Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

ДОСЛІДЖЕННЯ ШЛЯХІВ ОПТИМІЗАЦІЇ FRONT-END ІНТЕРНЕТ МАГАЗИНУ НА REACT.JS

RESEARCH ON PPTIMIZATION METHODS FOR FRONT-END DEVELOPMENT OF AN ONLINE STORE USING REACT.JS

спеціальність 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КНм-21
Давиденко Віктор Володимирович

_____
(підпис)

Керівник:
к.т.н., доцент
Лук'янчук Юрій Анатолійович

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2025р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Валерій Олександрович

_____
(підпис)

Луцьк – 2025

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *магістр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Давиденко Віктор Володимирович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Дослідження шляхів оптимізації front-end інтернет магазину на React.js»
Керівник роботи: к.т.н., доцент Лук'янчук Юрій Анатолійович
затверджені наказом закладу вищої освіти від «14» травня 2025 р. № 255/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи « 03 » грудня 2025 р.
3. Вихідні дані до роботи:
Аналіз сучасного стану технологій у front-end розробках вебдодатків, дослідження методів оптимізації React.js для їх застосування під час розробки вебдодатку інтернет-магазину.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Аналіз існуючих рішень для розробки front-end інтернет магазину. Огляд і аналіз загальних принципів забезпечення ефективності front-end інтернет магазину. Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання. Дослідження практичного застосування методів оптимізації в React.js. Експериментальне дослідження результативності методів оптимізації front-end інтернет магазину на React.js

5. Перелік графічного матеріалу:

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Лук'янчук Ю. А.*		
Теоретичне дослідження та практична реалізація оптимізації front-end	*Лук'янчук Ю. А.*		
Експериментальне дослідження результативності оптимізації front-end	*Лук'янчук Ю. А.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Савчук В. О.*		
Гарант ОПП	*Ліщина В. О.*		

7. Дата видачі завдання *« » р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 8.09.2025 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 20.09.2025 р.*	
3	*Огляд і аналіз загальних принципів забезпечення ефективності front-end інтернет магазину*	*до 06.10.2025 р*	
4	*Дослідження практичного застосування методів оптимізації в React.js*	*до 18.10.2025 р.*	
5	*Аналіз інструментів для визначення вузьких місць у продуктивності та формування набору показників продуктивності вебдодатку*	*до 25.10.2025 р.*	
6	*Оцінка продуктивності вебдодатку та виявлення вузьких місць*	*до 05.11.2025 р.*	
7	*Оцінка впливу методів оптимізації на продуктивність вебдодатку*	*до 29.11.2025 р.*	
8	*Здача чистового варіанту магістерської роботи на кафедру*	*до 03.12.2025 р.*	

Здобувач вищої освіти _____ Віктор ДАВИДЕНКО

Керівник роботи _____ Юрій ЛУК'ЯНЧУК

АНОТАЦІЯ

Давиденко В. В. Дослідження шляхів оптимізації front-end інтернет магазину на React.js. Рукопис.

Кваліфікаційна робота магістра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі досліджено сучасні аспекти та тенденції у front-end розробці вебдодатку інтернет-магазину.

У другому розділі описано принципи оптимізації продуктивності під час розробки front-end інтернет-магазину та практичного застосування методів оптимізації в React.js

У третьому розділі представлено аналіз інструментів для визначення вузьких місць у продуктивності та формування набору показників продуктивності вебдодатку. Розроблено стратегію аналізу результатів оцінки продуктивності вебдодатку та виявлення вузьких місць. Особливу увагу приділено застосуванню методів оптимізації продуктивності React.js та аналізу результативності їх впливу на продуктивність на вебдодатку інтернет-магазину.

У висновках узагальнено інформацію, відображену у попередніх частинах. Результати дослідження можуть бути використані підприємствами електронної комерції для виявлення потенціалу та можливостей удосконалення ефективності функціонування вебдодатку інтернет-магазину.

Ключові слова: продуктивність, показники продуктивності вебдодатку, оптимізація продуктивності.

ANNOTATION

Viktor Davydenko. Research on pptimization methods for front-end development of an online store using React.js. Manuscript.

Master's thesis for the «Computer Science» educational program. Lutsk National Technical University, Lutsk, 2025.

Master's thesis consists of an introduction, three sections, conclusions and proposals, a list of used sources and appendices.

The first section examines modern aspects and trends in the front-end development of an online store web application.

The second section describes the process principles of optimizing performance during the development of a front-end online store and the practical application of optimization methods in React.js

The third section presents an analysis of tools for identifying bottlenecks in performance and forming a set of web application performance indicators. A strategy for analyzing the results of evaluating the performance of a web application and identifying bottlenecks has been developed. Particular attention is paid to the application of React.js performance optimization methods and the analysis of the effectiveness of their impact on the performance of the web application

The conclusions summarize the information reflected in the previous parts. The results of the study can be used by e-commerce enterprises to identify the potential and opportunities for improving the efficiency of the functioning of the web application of the online store.

Keywords: performance, web application performance indicators, performance optimization.

ЗМІСТ

ВСТУП

Електронна комерція стала ключовим аспектом сучасного бізнесу, забезпечуючи неперевершену доступність та зручність для споживачів у всьому світі. Front-end відіграє важливу роль у створенні вебдодатків, а в електронній комерції – це рівень інтернет-магазину, орієнтований на клієнта, покликаний приваблювати клієнтів своєю функціональністю, можливостями та простотою. У зв'язку з цим front-end повинен мати інтуїтивно зрозумілий інтерфейс, візуальну привабливість, хорошу залученість та рівень конверсії. Покращення інтерфейсу вебдодатку, функцій та зручності використання за допомогою фреймворкових технологій стало важливим аспектом електронної комерції. У сучасному середовищі дедалі складніших вебдодатків та інтерактивних вебплатформ користувачі вимагають плавності, швидкодії та миттєвого зворотного зв'язку. Тому, у сфері веброзробки сформувалася тенденція до створення додатків, які не лише багаті на функції, але й ефективно працюють [1].

Сучасні вебдодатки стають дедалі складнішими, що призводить до потреби в адекватній продуктивності для досягнення успіху на конкурентному ринку. Через складність вебдодатків та кількість поширених бібліотек, великі розміри пакетів коду можуть впливати на час завантаження сторінки, який необхідно мінімізувати, щоб зменшити показник відмов користувачів. Оптимізація технології розробки front-end вебдодатку суттєво впливає на онлайн-досвід та досвід користувачів, що має важливе практичне значення. Складність сучасних вебдодатків призводить до підвищених вимог до продуктивності для успіху на конкурентному ринку. Щоб додаток був продуктивним, він повинен забезпечувати безперебійну взаємодію з користувачем та плавні переходи між інтерфейсами користувача. Продуктивність вебдодатку стає критично важливою областю досліджень через її вплив на користувацький досвід. Існує тісний взаємозв'язок між користувацьким досвідом і таким чинником, як продуктивність. Крім того, погана продуктивність сайту також впливає на його рейтинг у пошуковій видачі

Google. Тому під час розробки вебдодатку необхідним є застосування методів оптимізації продуктивності.

Завдяки динамічній взаємодії та реалізації складних функцій, JavaScript, як мова програмування для Інтернету, відіграє вирішальну роль у розробці front-end [2]. React.js, бібліотека JavaScript, має численні застосування як інструмент для створення інтерактивних, насичених та надійних інтерфейсів користувача шляхом реалізації деяких основних функцій JavaScript [3]. Завдяки ефективному механізму побудови динамічних користувацьких інтерфейсів, що складаються з великої кількості компонентів, React.js здобув популярність у front-end розробці для вебдодатків електронної комерції. Оптимізація React-додатків має вирішальне значення для покращення продуктивності та взаємодії з користувачем. React.js має кілька методів оптимізації, які він реалізує за замовчуванням, наприклад, віртуальна модель об’єктів документів (Document Object Model, DOM). Однак цих методів не завжди достатньо і відповідальність за подальшу оптимізацію покладається на розробників. Оптимізація продуктивності стає вирішальною в міру масштабування вебдодатку. При цьому, забезпечення підвищення продуктивності передбачає пошук вузьких місць і застосування правильного виправлення проблеми з продуктивністю за допомогою методів оптимізації продуктивності React.js

Метою дослідження є підвищення продуктивності вебдодатку інтернет-магазину шляхом застосування інструментів та методів оптимізації React.js для front-end на основі аналізу та оцінки їх результативності.

Об’єкт дослідження: загальна продуктивність вебдодатку.

Предмет дослідження: інструменти та методи оптимізації React.js, які забезпечують виявлення проблем з продуктивністю та підвищення загальної продуктивності front-end вебдодатку.

Для досягнення мети поставлені та вирішені такі завдання:

1) виконати аналіз предметної області та існуючих рішень для розробки front-end інтернет магазину;

2) проаналізувати переваги та недоліки застосування React.js для front-end розробки;

3) провести аналіз стратегій забезпечення ефективності веб додатку;

4) дослідити особливості практичного застосування методів оптимізації в React.js. під час розробки інтернет-магазину;

5) обґрунтувати механізм визначення вузьких місць у продуктивності вебдодатку та сформувати набір показників для аудиту продуктивності;

6) дослідити результативність застосування методів оптимізації front-end на продуктивність вебдодатку інтернет-магазину.

Новизна роботи полягає у розробці механізму застосування комбінації методів оптимізації продуктивності React.js під час розробки вебдодатків, який базується на урахуванні результативності їх впливу на швидкість завантаження контенту, обробку складних обчислень та взаємодію з користувачем з точки зору продуктивності завантаження, інтерактивності та візуальної стабільності, що забезпечує новий підхід до виявлення вузьких місць та підвищення продуктивності front-end інтернет-магазину, розширення функціональності та інтерактивності вебдодатку.

Практична цінність запропонованого механізму полягає у виявленні потенціалу та можливостей для удосконалення ефективності функціонування вебдодатку інтернет-магазину, що сприятиме покращення взаємодії з клієнтами, забезпеченню високої якості обслуговування, а отже підвищенню конкурентоспроможності компанії.

Апробація результатів дослідження (додаток А):

1) Давиденко Л.В., Давиденко В.В. Фреймворк web-додатку для торгівлі електроенергією з інтеграцією технологій блокчейн та інтернету речей. Матеріали VI Міжнародної науково-технічної конференції «Оптимальне керування електроустановками (ОКЕУ- 2025)», 22-23 жовтня 2025 року : збірник наукових праць. Вінниця : ВНТУ, 2025. С. 231-233.

2) Давиденко В.В. Технології оптимізації Front-end web-додатків на React.js для підвищення продуктивності. Студентський науковий вісник.

Луцький національний технічний університет. Луцьк: Видавництво «Вежа-Друк», 2025. Вип. 54.

Результати досліджень доповідались та обговорювались на науковій конференції VI Міжнародній науково-технічній конференції «Оптимальне керування електроустановками (ОКЕУ-2025)», Вінницький національний технічний університет (ВНТУ), 22-23 жовтня 2025 року (додаток Б).

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області

Front-end в електронній комерції – це рівень інтернет-магазину, орієнтований на клієнта, тобто, це те, що бачить користувач при доступі до інтернет-магазину, з чим він взаємодіє і орієнтується в процесі онлайн-покупок. Front-end система містить такі елементи, як шрифти, кольори, зображення, випадаючі меню, панелі пошуку, сторінки з детальною інформацією про продукт і кошики для покупок, тобто, охоплює всі компоненти інтерфейсу користувача та логіку на стороні клієнта. Front-end відіграє важливу роль у створенні вебдодатків, оскільки це обличчя вебсайту, яке може привабити користувачів своїми функціями, доцільністю та простотою [4]. Front-end повинен мати інтуїтивно зрозумілий інтерфейс, візуальну привабливість, хорошу залученість та рівень конверсії. Задоволення очікувань користувачів щодо плавності, швидкодії та миттєвого зворотного зв’язку вимагає відповідних архітектурних та розробницьких рішень з перших етапів проектування вебдодатку [5], особливо, сучасних односторінкових додатків (SPA) та інтерактивних вебплатформ, які є дедалі складнішими.

В останні роки у веброзробці відбувся зсув парадигми в бік створення додатків, які не лише багаті на функції, але й ефективно працюють [1]. У front-end розробці електронної комерції є три ключові елементи [6].

Вебпродуктивність – це один з основних елементів front-end системи. Надійна продуктивність сайту, час завантаження та здатність швидко реагувати на введення даних користувача, безперебійна взаємодія з користувачем – це ті чинники, які оцінюються користувачем під час відвідин сторінки вебдодатку. Вебпродуктивність також забезпечує хороші результати пошукової оптимізації (Search Engine Optimization, SEO), що є ключовим драйвером трафіку та доходів для підприємств електронної комерції. SEO – практика оптимізації вебсайту

електронної комерції для його відображення на початку списку результатів пошукових систем, залучаючи більше органічного трафіку. Видимість у пошуку безпосередньо впливає на успіх сайту електронної комерції. Погана SEO-оптимізація впливає на потенціал органічного (неоплачуваного) трафіку, збільшує витрати на маркетинг і обмежує впізнаваність бренду. Поєднання хорошого SEO та швидкої продуктивності допомагає залучити більше органічного трафіку до інтернет-магазину та збільшити кількість клієнтів.

Продуктивність вебдодатків включає як об’єктивні вимірювані показники (час завантаження, кількість кадрів за секунду, час до інтерактивності), так і суб’єктивне враження користувача про те, скільки часу займає завантаження контенту. Сучасні вимоги щодо створення вебдодатків охоплюють низку аспектів проблеми продуктивності (табл. 1.1): мінімізацію часу завантаження та відгуку, використання методів приховування затримки для забезпечення максимально інтерактивного та швидкого користувацького інтерфейсу.

Таблиця 1.1 – Аспекти проблеми вебпродуктивності

Назва	Пояснення
Сприйнята продуктивність	Стосується того, як користувач сприймає продуктивність вебдодатку
Вимірювання продуктивності	Стосується вимірювання фактичної та сприйнятої швидкості роботи програми, оптимізації та моніторингу продуктивності
Зменшення загального навантаження	Великі файли або розмір пакета призводять до затримки, що вимагає зменшення розмірів файлів, http-запитів
Плавність та інтерактивність	Стосується забезпечення інтерактивності користувацького інтерфейсу та його належної роботи шляхом мінімізації кількості необхідних ререндерів

Користувацький інтерфейс (User Interface, UI), який стосується інтерфейсу користувача та може бути визначений як точка взаємодії та комунікації між користувачами та цифровим магазином. Інтерфейс користувача повинен мати обтічний макет, який спрощує завдання і повністю відповідає вимогам

користувача. Складові UI, такі як адаптивність вебдодатку та візуальні елементи (форми, сторінки, кнопки, пункти меню), додають інтерактивності до інтерфейсу користувача.

Користувацький досвід (Use experience, UX): загальний досвід, оцінка або відчуття користувача. Від оптимального дизайну користувацького досвіду залежить те, чи хочуть користувачі повертатися на сайт чи ні. Користувацький досвід безпосередньо пов’язаний з коефіцієнтами конверсії та утримання, а ефективність вебсайту безпосередньо пов’язана з цим користувацьким досвідом.

Добре спроектована Front-end система електронної комерції пропонує більше, ніж просто естетику. Front-end система повинна забезпечувати швидку роботу та вирішувати завдання управління станом та рендеринг. Отже, Front-end система повинна збалансувати:

– вимоги до продуктивності – забезпечити час рендерингу менше секунди та плавну взаємодію на різних пристроях;

– гнучкість бізнесу – забезпечити легке оновлення акцій та сезонних кампаній;

– досвід розробника – використання модульної структури коду, що підтримує командну співпрацю та швидку ітерацію;

– SEO-оптимізацію – застосування структурованого контенту, який пошукові системи можуть сканувати та індексувати.

1.2 Аналіз існуючих рішень для розробки front-end інтернет магазину

Невпинний розвиток вебтехнологій докорінно змінив очікування користувачів та парадигми розробки, започаткувавши еру, де продуктивність та адаптивність мають першочергове значення [7]. Безперебійна взаємодія, адаптивність та швидке завантаження є необхідною умовою для хорошого користувацького досвіду та стали обов’язковими для задоволення вимог користувачів [8]. Розробники використовують різні фреймворки для досягнення

адаптивності, інтерактивної поведінки, підвищення ефективності [4]. Рисунок 1.1 відображає основні компоненти сучасної front-end розробки.

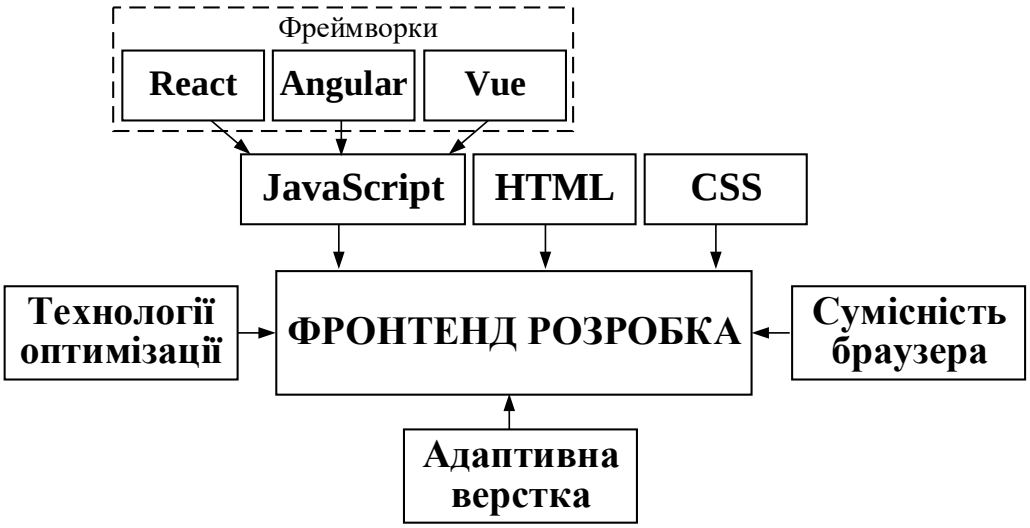

Рисунок 1.1 – Компоненти front-end розробки [4]

Роль React.js в електронній комерції. JavaScript, як мова програмування для Інтернету, відіграє вирішальну роль у розробці front-end, пропонуючи динамічну взаємодію та складні функції [2]. React, Angular та Vue є одними з провідних JavaScript-фреймворків, що використовуються для розробки динамічних вебзастосунків [1].

Одним з широко використовуваних фреймворків для створення динамічних вебдодатків і користувацьких інтерфейсів, який відіграє важливу роль для front-end розробки в електронній комерції, є React.js – розроблена Facebook бібліотека JavaScript з відкритим кодом [2, 5, 7-10].

React.js має низку переваг у створенні front-end і забезпечує створення красивих користувацьких інтерфейсів шляхом реалізації деяких основних функцій JavaScript [3], а також простий у використанні та впровадженні, оскільки надає синтаксис розмітки, тісно пов’язаний з HTML.

У React кожен елемент – це простий JavaScript-об’єкт, що описує компонент, який він представляє, разом з будь-якими відповідними властивостями або атрибутами. Компоненти – це елементи інтерфейсу користувача, які пропонують дані для перегляду та можуть змінюватися з часом

[3]. React.js використовує розбиття інтерфейсу користувача на прості, незалежні компоненти, що підлягають повторному використанню [3, 11, 12]. Компоненти повторного використання пов’язані між собою, створюючи цілісний інтерфейс користувача для програми. Кожен компонент інкапсулює власну логіку та рендеринг, що сприяє модульності та зручності обслуговування [12]. На відміну від традиційної веброзробки, модульні, повторно використовувані компоненти React.js пришвидшують створення складних вебдодатків для електронної комерції. Декларативний синтаксис спрощує розробку, роблячи компоненти більш інтуїтивними. Компонентна структура React.js дозволяє легко створювати інтерактивні функції, такі як галереї зображень продуктів, картки товарів та кошики для покупок, швидкий попередній перегляд продуктів, миттєві результати пошуку, фільтрація в реальному часі та автоматичне оновлення кошиків для покупок, що покращує мікровзаємодію з користувачами. Вирішальне значення для покращення користувацького досвіду має інтерфейс користувача, який швидко реагує [11]. React.js допомагає сторінкам продуктів, кошику для покупок і процесу оформлення замовлення працювати швидко та безперебійно. Завдяки швидкій взаємодії з компонентами React.js, користувачі відчувають легке прокручування в режимі реального часу, а сторінки заповнюються практично миттєво.

Запуск інтернет-магазину передбачає регулярне оновлення – нові товари, розпродажі, зміни дизайну та покращення функцій. Структура React.js, заснована на компонентах, робить ці оновлення набагато простішими.

React.js використовує віртуальну модель об’єктів документа (Document Object Model, DOM) – функцію, яка вирішує, чи потрібно перезавантажувати компонент, виходячи з його поточного стану та будь-яких змін, що відбуваються.

Рисунок 1.2 демонструє три фази процесу оновлення інтерфейсу користувача в React: події, ініційовані користувачем; оцінку DOM; оновлення інтерфейсу користувача.

