, чи очевидно для користувача, як закрити модальне вікно, чи інтуїтивно зрозуміле його розміщення – ці аспекти залишаються поза межами традиційних автоматизованих тестів.

Що стосується використання штучного інтелекту для тестування модальних вікон, то тут відкриваються нові перспективи. ШІ може бути застосований для розпізнавання шаблонів поведінки користувачів, виявлення аномалій у поведінці інтерфейсу та прогнозування помилок. Наприклад, моделі комп'ютерного зору, інтегровані з інструментами автоматизації, можуть «бачити» інтерфейс так, як його бачить користувач, і виявляти проблеми у візуальному відображенні модального вікна – зокрема, його зміщення, перекривання, неправильну адаптацію тощо. ШІ також може допомагати в генерації тестових сценаріїв, автоматично ідентифікуючи модальні компоненти в DOM і формуючи тест-кейси для взаємодії з ними.

Ба більше, завдяки інтеграції з нейромережами або алгоритмами машинного навчання можна автоматично пріоритезувати тестування найбільш критичних модальних вікон або сценаріїв, що найчастіше викликають помилки. Такі підходи не замінюють класичну автоматизацію, але суттєво покращують її гнучкість та ефективність.

Висновки. Реалізація модальних вікон у середовищі React та Next.js – це не лише технічне завдання, а комплексна інженерна проблема, яка поєднує вимоги до UX, доступності, архітектури та продуктивності. В статті, розглянуто різні проблеми при розробленні функціональних модальних вікон в інтерфейсі користувача. Зокрема це стосується проблем доступності, продуктивності та ефективності, інтеграції модальних вікон з маршрутизацією, проблем з мобільною версією, переривання поточної взаємодії, проблем з розміткою та стилями. Такий підхід до реалізації модального вікна не лише відповідає технічним вимогам WCAG, а й підвищує зручність користування для всіх користувачів, незалежно від їхніх фізичних можливостей. Він гарантує чітку структуру взаємодії, контроль над фокусом, підтримку клавіатурної навігації та повну сумісність з технологіями доступності. Усе це робить вебзастосунок більш інклюзивним і професійно

реалізованим. Систематизація підходів до реалізації цих компонентів дозволяє не лише забезпечити якість інтерфейсу, а й підвищити масштабованість та підтримуваність вебзастосунків у довгостроковій перспективі.

Оскільки традиційні засоби автоматизованого тестування подекуди мають проблеми з модальною взаємодією через їх тимчасовий характер і залежність від подій, ініційованих користувачем, автори дослідили питання тестування модальних вікон у динамічних вебсередовищах. У статті обговорюється те, як штучний інтелект (ШІ) здатен допомогти розв'язати ці складнощі, дозволяючи складний аналіз модальної поведінки, включно з динамічною адаптацією вмісту та реагування в режимі реального часу. Проведений аналіз може бути цінним як для новачків, так і для досвідчених розробників інтерфейсу, які прагнуть удосконалити свій підхід до модального дизайну в сучасних вебзастосунках.

Список використаних джерел:

1. Ruiz J., Serral E., Snoeck M. Unifying Functional User Interface Design Principles. *International Journal of Human-Computer Interaction*. 2020. Vol. 37(4). P. 1-21. DOI: https://doi.org/10.1080/10447318.2020.1805876
2. Cui H., Trimananda R., Markopoulou A. Understanding Privacy Norms through Web Forms. *arXiv*. 2024. Vol. 2408.16304. DOI: https://doi.org/10.48550/arXiv.2408.16304.
3. Gellert U., Cristea A. Creating Modal Windows and External Windows. In: *Web Dynpro ABAP for Practitioners*. Springer, Berlin, Heidelberg. 2013. P. 361-379. DOI: https://doi.org/10.1007/978-3-642-38247-5_15
4. Knut H., Lars H., Vegard S., Frode S. Form Feedback on the Web: A Comparison of Popup Alerts and In-Form Error Messages. *Smart Innovation, Systems and Technologies (SIST).* 2019. Vol. 145. P. 369-379. DOI: https://doi.org/10.1007/978-981-13-8566-7_35
5. Zhan C. File Upload and Popup Dialogs. In: *Selenium WebDriver Recipes in C#*. Apress, Berkeley, CA. 2024. P. 107-116. DOI: https://doi.org/10.1007/979-8-8688-0023-8_13
6. Visconti E., Tsigkanos Ch., Nenzi L. Automated Monitoring of Web User Interfaces. *ACM Transactions on the Web*. 2025. Vol. 19, Issue 2. No. 10. P. 1-27. DOI: https://doi.org/10.1145/3708512.
7. Трофименко О.Г., Пастернак Ю.Ю., Манаков С.Ю., Лобода Ю.Г. Автоматизація тестування вебсайтів електронної комерції. *Сучасна спеціальна техніка*. 2021. № 2(65). С. 46-59. DOI: https://doi.org/10.36486/mst2411–3816.2021.2(65).5
8. Trofymenko O. H., Dyka A. I. Problems of testing e-commerce websites. *International scientific conference «Information technologies and management in higher education and sciences»* (November 28, 2022). Riga, Latvia: "Baltija Publishing", 2022. Part 3. P. 195-199. DOI: https://doi.org/10.30525/978-9934-26-277-7-230
9. Boyev V. What is a modal in web design: all you need to know. 2024. URL: https://www.halo-lab.com/blog/modal-web-design
10. Peña N. S. Modals vs. Dialogs vs. Popups: How to Use Them Effectively in UX Design. 2025. URL: https://surl.li/wkumrj
11. The Dialog Element. Mdn web docs. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dialog
12. createPortal. React. URL: https://react.dev/reference/react-dom/createPortal
13. NextJS Docs. Routing. Parallel Routes. URL: https://nextjs.org/docs/app/building-your-application/routing/parallel-routes#modals
14. useRef. React. URL: https://react.dev/reference/react/useRef
15. Трофименко О.Г., Дика А.І., Лобода Ю.Г. Аналіз інструментів тестування вебзастосунків. *Кібербезпека: освіта, наука, техніка.* 2023. № 4(20). С. 62-71. DOI: https://doi.org/10.28925/2663-4023.2023.20.6271

