БАКАЛАВРСЬКА РОБОТА

БР.ІП – 23.00.00.000 ПЗ

Група ІП-22-1

Славніков Нікіта

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Славніков Нікіта Сергійович

(прізвище, ім'я, по батькові)

УДК 004.942
(індекс)

БАКАЛАВРСЬКА РОБОТА

Методи оптимізації продуктивності web-сервісів
(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня _____Славніков Н.С._____
(підпис, ініціали та прізвище здобувача)

Науковий керівник ____Ксенич Андрій Іванович, доцент, к.т.н._____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

_доц._____Бандура В.В.__
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доцент. В.В. Бандура
“ ____ ” _____2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Славніков Нікіта Сергійович

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) "Методи оптимізації продуктивності web-сервісів"

керівник проекту (роботи) доцент, к.т.н. Ксенич А.І.

затверджені наказом вищого навчального закладу від “ 21_ ” _квітня 2026 р. № ___179/7____

2. Строк подання студентом проекту (роботи) 10 червня 2026 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз вимог до програмного забезпечення
2. Аналіз вимог і постановка задачі
3. Проектування системи
4. Реалізація проекту
5. Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Тест продуктивності до оптимізації (рис. 3.1, ст. 60)

2 Тест продуктивності після оптимізації (рис. 3.2, ст. 60)

3 Навантажувальне тестування з кількістю користувачів від 1 до 10 000 (рис. 3.5, ст. 62)

4 Основні проблеми при застосуванні WPO на практиці (рис. 3.6, ст. 83)

5 Розподіл оцінок пріоритетності WPO (рис. 3.7, ст. 86)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

2. Дата видачі завдання **2026 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Визначення та обгрунтування теми роботи	17.03.2026	виконано
2	Огляд існуючих концепцій, рішень та сервісів в даній області	21.04.2026	виконано
3	Побудова моделі або алгоритму власного рішення	01.05.2026	виконано
4	Документування реалізації власного оригінального рішення вибраними засобами	21.05.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	09.06.2026	виконано

Студент _____Славніков Н.С.
(підпис)

Керівник роботи _____Ксенич А.І.
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 91 сторінка, 8 рисунків, 7 таблиць, список використаних джерел із 40 найменувань.

Метою роботи є дослідження методів оптимізації продуктивності web-сервісів та аналіз підходів до підвищення швидкодії, масштабованості й ефективності сучасних веб-застосунків.

Об'єкт дослідження: web-сервіси та веб-застосунки.

Предмет дослідження: методи, моделі та інструменти оптимізації продуктивності web-сервісів, включаючи підходи до оптимізації клієнтської та серверної частин, а також засоби моніторингу і тестування.

Результати дослідження: проведено аналіз факторів, що впливають на продуктивність web-сервісів, досліджено сучасні методи оптимізації, розглянуто ключові метрики продуктивності та інструменти їх вимірювання.

У першому розділі розглянуто теоретичні основи продуктивності web-сервісів, визначено основні поняття, фактори впливу та методи оптимізації клієнтської частини.

У другому розділі досліджено методи та інструменти оптимізації продуктивності, зокрема підходи до збору та аналізу даних, а також ключові метрики оцінювання продуктивності веб-застосунків.

У третьому розділі розглянуто практичні аспекти реалізації оптимізаційних рішень, використання сучасних інструментів моніторингу, а також проведено тестування та аналіз ефективності підвищення продуктивності системи.

Висновок: застосування сучасних методів оптимізації продуктивності дозволяє значно підвищити швидкодію web-сервісів, покращити користувацький досвід і забезпечити ефективне використання ресурсів системи.

КЛЮЧОВІ СЛОВА: WEB-СЕРВІСИ, ПРОДУКТИВНІСТЬ, ОПТИМІЗАЦІЯ, FRONTEND, BACKEND, МОНІТОРИНГ, НАВАНТАЖЕННЯ, CI/CD, МАСШТАБОВАНІСТЬ.

ANNOTATION

The bachelor's thesis contains of 91 pages,8 figures, 7 tables, and a reference list with 40 sources.

The aim of the work is to study methods for optimizing the performance of web services and to analyze approaches to improving the speed, scalability, and efficiency of modern web applications.

Research object: web services and web applications.

Research subject: methods, models, and tools for optimizing web service performance, including approaches to frontend and backend optimization, as well as monitoring and testing tools.

Research results: an analysis of factors affecting web service performance was conducted, modern optimization methods were studied, key performance metrics and measurement tools were reviewed, and the effectiveness of optimization solutions was evaluated.

The first section describes the theoretical foundations of web service performance, including key concepts, influencing factors, and frontend optimization methods.

The second section analyzes methods and tools for performance optimization, including data collection and analysis approaches, as well as key performance metrics for web applications.

The third section covers the practical implementation of optimization solutions, the use of modern monitoring tools, and the testing and evaluation of system performance improvements.

Conclusion: the application of modern performance optimization methods significantly improves web service speed, reduces response time, enhances user experience, and ensures efficient use of system resources.

KEY WORDS: WEB SERVICES, PERFORMANCE, OPTIMIZATION, FRONTEND, BACKEND, MONITORING, LOAD TESTING, CI/CD, SCALABILITY.

ЗМІСТ

ВСТУП

У сучасному світі веб-технології відіграють ключову роль у функціонуванні інформаційних систем, забезпечуючи доступ до даних, обробку запитів та взаємодію користувачів із сервісами в режимі реального часу. Зростання кількості користувачів, обсягів даних та складності програмних систем призводить до підвищених вимог щодо продуктивності веб-сервісів. Низька швидкодія, затримки у відповіді сервера або неефективне використання ресурсів можуть суттєво вплинути на якість користувацького досвіду та конкурентоспроможність програмного продукту. Саме тому оптимізація продуктивності веб-сервісів є одним із ключових завдань сучасної програмної інженерії.

Актуальність теми зумовлена необхідністю забезпечення високої швидкодії, масштабованості та стабільності веб-сервісів в умовах зростаючого навантаження та динамічного розвитку інформаційних технологій. Існуючі підходи до розробки веб-застосунків не завжди враховують вимоги до ефективного використання ресурсів, що призводить до зниження продуктивності систем. У зв’язку з цим виникає потреба в дослідженні сучасних методів оптимізації, інструментів моніторингу та аналізу продуктивності, а також впровадженні практичних рішень для підвищення ефективності роботи веб-сервісів.

Метою роботи є дослідження та застосування методів оптимізації продуктивності веб-сервісів, що забезпечують зменшення часу відгуку, підвищення пропускної здатності системи та ефективне використання обчислювальних ресурсів.

Завданнями дослідження є аналіз теоретичних основ продуктивності веб-сервісів, визначення основних факторів, що впливають на швидкодію, дослідження методів оптимізації фронтенду та бекенду, аналіз сучасних метрик продуктивності, вивчення інструментів моніторингу та профілювання, а також реалізація практичних підходів до оптимізації та оцінка їх ефективності.

Об’єктом дослідження є процеси функціонування веб-сервісів у сучасних інформаційних системах.

Предметом дослідження є методи, моделі та інструменти оптимізації продуктивності веб-сервісів, включаючи кешування, балансування навантаження, оптимізацію баз даних, мінімізацію ресурсів фронтенду та використання сучасних протоколів передачі даних.

Методи дослідження включають аналіз наукових і технічних джерел, порівняльний аналіз існуючих підходів до оптимізації продуктивності, моделювання процесів обробки запитів, експериментальне дослідження ефективності застосованих методів, а також аналіз отриманих результатів.