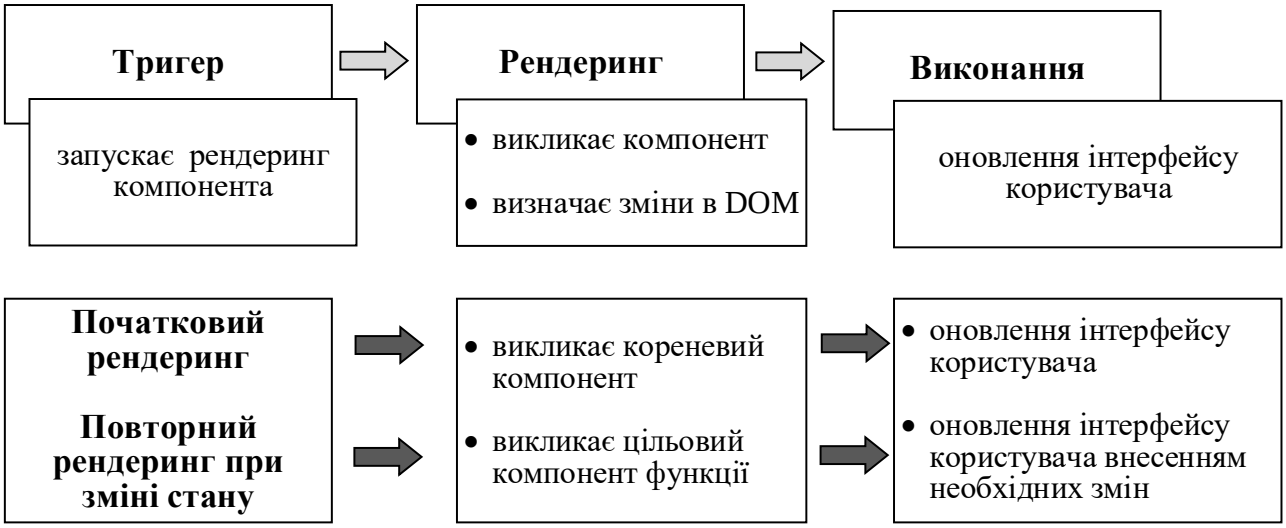

Рисунок 1.2 – Початковий рендеринг та повторний рендеринг відповідно до фаз оновлення інтерфейсу користувача

Джерело: розроблено автором

Фаза тригера стосується запуску рендерингу компонента. Це відбувається під час початкового рендерингу або в результаті зміни стану компонента чи його батьківського компонента. Отже, щоразу, коли стан оновлюється функцією встановлення стану, це запускає повторний рендеринг компонента.

Рендеринг компонента – це ситуація, коли компонент викликається після виконання коду, що міститься в компоненті або батьківському компоненті. Це проміжний етап, який визначає, чи є потреба модифікувати DOM, оцінюючи попередню та поточну версії віртуального DOM. Таким чином, фаза рендерингу являє собою виклик компонентів та оцінку того, що відображати на екрані. Під час початкового рендерингу React викликає кореневий компонент та створює вузли DOM для всіх елементів React, повернутих з JSX компонента. Після цього, для наступних рендерів, React викликає компонент-функцію, оновлення стану якої запустило рендеринг. Під час повторного рендерингу React виконує перевірку та обчислює, чи є якась різниця у властивостях елемента DOM, порівнюючи його з попереднім рендерингом.

React оновлює лише ту частину сторінки, яка змінилася, замість того, щоб перезавантажувати всю сторінку. Це запобігає повторному рендерингу застосунку, якщо це не потрібно [13], та підвищує загальну ефективність

застосунку [3, 9]. Це означає швидше завантаження сторінок та плавнішу взаємодію (зокрема, прискорення оновлення наявності товарів, додавання товарів у кошик, фільтрація товарів або оформлення замовлення), що є критично важливим для електронної комерції. Віртуальна DOM забезпечує ефективне оновлення у режимі реального часу.

Такі функції як віртуальна DOM та компоненти багаторазового використання скорочують час розробки та забезпечують зручність обслуговування. Різні частини вебсайту можуть оновлюватися незалежно один від одного, не впливаючи на інші розділи. Оновлення можна тестувати та розгортати швидше, щоб інтернет-магазин залишався актуальним і конкурентоспроможним. Чиста структура коду також полегшує пошук і виправлення будь-яких проблем, які можуть виникнути. У міру зростання інтернет-магазину React.js дозволяє легко додавати нові продукти, створювати нові функції та обслуговувати більше клієнтів, не порушуючи ваш сайт. Новий розділ можна додати на сайті, не зачіпаючи деталі, які вже добре працюють. Така гнучкість важлива, оскільки підприємства електронної комерції часто змінюються.

Завдяки своєму декларативному синтаксису, компонентній архітектурі та ефективному алгоритму порівняння віртуальних DOM, React.js дозволяє розробникам створювати великі масштабовані модульні вебдодатки, які мають швидкий рендеринг та ефективне оновлення у відповідь на зміни даних в режимі реального часу [2, 5, 8]. Основні засоби React.js, такі як управління станом, архітектура компонентів та оптимізація продуктивності (рис. 1.3), є ключовими для забезпечення ефективного функціонування додатків електронної комерції. React.js полегшує забезпечення вимог безперебійної роботи, високої продуктивності і швидкості завантаження, а також відмінного користувацького досвіду та дозволяє створювати швидкі, налаштовувані вебдодатки для інтернет-магазинів з плавним користувацьким інтерфейсом на всіх пристроях. Адаптивні функції дизайну в React.js гарантують, що інтернет-магазин виглядає професійно та безперебійно функціонує незалежно від розміру екрану. Гнучкість та

масштабованість React.js, підтримка багаторазових компонентів і можливість оновлювати лише те, що необхідно, роблять його ідеальним для платформ електронної комерції із великою кількістю каталогів продуктів та користувачів.

Рисунок 1.3 – Сильні сторони React.JS для вебдодатків електронної комерції [3]

Переваги та проблеми використання React.js для розробки високопродуктивного інтернет-магазину.

Хоча React.js отримує схвалення за те, як він спроектований для забезпечення швидкої та безперебійної взаємодії з інтерфейсом користувача, пропонує численні переваги з точки зору продуктивності розробників та можливості повторного використання коду, він стикається з певними обмеженнями в міру зростання застосунку.

React.js зосереджений на компонентах і побудові користувацьких інтерфейсів з частин та функціонує, відтворюючи контент у віртуальній DOM та ефективно керуючи ним [9]. Віртуальна DOM рендерить вузли за потреби,

підтримує безперебійну роботу, зменшує обчислювальне навантаження та допомагає пришвидшити час рендерингу, особливо для дуже складних або динамічних інтерфейсів [8], тому React.js вважається таким, що має найкращу продуктивність рендерингу [10]. Компонентна архітектура React та віртуальна DOM забезпечують швидший рендеринг та адаптивність вебсторінок. Це дозволяє додаткам React.js відповідати очікуванням користувачів, що призводить до їх кращої залученості та задоволеності [8]. Разом з тим, попри віртуальну DOM та механізми диференціації, розроблені для зменшення прямої взаємодії з фактичною DOM, щоб мінімізувати накладні витрати на рендеринг [8], тобто забезпечення оптимального рендерингу та прискорення роботи React додатку, середні та великі онлайн-додатки можуть мати низьку продуктивністю [10], оскільки чим складніший додаток, тим важче підтримувати його продуктивність на високому рівні.

Зі зростанням складності вебдодатків вони містять великі пакети JavaScript, зображення високої роздільної здатності та динамічний контент. Збільшення розміру та складності програми може сприяти повільному завантаженню, збільшенню використання пам’яті, зумовити зниження продуктивності вебдодатку на пристроях низького класу або повільному мережевому з’єднанні [5, 14]. Додаток великого обсягу може стати повільнішим вже під час початкового рендерингу, а також може зробити взаємодію інтерфейсу користувача несумісною під час наступних рендерів. Що стосується рендерингу та повторного рендерингу, компоненти, які обробляють ресурсоємні обчислення, є найбільш трудомісткими та впливають на швидкість реагування додатків. Час, необхідний для того, щоб додаток став реагуючим, збільшується зі збільшенням кількості ресурсоємних компонентів, які потребують повторного рендерингу. До поширених вузьких місць продуктивності належать надлишковий повторний рендеринг компонентів, неефективне керування станом, великі розміри пакетів та неправильне використання асинхронних операцій, затримки програми через фонові обчислення [13]. Великі розміри пакетів є основною проблемою в додатках електронної комерції React.js, що

впливає на користувацький досвід і бізнес-показники. Надмірно складні реалізації компонентів впливають на продуктивність застосунку, а також ускладнюють його підтримку та масштабування. Перевантаження, роздуття, повільний рендеринг та дорогі операції можуть негативно вплинути на взаємодію з користувачем [8]. У застосунках з високим трафіком або корпоративних застосунках ці вузькі місця проявляються у вигляді повільнішого часу завантаження, затримки взаємодії та зниження швидкості реагування, що негативно впливає на залученість користувачів та коефіцієнт конверсії [15]. Таким чином, хоча React.js відомий своєю легкістю, яку він надає для створення швидких інтерфейсів користувача, але в міру зростання програми він може стати повільним. Отже, незважаючи на широке поширення та потужний набір функцій, застосунки React.js можуть мати різні неефективності та вузькі місця [7].

З технічної точки зору, React-додатки можуть мати з такі проблеми, як [5]:

– надмірне та непотрібне повторне відтворення компонентів через неправильне використання властивостей, контексту або стану;

– великі пакети JavaScript, що збільшують час до взаємодії;

– неефективне управління станом призводить до складного потоку даних та надлишкових обчислень;

– погана обробка зображень та ресурсів, що роздуває інтерфейс користувача та споживає пам’ять.

Ці технічні неефективності часто залишаються непоміченими на ранніх етапах розробки, але стають критичними в міру зростання застосунку та масштабування трафіку користувачів. В результаті цього, зростають витрати на інженерію та обслуговування, накопичується технічний борг та страждає надійність застосунків [5].

Для вебдодатку електронної комерції важливою є видимість у пошукових системах. Для будь-якого інтернет-магазину вирішальне значення має бути знайденим у Google. Пошукові системи, такі як Google, у своїх алгоритмах ранжування враховують швидкість завантаження сторінки та показники взаємодії з користувачем. Низька продуктивність може перешкоджати SEO,

обмежуючи видимість та органічне охоплення [5]. Оскільки React.js це бібліотека на основі JavaScript, він часто розглядається як недружній до SEO за замовчуванням. React.js вебдодатки, що відображаються на стороні клієнта, іноді можуть перешкоджати SEO, оскільки боти пошукових систем можуть не повністю індексувати контент, відображений за допомогою JavaScript.

1.3 Огляд і аналіз загальних принципів забезпечення ефективності front-end інтернет магазину

Метою розробки ефективного вебсайту є не лише створення привабливого інтерфейсу, а й досягнення адаптивності, сумісності з різними браузерами та пристроями, вибір правильних фреймворків та оптимізація з точки зору часу завантаження, швидкості відгуку та взаємодії користувача з Інтернетом [4].

З поширенням багатофункціональних клієнтських інтерфейсів, користувачі очікують миттєвої взаємодії та безперебійного досвіду [5]. Швидке завантаження та швидкість рендерингу призводять до кращого користувацького досвіду та вищого рівня залученості. Дослідження показують, що навіть невеликі затримки в завантаженні сторінок можуть вплинути на поведінку користувачів та бізнес-показники [2]. Щоб забезпечити найкращий користувацький досвід, розробникам необхідно використовувати комбінацію стратегій та інструментів.

Зі зростанням складності вебдодатків керування великими пакетами JavaScript та надмірним завантаженням ресурсів стало критичним викликом, розробники все частіше стикаються з проблемою забезпечення швидкості та адаптивності React-додатків [5]. React.js, дозволяє створювати динамічні вебдодатки, але вимагає ретельної оптимізації для підтримки ефективності. Зосередження на оптимізації React-застосунку має вирішальне значення для підвищення швидкості, адаптивності та загальної ефективності застосунку, особливо враховуючи, що застосунки масштабуються за складністю та розміром (рис. 1.4). Крім того, оптимізований код легше підтримувати, налагоджувати та масштабувати з часом. Оптимізація React-застосунків є вирішальною для

покращення продуктивності. Без оптимізації добре побудовані React-додатки можуть страждати від повільного рендерингу та великого використання пам’яті.

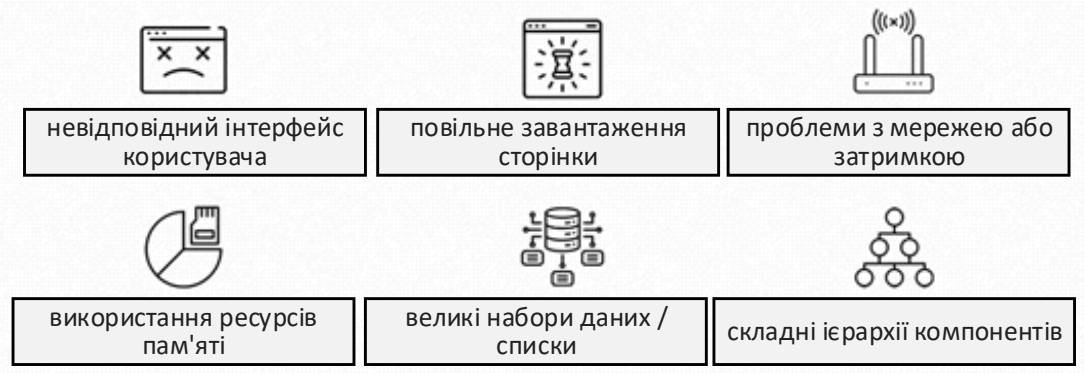

Рисунок 1.4 – Проблеми React-додатків, що вимагають застосування оптимізації

Джерело: розроблено автором

Повторний рендеринг компонентів є однією з основних причин зниження продуктивності в React-застосунках [5]. Щоразу, коли компонент повторно рендериться, React повинен виконати свою логіку рендерингу, провести диференціацію отриманої віртуальної DOM та застосувати зміни до реальної DOM. Хоча React розроблений для ефективного виконання цих операцій, непотрібні повторні рендери між глибоко вкладеними компонентами можуть суттєво вплинути на швидкість реагування.

Зі збільшенням масштабу застосунків, спосіб, яким компоненти React керують рендерингом, та те, як стан поширюється по всьому дереву компонентів, стає дедалі важливішим [5]. Маніпуляції з DOM (представлення коду у вигляді збалансованої деревоподібної структури даних) відіграють важливу роль в аналізі продуктивності фреймворків [9]. Внутрішня структура фреймворку та те, як він обробляє маніпуляції та оновлення DOM, мають найбільший вплив на продуктивність під час виконання.

Одним із базових рішень є оптимізація віртуальної DOM, що означає менше взаємодії з фактичною DOM для зниження накладних витрат на рендеринг [8]. DOM – це дерево, яке уповільнює оновлення, коли воно стає все

більшим, особливо в додатку з великою кількістю вузлів. Оскільки DOM має деревоподібну структуру, то оновлюється повільно і для оновлення потрібно перераховувати та змінювати кольори значної частини інтерфейсу. Кожне оновлення DOM викликає перерахунки, перекомпонування та перемальовування, які споживають багато обчислювальної потужності. Віртуальна DOM забезпечує абстракцію між додатком та DOM. Замість оновлення DOM, React обчислює «різницю» між поточним та минулим станами віртуального DOM (рис. 1.5) та лише певні зміни оновлюються та повторно відображаються у реальному DOM.

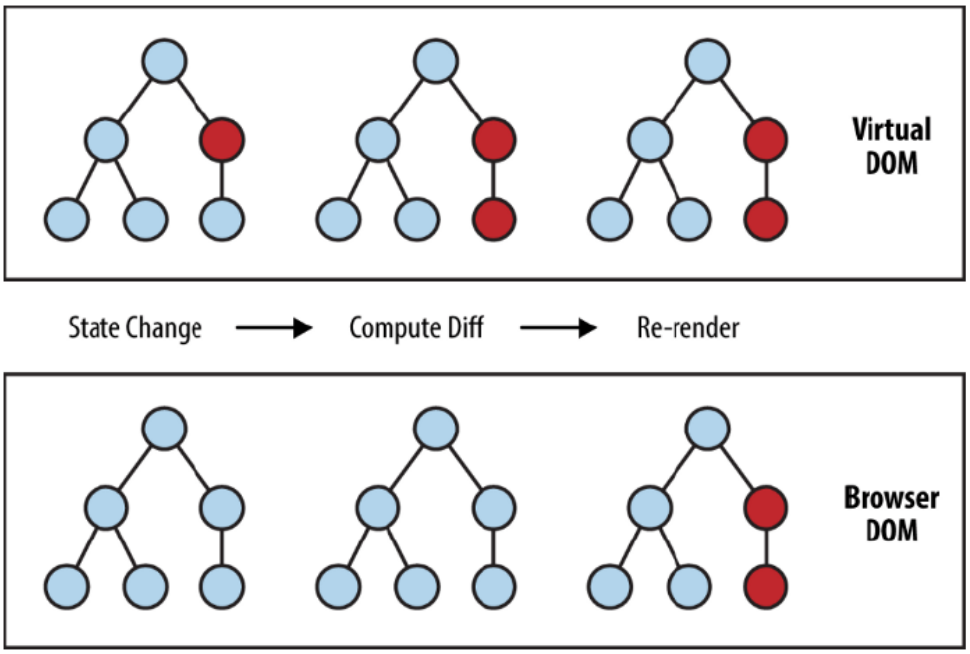

Рисунок 1.5 – Візуалізація роботи віртуальної DOM React [9]

Таке вибіркове оновлення економить багато обчислень та забезпечує швидший час рендерингу, особливо в динамічних додатках, які часто змінюють інтерфейс користувача [8]. Оптимізація віртуальної DOM є одним з основних методів, вбудованих в React, який дозволяє уникнути безпосередньої взаємодії з фактичною DOM. Завдяки шару абстракції віртуальної DOM React визначає, що змінюється в інтерфейсі користувача, та оновлює це контрольованим чином [8].

Віртуальна DOM займає центральне місце в React архітектурі забезпечує базовий механізм для покращення продуктивності [7], проте як доповнення до

оптимізації віртуальної DOM, React має деякі вбудовані інструменти, які пришвидшують роботу React-додатку [8].

Існує широкий спектр стратегій оптимізації на різних рівнях абстракції від мікрорівневих методів, таких як запам’ятовування, кешування функцій та поверхневе порівняння для уникнення непотрібних повторних рендерів, до макрорівневих рішень, таких як рендеринг на стороні сервера, розділення коду та віртуалізація списків, кожна стратегія стосується певного аспекту проблем продуктивності. У поєднанні вони створюють цілісну основу для створення високопродуктивних React-додатків [5].

Оптимізація технології front-end розробки має важливе практичне значення [4], оскільки вона суттєво впливає на онлайн-досвід користувачів, а також на розвиток підприємства електронної комерції. Без належної оптимізації користувачі стикаються з повільним завантаженням, невідповідними інтерфейсами та збільшеним споживанням ресурсів, що може негативно вплинути на залученість, утримання користувачів та бізнес-результати [14]. Безперервний моніторинг і оптимізація відіграють ключову роль у підтримці високопродуктивних React-додатків. Взаємодія з користувачами та зворотний зв’язок керують оптимізацією, допомагаючи пріоритетизувати покращення на основі реальних потреб користувача. Оптимізація React додатку, якщо вона виконана правильно, покращує завантаження сторінки та взаємодію з користувачем, зменшує показник відмов і підвищує залученість клієнтів, а також знижує загальні витрати. Оптимізований додаток має хороші шанси зайняти високі позиції в рейтингу пошукової видачі. Такий додаток стає більш помітним для користувача і отримує більше залучення.

Таким чином, у сфері сучасної веброзробки оптимізація front-end вебдодатку – це критично важлива фундаментальна вимога [5], яка безпосередньо впливає на взаємодію з користувачем, залученість, задоволеність та утримання користувачів [2], видимість у пошукових системах, а отже, сприйняття бренду та ключові показники ефективності бізнесу [5]. Оптимізація застосунків React.js – це не просто технічне вдосконалення, а бізнес-імператив.

1.4 Постановка завдання на кваліфікаційну роботу магістра

Завдання кваліфікаційної роботи – дослідження стратегій підвищення продуктивності вебдодатку, а саме методів оптимізації React.js для front-end.

Для реалізації поставленої мети потрібно вирішити наступне:

1) виконати аналіз предметної області та існуючих рішень для розробки front-end інтернет магазину;

2) проаналізувати переваги та недоліки застосування React.js для front-end розробки;

3) провести аналіз стратегій забезпечення ефективності вебдодатку;

4) дослідити особливості практичного застосування методів оптимізації в React.js. під час розробки інтернет-магазину;

5) обґрунтувати механізм визначення вузьких місць у продуктивності вебдодатку та сформувати набір показників для аудиту продуктивності;

6) дослідити результативність застосування методів оптимізації front-end на продуктивність вебдодатку інтернет-магазину.

Висновки до розділу 1

Front-end відіграє важливу роль у створенні вебдодатків. Покращення інтерфейсу вебдодатку, функцій та зручності використання за допомогою фреймворкових технологій стало важливим аспектом електронної комерції. Потужний набір функцій React.js надає низку переваг у створенні front-end та забезпечує створення красивих та швидких користувацьких інтерфейсів. Разом з тим, застосунки React.js можуть мати різні неефективності та вузькі місця, а в міру зростання програми він може стати повільним.

У світлі зростаючих вимог користувачів щодо безперебійності взаємодії, швидкості завантаження тощо оптимізація front-end React-додатку інтернет-магазину має важливе практичне значення для підвищення швидкості, адаптивності, загальної ефективності застосунку, онлайн-досвіду користувачів.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНЕ ЗАСТОСУВАННЯ СТРАТЕГІЙ ОПТИМІЗАЦІЇ FRONT-END ІНТЕРНЕТ МАГАЗИНУ

2.1 Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання

Продуктивність – характеристика front-end розробки, яка має вирішальне значення для адаптації до нових вимог сучасних користувачів та масштабованості вебзастосунків [8-9]. Продуктивний вебдодаток забезпечує послідовний та приємний користувацький досвід, що сприяє залученню, утриманню користувачів та конверсіям [15]. Ігнорування продуктивності призводить до зниження задоволеності користувачів, підвищеного показника відмов, нижчої конверсії та конкурентної невигідності [5].

Вимога продуктивності передбачає створення вебдодатків, які швидко завантажуються та реагують на взаємодію з користувачем.