References:

1. Ruiz, J., Serral, E., & Snoeck, M. (2020). Unifying Functional User Interface Design Principles. *International Journal of Human-Computer Interaction,* 37(4), 1-21. https://doi.org/10.1080/10447318.2020.1805876.
2. Cui, H., Trimananda, R., & Markopoulou, A. (2024). Understanding Privacy Norms through Web Forms. *arXiv*. 2408.16304. https://doi.org/10.48550/arXiv.2408.16304.
3. Gellert, U. & Cristea, A.D. (2013). Creating Modal Windows and External Windows. In: *Web Dynpro ABAP for Practitioners.* Springer, Berlin, Heidelberg, 361-379. https://doi.org/10.1007/978-3-642-38247-5_15
4. Knut, H., Lars, H., Vegard, S., & Frode, S. (2019). Form Feedback on the Web: A Comparison of Popup Alerts and In-Form Error Messages. *Smart Innovation, Systems and Technologies (SIST),* 145, 369-379. https://doi.org/10.1007/978-981-13-8566-7_35.
5. Zhan, C. (2024). File Upload and Popup Dialogs. In: Zhan, Z. (eds) Selenium WebDriver Recipes in C#. Apress, Berkeley, CA, 107-116. https://doi.org/10.1007/979-8-8688-0023-8_13
6. Visconti, E., Tsigkanos, Ch., & Nenzi, L. (2025). Automated Monitoring of Web User Interfaces. *ACM Transactions on the Web*, 19, 2(10), 1-27. https://doi.org/10.1145/3708512.
7. Trofymenko, O., Pasternak, Yu., Manakov, S., & Loboda, Yu. (2021). Avtomatyzatsiia testuvannia vebsaitiv elektronnoi komertsii [Automation of testing of e-commerce websites]. *Suchasna spetsialna tekhnika*, 2(65), 46-59. https://doi.org/10.36486/mst2411–3816.2021.2(65).5

8. Trofymenko, O. H. & Dyka, A. I. (2022). Problems of testing e-commerce websites. *International scientific conference «Information technologies and management in higher education and sciences»* (November 28, 2022). Riga, Latvia: “Baltija Publishing”, Part 3, 195-199. https://doi.org/10.30525/978-9934-26-277-7-230.

9. Boyev, V. (2024). What is a modal in web design: all you need to know. https://www.halo-lab.com/blog/modal-web-design

10. Peña, N. S. (2025). Modals vs. Dialogs vs. Popups: How to Use Them Effectively in UX Design. https://surl.li/wkumrj

11. The Dialog Element. Mdn web docs. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dialog.

12. NextJS Docs. Routing. Parallel Routes. https://nextjs.org/docs/app/building-your-application/routing/parallel-routes#modals

13. useRef. React. https://react.dev/reference/react/useRef

14. createPortal. React. https://react.dev/reference/react-dom/createPortal

15. Trofymenko, O., Dyka, A., & Loboda, Yu. (2023). Analiz instrumentiv testuvannia vebzastosunkiv. Kiberbezpeka: osvita, nauka, tekhnika, 4(20), 62-71. https://doi.org/10.28925/2663-4023.2023.20.6271