У роботі передбачається дослідити сучасні підходи до оптимізації продуктивності веб-сервісів, включаючи використання кешування, CDN, асинхронної обробки запитів, оптимізацію запитів до баз даних, застосування сучасних інструментів моніторингу та тестування продуктивності. Особлива увага приділятиметься забезпеченню масштабованості, зменшенню затримок та підвищенню ефективності використання ресурсів.

Бакалаврська робота містить 91 сторінок, 8 рисунків і 7 таблиць, 3 розділи, висновки та список використаних джерел.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ WEB-СЕРВІСІВ

1.1 Основні поняття продуктивності web-сервісів

Оптимізація продуктивності веб-сайтів (WPO) була широко досліджена в останні роки, охоплюючи методи на кількох рівнях веб-стеку. У ранніх фундаментальних оглядах зазначається, що сучасні веб-сторінки різко зросли в розмірі (наприклад, зростання середнього розміру мобільної сторінки на 600% за 2012-2022 роки [1]), що зумовлює потребу в систематичній оптимізації . У цьому розділі розглядається ключова література з WPO, зосереджуючись на комплексних оглядах методів оптимізації, дослідженнях, що досліджують вплив продуктивності на користувацький досвід та сприйняту якість, а також дослідженнях практик та інструментів розробки, що підтримують впровадження WPO. Вепсалайнен *та ін.* надають комплексний огляд сучасних методів WPO, класифікуючи як низькорівневі, так і високорівневі підходи [2]. Вони виділяють класичні методи, такі як кешування, стиснення та розвантаження клієнт-серверної роботи, а також новіші практики розробки, які неявно підвищують продуктивність. Наприклад, *прогресивне вдосконалення* та *розробка з використанням HTML* надають пріоритет завантаженню мінімального контенту спочатку (чистий HTML) та відкладають JavaScript/CSS, що призводить до швидшого початкового рендерингу [3]. Аналогічно, *стилізація, орієнтована на корисність*, та інші сучасні фреймворки зменшують складність CSS для оптимізації рендерингу. Вепсалайнен *та ін.* наголошують , що ці шаблони розробки, хоча спочатку були мотивовані продуктивністю розробників або проблемами UX, мають побічний ефект покращення часу завантаження та скорочення обробки клієнта [4]. Їхній огляд підкреслює тенденцію: WPO сьогодні — це не лише мережеві трюки, але й впровадження методів бережливого дизайну та кодування у вебфреймворках.

Хоча було запропоновано багато методів, кілька досліджень досліджують

інструменти та сервіси, що реалізують WPO в реальних умовах. Ян *та ін.* зазначають, що оптимізатори веб- продуктивності (WPO), тобто прискорювачі на основі проксі-серверів або на стороні сервера, такі як Google AMP, широко використовуються для підвищення швидкості за допомогою таких методів, як перекодування зображень, мініфікація JS/CSS, стисненняHTML та кешування [5]. Ці WPO можуть значно скоротити час завантаження сторінки (Ян *та ін.* зазначають, що AMP може пришвидшити завантаження сторінки в середньому в 2,5 раза для мільярдів сторінок [6]). Однак вони також часто призводять до непередбачуваних побічних ефектів. Зокрема, виявляють *візуальні спотворення,* спричинені трансформаціями WPO , які можуть серйозно погіршити взаємодію з користувачем [7]. Їхній інструмент «Vetter» автоматично виявляє відмінності в морфології сторінки до та після оптимізації, точно визначаючи, наприклад, коли зображення або стиль були змінені неправильно. Використовуючи Vet ter, автори виявили десятки дефектів у популярних сервісах WPO: наприклад, оптимізація головної сторінки новин на основі Ziproxy була настільки спотворена, що користувачі відхиляли оптимізований вигляд [8]. Це підкреслює, що агресивний WPO може мати зворотний ефект.

UX, якщо його не ретельно протестувати, тему підтримують і інші, зазначаючи, що багато користувачів відмовляються від WPO через пошкоджені макети або відсутній контент [9].

Вплив продуктивності на фактичний користувацький досвід та сприйняту якість обслуговування також було ретельно досліджено. Венер *та ін.* вивчають показники Core Web Vitals (CWV) від Google, такі як LCP, FID/INP, CLS тощо, у зв'язку із суб'єктивною якістю досвіду (QoE) [10]. Вони виявили, що самі по собі показники CWV є «набагато менш прогностичними для веб-QoE, ніж очікувалося», і що прості показники часу завантаження все ще домінують у сприйнятті користувачів. У контрольованих краудсорсингових експериментах вони показують, що оцінки користувачів щодо готовності сторінки сильніше корелюють із загальним часом завантаження, ніж з окремими CWV [11]. Аналогічно, у промисловому дослідженні ван Ріта та *ін.* було виміряно багато

традиційних показників (11 різних таймінгів) за десятки втручань у продуктивність на великій веб-панелі керування [12]. Вони досягли значних покращень (наприклад, зменшення першого відображення контенту на понад 97% після 13 оптимізацій) та співвіднесли їх з відгуками користувачів. Примітно, що їхнє дослідження користувачів виявило спеціальну метрику «часу до віджета», яка майже ідеально збігалася зі сприйнятою готовністю. Разом ці роботи вказують на ключову тенденцію: хоча нові метрики (Web Vitals) є корисними орієнтирами, кінцевим арбітром є те, як зміни продуктивності впливають на фактичне сприйняття користувачами враження. Іншими словами, будь-яка стратегія WPO повинна бути перевірена не лише числовими метриками, але й дослідженнями користувачів, щоб забезпечити матеріалізацію перцептивних переваг [13].

Практики розробки та автоматизація також розвиваються для підтримки WPO. Гупта та *ін.* пропонують використовувати методи штучного інтелекту для оптимізації самого коду CSS для підвищення продуктивності та зручності обслуговування. Їхня робота описує автоматизований рефакторинг таблиць стилів для усунення надлишків та неефективності, що призводить до більш компактного CSS з порівнянною семантикою [14]. Це відображає ширшу тенденцію використання інструментів (включаючи машинне навчання), щоб допомогти розробникам у написанні коду, що покращує продуктивність. Хан аналогічно аналізує схеми оптимізації фронтенду, пропагуючи структурування ресурсів на основі XML, оптимізацію зображень та кешу, а також експериментальну перевірку, що відповідає сучасним практикам адаптивного дизайну та лінивого завантаження зображень [15]. Загалом, література показує, що сучасні веб-фреймворки та інструменти збірки все частіше включають продуктивність як основну проблему: такі методи, як розділення коду, серверний рендеринг (наприклад, багатосторінковий проти односторінкового) та часткове завантаження (острівна архітектура), з'являються як поширені стратегії. Ці підходи вимагають від розробників по-іншому думати про те, як і
коли виконується код, вбудовуючи продуктивність у робочий процес розробки.