Продуктивність не слід розглядати як аудит після розробки. Розробники повинні добре володіти API, методами життєвого циклу React.js та розуміти, як сучасні браузери, движки JavaScript та шаблони взаємодії з користувачем впливають на продуктивність застосунків [5]. Оскільки веброзробка передбачає складну взаємодію з користувачем, розуміння того, як виявляти та вирішувати проблеми продуктивності, пов’язані з React.js, стає важливим завданням [5]. Рішення щодо проектування, архітектури компонентів та стратегії управління станом повинні прийматися з розумінням їхнього впливу на продуктивність [5]. React.js за замовчуванням базується на механізмі, який дозволяє створювати швидкі інтерактивні вебдодатки, і чудово підходить для створення швидких, динамічних інтерфейсів користувача. Однак, коли додаток React.js масштабується, особливо великого розміру, важливе значення має оптимізація продуктивності.

Зі зростанням кількості вебдодатків, оптимізація продуктивності стала критично важливим завданням в архітектурі сучасної веброзробки, що лежить в

основі плавності та швидкості, які вимагають користувачі [14], впливає на забезпечення безперебійного досвіду кінцевих користувачів, їх задоволеність та залученість [16], а також для забезпечення масштабованості додатків [8] та високої видимості в пошукових системах [15].

Оптимізація продуктивності вебдодатку – це процес, який змушує вебсторінки завантажуватися та працювати швидше за рахунок підвищення ефективності зовнішнього коду. Цей процес включає в себе різноманітні техніки та стратегії, спрямовані на скорочення часу завантаження сторінки, покращення користувацького досвіду, мінімізацію споживання ресурсів та покращення загальної продуктивності сайту. Слід зауважити, що оптимізація продуктивності в React-застосунках виходить далеко за рамки поверхневого покращення швидкості і не є одноразовим завданням [5]. Досягнення оптимальної продуктивності в React-застосунках є складним завданням через різні фактори, такі як великі розміри пакетів, неефективний рендеринг та погане управління станом [2]. Оптимізація продуктивності в React-застосунках є безперервним процесом оцінки, вимірювання та вдосконалення [5]. З огляду на зростання складності та масштабу вебдодатків, зростаючі вимоги користувачів до швидкості та швидкодії, необхідним є використання систематичних методів для досягнення швидкодії та ефективності рендерингу [8], що вимагає складних підходів до оптимізації продуктивності [7]. React.js пропонує різні можливості оптимізації для покращення продуктивності. Кожен з методів оптимізації спрямований на підвищення продуктивності вебдодатку, проте володіє певними властивостями, які визначають особливість його функціонування та результат впливу на роботу вебдодатку.

Одним з методів оптимізації є розділення коду та ліниве завантаження. Зменшення розміру пакета є ключем до швидшого та кращого реагування React-додатку. Менші компоненти легше оптимізувати, тестувати та підтримувати [5]. Крім того, менший пакет означає швидший час завантаження, що є критично важливим для підтримки зацікавленості користувачів.

Розділення коду [8] передбачає розбиття основного коду програми на менші пакети, які викликаються за та завантажуються за запитом, коли користувач переміщується по додатку. Замість того, щоб завантажувати всю програму одразу, розділення коду надає користувачу лише код, необхідний для поточної сторінки, яку бачить користувач, або функції. На практиці це означає ізоляцію часто оновлюваних частин інтерфейсу, таких як поля введення або індикатори стану в реальному часі, від відносно статичного контенту, такого як заголовки або панелі навігації. Це обмежує область повторного рендерингу лише тими частинами інтерфейсу, які потребують оновлення [5]. Розділення коду дозволяє браузерам динамічно завантажувати лише необхідні частини коду [8], зменшує час початкового завантаження, тим самим оптимізуючи продуктивність React-застосунку, особливо для великих застосунків, використання ресурсів [7].

Розділення коду тісно узгоджується з сучасними цілями вебпродуктивності, такими як швидке завантаження, адаптивна інтерактивність та ефективне використання мережі. За умови продуманого впровадження, воно сприяє плавнішому та ефективнішому користувацькому досвіду, одночасно покращуючи зручність обслуговування та масштабованість кодової бази [5].

Основні переваги розділення коду включають [5]:

– зменшений час початкового завантаження: під час першого відвідування сторінки завантажується лише необхідний JavaScript;

– покращена продуктивність на низькоякісних пристроях: менші пакети споживають менше ресурсів пам’яті та процесора під час розбору та виконання;

– масштабованість: модуляризація коду спрощує обслуговування та забезпечує чіткіші межі коду;

– прогресивне завантаження: ресурси можуть бути пріоритетизовані на основі потоку та поведінки користувачів.

Метод завантаження пакету лише тоді, коли потрібен будь-який з компонентів, що містяться в ньому, називається відкладеним (лінивим) завантаженням. Відкладене завантаження надає пріоритет критичним ресурсам [8] і відкладає (затримує) завантаження некритичних компонентів та ресурсів до

моменту необхідності їх використання, тим самим зменшуючи розмір початкового пакету [8]. Наприклад, вікно налаштувань, до якого рідко звертаються, не потребує завантаження під час початкового рендерингу сторінки. Натомість, їх можна завантажувати, коли користувач ініціює дію, тим самим зменшуючи початкове обчислювальне навантаження [5]. Зображення, відео або додаткові компоненти також можна завантажувати асинхронно, щоб програма могла спочатку виділити ресурси, необхідні для взаємодії. Завантажуючи менше даних на початку, відкладене завантаження максимально скорочує час, необхідний для переходу програми в інтерактивний режим [8], а також збільшує швидкість завантаження сторінки [7, 14], очікуючи, поки знадобиться завантажити несуттєві компоненти або ресурси. Це дозволяє динамічно завантажувати компоненти, а не включати їх до початкового пакету JavaScript [5], що пришвидшує час початкового завантаження та дозволяє користувачеві швидше звикнути до програми [8]. Таким чином, механізм відкладеного завантаження не лише оптимізує процес, зменшує час початкового завантаження сторінки, мінімізує використання пропускної здатності, але й покращує взаємодію з користувачем, дозволяючи значущим частинам інтерфейсу швидко відображатися, тоді як менш критичні частини завантажуються у фоновому режимі [5]. Відкладене завантаження підвищує продуктивність та покращує загальний користувацький досвід, особливо на мобільних пристроях та повільних мережевих з’єднаннях. Це чудовий метод для великих ресурсів або програм з динамічним контентом та великою кількістю елементів або медіафайлів, де першопріоритетність завантаження критичних ресурсів створює враження, що програма вже працює, не чекаючи на контент [8].

Основні переваги лінивого завантаження включають [5]:

– покращений час початкового завантаження: несуттєві ресурси виключені з початкового пакету;

– зменшене споживання пропускної здатності: користувачі завантажують лише те, що вони фактично переглядають або з чим взаємодіють;

– покращена сприйнята продуктивність: швидше відображення контенту у верхній частині сторінки призводить до кращої задоволеності користувачів;

– ресурсоефективність: особливо актуально на мобільних пристроях, оскільки менше ресурсів означає менше навантаження на процесор/графічний процесор і менше використання батареї.

Ліниве завантаження (lazyload) – це потужне доповнення до розділення коду, яке дозволяє ефективно використовувати мережеві та пристроєві ресурси в React-додатках [5]. Застосування його як до компонентів інтерфейсу користувача, так і до статичних ресурсів, дозволяє створювати додатки, які завантажуються швидше, працюють швидше та краще адаптуються до різних контекстів користувача. Розділення коду вирішує проблему фрагментації пакетів JavaScript, тоді як відкладене завантаження поширює цю філософію на зображення, компоненти та інші ресурси всередині застосунку [17].

Оптимізація зображень є дієвим методом, оскільки зображення складають значну частину загальної ваги більшості React-додатків. Завантаження ресурсів має великий вплив на продуктивність. Завантаження великих і неоптимізованих зображень у React-додатках може негативно вплинути на продуктивність додатку. Оптимізація зображень у додатках React може допомогти покращити їхню вебпродуктивність та загальний користувацький досвід [18].

Першим кроком є перетворення зображення у формат, який дозволяє отримати найменший можливий розмір, зберігаючи при цьому свою якість [18]. Хоча існують різні формати зображень, такі як JPEG і PNG, формати нового покоління, такі як WebP і AVIF пропонують значно краще стиснення, більш оптимальний розмір файлу з хорошою якістю в порівнянні з іншими форматами [18], що призводить до швидшого завантаження та меншого використання пропускної здатності, особливо на мобільних пристроях. Формат зображень WebP спеціально розроблений Google для Інтернету для показу менших зображень із гарною якістю, порівнянний з JPEG. AVIF пропонує ще кращі розміри файлів стиснення та зберігає ту ж якість, що й зображення WebP. AVIF

зазвичай надає навіть менші розміри файлів, ніж WebP, але його підтримка все ще менша в порівнянні з WebP, який підтримується у всіх браузерах.

Прогресивні методи завантаження зображень у поєднанні з сучасними форматами зображень, такими як WebP та AVIF, забезпечують оптимальний візуальний контент. Використовуючи HTML і атрибути, можна завантажувати адаптивні зображення в ситуаціях де потрібно відображати різні версії одного і того ж зображення для різних розмірів екрану. Недоліком використання атрибута є те, що потрібно надати кілька версій одного зображення для вебпереглядача на вибір.

Швидкість завантаження React-додатку може знизитись, якщо він містить велику кількість зображень. Це відбувається в результаті того, що DOM рендерить кожне зображення перед тим, як буде показаний інтерфейс користувача. Відкладене завантаження, окрім компонентів, рекомендовано застосовувати до медіа-ресурсів, особливо до великих зображень та відео [5]. Відкладене завантаження зображень не відображатиме зображення, доки не настане його черга на екрані користувача, скорочуючи початковий час завантаження та підвищуючи продуктивність сторінки. Зображення на вебсторінці завантажуються не всі одночасно. Коли користувач прокручує вниз до інших частин сторінки, тоді завантажуються зображення, що залишилися.

Поширеним підходом для відкладеного завантаження є використання Intersection Observer API, власного API браузера, який дозволяє розробникам виявляти, коли елемент потрапляє в область перегляду. Це дає змогу завантажувати зображення лише тоді, коли вони ось-ось стануть видимими для користувача [5]. Оскільки не всі зображення на вебсторінці завантажуються одночасно, скорочується час, необхідний для відображення зображень у поточному вікні перегляду користувача, тим самим створюючи враження, що сторінка завантажується швидше. Цей метод особливо корисний в інтерфейсах з нескінченною прокруткою, галереях продуктів та сторінках з довгим контентом. Він гарантує, що програма завантажує лише той контент, з яким користувач активно взаємодіє, що призводить до значного підвищення продуктивності [5].

Поєднання таких методів, як представлення форматів зображень WebP і AVIF, адаптивні зображення з srcsetsizes, ліниве завантаження та використання бібліотек React, дозволяє значно скоротити час завантаження зображень, забезпечити швидку роботу вебдодатку без погіршення якості зображення і підвищити загальну продуктивність програми.

Дослідження показують, що 13 % проблем із продуктивністю JavaScript були спричинені повторними операціями [19].

Однією з стратегій оптимізації є запам’ятовування (мемоізація) – техніка, яка працює на концепції повторного використання та кешування тих самих вхідних даних для викликів функцій, що потребують інтенсивних обчислень [8]. Центральний принцип мемоізації полягає у збереженні результатів обчислень на рівні програми для подальшого використання. Результати зберігаються за допомогою параметрів обчислення. Якщо результат знайдено за заданими параметрами, він повертається; в іншому випадку обчислення виконується та кешується [19]. Тобто, виклик непотрібних повторних рендерингів не здійснюється, якщо кешований результат однаковий для останніх вхідних аргументів. Шляхом кешування результатів ресурсомістких викликів функцій забезпечується покращення продуктивність застосунку.

Слід зазначити, що оптимізація віртуальної DOM означає швидше оновлення інтерфейсу, тоді як мемоізація дозволяє значно зменшити надлишкові обчислення [7], уникнути повторного рендерингу та повторних обчислень [8], що призводить до значного підвищення продуктивності [19].

Цей метод застосовується у front-end у вигляді зберігання результатів обчислень у пам’яті браузера. Мемоізацію рекомендується використовувати для ресурсомістких обчислень, таких як операції сортування та фільтрації під час рендерингу, щоб уникнути повторних обчислень та повторних рендерів, зумовлених змінами в посиланнях на об’єкти. Мемоізація вважається надлишковою для простих обчислень у функціях [8]

Оптимізація управління станом є ще одним з методів, який пропонує React. Управління станом є важливим для оптимізації продуктивності React-додатку,

забезпечення його масштабованості та безперебійної роботи. Розумне управлінні станом допомагає запобігти непотрібним повторним рендерингам, дозволяє підвищити продуктивність додатку, гарантує його адаптивність та забезпечує безперебійний користувацький досвід.

Щоб зменшити потребу в глобальних оновленнях стану застосовується локальне управління станом в межах компонентів. Це дозволяє мінімізувати обсяг змін стану та зменшує кількість компонентів, на які впливають оновлення. Локальне управління станом обмежує оновлення стану окремими компонентами, зменшуючи вплив на решту застосунку. Цей локалізований підхід допомагає мінімізувати повторні рендери та покращити загальну продуктивність [7].

Передовий досвід в управлінні станом охоплює [5]:

– нормалізацію станів для уникнення дублювання та невідповідностей;

– мінімізацію розміру компонентів, що підписуються на стан;

– використання селекторів для ефективного отримання даних;

– уникнення непотрібного підняття штату, коли достатньо реквізитів.

Мета полягає в тому, щоб забезпечити навмисне, локалізоване і ефективне поширення змін стану, зменшуючи непотрібні обчислювальні витрати [5].

Неправильно структуровані дерева станів або надмірне використання контексту можуть спричинити вузькі місця в продуктивності, особливо коли багато компонентів покладаються на спільний стан [5]. Тому, слід уникати розміщення занадто великого стану в одному компоненті або пропускати його через глибокі дерева компонентів. Такі методи, як пакетне оновлення стану та запам’ятовування селекторів, допомагають зменшити навантаження на додаток, забезпечуючи швидкість та ефективність роботи. Слід зазначити, що React Context API, хоча й зручний, проте часто використовується надмірно або неправильно. Оскільки зміни контексту зумовлюють повторний рендеринг компонентів, що споживають весь ресурс, навіть незначні оновлення можуть спричинити проблеми з продуктивністю під час поширення по всьому великому дереву компонентів. Тому, контекст слід резервувати для глобальних, рідко оновлюваних значень, таких як тема або статус автентифікації користувача [5].

Хуки – це функції JavaScript, які дозволяють керувати станом у React-додатках. Хуки визначаються як «прості функції JavaScript, які дозволяють компонентам використовувати локальний стан та виконувати побічні ефекти (або наскрізні проблеми) та інші функції React» [18]. По суті, хуки усувають необхідність використання класів та дозволяють використовувати логіку стану між компонентами [19].

Проте, складніші застосунки часто потребують глобальної обробки стану. Для застосунків з більш динамічними або масштабними вимогами до стану, сторонні бібліотеки, такі як Redux, Zustand та Recoil, пропонують масштабовані рішення та структуровані способи керування та оновлення стану, що спрощує обробку складних взаємодій. Redux, наприклад, впроваджує централізоване сховище, передбачувані переходи станів та можливості проміжного програмного забезпечення, що робить його придатним для корпоративних застосунків. Однак він вимагає шаблонного коду та кривої навчання. Натомість, Zustand надає мінімалістичний та інтуїтивно зрозумілий API, тоді як Recoil дозволяє керувати станом на основі атомів з детальною реактивністю [20].

2.2 Дослідження практичного застосування методів оптимізації в React.js

Властивості, призначення та особливості функціонування кожного методу оптимізації React.js визначають специфіку його практичної реалізації.

Реалізація методу розділення коду та ліниве завантаження є одним із широко використовуваних підходів для оптимізації вебдодатків. У контексті React, розділення коду на рівні компонентів за допомогою динамічного імпорту та лінивого завантаження, яке реалізується за допомогою React.lazy у поєднанні з Suspense. Компоненти React.lazy і Suspense дозволяють розбити додаток на менші керовані компоненти (наприклад, розбиття каталогу товарів на категорії), які завантажуються на вимогу [7]. Це стандартний метод для оптимізації великих програм з кількома компонентами. Затримка завантаження компонентів, які не

потрібні під час початкового рендерингу, дозволяє зменшити початковий розмір бандла (ліст. 2.1).

Лістинг 2.1 – Приклад використання React.lazy для реалізації лінивого завантаження

```
const lazyLoadingComponent = React.lazy(
  () =>  './lazyLoadingComponent'
);
```

кінець лістингу 2.1

Suspense використовується разом із React.lazy для відображення контенту, такого як індикатор завантаження, під час завантаження запитуваного контенту.

Для цієї мети використовується параметр fallback що приймає будь-які елементи React для візуалізації перед завантаженням компонента.

Компоненти, створені за допомогою React.lazy, мають бути поміщені в Suspense, який керує їхнім станом завантаження (ліст. 2.2).

Лістинг 2.2 – Приклад використання Suspense для реалізації лінивого завантаження

```
<Suspense
  fallback={<div>Loading...</div>}
>
  {/*
     lazy load component
  */}
</Suspense>
```

кінець лістингу 2.2

У складніших застосунках може використовуватись розділення коду на основі маршрутизації [5], що забезпечує завантаження лише JavaScript та CSS для поточної сторінки (ліст. 2.3).

Лістинг 2.3 – Приклад розділення коду на основі маршрутизації

```
import { lazy, Suspense } from 'react';
```

```
import {
  BrowserRouter as Router,
  Route,
  Switch
} from 'react-router-dom';

const HomePage = lazy(
  () => import('./routes/HomePage')
);
const AboutPage = lazy(
  () => import('./routes/AboutPage')
);

function App() {
  return (
    <Router>
      <Suspense
        fallback={
         <div>Loading...</div>
        }
       >
        <Switch>
          <Route exact path="/" component={HomePage} />
          <Route path="/about" component={AboutPage} />
        </Switch>
      </Suspense>
    </Router>
  );
}
```

кінець лістингу 2.3

Використання розділення коду може допомогти покращити час початкового завантаження, розбиваючи додаток на менші, керовані фрагменти, які завантажуються на вимогу [21].

Реалізація методів оптимізації зображень є дієвим способом їх стиснення та швидшого завантаження. Більшість браузерів вже підтримують зображення AVIF і WebP, але хорошою практикою є надання резервного формату на випадок, якщо пристрій користувача не підтримує ці сучасні формати. Щоб отримати максимальну віддачу від обох форматів, рекомендується використовувати елемент picture, щоб запропонувати браузеру кілька варіантів. Наприклад, можна використовувати версію AVIF, якщо браузер її підтримує, а також використовувати WebP для ширшого охоплення та застосувати традиційний

JPEG як останній запасний варіант. Це гарантує, що користувачі завжди отримують найкращий досвід на основі можливостей браузера.

Лістинг 2.4 демонструє використання елементу picture для забезпечення підтримки зображень AVIF та WebP і повертається до JPEG, якщо жодне з цих зображень не підтримується на пристрої користувача [10].

Лістинг 2.4 – Приклад використання елементу picture

```
import React from "react";

export default function App() {
  return (
    <picture>

      <source
       srcSet="/images/sample.avif"
       type="image/avif"
      />
      <source
       srcSet="/images/sample.webp"
       type="image/webp"
      />
      <img
        src="/images/sample.jpg"
        alt="Descriptive alt text"
      />

    </picture>
  );
}
```

кінець лістингу 2.4

Також зображення, які містять описові атрибути alt, є більш доступними та простішими для розуміння пошуковими системами, що може покращити видимість вашого сайту в результатах пошуку за зображеннями.

Для реалізації адаптивного відображення зображень у React-додатках використовуються атрибути srcSet і sizes, що дозволяє браузеру самостійно вибирати найоптимальніший файл залежно від параметрів пристрою. Атрибут srcSet містить перелік варіантів одного зображення з різною роздільністю (наприклад, 480w, 800w, 1200w), що дає змогу браузеру завантажити саме той

файл, який найкраще відповідає розміру вікна перегляду або щільності пікселів екрана. Водночас атрибут sizes визначає, яку ширину зображення слід використовувати в різних умовах відображення – наприклад, меншу для мобільних пристроїв і більшу для екранів із високою роздільністю. Такий підхід мінімізує обсяг переданих даних і забезпечує швидке завантаження контенту без втрати якості. У контексті React це є важливою практикою оптимізації, адже дозволяє покращити продуктивність інтерфейсу користувача та зменшити час початкового рендерингу сторінки (ліст. 2.5).

Лістинг 2.5 – Приклад використання атрибутів srcSet і sizes для реалізації адаптивного відображення зображень

```
export default function App() {

  return (
    <picture>

      <source
        srcSet="image-480w.webp 480w,
                image-800w.webp 800w,
                image-1200w.webp 1200w"
        sizes="(max-width: 600px) 480px,
               (max-width: 1024px) 800px,
               1200px"
        type="image/webp"
      />

      <img
        src="image-1200w.jpg"
        alt="Adaptive image example"
        loading="lazy"
        decoding="async"
        style={{
          width: "100%",
          height: "auto",
          display: "block"
        }}
      />

    </picture>
  );
}
```

кінець лістингу 2.5

Ефективною технікою оптимізації продуктивності React-вебдодатків є ліниве (відкладене) завантаження зображень. Техніка полягає у завантаженні медіаконтенту лише тоді, коли він потрапляє в область видимості користувача. Такий підхід реалізується за допомогою атрибута loading зі значенням lazy (ліст. 2.6), що вбудований у HTML5 і підтримується більшістю сучасних браузерів. У контексті React це дозволяє уникати завантаження всіх зображень одразу під час початкового рендерингу, що значно зменшує обсяг переданих даних і прискорює початкове відображення сторінки.