Підсумовуючи, нещодавні дослідження підкреслюють, що ефективне WPO – це багатогранний процес. З точки зору користувацького досвіду, дослідження, подібні до тих, підкреслюють, як оптимізація продуктивності перетворюється на реальні перцептивні результати, показуючи, що оптимізацію необхідно оцінювати за показниками, орієнтованими на користувача, а не лише за сирими технічними показниками [16]. Щодо практики розробки, опитування та фреймворки, ілюструють, що сучасна веб-інженерія, від філософій дизайну інтерфейсу користувача (прогресивне вдосконалення, HTML-first) до автоматизованих інструментів (рефакторинг ШІ), може безпосередньо враховувати міркування продуктивності в процесі розробки [17]. Наш проект, який досліджує вплив WPO на UX та те, як команди застосовують оптимізації, базується на цих висновках. У відповідній літературі пропонуються два ключові моменти: по-перше, методи WPO повинні збалансувати приріст швидкості з потенційними регресіями UX (наприклад, візуальними спотвореннями); по-друге, успішне WPO тісно пов'язане з робочими процесами розробників через інструменти та найкращі практики. У наступних розділах ця література буде використана для оцінки різних методів створення системи оцінки діяльності (WPO).

Швидке зростання та складність веб-застосунків зробили оптимізацію продуктивності веб-сайтів (WPO) ключовою проблемою в сучасній розробці [18]. Більші розміри сторінок та підвищена інтерактивність призвели до повільнішого завантаження та більших вимог до пристроїв. У відповідь з'явився широкий спектр стратегій оптимізації для покращення швидкості реагування та взаємодії з користувачем, багато з яких описані нижче.

WPO охоплює різноманітний набір стратегій, включаючи кешування, мінімізацію ресурсів, оптимізацію зображень, покращення мережевого протоколу та шаблони архітектурного проектування, такі як прогресивне покращення та рендеринг, керований сервером. Нещодавні галузеві ініціативи, такі як Core Web Vitals від Google, ще більше підкреслили важливість вимірюваної веб-продуктивності для рейтингу в пошукових системах та

утримання користувачів [19].

WPO також має етичні та екологічні наслідки. Швидші вебсайти покращують доступність, зменшують споживання енергії та підтримують інклюзію. Однак надмірно агресивна оптимізація може погіршити UX або призвести до візуальних спотворень [20].

Оскільки традиційні показники часто не відображають сприйняту якість досвіду (QoE), потрібне ширше розуміння як технічних, так і людиноорієнтованих впливів. У цій роботі досліджується, як вибрані методи WPO впливають на ключові показники ефективності та сприйняту продуктивність, а також труднощі, з якими стикаються розробники під час їх застосування.

1.2 Фактори, що впливають на продуктивність web-застосунків

Це дослідження зосереджено саме на впливі вибраних методів WPO фронтенду на показники продуктивності веб-застосунків та взаємодію з користувачем. Дослідження включає такі методи, як кешування, мінімізація ресурсів, оптимізація зображень та стратегії архітектури фронтенду. В аналіз інтегровано як емпіричні вимірювання, так і якісні відгуки від користувачів і розробників.

Дослідження не охоплює оптимізацію серверної інфраструктури, що виходить за рамки рівнів веб-обслуговування, таких як покращення продуктивності баз даних або серверної частини. Аналогічно, нові теми, такі як прогнозування продуктивності на основі штучного інтелекту або механізми спекулятивної оптимізації, залишаються поза межами основної сфери дослідження. Основний акцент залишається про практичні практики WPO, які можуть безпосередньо застосовувати веб-розробники, та їхній вплив на сприйняту та вимірювану продуктивність.

Звужуючи сферу дослідження до практичних, фронтенд-енд та видимих для клієнта стратегій оптимізації, дослідження має на меті надати

цілеспрямовані, засновані на доказах рекомендації, які безпосередньо актуальні для сучасної галузевої практики.

Для кращого розуміння ландшафту WPO у цьому розділі детально розглянуто кілька ключових методів, які є центральними для покращення продуктивності сучасних вебзастосунків.

Кешування покращує час завантаження, зберігаючи часто використовувані ресурси (наприклад, зображення, CSS, JS) у браузері або через CDN [21]. Це зменшує повторювані запити до сервера та знижує затримку. Розширені стратегії включають сервісні працівники для детального кешування та офлайн-підтримки.

Оптимізація зображень зменшує час завантаження шляхом стиснення файлів, використання сучасних форматів (таких як WebP/AVIF) та відображення адаптивних розмірів зображень залежно від пристрою та роздільної здатності екрана [22]. Ці стратегії покращують найбільший обсяг контенту (LCP) та загальний UX.

Мінімізація зменшує розміри файлів HTML, CSS та JS, видаляючи пробіли, коментарі та невикористаний код, покращуючи швидкість завантаження та виконання. Зазвичай використовуються такі інструменти, як Terser та Uglify.

Розділення коду поділяє JavaScript на менші фрагменти, що завантажуються на вимогу, зменшуючи час початкового завантаження та покращуючи інтерактивність.

Відкладене завантаження відкладає завантаження зображень або компонентів поза екраном до моменту потреби, покращуючи початкове завантаження сторінки та заощаджуючи пропускну здатність. Однак, неправильне використання цього методу для критично важливого контенту може зашкодити UX.

Струшування дерева усуває невикористаний JavaScript під час процесу збірки, що призводить до менших розмірів пакетів та швидшого завантаження.

Gzip та Brotli стискають текстові ресурси (HTML, CSS, JS) під час

передачі, зменшуючи розмір завантаження та час завантаження — Brotli зазвичай пропонує краще стиснення [23].

Критичне вбудовування CSS розміщує ключові стилі безпосередньо в HTML <head> для пришвидшення рендерингу вище згину та покращення першого візуалізації контенту (FCP).

Підказки щодо ресурсів (наприклад, preload, preconnect) інформують браузер про майбутні потреби в ресурсах, зменшуючи затримку та покращуючи послідовність завантаження [24].

Метою цього дослідження є вивчення реального впливу методів оптимізації веб-продуктивності (WPO) на продуктивність та показники в практиці розробки, зосереджуючись на практичному впровадженні, досвіді розробників та задоволеності користувачів. Це відповідає зростаючій потребі в швидших та ефективніших веб-додатках у сучасних цифрових сервісах.

Для досягнення цієї мети ми застосували змішаний метод дослідження, який поєднує цільове опитування, розроблене спеціально для веб-розробників, з емпіричним тестуванням методів WPO в змодельованих мережевих умовах. Такий підхід дозволяє зрозуміти як організаційні, так і технічні проблеми за допомогою даних опитування, а також оцінити ефективність стратегій оптимізації за допомогою контрольованих експериментів, які ми розробили та впровадили спеціально для цього дослідження.

Змішаний метод був обраний для того, щоб наші дослідницькі питання можна було розглянути з різних точок зору, що підвищило як глибину, так і достовірність наших висновків.

У цьому дослідженні було використано змішаний методологічний підхід, що поєднує якісні, кількісні та теоретичні методології для забезпечення всебічного розуміння оптимізації веб-продуктивності (WPO). Конкретні використані методи були такими:

Теоретичний аналіз: Було проведено комплексний *огляд літератури для* встановлення теоретичних основ WPO, визначення існуючих методів та проблем, а також визначення поточних прогалин у дослідженнях.

Емпіричний кількісний аналіз: Контрольоване *емпіричне тестування* було проведено за допомогою спеціального веб-застосунку. Об'єктивні, кількісні дані про показники продуктивності (наприклад, Core Web Vitals, час завантаження сторінки) були зібрані за допомогою стандартизованих інструментів, таких як Google Lighthouse, за умов модельованої мережі (наприклад, 3G, 4G). Емпіричний якісний аналіз:

- Фаза емпіричного тестування також включала збір якісних даних за допомогою користувацького тестування , збираючи відгуки про сприйняту продуктивність та користувацький досвід від реальних людей, які взаємодіють з різними рівнями оптимізації застосунку.
- опитування, спрямоване на розробників та відповідних зацікавлених сторін, з метою збору якісної інформації (і потенційно деяких кількісних оцінок) щодо практичних проблем, компромісів, процесів прийняття рішень та сприйнятої ефективності, пов'язаної з впровадженням методів WPO.