Лістинг 2.6 – Приклад реалізації лінивого завантаження зображень

```
export default function App() {

  return (
    <img
      src="image-default.avif"
      alt="A responsive image example"
      loading="lazy"
    />
  );
}
```

кінець лістингу 2.6

Для реалізації кешування на рівні застосунку React.js включає кілька JavaScript-хуків (інструментів, що надають методи стану та життєвого циклу React.js для компонентів функцій) та функцій, таких як React.memo, useCallback та useMemo, використання яких забезпечує полегшення оптимізації продуктивності програми.

Повторний рендеринг у React відбувається, коли змінюється значення пропсу або стану. Пропс у дочірніх компонентах часто залишається незмінним, але реквізит його батьківського компонента змінюється.

React.memo та useMemo використовуються для «запам’ятовування» компонентів або результатів обчислень для їх використання за потреби [8].

React.memo, компонент вищого порядку, який запам’ятовує результат і не дозволяє функціональному компоненту повторно відображатися, якщо його

властивості не оновлюються [5, 7, 8]. Це є корисним для компонентів, які відображають однаковий результат за однаковими властивостями. React.memo виконує поверхневе порівняння попередніх та нових властивостей. Якщо вони однакові, це запобігає повторному рендерингу компонента, тим самим заощаджуючи обчислювальні ресурси та забезпечуючи оптимізацію продуктивності у великих застосунках, де багато компонентів можуть повторно рендеритися без потреби через незмінні реквізити [7].

Лістинг 2.7 – Приклад використання React.memo

```
const MyComponent =
  React.memo(
    function MyComponent(props) {
      /*
        render using props
      */
    }
  );
```

кінець лістингу 2.7

За замовчуванням React.memo поверхнево порівнює (shallow comparison) прості значення. Якщо компонент отримує об’єкти або масиви як props, просте поверхневе порівняння не спрацює, бо об’єкти у JavaScript порівнюються за посиланням. У таких випадках потрібна власна функція порівняння (ліст. 2.8), яка визначає, чи зміни в props дійсно вимагають нового рендеру.

Лістинг 2.8 – Приклад мемоізованого компоненту з власною функцією порівняння

```
const MyComponent = (props) => {
  /*
    render using props
  */
};

const areEqual =
  (prevProps, nextProps) => {
    /* return true if passing nextProps to render would return
the same result as passing prevProps to render,
```

```
otherwise return false */
  };

export default React.memo(MyComponent, areEqual);
```

кінець лістингу 2.8

UseMemo – використовується для запам’ятовування результатів ресурсомістких обчислень. UseMemo зберігає ресурсоємні обчислення та обчислює їх знову лише тоді, коли змінилися залежності [8]. Він гарантує, що похідне значення перераховується лише тоді, коли змінюються його залежності, уникаючи непотрібних перерахунків під час рендерингу [5].

UseMemo запам’ятовує повернене значення функції (ліст. 2.9). UseMemo рекомендується використовувати для ресурсоємних обчислень з великим розміром даних, тому одним із варіантів використання може бути кешування результатів пошуку.

Лістинг 2.9 – Мемоізація результуючого значення функції

```
const memoizedValue = useMemo
(
  /*
    Memoized function execution
  */
  () =>
    computeExpensiveValue(a, b),
  [a, b]
);
```

кінець лістингу 2.9

Наприклад, є компонент, який фільтрує великий список. Без мемоізації логіка фільтра виконувалася б при кожному рендері, навіть коли в цьому немає потреби.

Отже, React.memo та useMemo зберігають компоненти або обчислені значення, щоб повторно генерувалися лише компоненти з новими набором вхідних даних. Це значно зменшує використання ресурсів і допомагає швидше реагувати на застосунки під час роботи зі складними обчисленнями або

великими даними [8]. Ці інструменти корисні, коли є програми, які інтенсивно використовуються, такі як великі набори даних або складні обчислення, особливо, коли йдеться про завдання з великою кількістю ресурсів, такі як візуалізація даних, виклики API або обчислення в ітерації [8]. Завдяки видаленню непотрібних повторних рендерів та обчислень менше навантажується система та забезпечується адаптивність програми [8].

useCallback використовується для запам’ятовування функцій, що передаються як параметри, та запобігає непотрібному повторному створенню функцій під час рендерів (ліст. 2.10). useCallback повертає збережену версію функції зворотного виклику, яка змінюється лише у випадку зміни її залежностей. Це важливо під час передачі зворотних викликів дочірнім компонентам, щоб запобігти запуску непотрібних рендерів [5]. Коли залежність змінюється, тільки тоді цей зворотний виклик буде змінюватися. Оптимізовані дочірні компоненти залежать від еталонної рівності. Отже, коли зворотні виклики передаються дочірнім компонентам, це допомагає запобігти непотрібним рендерингам.

Лістинг 2.10 – Приклад використання useCallback

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
  },
  [a, b]
);
```

кінець лістингу 2.10

UseCallback запам’ятовує функцію зворотного виклику, щоб уникнути перевизначення функцій під час кожного повторного рендерингу компонента. UseCallback рекомендується використовувати щоразу, коли функція передається до замемоізованого компонента або функція передається як аргумент ефекту.

Інструменти useMemo та useCallback корисні в компонентах, чутливих до продуктивності, де обчислення включають перетворення даних, фільтрацію або

агрегацію. Вони також відіграють вирішальну роль при використанні разом з React.memo для збереження референційної рівності між рендерами.

Що стосується хуків useMemo та useCallback, слід зазначити, що хоча хуки useMemo та useCallback забезпечують переваги в продуктивності, вони мають різні варіанти використання. У той час як хук useMemo дозволяє кешувати результуюче значення ресурсомісткого обчислення, хук useCallback призначений для кешування визначення функції.

Проте, використання цих хуків по всьому додатку або неправильне застосування мемоізації може негативно вплинути на продуктивність додатку, оскільки ці методи використовують пам’ять і можуть збільшити накладні витрати. Стратегічне управління часом і способом отримання даних у React-застосунку може значно покращити продуктивність.

UseCallback часто використовується з useEffect, який може зберігати змінні значення без запуску повторного рендерингу. Це робить його ідеальним для зберігання інтервалів, попередніх значень, посилань на DOM або будь-яких даних, що зберігаються протягом рендерингу, не впливаючи на цикл рендерингу [22]. UseEffect рекомендується використовувати для отримання даних під час монтування компонентів, і, де це можливо, затримувати або розбивати запити даних на сторінки, щоб уникнути перевантаження ресурсів користувача та системи. UseEffect запускається щоразу, коли змінюються залежності, а useCallback може запобігти нескінченним циклам, що виникають внаслідок повторного створення функції, що використовується в useEffect.

Стратегія оптимізації управління станом реалізовується вбудованими в React хуками, зокрема, useState та useReducer, використовуються для локального керування станом компонента, а хук useContext використовується для керування даними в масштабі всієї програми. Що стосується конкретного випадку використання, useState використовується для простих перетворень, тоді як useReducer використовується для складної логіки стану [19]. Стан можна ініціалізувати та оновлювати за допомогою хука useState (ліст. 2.11), який використовується, наприклад, для обробки зміни стану в компонентах, що

використовуються у додатку. Хук useState повертає масив, що складається зі змінної стану, яка зберігає дані між рендерингами, та функції-сеттера, яка оновлює змінну. Це призводить до повторного рендерингу компонента.

Лістинг 2.11 – Приклад використання useState

```
function MyComponent() {

  const [
    localState,
    setLocalState
  ] = useState(initialValue);

  return (
    <div>
      <button onClick={
       () => setLocalState(
         localState + 1
       )
      }>
        {localState}
      </button>
    </div>
  );

}
```

кінець лістингу 2.11

Висновки до розділу 2

Оптимізація front-end вебдодатку є важливою стратегією для підвищення продуктивності, взаємодії з користувачем та масштабованості та передбачає систематичне використання різноманітних технік, стратегій та методів для досягнення швидкодії, ефективності рендерингу, мінімізації споживання ресурсів.

React.js пропонує низку підходів оптимізації для покращення продуктивності. Методи оптимізації, такі як розділення коду та ліниве завантаження спрямовані на зменшення обсягу JavaScript, мінімізацію повторного рендерингу та забезпечення ефективного завантаження ресурсів.

Для оптимізації зображень ефективним є використання комбінації методів, таких як сучасні формати зображень WebP і AVIF, адаптивні зображення з srcSet та sizes, ліниве завантаження, що дозволяє значно скоротити час завантаження зображень, забезпечити швидку роботу вебдодатку без погіршення якості зображення і підвищити загальну продуктивність вебдодатку.

Застосування техніки запам’ятовування (мемоізації) має на меті зменшити надлишкові обчислення, уникнути повторного рендерингу та повторних обчислень.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ ОПТИМІЗАЦІЇ FRONT-END ІНТЕРНЕТ МАГАЗИНУ НА REACT.JS

3.1 Методика проведення дослідження

Продуктивність вебсайту охоплює різні показники та аспекти вебдосвіду користувача, включаючи час завантаження сторінки, реакцію на взаємодію, стабільність контенту під час завантаження. Вибір та застосування інструментів оптимізації починається з виявлення вузьких місць у продуктивності [8].

Отже першим етапом є вибір інструментів для визначення вузьких місць у продуктивності. Найчастіше вузькі місця є результатом неефективних процесів рендерингу (багато або навіть непотрібних повторних рендерів) або ресурсомістких компонентів, які потребують багато обчислювальної потужності [8]. Підтримка оптимальної продуктивності вимагає постійного моніторингу, ітеративних удосконалень та дотримання найкращих практик [16].

Правильне вимірювання продуктивності є важливим етапом для виявлення причин збоїв або спрямування зусиль з оптимізації [8]. Існує два підходи до систематичного відстеження вимірюваних показників для виявлення прогресу в досягненні цілей забезпечення продуктивності вебдодатку [23]:

– моніторинг, який відповідає на питання, наскільки добре працює додаток;

– профілювання, яке відповідає на питання, чому додаток має таку продуктивність.

Інструменти моніторингу та профілювання відіграють ключову роль у виявленні вузьких місць та зниженні використання ресурсів [24], при цьому, не лише дозволяють визначити області для вдосконалення, але й дають уявлення про те, як зміни впливають на загальну продуктивність сайту. Профілювання продуктивності використовується для вирішення питання про те, чи існує потреба в оптимізації та чи варто витрачати час на оптимізацію розробки для покращення взаємодії з користувачем, дозволяє визначити, як змінилась

продуктивність в результаті застосування методів оптимізації, та виконати аналіз переваг методів оптимізації.

Для розуміння проблем, що впливають на продуктивність додатка, та прийняття рішення про застосування способів оптимізації покращення продуктивності в React доцільним є використання інструментів розробника для аналізу продуктивності. React надає низку інструментів, зокрема React DevTools та Chrome DevTools, які пропонують аналітику продуктивності додатка, допомагаючи виявляти проблеми та неефективність.

React DevTools – набір інструментів для моніторингу та профілювання [21], який дозволяє аналізувати продуктивність, відстежувати оновлення та перевіряти ієрархії компонентів, пропси, стан і методи життєвого циклу, щоб виявити джерело проблеми. Якщо використовується Chrome або Firefox, то React DevTools рекомендується використовувати через розширення для браузера.

Панель продуктивності Developer Tools записує та аналізує активність вебсайту під час його завантаження, що дозволяє перевіряти ієрархії компонентів, відстежувати властивості та зміни стану, а також аналізувати поведінку рендерингу в режимі реального часу [5, 8]. React DevTools забезпечує розуміння продуктивності додатка на рівні компонентів. Він дозволяє бачити, як відображаються компоненти та як зміни стану впливають на них.

Однією з функцій React DevTools є Profiler, який використовуватися для збору інформації про тривалість рендерингу та надає інтерактивний метод для визначення причини існування проблеми з продуктивністю[5, 8]. Profiler показує поведінку компонентів та шаблони рендерингу, забезпечує відстеження часу відображення кожного компонента (визначає, які компоненти React і чому були повторно відрендерені, вимірює частоту рендерингу компонентів, тривалість рендерингу компонентів) і надає візуалізацію дерева компонентів у вигляді полум’яного графіка, яка дозволяє виявити компоненти, які повільно працюють. Аналіз результатів Profiler дозволяє виявити вузькі місця в продуктивності, такі як частий повторний рендеринг компонентів, дорогі оновлення [5]. Діагностична інформація Profiler допомагає виявити вузькі місця продуктивності та дозволяє

проводити цілеспрямовану оптимізацію. Результати оптимізованої та неоптимізованої версій програми використовуються для оцінки продуктивності.

Браузер Google Chrome надає вбудований набір інструментів веброзробки Chrome DevTools оснащений вкладкою продуктивності. Chrome DevTools пропонує набір функцій для більш глибокої перевірки продуктивності. Використання панелі «Продуктивність» дає змогу записувати та аналізувати час виконання програми, переглядати детальну інформацію про використання процесора та виявляти будь-які проблеми з продуктивністю JavaScript. Панель «Пам’ять» забезпечує виявлення та виправлення витоків пам’яті, тоді як панель «Мережа» допомагає відстежувати швидкість завантаження ресурсів, що має вирішальне значення для оптимізації часу завантаження. Для профілювання застосунку записується сеанс браузера. Дані про продуктивність збираються під час кожного рендерингу застосунку під час сеансу. Аналіз зібраної інформації дозволяє визначити вузькі місця продуктивності.

Для виявлення вузьких місць у продуктивності рекомендується використовувати спеціалізовані інструменти профілювання продуктивності, такі як Google Lighthouse, WebPageTest, Google PageSpeed Insights.

Google Lighthouse – це комплексний інструмент з відкритим вихідним кодом, інтегрований у Chrome DevTools, який надає цілісний аудит продуктивності [8]. Google Lighthouse оцінює різні сфери: продуктивність вебдодатку, SEO (пошукова оптимізація), доступність та найкращі практики веброзробки. Google Lighthouse враховує різні показники (ефективність рендерингу, використання мережі, завантаження сторінки), генерує звіт із оцінками за кожною сферою та пропонує конкретні дієві рекомендації щодо покращення, такі як стиснення ресурсів, кешування та оптимізація шляхів рендерингу [8]. Такі дані гарантують, що оптимізація відповідає ширшим цілям, таким як ранжування в результатах пошуку та пропонування чогось користувачам різного віку. Аудит продуктивності, проведений Lighthouse, охоплює різні показники, пов’язані зі швидкістю завантаження сторінки, аналіз яких допомагає кількісно визначити, як покращення продуктивності в React

впливає на користувацький досвід [5].

WebPageTest надає детальну інформацію про час завантаження та продуктивність вебдодатку. Він дозволяє проводити тестування з різних місць по всьому світу та на різних браузерах і пристроях, забезпечуючи глибоке розуміння продуктивності вебдодатку в різних сценаріях користувача. Сильна сторона WebPageTest полягає в його здатності моделювати кілька умов, включаючи різну швидкість мережі, що дозволяє розробникам оптимізувати його для широкої аудиторії.

Google PageSpeed Insights аналізує вміст вебсторінки (при цьому, поєднуючи дані контрольованого тестового середовища з реальними показниками користувацького досвіду), а потім генерує пропозиції щодо швидшого її використання. Цей інструмент забезпечує розуміння того, як вебдодаток працює в реальних умовах користувача і як його покращити.

Поєднання методів оптимізації з інструментами аудиту продуктивності, які надають візуалізацію тенденцій рендерингу, поведінки користувачів та продуктивності вебдодатків у режимі реального часу, дозволяють постійно контролювати та оптимізувати додаток [8]. Інструменти Lighthouse, WebPageTest або PageSpeed Insights надають інформацію про продуктивність додатка, допомагають визначити, що потрібно покращити, які ресурси можна ліниво завантажити або стиснути для подальшої оптимізації, а також перевірити успішність оптимізації [8]. Google Lighthouse оцінює вебдодаток з точки зору SEO, доступності та загальної продуктивності, тоді як React Profiler додає більш детальний аналіз часу рендерингу та продуктивності компонентів [8]. Регулярний аудит продуктивності за допомогою Google Lighthouse або React Profiler дає змогу виявити будь-які нові проблеми з продуктивністю.

Для моніторингу продуктивності необхідно сформувати набір показників.

Загальну продуктивність вебдодатку визначає швидкість і плавність вебдодатку з моменту ініціювання користувачем запиту до кінцевого відображення сторінки, включаючи всі її інтерактивні функції.

Ключові складові продуктивності вебсайту включають:

– час завантаження: час, необхідний для того, щоб сторінка стала повністю видимою та придатною для використання (враховуючи завантаження всього тексту, зображень і коду, необхідного для функціонування сторінки);

– адаптивність: як швидко вебдодаток реагує на взаємодію з користувачем (кліки, прокручування, введення тексту), тобто, адаптивний вебдодаток відчувається «швидким» і реагує без помітних затримок;

– стабільність візуалізації: візуальна стабільність вебдодатку гарантує, що під час його завантаження елементи не зміщуються, що не порушує взаємодію користувача зі сторінкою;

– ефективність коду: оптимізація ресурсів і коду гарантує, що вебдодаток використовує найменшу кількість обчислювальної потужності та пропускної здатності для завантаження та роботи, що особливо важливо для користувачів на мобільних пристроях або з обмеженими тарифними планами.

У відповідь на зростаючу складність вебдодатків, Google щоб надати розробникам єдину основу для оцінки показників продуктивності представив стандартизований набір показників Web Vitals, який стосується взаємодії з користувачем у режимі реального часу. Показники Web Vitals кількісно визначають взаємодію з користувачем з точки зору продуктивності завантаження, інтерактивності та візуальної стабільності [16].

Основними показниками продуктивності вебсайту є [1, 5, 8, 16]:

– найбільше відображення контенту (Largest Contentful Paint, LCP): вимірює час з моменту початку завантаження сторінки до моменту відображення на екрані найбільшого видимого елемента контенту (текстового блоку або елемента зображення) у вікні перегляду, який користувачі можуть бачити «вище згину», тобто, без прокручування вниз;

– затримка першого введення (First Input Delay, FID): кількісно визначає час між першою взаємодією користувача (кліканням, дотиком) та відповіддю браузера (початком оброки запиту);

– кумулятивне зміщення макета (Cumulative Layout Shift, CLS): вимірює візуальну стабільність, відстежуючи неочікувані зміни візуального контенту

макета, тобто, частоту всіх несподіваних зсувів макета, що відбуваються між початком завантаження сторінки та моментом, коли стан її життєвого циклу змінюється на приховане.

Крім того існує низка додаткових показників продуктивності :

– перше відображення контенту (First Contentful Paint, FCP): вимірює час з моменту початку завантаження сторінки до моменту відображення будь-якого контенту на екрані;

– час до взаємодії (інтерактивності) (Time To Interactive, TTI): вимірює час, коли додаток стає зручним для використання та адаптивним, тобто, час від початку завантаження до моменту повної інтерактивності сторінки, коли не лише відображається візуальний контент, а й обробляються події користувача (кліки, введення тексту тощо);

– загальний час блокування (Total Blocking Time, TBT): вимірює загальний час всіх тривалих завдань (>50 мс) між фазами між FCP і TTI, протягом якого основний потік був заблокований на досить довгий проміжок часу, що перешкоджало взаємодії користувача;

– час до першого байта (Time to First Byte**,** TTFB): вимірює час відповіді сервера, необхідний мережі для доставки першого байту ресурсу контенту у відповідь на запит користувача;

– перше повне відображення (First Contentful Paint, FCP): вимірює, скільки часу потрібно для відображення першого видимого елементу вмісту вебсайту

– індекс швидкості (Speed Index, SI): вимірює, наскільки швидко контент візуально відображається під час завантаження екрану.

Моніторинг продуктивності за допомогою Web Vitals надає розробникам практичні показники, пов’язані з користувацьким досвідом [5]. Користувачі часто пов’язують швидший час завантаження з вищою якістю вебдодатків. Великі затримки можуть дратувати користувачів, що призводить до відмови від послуг. Нестабільні макети можуть призвести до поганого користувацького досвіду, такого як випадкові кліки на зміщуваних елементах. Таким чином, основні показники Web Vitals, такі як LCP, FID та CLS, представляють

найважливіші аспекти продуктивності завантаження, інтерактивності та візуальної стабільності. Ці показники забезпечують кількісний спосіб оцінки та покращення взаємодії з користувачем, є невід'ємною частиною рейтингу SEO та орієнтирами для оцінки та оптимізації продуктивності вебдодатків. Існує прямий вплив LCP, FID та CLS на взаємодію з користувачем, рейтинг у пошуку та бізнес-результати [16]. FID базується на реальних даних користувачів. FID також пов’язана з метриками TBT та TTI. TTFB та FCP використовуються для діагностики проблем з LCP. TBT та TTI використовуються для діагностики проблем з інтерактивністю.

Стратегія аналізу результатів аудиту продуктивності вебдодатку є важливим інструментом розуміння отриманих результатів моніторингу продуктивності.

Показники продуктивності, орієнтовані на користувача, є важливим інструментом для розуміння та покращення роботи сайту. Розуміння результатів їх вимірювання є важливим моментом в процесі прийняття рішення щодо необхідності оптимізації вебдодатку. З цією метою виконується візуалізація профілювання продуктивності (рис. 3.1), тобто ранжування за принципом: «хороша» – значення, що відповідають зеленій зоні, «потребує покращення» – помаранчевій зоні, «погана» – червоній зоні.

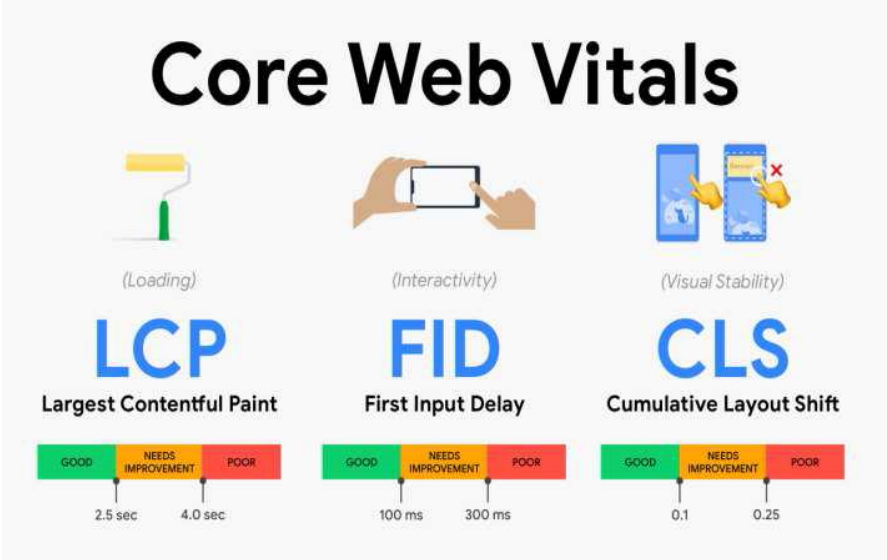

Рисунок 3.1 – Візуалізація профілювання продуктивності за Web Vitals [25]

Оцінки показників продуктивності за Web Vitals визначаються за попередньо заданими діапазонами. Вважається, що хороший LCP становить менше 2,5 секунд. Хороший FID становить менше 100 мілісекунд. Хороший бал CLS нижче 0,1. Неочікувана зміна макета означає будь-який видимий елемент, який змінює своє положення без взаємодії з користувачем. Спалах змін макету складається з кількох змін протягом короткого часу, максимум 5 секунд. Для гарного користувацького досвіду слід уникати високого балу CLS [26].

Для аналізу продуктивності використовується розширення Google Lighthouse та Web Vitals. Google Lighthouse забезпечує автоматизовані аудити продуктивності, включаючи основні показники, а також додаткові показники, такі як час до першого байта (TTFB), перше відображення контенту (FCP) та час до взаємодії (TI), індекс швидкості тощо. Зважений бал продуктивності оцінюється в діапазоні від 0 до 100. Показники оцінюються та класифікуються за кольорами від червоного до зеленого, де червоний та помаранчевий бали означають потребу в покращенні (рис. 3.2). Загальний бал 90-100, позначений зеленим кольором, ідеально підходить для гарного користувацького досвіду. Для визначення балів значення показників порівнюються з даними продуктивності, зібраними з найбільш відвідуваних вебсайтів Інтернету [26].

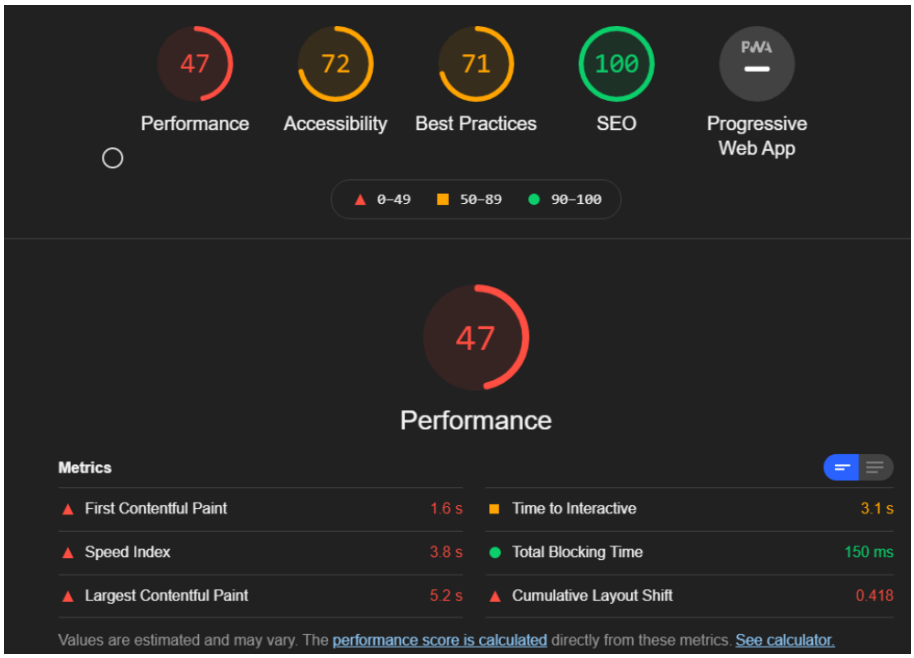

Рисунок 3.2 – Приклад візуалізації показників продуктивності Lighthouse [26]

Для визначення проблемних областей та розуміння шляхів покращення загального показника продуктивності використовується звіт Lighthouse, який містить розділи «Можливості» та «Діагностика» (рис. 3.3): найважливіші можливості або діагнози позначені червоним кольором і відображаються вище у списку [26].

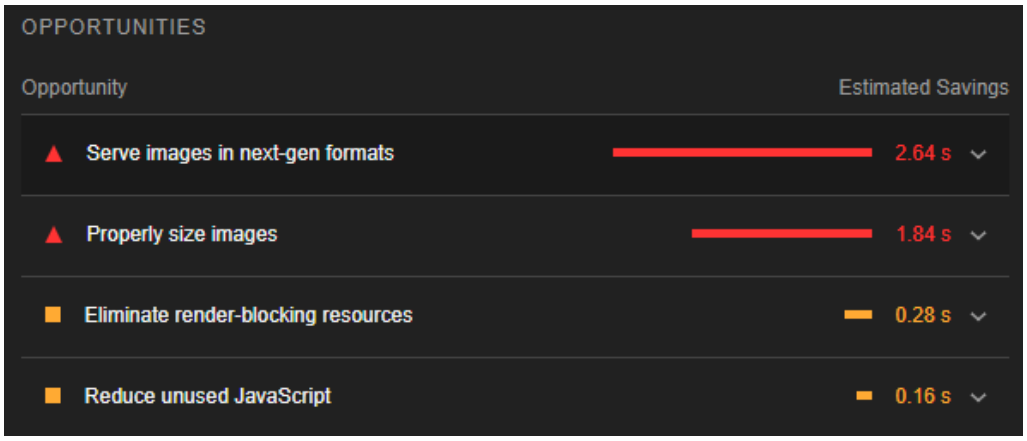

Рисунок 3.3 – Приклад пропозицій щодо покращення продуктивності Lighthouse [26]

3.2 Обробка та аналіз результатів дослідження

На першому етапі дослідження виконано оцінку продуктивності вебдодатку з метою виявлення вузьких місць.

Тестування впливу методів оптимізації front-end інтернет магазину на React.js виконуватиметься на основі сторінки продукту інтернет магазину товарів для тварин PawShop (рис. 3.4), так як саме сторінка продукту найбільш важливою для конверсії та найбільш навантаженою контентом.

Сторінка продукту містить:

– хедер з логотипом та навігацією;

– слайдер з фотографіями продукту, а також мініатюрами фотографій для навігації;

– блок з вибором опцій продукту, кількості та кнопками «Додати в кошик» та «Придбати в один клік»;

– секція з блоками опису продукту, відгуками та формою написання нового відгуку з перемикачами зверху;

– секція з рекомендованими продуктами;

– футер з формою підписки на розсилку, фоновим зображенням, логотипом та посиланнями на сервісні сторінки.

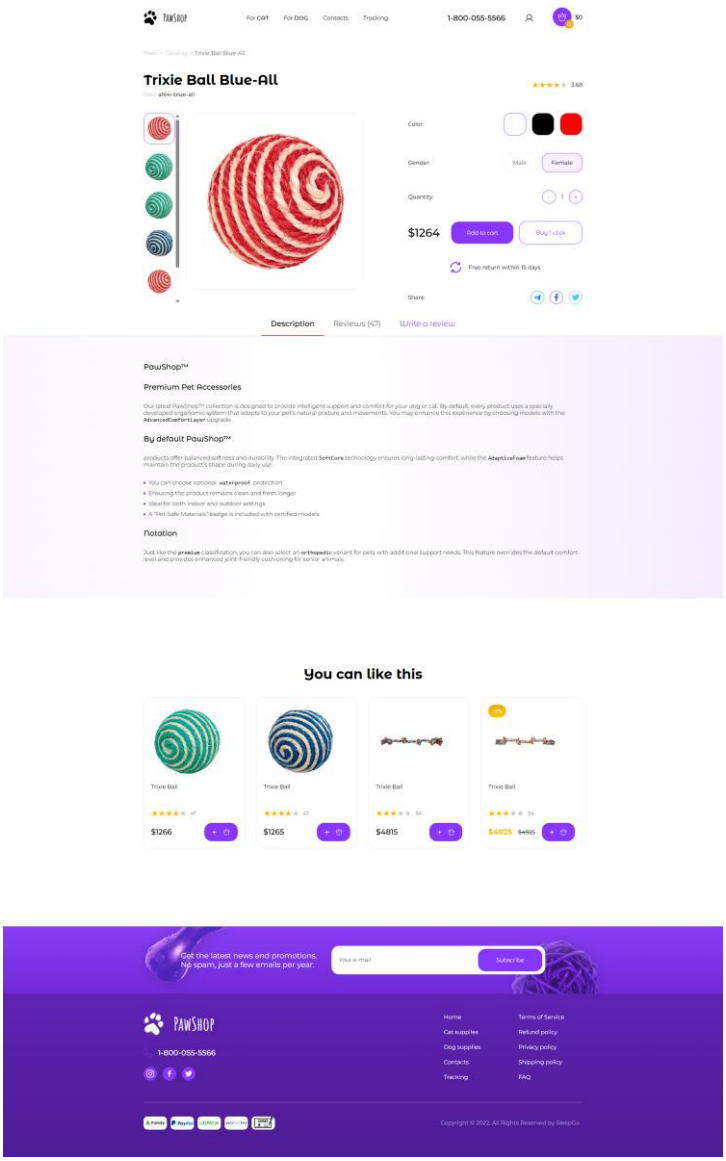

Рисунок 3.4 – Сторінка продукту інтернет-магазину товарів для тварин PawShop

Джерело: розроблено автором

Початкове тестування вебдодатку інтернет-магазину проводилося з метою визначення базових показників продуктивності, що дозволяють оцінити як

фактичну швидкодію інтерфейсу, так і ефективність його реалізації з точки зору архітектури React.js. На даному етапі важливо було не лише отримати числові значення показників, а й проаналізувати їх у контексті роботи окремих компонентів, логіки управління станом, а також структури бандлів JavaScript і завантажуваних ресурсів.

У рамках даного дослідження продуктивності було обрано саме мобільну емуляцію, оскільки мобільні пристрої є найбільш чутливими до нефункціональних недоліків вебдодатків. Мобільні браузери мають нижчу обчислювальну потужність, обмежений обсяг оперативної пам’яті та частіше працюють в умовах нестабільного або низькошвидкісного інтернет-з’єднання. Внаслідок цього навіть незначні затримки, надмірний JavaScript-бандл, неоптимізовані зображення чи дублювання рендерів можуть суттєво вплинути на загальне сприйняття швидкості роботи сайту.

Для проведення оцінки було використано мобільну емуляцію сервісу Google PageSpeed Insights, який надав детальний звіт (рис. 3.5) щодо показників метрик, які впливають на загальний показник продуктивності.

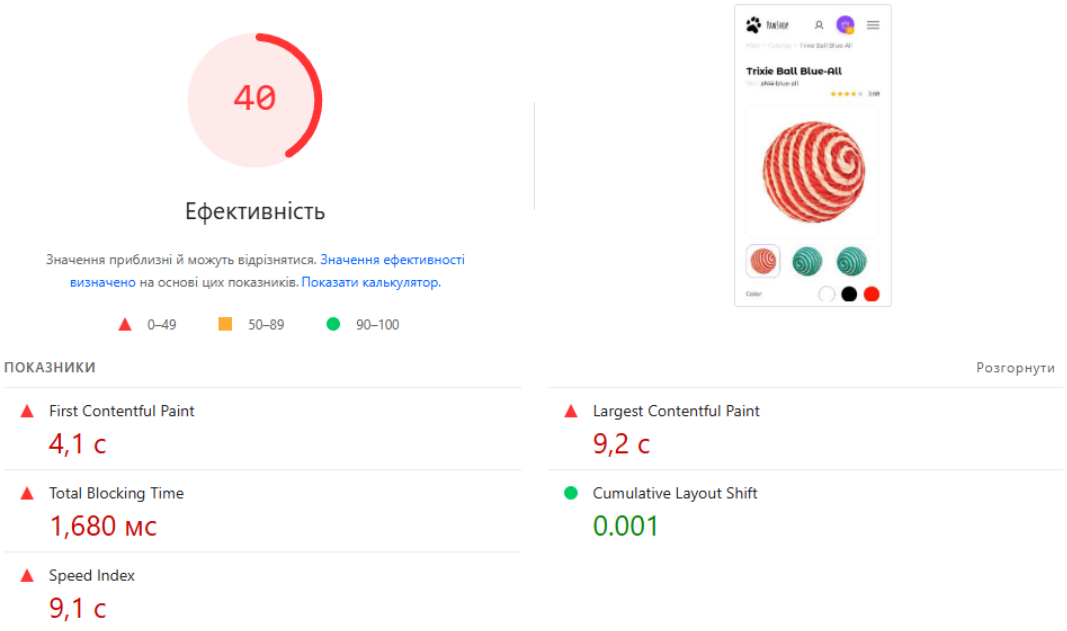

Рисунок 3.5 – Початковий звіт показників продуктивності Google PageSpeed Insights

Джерело: розроблено автором

З початкового звіту отримано показник продуктивності 40 балів, при нижній межі рекомендованої норми 50 балів. Також усі показники, крім сукупного зміщення макета (Cumulative Layout Shift, CLS) з показником 0,001, є нижчими мінімально допустимої норми, а саме:

– перша візуалізація контенту (First Contentful Paint, FCP) – 4,1 с;

– візуалізація великого контенту (Largest Contentful Paint, LCP) – 9,2 с;

– загальний час блокування (Total Blocking Time, TBT) – 1680 мс;

– індекс швидкості (Speed Index, SI) – 9,1 с.

Таким чином, результати початкового вимірювання продуктивності підтверджують наявність низки суттєвих проблем, що впливають на швидкодію сторінки продукту. Підвищені значення FCP, LCP, TBT та Speed Index свідчать про значне навантаження як на мережевий канал, так і на головний потік браузера, що характерно для інтерфейсів із великою кількістю медіаконтенту та розгалуженою компонентною структурою. Виявлені недоліки зумовлені як неоптимізованими зображеннями, так і особливостями роботи React-додатку – зокрема, надмірним обсягом JavaScript-коду та потенційними повторними рендерами окремих компонентів. Сукупність цих факторів визначає необхідність впровадження цілеспрямованих методів оптимізації, що охоплюють роботу з графічними ресурсами, розділенням коду, мемоізацією та оптимізацією управління станом. Отримані значення слугують базовою точкою відліку, яка дозволить надалі об’єктивно оцінити ефективність застосованих оптимізаційних підходів у межах експериментального дослідження.

Для виявлення результативності різних методів оптимізації front-end виконано оцінку впливу методів оптимізації React на продуктивність вебдодатку.

Після проведення базових вимірювань продуктивності було здійснено впровадження комплексу оптимізаційних заходів, спрямованих на покращення швидкодії вебдодатку інтернет-магазину. Метою даного етапу є визначення реального впливу кожної групи методів оптимізації на ключові показники продуктивності та аналіз ефективності застосованих технічних рішень. Оцінювання проводилося в умовах мобільної емуляції Google PageSpeed Insights

за аналогічних параметрів, що й під час початкового тестування, що забезпечило коректність та репрезентативність результатів.

На першому етапі було реалізовано оптимізацію зображень, які становили значну частину загального обсягу завантажуваних даних. Було впроваджено використання форматів нового покоління (WebP), а також адаптивних зображень за допомогою атрибутів srcset і sizes, що дозволило браузеру підбирати відповідний розмір медіафайлу залежно від ширини екрану. Крім того, було застосовано ліниве завантаження (lazy-loading) для медіа, що не входять до першої області видимості (ліст. 3.1).

Лістинг 3.1 – Використання атрибутів srcSet, sizes та loading в універсальному компоненті Image (додаток В) для реалізації адаптивного відображення та лінивого завантаження зображень

```
     <picture>

           {big && ( <source srcSet={big} media="(min-width: 1025px)"
type="image/webp"/> )}

           {preview && ( <source srcSet={preview} media="(min-width:
481px)" type="image/webp" /> )}

           {thumb && (
             <source srcSet={thumb} media="(max-width: 480px)"
     type="image/webp" /> )}

           <img
             className={className} src={mainUrl} srcSet={srcset}
sizes={sizes} width={width} height={height} loading={loading}
onClick={onClick} alt={alt} title={title} onError={handleImageError}
            />

     </picture>
```

кінець лістингу 3.1

Внаслідок цих змін зменшився загальний розмір сторінки, скоротилося мережеве навантаження, що в результаті покращило метрики (рис. 3.6) наступним чином:

– Загальний показник продуктивності змінився з 40 балів до 48 балів (покращення на 22 %);

– FCP змінився з 4,1 с до 3,8 с (покращення на 7 %);

– LCP змінився з 9,2 с до 7,1 с (покращення на 22 %);

– SI змінився з 9,1 с до 7,9 с (покращення на 13 %);

– CLS зменшився до нуля;

– TBT не зазнав суттєвих змін, оскільки зображення не впливають на блокування головного потоку.

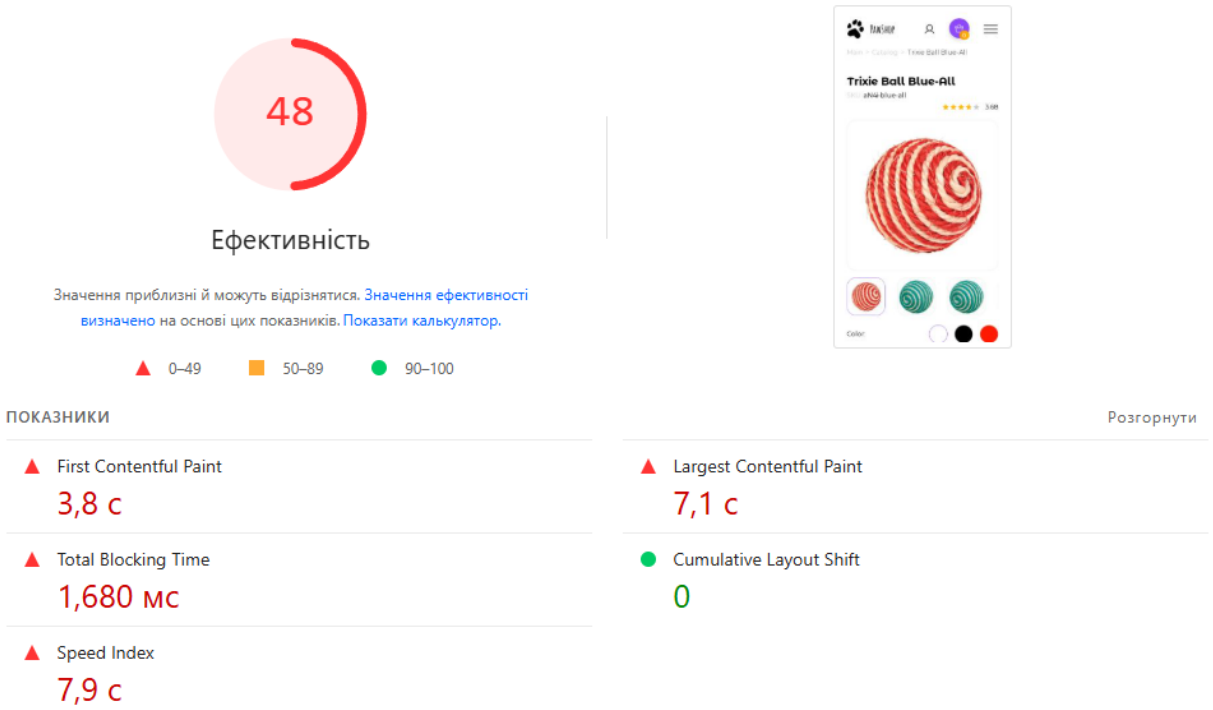

Рисунок 3.6 – Звіт показників продуктивності Google PageSpeed Insights після реалізації методів оптимізації зображень

Джерело: розроблено автором

На наступному етапі оптимізації front-end вебдодатку було виконано реалізацію методу розділення коду та лінивого завантаження компонентів за допомогою механізму React.lazy у поєднанні з Suspense (ліст. 3.2). Такий підхід дав змогу уникнути завантаження важких компонентів, які не потрібні користувачеві на початковому етапі взаємодії.

Лістинг 3.2 – Використання React.lazy у поєднанні з Suspense в компоненті сторінки продукту ProductPage (додаток Г)

```
const PopularProducts = React.lazy(
  () =>
    import("components/Home/PopularProducts")
);

const ProductContent = React.lazy(
  () =>
    import("components/Product/ProductContent/ProductContent")
);

const ProductOptions = React.lazy(
  () =>
    import("components/Product/ProductOptions")
);

const ProductPhotos = React.lazy(
  () =>
    import("components/Product/ProductPhotos")
);