Синтез даних: Заключний етап включав *інтеграцію даних,* де результати огляду літератури, кількісних емпіричних тестів, якісних відгуків користувачів та опитування були синтезовані для створення цілісних висновків та рекомендацій на основі доказів, як представлено в наступних розділах.

Поєднання цих методів дозволило провести тріангуляцію даних, забезпечивши надійну основу для вирішення дослідницьких питань.

Виходячи зі своїх цілей та завдань, це дослідження класифікується переважно як емпіричне оціночне та пошукове дослідження .

Оцінювальний: Основною метою було емпірично *оцінити* ефективність різних методів WPO шляхом вимірювання їхнього впливу на ключові показники ефективності. Це включало порівняння різних методів за контрольованих умов.

Дослідницьке: Метою дослідження було *дослідити* та зрозуміти проблеми, з якими стикаються розробники та організації під час впровадження методів WPO. Це включало вивчення менш вивченого аспекту практики WPO за допомогою опитувань та аналізу.

Опис: У дослідженні також *описано* зв'язок між конкретними методами

WPO, показниками продуктивності, користувацьким досвідом та практиками розробників на основі зібраних даних.

Конструктивний/Рекомендаційний: Зрештою, дослідження мало на меті синтезувати результати в практичні рекомендації, засновані на доказах, для оптимізації продуктивності веб-сайтів на стабільний спосіб, враховуючи доступність та задоволеність користувачів, надаючи їм конструктивного характеру.

Дослідження міцно ґрунтується на емпіричному зборі даних (метрики продуктивності, відгуки тестувальників, відповіді на опитування) та включає експериментальні маніпуляції (застосування методів WPO до контрольованого застосунку)

1.3 Методи оптимізації продуктивності клієнтської частини

Оптимізація веб-продуктивності - це практика пришвидшення завантаження веб-сайтів та їхньої швидшої реакції на взаємодію з користувачами. Вона безпосередньо впливає на поведінку користувачів, моделі взаємодії та бізнес-результати через численні механізми. Коли користувачі стикаються з повільними веб-сайтами, вони розчаровуються та часто взагалі їх покидають. Цей зв'язок між швидкістю та задоволеністю користувачів не є просто суб'єктивним, він має неврологічну основу. Дослідження в когнітивній психології показують, що користувачі сприймають взаємодію як «миттєву», коли вона відбувається протягом 100 мс, як «швидку реагування», коли вона відбувається протягом 300 мс, і як «повільну», коли вона триває довше 1 секунди.

Психологія користувацького досвіду показала, що сприйнята продуктивність часто має більше значення, ніж фактичні показники продуктивності. Користувачі оцінюють швидкість сайту не лише за тим, скільки часу потрібно для повного завантаження, але й за тим, як швидко вони можуть побачити корисний контент і почати взаємодіяти з ним. Це сприйняття стимулює розробку методів прогресивного покращення та шаблонів скелетного

завантаження, які забезпечують візуальний зворотний зв'язок під час завантаження контенту у фоновому режимі. До оптимізації продуктивності потрібно підходити цілісним чином, охоплюючи кілька взаємопов'язаних рівнів:

· Оптимізація мережі : зменшення кількості та розміру HTTP-запитів за допомогою таких методів, як групування, мініфікація, стиснення та пріоритизація ресурсів. Це включає впровадження ефективних мереж доставки контенту (CDN) для фізичного наближення ресурсів до користувачів.

· Оптимізація рендерингу: усунення ресурсів, що блокують рендеринг, шляхом відкладення некритичного CSS/JavaScript, реалізації критичної оптимізації шляху рендерингу та використання рендерингу на стороні сервера, де це доречно. Це гарантує, що браузер може якомога швидше створити візуальне представлення сторінки.

· Оптимізація виконання JavaScript: зменшення блокування основного потоку за допомогою розділення коду, лінивого завантаження та ефективної обробки подій. Це включає оптимізацію сторонніх скриптів, які часто значно знижують продуктивність.

· Оптимізація доставки ресурсів : впровадження інтелектуальних стратегій кешування, сервісних працівників та прогнозного попереднього вибору для мінімізації повторних завантажень та підготовки ресурсів до їх знадоблення. Це створює стійкий інтерфейс, який добре працює в різних мережевих умовах.

У швидкозмінному цифровому середовищі, веб

Продуктивність додатків стала фундаментальним фактором успіху бізнесу в різних галузях промисловості. Нещодавнє дослідження, опубліковане в журналі «Journal of King Saud University - Computer and Information Sciences», демонструє, що показники продуктивності веб-сайтів безпосередньо корелюють із моделями залучення користувачів та бізнес-результатами. Дослідження, в якому аналізуються дані понад 12 000 користувацьких сеансів, показало, що веб-сайти, які досягають першого завантаження контенту (FCP) менш ніж за 1,8 секунди, демонструють на 23% вищий рівень утримання користувачів порівняно

з повільнішими альтернативами. Крім того, дослідження показало, що мобільні користувачі, які зараз складають 63,7% веб-трафіку, демонструють особливо підвищену чутливість до проблем продуктивності, причому 89% залишають сайти, які завантажуються довше 3 секунд [25]. Кореляція між оптимізацією продуктивності та бізнес-показниками глибоко поширюється на платформи електронної комерції. Комплексний аналіз коефіцієнтів конверсії електронної комерції на кількох платформах показав, що час завантаження сторінки суттєво впливає на поведінку користувачів та рішення про покупку. Дослідження, в якому розглянуто 850 000 переглядів сторінок у різних секторах електронної комерції, показало, що коефіцієнти конверсії знижувалися в середньому на 4,42% на кожну додаткову секунду часу завантаження від 0 до 4 секунд, причому вплив експоненціально зростав після перевищення 4-секундного порогу. У дослідженні зокрема зазначалося, що веб-сайти, які оптимізували час завантаження з 8 секунд до 2 секунд, зазнали середнього збільшення коефіцієнта конверсії на 74,3%, що є значними альтернативними витратами для неоптимізованих платформ [26]. Технічний ландшафт фронтенд-оптимізації значно розвинувся для вирішення цих проблем. Сучасні веб-додатки повинні враховувати складні міркування продуктивності в різних мережевих умовах, можливостях пристроїв та очікуваннях користувачів. Впровадження сучасних методів оптимізації вимагає глибокого розуміння конвеєрів візуалізації браузера, мережевих протоколів та стратегій кешування. Аналіз даних про продуктивність провідних платформ електронної комерції показує, що ефективні стратегії оптимізації можуть скоротити час відгуку сервера на 47,3%, зменшити споживання пропускної здатності на 38,2% та покращити коефіцієнти потрапляння в кеш на 56,7%. Метрики користувацького досвіду стають дедалі нюансованішими, а дослідження показують, що сприйнята продуктивність часто відрізняється від технічних вимірювань. Дослідження показують, що користувачі сприймають сторінки як «швидкі», коли інтерактивні елементи стають адаптивними протягом 100 мс, незалежно від загального часу завантаження сторінки. Цей психологічний аспект продуктивності призвів до

розробки методів прогресивного покращення та шаблонів скелетного завантаження, які, як показали, покращують сприйняті рейтинги продуктивності на 31,4%, навіть коли фактичний час завантаження залишається незмінним [27]. Фінансові наслідки оптимізації продуктивності є суттєвими та кількісно вимірюваними. Організації, що впроваджують комплексні стратегії оптимізації фронтенду, повідомляють про середнє збільшення доходу на 8,4% та покращення утримання клієнтів на 12,7%. Якщо розглядати конкретно мобільну комерцію, оптимізовані веб-сайти демонструють на 27,3% вищий коефіцієнт конверсії порівняно з їхніми неоптимізованими аналогами, причому користувачі витрачають в середньому на 12,8 хвилини довше за сеанс на добре оптимізованих сайтах [28].