/*…*/
const ProductPage = () => {
  /*…*/

  return (
    /*…*/
                <Suspense fallback={<Preloader />}>
                  <ProductPhotos items={productData.data.images} />
                  <ProductOptions item={productData} />
                </Suspense>
      /*…*/
    </>
  );
};
export default ProductPage;
```

кінець лістингу 3.2

Внаслідок реалізації даного методу оптимізації зменшився обсяг JavaScript, який виконується одразу після завантаження сторінки, що сприяло помітному прискоренню первинного рендерингу інтерфейсу. На цьому етапі було досягнуто таких значень метрик (рис. 3.7):

– Загальний показник продуктивності змінився з 48 балів до 60 балів (покращення на 25 %);

– FCP змінився з 3,8 с до 3,6 с (покращення на 5 %);

– LCP змінився з 7,1 с до 6 с (покращення на 15 %);

– SI змінився з 7,9 с до 6,8 с (покращення на 14 %);

– CLS не зазнав змін.

– TBT змінився з 1500 мс до 600 мс (покращення на 60 %).

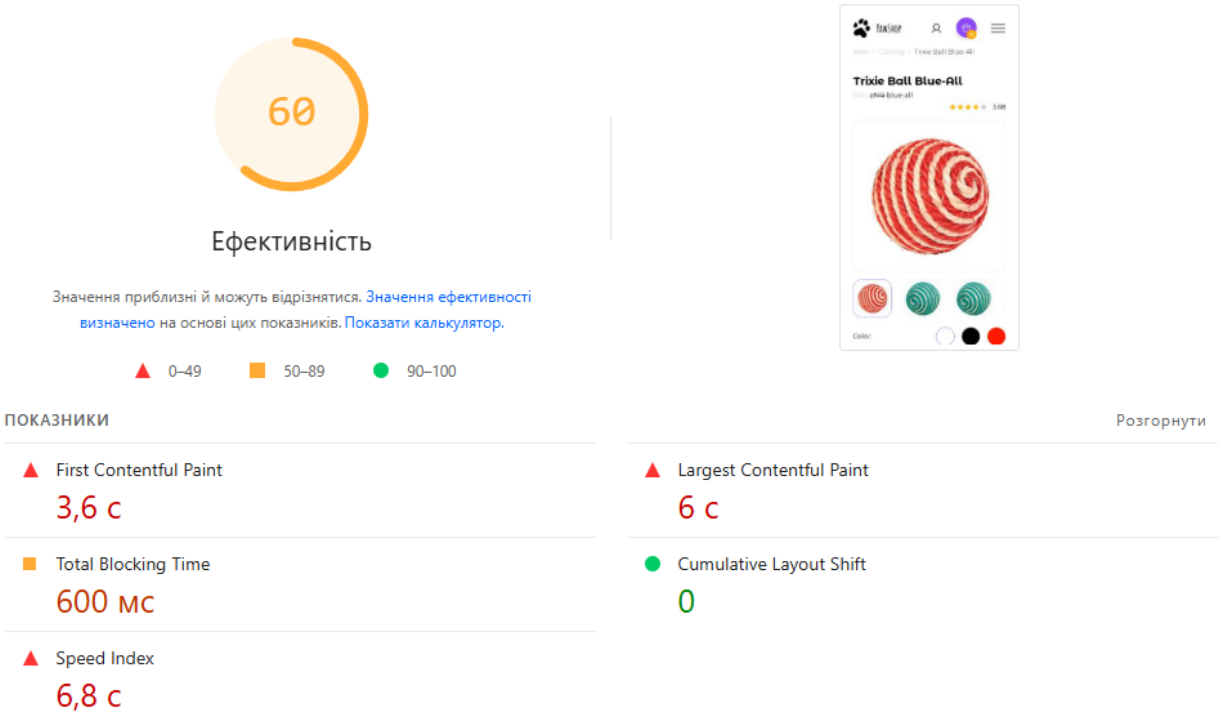

Рисунок 3.7 – Звіт показників продуктивності Google PageSpeed Insights після реалізації розділення коду та лінивого завантаження компонентів

Джерело: розроблено автором

Третім етапом було скорочення кількості незатребуваних повторних рендерів компонентів, що було реалізовано через використання механізмів мемоізації (ліст. 3.3): React.memo, useMemo, useCallback та корекцію залежностей у useEffect. Це дозволило зменшити кількість обчислень у головному потоці браузера, стабілізувати роботу компонентів і підвищити реактивність інтерфейсу.

Лістинг 3.3 – Використання React.memo, useMemo, useCallback та useEffect в компоненті сторінки продукту ProductPage (додаток Г) для реалізації мемоізації

```
const ProductPage = () => {
  const {
    slug
  } = useParams();
  const {
    state,
    fetchProduct
  } = useProduct();
  const { data: productData, isLoading } = state;
  useEffect(() => {
  catalogItemGET(slug);
  }, [catalogItemGET, slug]);

  useEffect(() => {
    fetchItem();
  }, [fetchItem]);

  /*…*/

  const breadcrumbsItem = useMemo(
    () =>
      productData?.data
        ? { name: productData.data.name }
        : { name: "Product" },
    [productData]
  );

  const ratingValue = useMemo(() => {
    return productData?.data
      ? parseFloat(
          productData.data.product.rating
       ) :
       0;
  }, [productData]);
};

export default ProductPage;
```

кінець лістингу 3.3

Хоча мемоізація не завжди має прямий вплив на мережеві метрики, вона суттєво знижує навантаження на JavaScript-рушій, що позитивно позначається на TBT та плавності взаємодії. Завдяки реалізації механізмів мемоізації було досягнуто наступних метрик (рис. 3.8):

– Загальний показник продуктивності змінився з 60 балів до 69 балів (покращення на 25 %);

– FCP змінився з 3,6 с до 3,5 с (покращення на 3 %);

– LCP змінився з 6,0 с до 5,1 с (покращення на 15 %);

– SI змінився з 6,8 с до 6 с (покращення на 12 %);

– CLS не зазнав змін.

– TBT різке скорочення з 600 мс до 70 мс (покращення на 88 %).

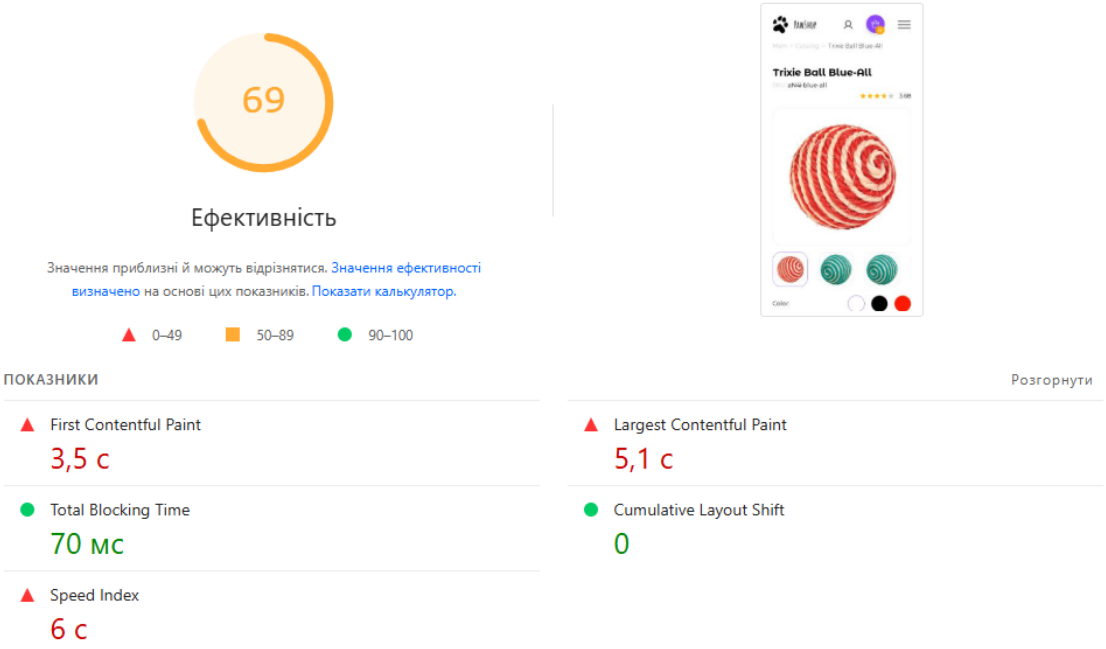

Рисунок 3.8 – Звіт показників продуктивності Google PageSpeed Insights після реалізації механізмів мемоізації

Джерело: розроблено автором

Завершальним етапом експериментального дослідження була оптимізація управління станом, яка передбачала винесення логіки завантаження та обробки даних у глобальний контекст ProductContext із використанням Context та useReducer, а також застосування вибору частин стану та мемоізацію похідних значень, що дозволило передавати компонентам лише ті дані, які їм дійсно необхідні. Це дозволило уникнути каскадних оновлень компонентів та знизити частоту обчислювальних операцій, що виконуються під час рендерингу.

Лістинг 3.4 – Використання Context та useReducer в ProductContext.jsx для локалізації стану та вибору частин стану для уникнення зайвих рендерів

```
      const ProductContext = createContext();
      const initialState = { data: null, isLoading: false, error: null };

      function productReducer(state, action) {
        switch (action.type) {
          case "FETCH_START":
            return { ...state, isLoading: true, error: null };
          case "FETCH_SUCCESS":
            return { ...state, isLoading: false, data: action.payload };
          case "FETCH_LOCAL_FALLBACK":
            return { ...state, isLoading: false, data: action.payload };
          case "FETCH_ERROR":
            return { ...state, isLoading: false, error: action.payload };
          default:
            return state;
        }
      }
      export const ProductProvider = ({ children }) => {
        const [state, dispatch] = useReducer(productReducer,
initialState);
        const [catalogItemGET] = useLazyCatalogItemGETQuery();
        const fetchProduct = useCallback(async (slug) => {
          dispatch({ type: "FETCH_START" });
          const res = await catalogItemGET(slug);
          if (res.error && res.error.status === 500) {
            const fallback = localVariants.data.find((item) =>
item.data.slug === slug) || null;
            dispatch({ type: "FETCH_LOCAL_FALLBACK", payload: fallback
}); return;
          }
          if (res.data) {
            dispatch({ type: "FETCH_SUCCESS", payload: res.data });
          } else {
            dispatch({type:"FETCH_ERROR", payload: res.error });
          }
        }, [catalogItemGET]);
        return (
          <ProductContext.Provider value={{ state, fetchProduct }}>
            {children}
          </ProductContext.Provider>
        );
      };

      export const useProduct = () => useContext(ProductContext);
```

кінець лістингу 3.4

Зменшення навантаження на цикл оновлення React знизило час виконання скриптів та позитивно вплинуло на TBT та загальну оцінку продуктивності. На цьому етапі було досягнуто наступних метрик (рис. 3.9):

– Загальний показник продуктивності змінився з 69 балів до 78 балів (покращення на 25 %);

– FCP змінився з 3,5 с до 3,4 с (покращення на 3 %);

– LCP змінився з 5,1 с до 3,9 с (покращення на 23 %);

– SI змінився з 6,0 с до 5,2 с (покращення на 13 %);

– CLS не зазнав змін.

– TBT змінився з 70 мс до 20 мс (покращення на 71 %).

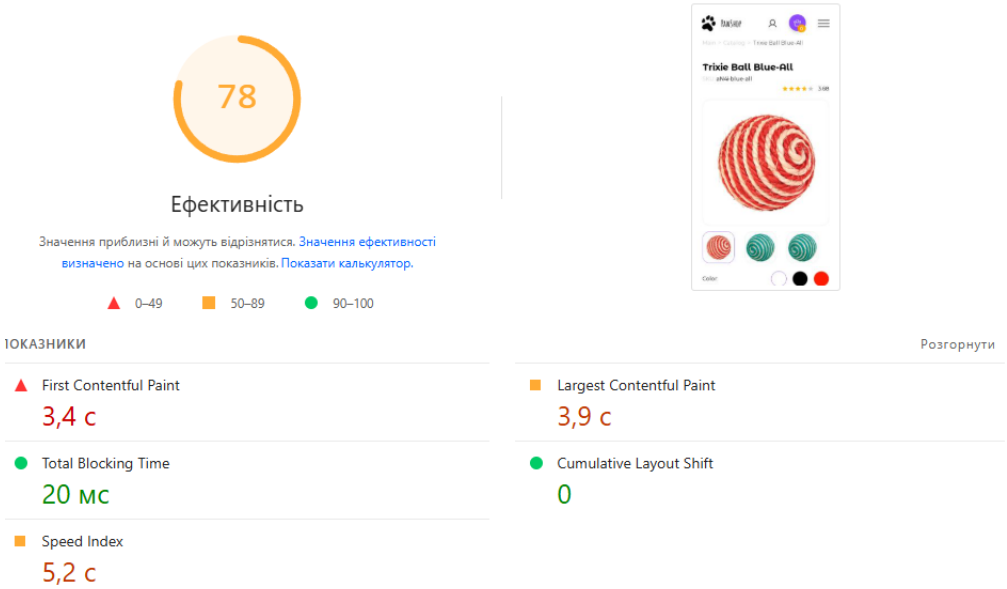

Рисунок 3.9 – Звіт показників продуктивності Google PageSpeed Insights після реалізації методів оптимізації управління станом

Джерело: розроблено автором

В таблиці 3.1 наведено результати оцінки продуктивності, отримані на кожному етапі дослідження. Аналіз результатів, показав, що найбільший позитивний вплив на метрики LCP та Speed Index дали саме оптимізація зображень. Водночас найбільше зменшення TBT було досягнуто завдяки розділенню коду та мемоізації. Оптимізація управління станом забезпечила загальне підвищення стабільності роботи інтерфейсу під час завантаження.

Таблиця 3.1 – Зведений звіт показників продуктивності

Назва етапу	Загальний бал	FCP, с	LCP, с	TBT, мс	CLS	SI, с
Початкові дані	40	4,1	9,2	1680	0,001	9,1
Оптимізація зображень	48	3,8	7,1	1680	0	7,9
Розділення коду та ліниве завантаження компонентів	60	3,6	6	600	0	6,8
Мемоізація	69	3,5	5,1	70	0	6
Оптимізація управління станом	78	3,4	3,9	20	0	5,2

Результати дослідження впливу методів оптимізації продуктивності front-end вебдодатку інтернет-магазину дозволяють стверджувати, що комплексний підхід до оптимізації продуктивності, який охоплює роботу як із мережевими, так і з обчислювальними аспектами вебдодатку, дозволив досягти істотного покращення всіх ключових метрик. Це підтверджує доцільність застосування сучасних методів оптимізації у React-додатках та демонструє їхню ефективність у контексті реальних вебпроєктів.

Висновки до розділу 3

Продуктивність вебсайту охоплює різні показники та аспекти досвіду користувача, включаючи час завантаження сторінки, реакцію на взаємодію, стабільність контенту під час завантаження. Загальну продуктивність вебдодатку визначає швидкість і плавність вебдодатку з моменту ініціювання користувачем запиту до кінцевого відображення сторінки, включаючи всі її інтерактивні функції. Показники продуктивності, орієнтовані на користувача, є важливим інструментом для розуміння та покращення роботи сайту. Правильне вимірювання продуктивності є важливим етапом для виявлення причин збоїв або спрямування зусиль з оптимізації. Для розуміння проблем, що впливають на продуктивність додатка, та прийняття рішення про застосування способів оптимізації покращення продуктивності в React доцільним є використання

інструментів аудиту та моніторингу показників продуктивності, зокрема, інструментів розробника для аналізу продуктивності.

Дослідження впливу методів оптимізації на продуктивність вебдодатку проведено згідно обраного плану експерименту, який передбачав застосування різних методів оптимізації front-end до зразка React-застосунку (сторінки продукту інтернет-магазину товарів для тварин PawShop) та вимірювання показників продуктивності на кожному етапі (до та після застосування певного методу оптимізації) шляхом використання мобільної емуляції сервісу Google PageSpeed Insights. Набір показників продуктивності вебдодатку, сформований для реалізації дослідження, містить стандартизований набір показників Web Vitals, а їх якісна оцінка виконувалась з урахуванням попередньо заданих діапазонами. Вимірювання показників продуктивності на кожному етапі за аналогічних параметрів забезпечило коректність та репрезентативність результатів вимірювання.

Аналіз результатів аудиту продуктивності вебдодатку свідчить про позитивний вплив різних методів оптимізації React.JS та їх комбінації на показники продуктивності. Спільне застосування таких методів як оптимізація зображень, розділення коду та ліниве завантаження забезпечують скорочення часу завантаження сторінки, покращуючи сприйняту продуктивність вебдодатку, а застосування оптимізації управління станом забезпечує загальне підвищення стабільності роботи інтерфейсу під час завантаження.

ВИСНОВКИ

Кваліфікаційна робота присвячена питанню підвищення продуктивності вебдодатку інтернет-магазину шляхом застосування методів оптимізації React.js, які забезпечують виявлення проблем з продуктивністю та можливостей для удосконалення ефективності функціонування вебдодатку інтернет-магазину. Результати дослідження підтвердили ефективність запропонованого підходу підвищення продуктивності вебдодатку, що базується на застосуванні комбінації методів оптимізації продуктивності React.js під час розробки вебдодатків.

Щодо виконаних завдань, можна підсумувати наступне:

1) аналіз предметної області та існуючих рішень для розробки front-end інтернет магазину дозволяє стверджувати, що вимога покращення інтерфейсу вебдодатку, функцій та зручності використання за допомогою фреймворкових технологій є важливим аспектом електронної комерції;

2) аналіз особливостей застосування React.js для front-end розробки показав, що потужний набір функцій React.js надає низку переваг у створенні front-end, що забезпечує створення красивих та швидких користувацьких інтерфейсів, проте із зростанням складності вебдодатку інтернет-магазину застосунки React.js можуть мати різні неефективності та вузькі місця, а в міру зростання програми він може стати повільним вже під час початкового рендерингу, а також може зробити взаємодію інтерфейсу користувача несумісною під час наступних рендерів;

3) аналіз стратегій забезпечення ефективності веб додатку показав, що оптимізація front-end інтернет-магазину, яка передбачає систематичне використання різноманітних технік, стратегій та методів для досягнення швидкодії, ефективності рендерингу, мінімізації споживання ресурсів є забезпечує підвищення продуктивності, адаптивності, загальної ефективності застосунку та покращення взаємодії з користувачем, що є важливим інструментом задоволення зростаючих вимог користувачів щодо безперебійності взаємодії, швидкості завантаження вебдодатку тощо;

4) з точки зору практичної реалізації оптимізації front-end під час розробки інтернет-магазину, React.js має власні методи та інструменти для вирішення проблем продуктивності, кожен з яким забезпечує підвищення загальної ефективності веб-додатку. Методи оптимізації React.js, такі як розділення коду та ліниве завантаження, спрямовані на зменшення обсягу JavaScript, мінімізацію повторного рендерингу та забезпечення ефективного завантаження ресурсів. Для оптимізації зображень ефективним є використання комбінації методів, таких як сучасні формати зображень WebP і AVIF, адаптивні зображення з srcSet та sizes, ліниве завантаження, що дозволяє значно скоротити час завантаження зображень, забезпечити швидку роботу вебдодатку без погіршення якості зображення і підвищити загальну продуктивність вебдодатку. Застосування техніки запам’ятовування має на меті зменшити надлишкові обчислення, уникнути повторного рендерингу та повторних обчислень;

5) для виявлення вузьких місць у продуктивності вебдодатку, доцільним є використання інструментів аудиту та моніторингу показників продуктивності, зокрема, інструментів розробника для аналізу продуктивності, які забезпечують розуміння проблем, що впливають на продуктивність додатка, та прийняття рішення про застосування способів оптимізації в React для покращення продуктивності. Продуктивність вебсайту охоплює різні показники та аспекти вебдосвіду користувача, включаючи час завантаження сторінки, реакцію на взаємодію, стабільність контенту під час завантаження. Загальну продуктивність вебдодатку визначає швидкість і плавність вебдодатку з моменту ініціювання користувачем запиту до кінцевого відображення сторінки, включаючи всі її інтерактивні функції. При цьому, показники продуктивності, орієнтовані на користувача, є важливим інструментом для розуміння та покращення роботи сайту. Урахування стандартизованого набору показників Web Vitals та застосування попередньо заданих діапазонів для їх якісної оцінки під час моніторингу продуктивності вебдодатку забезпечує виявлення проблем з продуктивністю та шляхів їх усунення;

6) дослідження впливу оптимізації front-end на продуктивність вебдодатку інтернет-магазину проведено згідно обраного плану експерименту, який передбачав застосування різних методів оптимізації до зразка React-застосунку (сторінки продукту інтернет-магазину товарів для тварин PawShop) та вимірювання показників продуктивності на кожному етапі шляхом використання мобільної емуляції сервісу Google PageSpeed Insights. Аналіз результатів аудиту продуктивності вебдодатку інтернет-магазину підтверджує позитивний вплив застосування різних методів оптимізації React.js та їх комбінації на показники продуктивності. Спільне застосування таких методів як оптимізація зображень, розділення коду та ліниве завантаження забезпечують скорочення часу завантаження сторінки, покращуючи сприйняту продуктивність вебдодатку, а застосування оптимізації управління станом сприяє загальному підвищення стабільності роботи інтерфейсу під час завантаження.

Результати дослідження впливу методів оптимізації продуктивності front-end вебдодатку інтернет-магазину дозволяють стверджувати, що комплексний підхід до оптимізації продуктивності, який охоплює роботу як із мережевими, так і з обчислювальними аспектами вебдодатку, дозволив досягти істотного покращення всіх ключових метрик. Це демонструє їхню ефективність у контексті реальних вебпроєктів та підтверджує доцільність застосування сучасних методів оптимізації у React-додатках для швидшого завантаження сторінки, покращення швидкості рендерингу та взаємодії з користувачем.

За результатами досліджень за темою кваліфікаційної роботи опубліковано 2 праці, в тому числі: 1 стаття у науковому виданні «Студентський вісник»; 1 тези доповіді у збірнику матеріалів міжнародної конференції. Перелік публікацій за матеріалами магістерського дослідження наведено в додатку А.

Результати дослідження доповідались та обговорювались на міжнародній науковій конференції. Підтвердження апробацію магістерського дослідження наведено в додатку Б (сертифікат учасника міжнародної конференції, який засвідчує публічний виступ та демонстрацію результатів дослідження).

Результати дослідження мають практичну цінність з точки зору їх застосування для виявлення потенціалу та можливостей для удосконалення ефективності функціонування вебдодатку інтернет-магазину, що сприятиме покращенню взаємодії з клієнтами, забезпеченню високої якості обслуговування, а отже підвищенню конкурентоспроможності компанії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kurapati L. Balancing accessibility and performance in progressive web applications using micro front-end architecture: a comprehensive study of REACTJS, ANGULARJS, and VUE-JS. International Journal of Advanced Research (IJAR). 2024. P. 31-34.

2. Kumar S. Optimizing Front-end Performance with React and JavaScript: Techniques and Impact Analysis. International Journal Of Novel Research And Development. 2024. Vol. 9, Is.7. P. b866-b869.

3. Rananavare A. Front-end development with REACT.JS. International Research Journal of Engineering and Technology (IRJET). 2022. Vol. 09, Is.06. P. 2339-2342.

4. Mohane A., Ghate K. Understanding Web Front-end Development Technology based upon Responsiveness, Optimization and Interactive Nature. International Research Journal of Engineering and Technology (IRJET). 2021. Vol.08, Is.06. P. 3324-3327.

5. Toprak A., Toprak F. S. Improving and Optimizing React Web Applications: Strategies and Techniques. 5th International Congress of Engineering and Natural Sciences (ICENSS 2025), Ankara. Turkey. 2025. P. 1-17.

6. eCommerce Front-End Development: The Ultimate Guide to 2024. April 19, 2024. URL: https://www.arrowhitech.com/market-insight/ecommerce-front-end-development/ (дата звернення 11.09.2025 року).

7. Jani Y. The Critical Importance of React Performance Optimization: Strategies for Success. North American Journal of Engineering and Research. 2020. Vol.1, Is.1. P. 39-42.

8. Mondal S. Enhancing React Application Performance: Proven Strategies and Best Practices. International Research Journal of Engineering and Technology (IRJET). 2024. Vol..11, Is.12. P. 309-312.

9. Vyas R. Comparative Analysis on Front-End Frameworks for Web Applications. International Journal for Research in Applied Science & Engineering Technology (IJRASET). 2022. Vol.10, Is.VII. P. 298-307.

10. Patel S. React Performance Optimization: Write Perfect Codes for Best Performance. 2024. URL: https://www.cmarix.com/blog/react-performance-optimization/ (дата звернення 29.09.2025 року).

11. Tiwari A., Srivastava S. Exploring Front End Framework React: A review. International Research Journal of Engineering and Technology (IRJET). 2020. Vol.07, Is.07. P. 2808-2811.

12. Javeed A. Performance Optimization Techniques for React.JS. URL: https://www.researchgate.net/publication/336621924_Performance_Optimization_Techniques_for_ReactJS (дата звернення 29.09.2025 року).

13. Gowda V., Rangaswamy Sh. Addressing the Limitations of React JS. International Research Journal of Engineering and Technology (IRJET). 2020. Vol.07, Is.04. P. 5065-5068.

14. Jain V. Optimizing WEB performance with LAZY LOADING and CODE SPLITTING. International Journal of Core Engineering & Management. 2022. Vol.7, Is.03. P. 193-199.

15. Chavan S., Kalwar J., Golatkar M., Ransing R. A web-based project repository using ReactJS. 13th International Conference on Computing Communication and Networking Technologies (ICCCNT), Kharagpur, India. 2022. P. 1-3.

16. Jain V. WEB vitals and core metrics for WEB performance optimization. International Journal of Core Engineering & Management. 2023. Vol.7, Is.06. P. 198-205.

17. Lakshmi Dr., Rakshitha L. The Power of React JS for Business Applications. International Research Journal on Advanced Engineering and Management (IRJAEM). 2024. Vol.2. P. 1637-1639.

18. Okpukoro Tr. J. React image optimization: Techniques to speed up your web apps. 29.08.2025. URL: https://uploadcare.com/blog/react-image-optimization-techniques/ (дата звернення 29.09.2025 року).

19. Mertz J., Nunes I., Della Toffola L., Selakovic M., Pradel M. Satisfying Increasing Performance Requirements With Caching at the Application Level. IEEE Soft-ware. 2021. Vol.38, Is.3. P. 87–95.

20. Soares F. L. et al. A React Style Guide Library for MUI Web Apps. 9th International HCI and UX Conference in Indonesia (CHIuXiD), Bali, Indonesia. 2023. P. 77-82.

21. React.JS Performance Optimization Techniques. URL: https://www.tatvasoft.com/outsourcing/2024/07/reactjs-performance-optimization.html (дата звернення 29.09.2025 року).

22. Sharma T., Gupta S., Singh U. R. Analyzing the difference between React.JS and Angular.JS. International Conference on Computational Intelligence, Communication Technology and Networking (CICTN), Ghaziabad, India. 2023. P. 37-42.

23. Jugo I., Kermek D., Meštrović A. Analysis and Evaluation of Web Application Performance Enhancement Techniques. URL: https://scispace.com/pdf/analysis-and-evaluation-of-web-application-performance-itp5r2i1yi.pdf (дата звернення 25.09.2025 року).

24. Mitawa S. A. Enhancing E-commerce Platforms through MERN Stack. Implementation Journal of Emerging Technologies and Innovative Research (JETIR). 2024. Vol.11, Is.5. P. k709-k716.

25. Google Developers. Core Web Vitals. 2021. URL: https://developers.google.com/search/docs/appearance/core-web-vitals (дата звернення 22.09.2025 року).

26. Google Developers. Lighthouse performance scoring. 2021. Accessed 25.3.2022. URL: https://web.dev/performance-scoring/ (дата звернення 25.09.2025 року).

ДОДАТКИ

Додаток А

Публікації за матеріалами магістерського дослідження

1) Давиденко Л.В., Давиденко В.В. Фреймворк web-додатку для торгівлі електроенергією з інтеграцією технологій блокчейн та інтернету речей. Матеріали VI Міжнародної науково-технічної конференції «Оптимальне керування електроустановками (ОКЕУ- 2025)», 22-23 жовтня 2025 року : збірник наукових праць. Вінниця : ВНТУ, 2025. С. 231-233.

ОКЕУ-2025

ОПТИМАЛЬНЕ КЕРУВАННЯ ЕЛЕКТРОУСТАНОВКАМИ (ОКЕУ-2025)

Матеріали VI міжнародної науково-технічної конференції
22-23 жовтня, 2025 р.

Збірник наукових праць

Вінниця, Україна

УДК 621.316.1
О-62

Видається за рішенням Вченої ради Вінницького національного технічного університету Міністерства освіти і науки України

Редакційна колегія: В. Біліченко, доктор технічних наук, професор
І. Єпіфанова, доктор економічних наук, професор
В. Комар, доктор технічних наук, професор

Рецензенти: В. Тептя, к.т.н., доцент
О. Рубаненко, к.т.н., професор

О-62 **Оптимальне керування електроустановками (ОКЕУ- 2025). Матеріали VI Міжнародної науково-технічної конференції, 22-23 жовтня 2025 року : збірник наукових праць [Електронний ресурс]. – Вінниця : ВНТУ, 2025. - (PDF, 267 с.)**

ISBN 978-617-8163-64-8

Збірник містить Матеріали VI МНТК за такими основними напрямками: методи і засоби оптимального керування електроустановками, електромеханічні системи, електротехнічні комплекси та керування ними, електротехнологічні процеси й енергозбереження.

Роботи подаються в авторській редакції. Редакційна колегія не несе відповідальності за достовірність інформації, яка наведена в роботах, та залишає за собою право не погоджуватися з думками авторів на розглянуті питання.

УДК 621.316.1

ISBN 978-617-8163-64-8

ЗМІСТ

УДК 621.311: 004.451.2.041

Л.В. Давиденко[1]
В.В. Давиденко[1]

ФРЕЙМВОРК WEB-ДОДАТКУ ДЛЯ ТОРГІВЛІ ЕЛЕКТРО-ЕНЕРГІЇ З ІНТЕГРАЦІЄЮ ТЕХНОЛОГІЙ БЛОКЧЕЙН ТА ІНТЕРНЕТУ РЕЧЕЙ

[1] Луцький національний технічний університет

Запропоновано спосіб взаємодії просмьюмера і споживача для реалізації механізму P2P-торгівлі, який базується на застосуванні технології блокчейн та інтернету речей та забезпечує безпечну комунікацію для обміну електроенергією та даними між учасниками платформи для торгівлі електроенергією без залучення третьої сторони. Технології інтернету речей використано для моніторингу виробництва і споживання електроенергії, передачі інформації, що генерується пристроями інтернету речей фізичного рівня, хмарним сервісам та обміну між вузлами мережі. Технологія блокчейн застосовується для підтримки зв'язку між пристроями інтернету речей, забезпечення надійності даних, конфіденційності та безпеки транзакцій. Інтерфейс користувача веб-додатку платформи торгівлі електроенергією забезпечує взаємодію просьюмера і споживача. Інформаційна панель користувацького інтерфейсу, фронтенд якого розроблено з використанням бібліотеки React.JS, забезпечує легкість її використання для проведення операцій з електроенергією. Переказ коштів у формі криптовалюти та контроль гаманця покупця забезпечує смарт-контракт, для розгортання якого використано публічну мережу блокчейну Ethereum. Для зв'язування облікового запису Ethereum з веб-додатком використовується блокчейн-гаманець MetaMask, який використовується для входу користувача та збереження анонімності під час запиту на операцію та забезпечує безпеку гаманця Ethereum за допомогою закритого ключа, автентифікації користувача та секретної фрази. Мережа блокчейн, розгорнута через абонентську мережу інтернет, забезпечує автентичність даних пристроїв інтернету речей за допомогою консенсусного алгоритму та безпечне збереження інформації про торгову діяльність, яка відображається в інтерфейсі користувача.

Ключові слова: P2P-торгівля, інтернет речей, блокчейн, транзакції, веб-інтерфейс користувача, бібліотека React.JS.

A method of interaction between a prosumer and a consumer is proposed to implement a peer-to-peer (P2P) trading mechanism, which is based on the use of blockchain technology and the Internet of Things and provides secure communication for the exchange of electricity and data between participants of the electricity trading platform without involving a third party. Internet of Things technologies are used to monitor electricity production and consumption, transfer information generated by physical-level Internet of Things devices to cloud services and exchange between network nodes. Blockchain technology is used to support communication between Internet of Things devices, ensure data reliability, confidentiality and security of transactions. The user interface of the electricity trading platform web application provides interaction between the prosumer and the consumer. The information panel of the user interface, the front end of which is developed using the React.JS library, ensures its ease of use for conducting electricity transactions. The transfer of funds in the form of cryptocurrency and control of the buyer's wallet is provided by a smart contract, for the deployment of which the public Ethereum blockchain network is used. To link the Ethereum account with the web application, the MetaMask blockchain wallet is used, which is used for user login and maintaining anonymity during the transaction request and ensures the security of the Ethereum wallet using a private key, user authentication and a secret phrase. The blockchain network, deployed over the subscriber Internet network, ensures the authenticity of data from IoT devices using a consensus algorithm and secures information about trading activities that is displayed in the user interface.

Keywords: *P2P trading, Internet of Things, blockchain, transactions, web user interface, React.JS library.*

Вступ

Декарбонізація, децентралізація та цифровізація є рушійними силами трансформації енергетичного сектору [1]. Зростання кількості відновлюваних джерел енергії (ВДЕ) та дерегуляція електромережі зумовлює процес децентралізації ринків електроенергії. Просьюмер, який має надлишок електроенергії, може зберігати її за допомогою систем накопичення енергії, експортувати в електромережу, продавати іншим споживачам. Впровадження торгівлі електроенергією між просьюмером та споживачем (P2P торгівлі) без посередника дозволяє обмінюватися електроенергією з ВДЕ у мікромережі, приносить певні вигоди просьюмерам, споживачам та оператору системи розподілу, оскільки полегшує балансування попиту і пропозиції на місцевому рівні. P2P торгівля передбачає розширену комунікацію для обміну електроенергією та даними. Проте вимагає вирішення питання інформаційної безпеки та конфіденційності, попередження витоку інформації, втрати даних тощо.

Метою роботи є розробити спосіб реалізації P2P торгівлі електроенергією, який би сприяв комунікації для обміну електроенергією та даними, а також безпечній взаємодії просьюмера і споживача без залучення третьої сторони.

Результати дослідження

Система P2P торгівлі містить фізичний рівень електромереж (фізичну інфраструктуру, яка потрібна мікромережі для забезпечення функції торгівлі енергією) і рівень інформаційно-комунікаційних технологій та контролю. З впровадженням розширеної інфраструктури вимірювання комунальні компанії, споживачі та виробники в розумних електромережах можуть взаємодіяти один з одним за допомогою обліку електроенергії на базі інтелектуальних лічильників із підтримкою двостороннього зв'язку. Розширена інфраструктура вимірювання, яка включає інтелектуальні лічильники, комунікаційні мережі, системи управління даними лічильників та засоби інтеграції зібраних даних у платформи/інтерфейси програмних додатків. Фізичний рівень генерує велику кількість даних, пов'язаних з електроенергією. Для моніторингу виробництва і споживання електроенергії використано технології інтернету речей (IoT). Пристрої IoT, якими в контексті електромережі є інтелектуальні лічильники, збирають дані щодо вимірювання електроенергії та передають дані хмарним службам в Інтернеті. Хмарні сервіси, такі як інтерфейс прикладного програмування (API), надають доступ до даних про споживання та генерацію електроенергії.

Незважаючи на численні переваги розширеної інфраструктури вимірювань, безпечна передача даних між пристроями є складним завданням. Для забезпечення безпеки, конфіденційності та надійності даних, що генеруються пристроями IoT, застосовується технологія блокчейн. Блокчейн - це децентралізований цифровий розподілений реєстр, який безпечно зберігає транзакції мережі однорангових вузлів та використовується для підключення великої кількості анонімних вузлів без необхідності використання центрального керуючого агента. Учасники блокчейну представлені як вузли, які співпрацюють для підтримки даних, що зберігаються в блокчейні, спільного захисту та ведення спільного запису транзакцій або цифрових подій, не покладаючись на будь-яку довірену сторону [1].

Розумні лічильники можуть бути безпосередньо підключені до мережі блокчейн через шлюз. Дані з лічильників містять ідентифікатори лічильників та іншу інформацію, пов'язану з комунальними послугами, відповідно до протоколу IEC 62056 [2]. Всі вузли мережі блокчейн зберігають копію перевірених даних, що гарантує довіру та автентичність інформації IoT. Блокчейн підтримує зв'язок між пристроями IoT, минаючи центральні сервери та дозволяючи швидше обмінюватися повідомленнями та даними між вузлами. Віртуальний рівень архітектури IoT-блокчейн для P2P-торгівлі енергією в мережі охоплює дані про енергію та транзакції торгівлі енергією, що генеруються пристроями IoT фізичного рівня. Впровадження технології блокчейн в платформу P2P-торгівлі забезпечує незмінні і прозорі записи транзакцій всім учасникам мережі [1, 3]. Автентифікація даних здійснюється за допомогою консенсусного алгоритму, а аналіз транзакцій – за допомогою смарт-контракту (розгорнутого на блокчейні фрагменту комп'ютерного коду, що реалізує виконання певних умов).

Модель P2P торгівлі електроенергією включає сервер IoT для передачі енергії між вузлами, а також публічну мережу блокчейну Ethereum для розгортання смарт-контракту для переказу коштів у формі криптовалюти та контролю гаманця покупця (якщо балансу недостатньо, це не дозволить покупцеві придбати електроенергію). Об'єкти інформації про замовлення та транзакції надсилаються в мережі блокчейн. Мережі блокчейн розгортаються через абонентську мережу інтернет. Блокчейн-гаманець використовується для входу користувача та збереження анонімності під час запиту на операцію. MetaMask забезпечує безпеку гаманця Ethereum за допомогою закритого ключа, автентифікації користувача та секретної фрази [4]. Сервер IoT дозволяє вузлам контролювати та моніторити вироблену енергію. Зв'язок інтелектуального вимірювання надсилається через TCP/IP.

Платформа для P2P-торгівлі енергією на основі Інтернету речей та блокчейну забезпечує взаємодію учасників за допомогою веб-інтерфейсу користувача. Для розробки фронтенду web-додатку використана бібліотека React.JS, що допомагає візуалізації, взаємодіючи з бекендом через API для отримання даних для відображення. Бібліотека JavaScript Web3 була використана як проміжний шар для сприяння комунікації між фронтендом та бекендом децентралізованого додатку (DAPP). MetaMask, гаманець Ethereum, що використовується для зв'язування облікового запису Ethereum з веб-додатком, було встановлено як розширення для Google Chrome. Сервер, на якому розміщено блокчейн Ethereum та інтерфейс користувача на основі React.JS, це віддалений сервер, налаштований для торгової діяльності. Конфігурація користувацького інтерфейсу має інформаційну панель із трьома розділами: торгівля,

гаманець і статистика, посилання для купівлі, продажу, перегляду поточних лістингів та транзакцій, схвалених шляхом майнінгу після завершення передачі потужності, що робить всю систему легкою для використання та розуміння. Веб-сайт дозволяє користувачам входити в систему та отримувати спеціальний ключ для керування. Після входу вони можуть здійснювати операції з електроенергією, такі як запити на купівлю або продаж, передавати енергію, вимірювати передану енергію та здійснювати платежі. Також є опції відмови покупцеві, якщо енергія недоступна для продажу. Вся торгова діяльність в інтерфейсі користувача записується на блокчейн-сервері.

Висновки

Платформа для P2P-торгівлі енергією на основі Інтернету речей та блокчейну забезпечує безпечну комунікацію для обміну електроенергією та даними між учасниками. У P2P-системах блокчейн фіксує транзакції миттєво та автономно, усуваючи потребу в посередниках та надаючи енергетичним трейдерам можливість приймати рішення на основі своїх уподобань. Просьюмери та споживачі можуть вводити свої налаштування через інтерфейс користувача, тоді як інтелектуальні лічильники надають дані про попит на енергію та надлишок енергії. Застосування блокчейну сприяє розвитку безпечної, автономної та надійної цифрової інфраструктури, яка здатна забезпечити координацію підключених до мережі енергетичних активів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Dorokhova M. Blockchain-Supported Framework for Charging Management of Electric Vehicles / Dorokhova M., Vianin J., Alder J.-M., Ballif C., Wyrsch N., Wannier D. A. // Energies. – 2021. - №14. – ID7144. https://doi.org/10.3390/en14217144

2. Appasani B. Blockchain-Enabled Smart Grid Applications: Architecture, Challenges, and Solutions / Appasani B., Mishra S.K., Jha A.V., Mishra S.K., Enescu F.M., Sorlei I.S., Bîrleanu F.G., Takorabet N., Thounthong P., Bizon N. // Sustainability/ - 2022. - 14. - ID 8801. https://doi.org/10.3390/su14148801

3. Wang Y. Distributed meter data aggregation framework based on Blockchain and homomorphic encryption. / Wang Y., Luo F., Dong Z., Tong Z., Qiao, Y. // IET Cyber-Physical Systems: Theory & Applications. – 2019. - №4. – Pp. 30-37. https://doi.org/10.1049/iet-cps.2018.5054

4. Al-Zoubi A. An Interactive IoT-Blockchain System for Big Data Management / Al-Zoubi A., Saadeddin T., Dmour M., Adi L. // 2022 4th IEEE Middle East and North Africa COMMunications Conference (MENACOMM), Amman, Jordan. - 2022. - pp. 71-76. https://doi.org/10.1109/MENACOMM57252.2022.9998263.

Давиденко Людмила Валеріївна – д-р техн. наук, професор кафедри електричної інженерії, Луцький національний технічних університет, м. Луцьк, e-mail: l.davydenko@lutsk-ntu.com.ua

Давиденко Віктор Володимирович – студент групи КНм-21, факультет комп'ютерно-інтегрованих технологій, Луцький національний технічний університет, м. Луцьк, e-mail: davydenko.v2211.122.24@lntu.edu.ua

Науковий керівник: ***Давиденко Людмила Валеріївна*** — д-р техн. наук, професор кафедри електричної інженерії, Луцький національний технічних університет, м. Луцьк

Davydenko Liudmyla V. – Dr. Sc. (Eng.), Professor of Electrical Engineering, Lutsk National Technical University, Lutsk, email : l.davydenko@lutsk-ntu.com.ua

Davydenko Viktor V. — Faculty of Computer Integrated Technologies, Lutsk National Technical University, Lutsk, email : davydenko.v2211.122.24@lntu.edu.ua

Supervisor: ***Davydenko Liudmyla V.*** – Dr. Sc. (Eng.), Professor of Electrical Engineering, Lutsk National Technical University, Lutsk.

2) Давиденко В.В. Технології оптимізації Front-end web-додатків на React.js для підвищення продуктивності. Студентський науковий вісник. Луцький національний технічний університет. Луцьк: Видавництво «Вежа-Друк», 2025. Вип. 54.

УДК 004.5:428:051
Давиденко В.В., ст. гр. КНм-21
Науковий керівник: к.т.н., доц. Лук'янчук Ю.А.

ТЕХНОЛОГІЇ ОПТИМІЗАЦІЇ FRONT-END WEB-ДОДАТКІВ НА REACT.JS ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ

У статті окреслено принципи підвищення продуктивності веб-додатку шляхом застосування стратегій оптимізації. Зазначено важливість застосування методів оптимізації для front-end інтернет-магазину. Охарактеризовано принципи застосування стратегій оптимізації React.js, які направлені на оптимізацію структури пакету та наявних зображень для зменшення розмірів веб-додатку та збільшення швидкості завантаження його компонент; а також зменшення повторних обчислень для оптимізації використання ресурсів.
Ключові слова: розбиття коду, ліниве завантаження, оптимізація зображень, мемоізація.

Davydenko V.

OPTIMIZATION TECHNOLOGIES FOR FRONT-END WEB APPLICATIONS ON REACT.JS TO INCREASE PRODUCTIVITY

The article outlines the principles of increasing the productivity of a web application by applying optimization strategies. The importance of applying optimization methods for the front-end of an online store is noted. The principles of applying React.js optimization strategies are described, which are aimed at optimizing the package structure and existing images to reduce the size of the project's web application and increase the speed of loading its components; as well as reducing recalculations to optimize resource usage.
Keywords: code splitting, lazy loading, image optimization, memorization.

Постановка проблеми. Електронна комерція перетворилася на один із ключових елементів сучасного бізнесу, забезпечуючи споживачам у всьому світі зручність і легкий доступ до товарів та послуг. Front-end відповідає за створення інтерфейсу веб-сайту або веб-додатка електронної комерції, тобто, відіграє важливу роль у забезпеченні клієнтської частини інтернет-магазину, яка має приваблювати користувачів функціональністю, простотою та зручністю. Тому front-end повинен вирізнятися інтуїтивно зрозумілим дизайном, естетичною привабливістю, високим рівнем залученості користувачів та ефективною конверсією. Покращення інтерфейсу, функціональності та зручності веб-сайтів за допомогою сучасних фреймворків стало важливою складовою розвитку електронної комерції. У сучасних умовах, коли веб-додатки стають дедалі складнішими й інтерактивнішими, користувачі очікують від них швидкої роботи, плавності та миттєвого відгуку. Саме тому у веб-розробці з'явилася тенденція до створення застосунків, які поєднують багатий функціонал із високою ефективністю роботи.

Аналіз останніх досліджень і публікацій. JavaScript відіграє ключову роль у front-end розробці, забезпечуючи динамічну взаємодію користувачів із веб-додатками та підтримуючи реалізацію складних функцій [1]. Серед найпопулярніших JavaScript-фреймворків, що використовуються для створення інтерактивних веб-застосунків, вирізняються React, Angular та Vue.

Відкрита бібліотека React.js є однією з найпоширеніших технологій для розробки сучасних динамічних веб-додатків і користувацьких інтерфейсів у сфері електронної комерції [1–4]. Завдяки використанню базових можливостей JavaScript, React має низку переваг у побудові front-end рішень та дозволяє ефективно створювати сучасні, привабливі та зручні інтерфейси. Його компонентна архітектура та віртуальна модель об'єктів документа (Virtual Document Object Model, DOM) забезпечують швидке відтворення та оновлення елементів сторінки, високу адаптивність і плавну взаємодію з користувачем, що позитивно впливає на залученість і задоволення клієнтів [2]. Водночас, у процесі розробки середніх і великих веб-додатків можуть виникати певні проблеми продуктивності, пов'язані з неефективністю або вузькими місцями в архітектурі [3], що може призводити до зниження загальної швидкодії системи [2].

Цілі статті. Метою дослідження є узагальнення принципів підвищення продуктивності веб-додатку інтернет-магазину шляхом застосування методів оптимізації React.js для front-end.

Виклад основного матеріалу дослідження. Із розвитком та зростанням популярності багатофункціональних клієнтських інтерфейсів користувачі дедалі більше очікують миттєвої реакції системи та безперебійної взаємодії для задоволення користувацького досвіду [4]. Метою розробки ефективного веб-додатку є не лише створення привабливого інтерфейсу, а й досягнення адаптивності, сумісності з різними браузерами та пристроями, вибір правильних фреймворків та методів оптимізації для досягнення високої продуктивності з точки зору швидкості завантаження та інтерактивності, часу відгуку та взаємодії користувача, зручності користування. Із підвищенням складності сучасних веб-додатків вони дедалі частіше містять великі пакети JavaScript, зображення високої роздільної якості та динамічний контент. Тому однією з важливих вимог якості під час веб-розробки є забезпечення продуктивності, тобто, здатності веб-додатків швидко завантажуватися, оперативно реагувати на дії користувача та забезпечувати комфортний досвід взаємодії.

Збільшення розміру та складності веб-додатків може призвести до повільнішого завантаження сторінок, підвищеного споживання пам'яті та зниження загальної продуктивності, особливо на пристроях із низькою потужністю чи при повільному інтернет-з'єднанні [4, 5]. Однією з основних проблем у React.js-додатках для електронної комерції є надмірно великий розмір пакетів, що безпосередньо впливає на користувацький досвід і ключові бізнес-показники. Додатки з великим обсягом коду часто демонструють затримки під час початкового рендерингу і можуть створювати проблеми з плавністю роботи інтерфейсу користувача під час подальших оновлень. Надмірно складна архітектура компонентів не лише знижує продуктивність, але й ускладнює обслуговування та масштабування додатку. Перевантаженість та надлишковість коду, тривалий рендеринг і ресурсомісткі операції негативно позначаються на взаємодії користувачів із додатком [2]. У веб-додатках із високим трафіком або корпоративних системах такі вузькі місця проявляються у вигляді збільшеного

часу завантаження, затримок при взаємодії та зниження швидкості реакції, що, у свою чергу, негативно впливає на рівень залученості користувачів.

Однією з головних причин зниження продуктивності у React-застосунках є повторний рендеринг компонентів [4]. Вплив на продуктивність та швидкодію виконання значною мірою визначається внутрішньою структурою фреймворку та тим, як він обробляє маніпуляції й оновлення DOM – деревоподібного представлення елементів веб-сторінки. Хоча React розроблений для ефективного виконання цих операцій, надмірні повторні рендери, особливо між глибоко вкладеними компонентами, можуть суттєво знижувати швидкість реагування інтерфейсу. Зі збільшенням масштабу застосунків все більшу роль відіграє те, як саме React-компоненти керують процесом рендерингу та поширенням стану в межах дерева компонентів [4]. DOM як структура має властивість уповільнювати оновлення в міру збільшення кількості елементів, що особливо помітно у великих додатках із численними вузлами. Оскільки DOM є деревоподібною структурою, будь-яке оновлення вимагає повторного обчислення та перефарбовування значної частини інтерфейсу, що робить процес оновлення повільним. Під час повторного рендерингу React.js виконує логіку порівняння станів, визначаючи різницю між поточною та попередньою версіями віртуальної DOM, після чого застосовує лише необхідні зміни до реального DOM [2]. Віртуальна DOM виступає проміжним шаром абстракції між додатком і реальною DOM, що дозволяє уникнути надмірних оновлень і мінімізувати кількість обчислень. Такий підхід значно прискорює рендеринг, особливо в динамічних застосунках, де інтерфейс часто змінюється [2]. Оптимізація роботи віртуальної DOM є одним із ключових механізмів підвищення продуктивності, вбудованих в React.js, який усуває потребу в безпосередній взаємодії з фактичною DOM [2, 3]. Віртуальна DOM займає центральне місце в React архітектурі та забезпечує базову основу для підвищення ефективності рендерингу. Водночас React.js також має низку вбудованих інструментів, які додатково оптимізують роботу застосунку та покращують його швидкодію.

Зменшення розміру пакета є важливим чинником для підвищення швидкодії, покращення реакції та чутливості React-додатку. Менші компоненти легше оптимізувати, тестувати та підтримувати [4]. Одним із ефективних підходів є розділення коду, яке полягає у поділі основного коду програми на менші пакети, що завантажуються за запитом під час навігації користувача по додатку. Ця стратегія дозволяє браузеру динамічно підвантажувати лише ті частини коду, які потрібні в конкретний момент [2, 5], скорочуючи час початкового завантаження та оптимізуючи продуктивність React-додатку, особливо у великих проєктах, а також сприяє більш ефективному використанню ресурсів [3]. Розділення коду відповідає сучасним вимогам веб-продуктивності, забезпечуючи швидке завантаження, адаптивну інтерактивність та раціональне використання мережевих ресурсів.

Метод, при якому пакети завантажуються лише тоді, коли потрібні їхні компоненти, називається відкладеним (або лінивим) завантаженням. Цей підхід надає пріоритет критично важливим ресурсам [2] і відкладає завантаження

Луцький національний технічний університет, «Студентський науковий вісник», Випуск 54 – 2025

некритичних компонентів до моменту їх реальної потреби, що зменшує розмір початкового пакету. Завдяки відкладеному завантаженню на початковому етапі передається менше даних, що значно скорочує час, необхідний для переходу програми в інтерактивний режим [2], а також підвищує швидкість завантаження сторінки [3, 5], очікуючи на завантаження несуттєвих компонентів або ресурсів лише за потреби. Це не лише пришвидшує початкове завантаження, але й допомагає користувачеві швидше ознайомитися з додатком та адаптуватися до роботи з програмою [2]. Некритичні компоненти завантажуються лише тоді, коли вони стають необхідними, що дозволяє динамічно підвантажувати функціонал замість включення його у стартовий пакет JavaScript [4]. Таким чином, механізм відкладеного завантаження оптимізує процес завантаження, зменшує час старту сторінки, економить пропускну здатність і покращує взаємодію з користувачем і користувацький досвід, забезпечуючи швидке відображення критично важливих елементів інтерфейсу, тоді як менш критичні елементи завантажуються у фоновому режимі [4].

Зображення становлять значну частину загального обсягу більшості React-додатків, а процес їх завантаження істотно впливає на продуктивність. Завантаження великих і неоптимізованих зображень у React-додатках може погіршувати швидкодію додатку та його загальну продуктивність. Оптимізація зображень у додатках React.js дозволяє підвищити їхню веб-продуктивність та покращити загальний користувацький досвід [6].

Першим етапом оптимізації зображень є їхнє перетворення у формат, який забезпечує мінімальний розмір файлу при збереженні високої якості. Хоча традиційно використовуються формати JPEG і PNG, сучасні формати, такі як WebP і AVIF, забезпечують значно ефективніше стиснення та пропонують оптимальний розмір файлу при збереженні високої якості [6]. Це сприяє швидшому завантаженню сторінок і зменшенню використання пропускної здатності, що особливо важливо для мобільних пристроїв. Формат WebP, розроблений Google спеціально для Інтернету, забезпечує менші файли зображення із якістю, порівнянною з JPEG. AVIF пропонує ще більш ефективне стиснення при збереженні аналогічної якості, пропонуючи зазвичай менший розмір файлів, ніж WebP. Однак підтримка AVIF у браузерах поки що обмежена, тоді як WebP широко підтримується у всіх сучасних браузерах. Поєднання прогресивних методів завантаження зображень із сучасними форматами, такими як WebP та AVIF, дозволяє забезпечити оптимальний візуальний контент і покращити продуктивність веб-додатків.

За допомогою HTML та відповідних атрибутів можна реалізувати адаптивне завантаження зображень, коли для різних розмірів екрану застосовуються різні версії одного й того ж зображення. Основним недоліком цього підходу є необхідність створення кількох версій одного зображення для вибору браузером. Однак сучасні інструменти та бібліотеки для оптимізації зображень дозволяють автоматизувати цей процес.

Наявність великої кількості зображень може знизити продуктивність React-додатку та уповільнити його завантаження, оскільки DOM рендерить кожне

зображення перед відображенням інтерфейсу користувача. Щоб уникнути цього, зображення, відео та інші додаткові компоненти можна завантажувати асинхронно, виділяючи пріоритетні ресурси для взаємодії користувача. Для підвищення продуктивності рекомендується застосовувати відкладене завантаження (lazy loading) не лише до компонентів, але й до медіа-ресурсів, особливо великих зображень і відео [4]. При відкладеному завантаженні зображення не відображаються до тих пір, доки вони не з'являться у видимій області екрану користувача, що скорочує початковий час завантаження та підвищує продуктивність сторінки. Зображення завантажуються не всі одночасно, а поступово, по мірі прокручування користувачем сторінки вниз до інших частин. Для реалізації такого завантаження широко застосовується Intersection Observer API – вбудований API браузер – інструмент, який визначає, коли елемент потрапляє в область перегляду. Такий підхід гарантує, що додаток завантажує лише той контент, з яким користувач активно взаємодіє, що значно підвищує продуктивність веб-додатку [4].

Завдяки сервісам для оптимізації зображень, таким як Uploadcare, процес оптимізації відбувається автоматично під час виконання, без потреби в ручному втручанні. Використання мережі доставки контенту (CDN) у поєднанні із сервісами оптимізації забезпечує ефективну обробку зображень у реальному часі під час завантаження веб-додатку та знижує навантаження на сервери, оскільки контент подається через глобально розподілену мережу серверів. CDN дозволяє ефективно кешувати файли та доставляти їх користувачам із найближчого сервера, а одночасна автоматична обробка зображень – включно зі стисненням, зміною формату та адаптацією розміру – забезпечує їхню оптимізацію. Це значно скорочує час завантаження зображень і покращує продуктивність веб-додатків.

Ефективним підходом є поєднання кількох методів, синергійна робота яких забезпечує зменшення часу завантаження зображень і покращення їхньої візуальної якості. Одночасне використання сучасних форматів зображень, таких як WebP та AVIF, адаптивних зображень, лінивого завантаження та спеціалізованих бібліотек React забезпечує швидке завантаження контенту та швидку роботу веб-додатку з підтримкою високої якості зображень і підвищує загальну продуктивність веб-додатку.

Дослідження показують, що 13% проблем із продуктивністю JavaScript були спричинені повторними операціями [7]. Однією з стратегій оптимізації є запам'ятовування (мемоізація) - техніка, яка працює на концепції повторного використання та кешування тих самих вхідних даних для викликів функцій, що потребують інтенсивних обчислень [2]. Центральний принцип мемоізації полягає у збереженні результатів обчислень на рівні програми для подальшого використання. Результати зберігаються за допомогою параметрів обчислення. Якщо результат знайдено за заданими параметрами, він повертається; в іншому випадку обчислення виконується та кешується [7]. Тобто, виклик непотрібних повторних рендерингів не здійснюється, якщо кешований результат однаковий для останніх вхідних аргументів. Шляхом кешування результатів ресурсомістких викликів функцій забезпечується покращення продуктивність

Луцький національний технічний університет, «Студентський науковий вісник», Випуск 54 – 2025

застосунку. Слід зазначити, що оптимізація віртуального DOM означає швидше оновлення інтерфейсу, тоді як мемоізація дозволяє значно зменшити надлишкові обчислення [3], уникнути повторного рендерингу та повторних обчислень [2], що призводить до значного підвищення продуктивності [7]. Цей метод застосовується у front-end у вигляді зберігання результатів обчислень у пам'яті браузера. Мемоізацію рекомендується використовувати для ресурсомістких обчислень, таких як операції сортування та фільтрації під час рендерингу, щоб уникнути повторних обчислень та повторних рендерів, зумовлених змінами в посиланнях на об'єкти. Мемоізація вважається надлишковою для простих обчислень у функціях [2].

Висновки. React.js за замовчуванням базується на механізмі, який дозволяє створювати швидкі інтерактивні веб-додатки. Однак, коли додаток (особливо великого розміру) масштабується, важливе значення має оптимізація продуктивності. Існує широкий спектр стратегій оптимізації, таких як розділення коду, відкладене завантаження компонент, запам'ятовування, кешування функцій та поверхневе порівняння для уникнення непотрібних повторних рендерів. При цьому, кожна стратегія стосується певного аспекту проблем продуктивності. У поєднанні вони створюють цілісну основу для створення високопродуктивних React-додатків.

Перелік джерел посилання

1. Kumar S. Optimizing Frontend Performance with React and JavaScript: Techniques and Impact Analysis. International Journal Of Novel Research And Development. 2024. Vol. 9. Is. 7. Pp. b866- b869

2. Mondal S. Enhancing React Application Performance: Proven Strategies and Best Practices. International Research Journal of Engineering and Technology (IRJET). 2024. Vol. 11. Is. 12. Pp. 309- 312.

3. Jani Y. The Critical Importance of React Performance Optimization: Strategies for Success. North American Journal of Engineering and Research. 2020. Vol. 1. Is. 1. Pp. 109-112.

4. Toprak A., Toprak F. S. Improving and Optimizing React Web Applications: Strategies and Techniques. 5th International Congress of Engineering and Natural Sciences (ICENSS 2025), Ankara, Turkey. 2025. Pp. 1-17

5. Jain V. Optimizing WEB performance with LAZY LOADING and CODE SPLITTING. International Journal of Core Engineering & Management. 2022. Vol.7. Is.3. Pp. 193- 199.

6. OkpukoroT. J. React image optimization: Techniques to speed up your web apps. Uploadcare. 2025. URL: https://uploadcare.com/react-image-optimization-techniques/

7. Mertz J., Nunes I., Della T.L., Selakovic M., Pradel M. Satisfying In-creasing Performance Requirements With Caching at the Application Level. IEEE Soft-ware. 2021. Vol.38. Is. 3. Pp. 87–95.

Додаток Б

Апробація результатів магістерського дослідження

Відбувся публічний виступ та демонстрація результатів на секційному засіданні VI Міжнародної науково-технічної конференції «Оптимальне керування електроустановками (ОКЕУ-2025)» 22-23 жовтня 2025 року, Вінниця, Вінницький національний технічний університет.

Міністерство освіти і науки України
Вінницький національний технічний університет

СЕРТИФІКАТ №59-25

засвідчує, що

Віктор Давиденко

взяв участь в VI Міжнародній науково-технічній конференції
«ОПТИМАЛЬНЕ КЕРУВАННЯ ЕЛЕКТРОУСТАНОВКАМИ (ОКЕУ-2025)»
в період з 22 по 23 жовтня 2025 року, **30 ГОДИН – 1 КРЕДИТ ECTS**, з напрямків:

– теоретичні проблеми оптимального керування;
– електроенергетичні системи та керування ними;
– електромеханічні системи, електротехнічні комплекси та керування ними;
– електротехнологічні процеси й енергозбереження;
– відновлювані та низьковуглецеві джерела енергії в електроенергетичних системах.

Ректор ВНТУ **Віктор БІЛІЧЕНКО**

м. Вінниця, 23 жовтня 2025 року

Додаток В

Лістинг програмного коду компонента Image.jsx