Основні показники веб-сторінок (Core Web Vitals) – це набір специфічних показників, які Google визначив як критично важливі для вимірювання взаємодії користувача з веб-сторінкою. Вони виходять за рамки традиційних технічних показників, таких як «час до першого байта», і зосереджуються на аспектах, які безпосередньо впливають на те, як користувачі сприймають продуктивність.

Найбільший показник контентного відображення (LCP) вимірює продуктивність завантаження, визначаючи, як швидко найбільший елемент контенту у вікні перегляду стає видимим для користувачів. Це може бути головне зображення, великий текстовий блок або відеоелемент. LCP є точнішим відображенням сприйнятої продуктивності завантаження, ніж старіші показники, оскільки він зосереджений на тому, що найважливіше для користувачів – бачення основного контенту.

Детальні стратегії впровадження:

· Оптимізуйте час відгуку сервера за допомогою ефективного бекенд-коду, належної індексації бази даних та шарів кешування

· Усуньте блокування рендерингу CSS та JavaScript за допомогою оптимізації критичного шляху

· Реалізація підказок щодо ресурсів, таких як попереднє завантаження,

попереднє підключення та попередня вибірка критично важливих ресурсів

· Використовуйте адаптивні методи роботи з зображеннями з правильним вибором розміру та формату

· Розгляньте рендеринг на стороні сервера або статичний рендеринг генерація для швидшої початкової доставки контенту

· Оптимізуйте завантаження веб-шрифтів, щоб запобігти невидимості тексту під час завантаження сторінки

Затримка першого вводу (FID) вимірює інтерактивність, кількісно визначаючи час між першою взаємодією користувача зі сторінкою (наприклад, натисканням кнопки) та моментом, коли браузер здатний відреагувати на цю взаємодію. Цей показник відображає, наскільки швидко реагує сайт, і на нього безпосередньо впливають час виконання JavaScript та блокування основного потоку.

Детальні стратегії впровадження:

Розбийте тривалі завдання JavaScript на менші фрагменти тривалістю до 50 мс

· Відкласти або видалити некритичні сторонні скрипти, які можуть конкурувати за ресурси головного потоку

· Використовуйте веб- робітників для переміщення інтенсивних ресурсів обчислення поза основним потоком

· Впроваджуйте ефективні шаблони делегування подій замість численних окремих слухачів подій

· Оптимізуйте виконання JavaScript, видаляючи непотрібний код і залежності

· Застосувати requestAnimationFrameandrequestIdleCallback для необов'язкової обробки

· Впроваджуйте методи усунення стрибків та дроселювання для частих подій, таких як прокручування або зміна розміру

Кумулятивне зміщення макета (CLS) вимірює візуальну стабільність,

обчислюючи, скільки неочікуваних рухів видимих елементів відбувається під час процесу завантаження сторінки. Цей показник відображає один із найбільш неприємних аспектів поганого веб-досвіду – коли елементи сторінки неочікувано рухаються, що призводить до неправильних кліків користувачів або їх втрати місця.

Детальні стратегії впровадження:

· Завжди вказуйте атрибути ширини та висоти для зображень, відео, фреймів iframe та іншого вбудованого контенту

· Зарезервуйте достатньо місця для динамічного контенту, такого як реклама або вбудовування, використовуючи блоки співвідношення сторін

· Уникайте вставки нового контенту поверх існуючого, окрім випадків, коли це відбувається у відповідь на взаємодію з користувачем

· Впроваджуйте заповнювачі контенту, які відповідають очікуваним кінцевим розмірам

· Використовуйте анімацію перетворення замість властивостей, таких як ширина/висота, які запускають перерахунок макета

· Забезпечте ефективне завантаження веб-шрифтів, щоб мінімізувати перекомпонування тексту

· Структуруйте CSS для встановлення макета на ранніх етапах процесу завантаження сторінки

Core Web Vitals стали остаточною основою для кількісної оцінки продуктивності веб-сайтів та якості користувацького досвіду в сучасному цифровому середовищі. Нещодавнє дослідження, в якому розглянуто 2500 веб-сайтів електронної комерції по всій Індії, виявило прямий зв'язок між дотриманням вимог Core Web Vitals та показниками залученості користувачів. Дослідження показало, що веб-сайти, які досягли оптимальних показників Core Web Vitals, зазнали значного збільшення тривалості сеансу користувача на 47% та зниження показника відмов на 39% порівняно з сайтами, які не відповідають вимогам. Крім того, платформи мобільної комерції, які підтримували стабільні показники продуктивності, показали на 52% вищий коефіцієнт конверсії, що

особливо важливо, враховуючи, що мобільні пристрої становлять 78,3% трафіку електронної комерції на ринках, що розвиваються [29].

Затримка першого введення (FID), що вимірює швидкість реагування на взаємодію, виявилася критичним фактором задоволеності користувачів на цифрових платформах. Комплексний аналіз індійських платформ електронної комерції показав, що веб-сайти, які підтримують показники FID нижче 100 мілісекунд, досягли на 43,2% вищого рівня успішних транзакцій. Примітно, що мобільні користувачі продемонстрували особливу чутливість до затримок взаємодії, при цьому спостерігалося різке зниження показників заповнення форм на 67%, коли FID перевищував 130 мілісекунд. Дослідження також виявило сильну кореляцію між оптимізацією FID та довірою користувачів, причому 82% користувачів повідомляють про вищу довіру до платформ, які демонструють швидку реагування на взаємодію [30]. Найбільший контентний малюнок (LCP) зарекомендував себе як ключовий показник сприйнятої продуктивності та ефективність доставки контенту. Аналіз даних про ефективність електронної комерції показує, що сторінки, які досягли LCP протягом 2,5 секунд, зазнали суттєвого збільшення переглядів сторінок товарів на 41% та покращення середньої вартості замовлення на 28%. Дослідження особливо підкреслило вплив на користувачів мобільних пристроїв, де оптимальні показники LCP відповідали зниженню рівня покидання кошика на 56%. Крім того, платформи, які підтримували стабільну ефективність LCP нижче порогу 2,5 секунди, повідомили про збільшення показників повернення клієнтів на 33%, що демонструє довгостроковий вплив оптимізації продуктивності на лояльність клієнтів [31]. Вимірювання сукупної зміни макета (CLS) виявили глибокі наслідки для взаємодії з користувачем та оптимізації конверсій. Нещодавній галузевий аналіз показує, що веб-сайти електронної комерції, які підтримують показники CLS нижче 0,1, досягли значного зниження випадкових кліків на 49% та покращення успішного завершення замовлення на 37%. Вплив був особливо помітним у мобільній комерції, де оптимізовані показники CLS відповідали зниженню помилок надсилання форми на 61% та збільшенню успішної обробки