```
import images from "imports/ImagesImport";
import React from "react";
const Image = ({ src, className, width, height, alt, onClick, loading,
title,}) => {
  const handleImageError = (event) => {
    event.target.src = images["defaultImg"];
  };
  const img = typeof src === "object" ? src : null;
  const big = img?.conversions?.big?.url;
  const preview = img?.conversions?.preview?.url;
  const thumb = img?.conversions?.thumb?.url;
  const mainUrl =
    img?.url || (typeof src === "string" ? src : images["defaultImg"]);
  const hasSrcset = big || preview || thumb;
  const srcset = [
    big ? `${big} 1200w` : null,
    preview ? `${preview} 800w` : null,
    thumb ? `${thumb} 400w` : null,
  ]
    .filter(Boolean)
    .join(", ");
  const sizes = `
    (max-width: 480px) 400px,
    (max-width: 1024px) 800px,
    1200px
  `;
  if (!hasSrcset) {
    return (
      <img className={className} src={mainUrl} width={width}
height={height} loading={loading} onClick={onClick} alt={alt}
title={title} onError={handleImageError} />
    );
  }
  return (
    <picture>
      {big && (
        <source srcSet={big} media="(min-width: 1025px)"
type="image/webp" />
      )}
      {preview && (
        <source srcSet={preview} media="(min-width: 481px)"
type="image/webp" />
      )}
      {thumb && (
        <source srcSet={thumb} media="(max-width: 480px)"
type="image/webp" />
      )}
```

```
      <img className={className} src={mainUrl} srcSet={srcset}
sizes={sizes} width={width} height={height} loading={loading}
onClick={onClick} alt={alt} title={title} onError={handleImageError} />
    </picture>
  );
};
export default Image;
```

Додаток Г

Лістинг програмного коду компонента сторінки продукту ProductPage.jsx

```
import Breadcrumbs from "components/base/Breadcrumbs";
import Preloader from "components/base/Preloader";
import RatingStars from "components/base/RatingStars";
import React, { useEffect, Suspense, useMemo } from "react";
import { useParams } from "react-router-dom";
import { useProduct } from "context/ProductContext";
const PopularProducts = React.lazy(() =>
  import("components/Home/PopularProducts")
);
const ProductContent = React.lazy(() =>
  import("components/Product/ProductContent/ProductContent")
);
const ProductOptions = React.lazy(() =>
  import("components/Product/ProductOptions")
);
const ProductPhotos = React.lazy(() =>
  import("components/Product/ProductPhotos")
);
const ProductPage = () => {
  const { slug } = useParams();
  const { state, fetchProduct } = useProduct();
  const { data: productData, isLoading } = state;
  useEffect(() => {
    fetchProduct(slug);
  }, [slug, fetchProduct]);
  const images = useMemo(() => productData?.data?.images || [],
[productData]);
  const name = useMemo(() => productData?.data?.name || "Product",
[productData]);
  const sku = useMemo(() => productData?.data?.sku || "", [productData]);
  const ratingValue = useMemo(
    () => productData?.data ? parseFloat(productData.data.product.rating)
: 0,
    [productData]
  );
  const breadcrumbsItem = useMemo(
    () => ({ name }),
    [name]
  );
  return (
    <>
      {isLoading && <Preloader />}

      {productData?.data && (
        <div className="container-vertical page-container product">
          <section className="container-vertical outer-container
product__wrapper">
```

```
            <div className="container product__breadcrumbs">
              <Breadcrumbs item={breadcrumbsItem} />
            </div>
            <div className="container product__top">
              <div className="product__header">
                <h2 className="title product__title">{name}</h2>
                <span className="text product__code">
                  <span className="product__code_SKU">SKU:</span> {sku}
                </span>
                <div className="container-horizontal rate">
                  <RatingStars value={ratingValue} />
                  <span className="text rate__text">{ratingValue}</span>
                </div>
              </div>
              {/* Lazy-loaded blocks */}
              <Suspense fallback={<Preloader />}>
                <ProductPhotos items={images} />
                <ProductOptions item={productData} />
              </Suspense>
            </div>
          </section>
          <section className="container-vertical product__middle">
            <Suspense fallback={<Preloader />}>
              <ProductContent item={productData} />
            </Suspense>
          </section>
          <Suspense fallback={<Preloader />}>
            <PopularProducts
              title="You can like this"
              buttons={false}
              className="product__recommendations"
            />
          </Suspense>
        </div>
      )}
    </>
  );
};
export default ProductPage;
```