платежів на 44%. Ці результати підкреслюють критичний зв'язок між візуальною стабільністю та впевненістю користувачів у завершенні транзакцій [32]. Впровадження стратегій оптимізації Core Web Vitals продемонструвало значний вплив на бізнес за різними показниками продуктивності. Індійські платформи електронної комерції, які досягли оптимальних показників Core Web Vitals, повідомили про середнє збільшення доходу на 51,3%, а коефіцієнти конверсії з мобільних пристроїв покращилися на 73,6% порівняно з показниками до оптимізації. Дослідження задокументувало, що веб-сайти, які підтримують стабільні показники продуктивності, зазнали збільшення середньої вартості замовлення на 47,8% та покращення показників утримання клієнтів на 62,4% протягом шестимісячного періоду [33]. Зв'язок між Core Web Vitals та видимістю в пошукових системах показав переконливі результати в нещодавніх аналізах. Платформи електронної комерції, які відповідають усім пороговим значенням Core Web Vitals, продемонстрували покращення видимості в органічному пошуку на 38,7% та збільшення показників кліків з результатів пошуку на 42,3%. Крім того, веб-сайти, які підтримують оптимальні показники продуктивності, досягли зниження витрат на залучення клієнтів на 29,4% завдяки покращеному органічному охопленню та посиленню сигналів довіри користувачів [34].

Висновок по розділу

У першому розділі досліджено теоретичні основи оптимізації продуктивності web-сервісів. Розглянуто основні поняття продуктивності, визначено фактори, які впливають на швидкодію та стабільність роботи web-застосунків. Особливу увагу приділено методам оптимізації клієнтської частини, зокрема зменшенню обсягу передаваних даних, кешуванню ресурсів та оптимізації завантаження сторінок. Встановлено, що комплексний підхід до підвищення продуктивності є важливою умовою забезпечення якісного користувацького досвіду.

РОЗДІЛ 2. МЕТОДИ ТА ІНСТРУМЕНТИ ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ

2.1 Методи збору та аналізу даних продуктивності

Щоб відповісти на дослідницькі питання, ми зібрали дані за допомогою двох різних методів: структурованого опитування, спрямованого на веб-розробників, та контрольованого експерименту, що включав тестування користувачами трьох версій веб-додатків. Ми обрали кілька методів для забезпечення тріангуляції методів [35]. Ці методи дозволили нам зібрати як широкі погляди практиків галузі, так і поглиблені спостереження за фактичним впливом на продуктивність.

Опитування було поширене онлайн серед фахівців, що працюють у сфері IT та розробки програмного забезпечення, з метою охоплення таких ролей, як фронтенд-розробники, бекенд-розробники , фулстек-розробники, технічні керівники та архітектори. Щоб охопити цю цільову групу, опитування було поширене через професійні мережеві платформи, такі як LinkedIn, а також безпосередньо серед контактів у різних компаніях IT-сектору, щоб заохотити ширший набір відповідей. Основним критерієм участі була активна професійна діяльність на відповідній посаді веб-розробника. Загалом було зібрано 32 відповіді.

Опитування включало як питання з вибором однієї правильної відповіді, так і питання з відкритою відповіддю. Серед розглянутих тем були використання та сприйнята ефективність різних методів WPO, проблеми, з якими стикаються під час їх впровадження, та важливість оптимізації продуктивності в організації. Учасників також запитали про те, як їхні команди пріоритезують продуктивність стосовно інших факторів, таких як розробка функцій, терміни та енергоефективність. Повна анкета опитування та відповіді, надані учасникам. 32 респонденти опитування представляли низку ролей в галузі IT та розробки програмного забезпечення: 15 визначили себе як Full-stack розробники (46,8%),

8 як Front-end розробники (25%), 5 як технічні керівники/архітектори (15,6%), 2 як Back-end розробники (6,2%) та 1 як генеральний директор (3,1%); один респондент не вказав роль. Що стосується досвіду, 8 респондентів (приблизно 25%) мали понад 6 років досвіду в веб-розробці, 9 (приблизно 28,1%) – 4-6 років, 11 (приблизно 34,3%) – 1-3 роки, а 3 (приблизно 9,3%) – менше 1 року. Учасники переважно працювали в таких галузях, як технології (12 респондентів, приблизно 37,5%), консалтинг (8 респондентів, приблизно 25%), електронна комерція (5 респондентів, приблизно 15,6%) та фінанси (3 респонденти, приблизно 9,3%), а також кілька респондентів з освіти та фінтех. Детальний огляд індивідуальних відповідей щодо ролей учасників, досвіду та галузей промисловості можна знайти в Додатку В.2. Розмір вибірки з 31 респондента було визначено кількістю відповідей, отриманих протягом періоду збору даних для цього дослідницького опитування. Хоча результати опитування й містять цінні знання від практикуючих фахівців, вони вважаються орієнтовними для цієї групи, а не узагальнюваними для всієї популяції веб-розробників через неімовірнісний метод вибірки та розмір вибірки.

Окрім опитування, ми розробили експеримент для збору якісних відгуків від учасників, які взаємодіяли з трьома різними версіями веб-застосунку:

1. Неоптимізована версія,
2. Частково оптимізована версія
3. Повністю оптимізована версія.

В експерименті використовувався спеціально розроблений веб-застосунок, призначений для імітації типової платформи електронної комерції. Основні функції включали:

- Головна сторінка, на якій відображаються представлені товари, рекламні банери та категорії товарів.
- Сторінки зі списками товарів з можливостями фільтрації та сортування.
- Сторінки з детальною інформацією про окремі продукти.
- Кошик для покупок та імітація процесу оформлення замовлення.

- Навігація користувача за допомогою React Router.

Додаток було розроблено як односторінковий додаток (SPA) з використанням React (версії 18) та Vite як інструменту збірки. Основною метою було створення реалістичного фронтенд-середовища, де можна було б систематично застосовувати та спостерігати вплив різних методів оптимізації веб-продуктивності (WPO).

Три різні версії веб-застосунку для електронної комерції, конкретні конфігурації яких детально описані в Додатку А, були систематично визначені для представлення чітких етапів оптимізації. Неоптимізована версія спочатку була розроблена як стандартний, функціональний базовий додаток без специфічної оптимізації продуктивності. Щоб встановити чіткий орієнтир зі значним потенціалом для вимірюваного покращення, були навмисно включені або перевірені на наявність поширені анти-шаблони продуктивності та неефективності, взяті з відомої літератури з веб-розробки та діагностичних інструментів. Для помірно оптимізованої версії було обрано «базальний набір» методів WPO. Цей набір було обрано для представлення поширених оптимізацій з високим рівнем впливу, які часто вважаються важливими. для базового рівня веб-продуктивності, при цьому вибір ґрунтувався на усталених практиках WPO та додатково керувався методами, які, як повідомляли учасники -нашого початкового опитування розробників, часто використовуються (як детально описано далі в цьому розділі). Нарешті, повністю оптимізована версія була розроблена для досягнення високого рівня оптимізації шляхом впровадження комплексного набору передових методів WPO, дотримуючись сучасних найкращих практик. Хоча термін «Повністю оптимізований» має на меті представити високий рівень оптимізації (див. Додаток А.3.1 для отримання детальної інформації), він не означає включення кожного можливого методу WPO; радше метою було продемонструвати значне покращення продуктивності шляхом застосування широкого та ефективного набору оптимізацій, загальновизнаних як найкращі практики. Конкретні антишаблони та оптимізації для кожної версії ітеративно документувалися в міру розробки відповідних

експериментальних версій програми.

Для експерименту було створено три різні версії цього веб-застосунку для електронної комерції:

1. Неоптимізована версія: Ця версія слугувала базовою. Вона була навмисно розроблена з використанням поширених шаблонів зниження продуктивності та не мала жодних специфічних методів WPO. Це включало такі проблеми , як великі розміри пакетів без розділення коду, неоптимізовані зображення, неефективне завантаження ресурсів та операції, що блокують рендеринг.

2. Помірно оптимізована версія: Ця версія включала базовий набір методів WPO. Цей набір був обраний для представлення поширених, високоефективних оптимізацій, які часто вважаються важливими для базового рівня продуктивності веб-сайтів. Вибір був заснований на усталених практиках WPO та додатково керувався методами, які часто використовувалися учасниками нашого початкового опитування розробників (як детально описано далі в цьому розділі). Ключові оптимізації включали базове розділення пакетів за допомогою React.lazy та Suspense , перетворення деяких ключових зображень у формат WebP з відкладеним завантаженням, мінімізацію збірки через Vite та базові заголовки кешування HTTP.

3. Повністю оптимізована версія: У цій версії реалізовано комплексний набір передових методів WPO, що відповідають сучасним передовим практикам. Хоча термін «Повністю оптимізований» має на меті представити стан високої оптимізації, досягнутий за допомогою широкого спектру встановлених та передових методів, він не означає включення кожної можливої методики WPO. Швидше, метою було продемонструвати значне покращення продуктивності шляхом застосування широкого та ефективного набору оптимізацій, які зазвичай визнаються найкращими практиками. Оптимізація була обширною, охоплюючи розширене розділення коду з попередньою вибіркою, комплексну оптимізацію зображень (WebP, розширене відкладене завантаження, техніка розмиття, пріоритетне завантаження), агресивну оптимізацію збірки (Terser , струшування

дерев, Gzip/Brotli), критичне вбудовування CSS, розширене кешування за допомогою сервісних працівників (функції PWA), оптимізацію візуалізації інтерфейсу користувача (запам'ятовування, віртуалізовані списки) та використання залежностей, орієнтованих на продуктивність.

Детальний, деталізований розбивка конкретних антишаблонів, присутніх у неоптимізованій версії, та повний перелік оптимізацій, застосованих до помірно та повністю оптимізованих версій.

Кожен учасник протестував усі три версії веб-сайту (неоптимізовану, помірно оптимізовану та повністю оптимізовану) за умов імітації мережі 3G, розроблених для виявлення відмінностей у продуктивності. Експериментальна процедура для кожного учасника включала початковий етап вільного дослідження кожної версії програми. Під час цього дослідження учасникам було запропоновано взаємодіяти з функціями сайту, як вони це роблять зазвичай. Після цього, як спеціальне завершальне завдання, учасникам було доручено оновити кожен із трьох веб-сайтів окремо та уважно занотувати свої спостереження щодо поведінки перезавантаження. Протягом як вільного дослідження, так і завдання оновлення учасників просили зосередитися на конкретних аспектах, таких як час завантаження, завантаження зображень та загальна швидкість реагування. Учасникам не повідомляли, яку версію вони використовують, щоб зменшити упередженість.

Після взаємодії з усіма трьома версіями та їх оцінки таким чином, вони надали якісний зворотний зв'язок про свій досвід (стислі нотатки для кожного учасника доступні в Додатку С) та класифікували версії від найбільш до найменш бажаної. 30 учасників для користувацького експерименту були набрані виключно через особисті мережі авторів, що склало вибірку для зручності. Такий підхід дозволив сформувати різноманітну групу осіб, яка, на основі загальної пам'яті авторів, включала студентів університетів з різними академічними рівнями (від першого курсу бакалаврату до магістерських програм), низку осіб з професійним досвідом веб-розробки та інших учасників із широкої громадськості, що представляють різноманітний нетехнічний досвід та

професійний статус (наприклад, ті, хто працює в охороні здоров'я, інших сферах послуг, або особи, які були безробітними на момент дослідження). Хоча конкретні демографічні дані, такі як точний розподіл за віком або точний розподіл за професійною діяльністю та рівнем досвіду, не реєструвалися систематично для всіх учасників експерименту, метою цієї стратегії набору було зібрати якісний зворотний зв'язок щодо сприйнятої продуктивності веб-сайту від різноманітного набору користувачів з різним ступенем технічної обізнаності та загальними моделями використання веб-сайту. Таке різноманіття вважалося корисним для охоплення ширшого спектру реакцій користувачів на різні версії застосунку . Загалом експеримент пройшли 30 учасників, що дозволило нам виявити закономірності у сприйнятті продуктивності.

Разом ці два джерела даних дозволили нам проаналізувати методи WPO як з точки зору розробника, так і з точки зору досвіду кінцевого користувача.

Вибір методів оптимізації веб-продуктивності (WPO) для фази емпіричного тестування цього дослідження був зумовлений їхньою поширеністю в сучасних практиках розробки. Щоб забезпечити практичну значущість наших експериментальних результатів, ми пріоритезували методи, які були визначені як найчастіше використовувані фахівцями з веб-розробки в початковому опитуванні, проведеному в рамках цього дослідження. Зокрема, учасники опитування часто повідомляли про такі методи, як стратегії кешування (визнані як давній та широко використовуваний метод зменшення затримки та навантаження на сервер), ліниве завантаження (яке здобуло значну популярність для покращення часу початкового завантаження сторінки шляхом відкладення позаекранних або некритичних ресурсів [36]) та комплексна оптимізація зображень (критичний аспект, оскільки зображення часто є основним фактором, що впливає на вагу сторінки та LCP [37]). Зосереджуючи експеримент на цих широко впроваджених методах, це дослідження має на меті надати розуміння, безпосередньо застосовне до викликів та можливостей, з якими стикається широке коло веб-розробників та організацій. Повний аналіз конкретних антишаблонів у неоптимізованій версії та різних наборів оптимізацій, -

реалізованих у помірно оптимізованій та повністю оптимізованій версіях експериментального за стосунку.

Для аналізу даних опитування було використано описову статистику. Це, головним чином, включало обчислення частоти відповідей та відсотків для категоріальних запитань з метою виявлення загальних тенденцій (наприклад, які методи WPO, за повідомленнями розробників, використовують найчастіше, та з якими труднощами вони стикалися). Для запитань, що стосуються шкалованих відповідей (таких як сприйнятий пріоритет WPO), тенденції спостерігалися шляхом вивчення розподілу відповідей за шкалою. Ми зосередилися на визначенні таких тенденцій, як те, які методи WPO розробники використовують найчастіше, з якими труднощами вони стикаються та наскільки важлива продуктивність у їхніх організаціях. Були створені діаграми та таблиці, щоб допомогти візуалізувати дані та виділити ключові закономірності.

Ми використали тематичний аналіз Брауна та Кларка [38] для вивчення відповідей у відкритому опитуванні, а також відповідей, отриманих в результаті експерименту. Цей метод дозволив нам виявити повторювані закономірності, теми та змістовні висновки, які доповнювали дані опитування.

Для експерименту ми попросили учасників взаємодіяти з трьома версіями одного й того ж веб-сайту: однією неоптимізованою, однією частково оптимізованою та однією повністю оптимізованою. Учасникам було доручено зосередитися на таких аспектах, як час завантаження, завантаження зображень та загальна швидкість реагування веб-сайтів.

Усі три версії були протестовані в умовах імітації мережі 3G, щоб зробити різницю в продуктивності більш помітною. Учасникам не повідомляли, яка версія яка . Після взаємодії з усіма трьома версіями кожен учасник надав якісний відгук і його попросили оцінити три версії від найбільш до найменш бажаної.

Відгуки були проаналізовані тематично, щоб виявити спільні закономірності у сприйнятті учасниками відмінностей у продуктивності. Ключові області уваги включали швидкість завантаження, поведінку рендерингу зображень та загальний користувацький досвід.

Щоб забезпечити валідність, ми орієнтувалися на досвідчених веб-розробників і розробили питання анкети чіткими та релевантними дослідницьким питанням. В експерименті учасники не знали, яка версія була оптимізована, що зменшило ризик упередженості.

Надійність експерименту була підтверджена використанням однакового мережевого моделювання (3G) та тестової установки для всіх учасників. Результати опитування були проаналізовані за допомогою базової статистики, тоді як відкриті та експериментальні відгуки були перевірені на наявність повторюваних закономірностей.

Щодо аналізу даних опитування, для узагальнення відповідей було використано описову статистику. Важливо зазначити, що висновкова статистика не застосовувалася для перевірки статистичної значущості або узагальнення результатів за межі вибірки учасників. Цей підхід було обрано через дослідницький характер опитування та склад групи респондентів. Отже, результати опитування, перш за все, пропонують розуміння перспектив саме цієї вибірки.

2.2 Підходи до оптимізації продуктивності web-сервісів

Оптимізація продуктивності веб-застосунків є критично важливою з кількох причин. З точки зору користувацького досвіду, повільні або невідповідні застосунки можуть призвести до розчарування, збільшення показника відмов та зниження загальної задоволеності користувачів. Продуктивність — це не лише швидкість; це забезпечення безперебійного досвіду, який відповідає очікуванням користувачів. Згідно з дослідженнями, затримка завантаження сторінки навіть на кілька секунд може суттєво вплинути на утримання користувачів та коефіцієнти конверсії. Для бізнесу низька продуктивність може безпосередньо вплинути на дохід, лояльність клієнтів та репутацію бренду[39].

Більше того, з розвитком цифрового ландшафту користувачі отримують доступ до веб-додатків на широкому спектрі пристроїв, від настільних

комп'ютерів до мобільних телефонів і планшетів. Ця різноманітність пристроїв, розмірів екранів і мережевих умов додає ще один рівень складності до оптимізації продуктивності. Методи оптимізації фронтенду, такі як адаптивний дизайн і дизайн, орієнтований на мобільні пристрої, є важливими для забезпечення безперебійної роботи користувачів на всіх пристроях. Аналогічно, стратегії оптимізації бекенду повинні враховувати масштабованість, гарантуючи, що додаток може обробляти зростаючу кількість користувачів без шкоди для продуктивності. Окрім прямого впливу на користувацький досвід, оптимізація продуктивності веб-додатків відіграє життєво важливу роль у SEO (пошуковій оптимізації) та загальній видимості сайту [40]. Пошукові системи, такі як Google, надають пріоритет швидкому завантаженню та адаптивним веб-сайтам у своїх рейтингах, а це означає, що погано оптимізований веб-додаток може негативно вплинути на його видимість і потенціал трафіку. Оскільки використання мобільного Інтернету продовжує зростати, Google та інші пошукові системи також розглядають мобільну продуктивність як фактор ранжування. Це підкреслює важливість оптимізації як фронтенд-, так і бекенд-компонентів, забезпечуючи хорошу роботу додатків на всіх пристроях і в усіх мережевих умовах. Крім того, оптимізація продуктивності може зменшити експлуатаційні витрати, пов'язані з надмірним використанням сервера, зменшити споживання пропускної здатності та покращити масштабованість, що робить її економічно ефективним заходом для підприємств, які прагнуть забезпечити ефективне та стабільне обслуговування користувачів.

Інвестуючи в оптимізацію продуктивності, організації можуть не лише підвищити залученість користувачів, але й покращити свою конкурентну позицію на дедалі більш насиченому онлайн-ринку [9].

Оптимізація фронтенду відіграє вирішальну роль у забезпеченні швидкої, чуйної та плавної взаємодії користувачів із веб-додатками. Оскільки фронтенд – це частина веб-додатку, яка безпосередньо взаємодіє з користувачем, вкрай важливо, щоб кожен аспект користувацького інтерфейсу (UI) та користувацького досвіду (UX) був оптимізований для скорочення часу завантаження, покращення

швидкості реагування та мінімізації використання ресурсів. У цьому розділі буде розглянуто деякі з найефективніших методів оптимізації фронтенду, зосереджуючись на тому, як вони можуть безпосередньо впливати на продуктивність, задоволеність користувачів та загальний успіх веб-додатку [10].

Одним з найважливіших методів оптимізації фронтенду є мінімізація та стиснення ресурсів. Веб-сторінки часто містять великі файли, такі як HTML, CSS, JavaScript та зображення. Ці файли можуть значно збільшити час завантаження, особливо на мобільних пристроях або при повільнішому мережевому з'єднанні. Мінімізація цих ресурсів шляхом видалення непотрібних пробілів, коментарів та невикористаного коду допомагає зменшити їхній розмір і, в свою чергу, зменшує обсяг даних, які потрібно завантажувати. Крім того, для подальшого зменшення розмірів файлів перед їх надсиланням із сервера клієнту можна використовувати методи стиснення , такі як Gzip або Brotli. Стискаючи та мінімізуючи ресурси, розробники можуть значно підвищити швидкість завантаження сторінок, покращуючи загальний користувацький досвід [12].

Ще однією важливою технікою є оптимізація зображень. Зображення часто є найбільшими ресурсами на веб-сторінці та можуть суттєво вплинути на час завантаження, якщо їх не оптимізувати. Щоб зменшити розміри зображень без шкоди для якості, розробники можуть використовувати методи та інструменти стиснення зображень, такі як формат WebP , який забезпечує кращу компресію порівняно з традиційними форматами зображень, такими як JPEG та PNG. Крім того, адаптивні методи обробки зображень, такі як використання атрибута srcset , гарантують, що зображення відображаються з правильною роздільною здатністю для різних розмірів екранів та пристроїв. Це зменшує кількість непотрібних даних, що надсилаються користувачам на менших пристроях або пристроях з нижчою роздільною здатністю екранів, покращуючи час завантаження та загальну продуктивність [13].

Відкладене завантаження – ще один потужний метод оптимізації фронтенду, особливо для веб-сайтів з великою кількістю медіафайлів. Цей метод гарантує, що зображення, відео чи інші елементи завантажуються лише тоді,

коли вони потрібні, тобто коли користувач їх збирається переглянути. Замість завантаження всіх ресурсів під час першого запиту сторінки, відкладене завантаження затримує завантаження некритичних ресурсів до тих пір, поки вони не знадобляться, зменшуючи час початкового завантаження та заощаджуючи пропускну здатність. Це особливо корисно для сторінок з довгим прокручуваним контентом, таким як галереї зображень або списки продуктів, оскільки це гарантує, що спочатку завантажується лише видима частина сторінки, покращуючи як продуктивність, так і користувацький досвід [14].

Одним з ключових компонентів покращення швидкості реагування веб-застосунку є оптимізація JavaScript. JavaScript відповідає за більшу частину інтерактивності та динамічної поведінки сучасних веб-застосунків. Однак погано написаний або неефективний код JavaScript може суттєво вплинути на час завантаження сторінки та зробити роботу застосунку повільною. Оптимізація JavaScript включає різні методи, такі як розділення коду , яке розділяє великі файли JavaScript на менші, більш керовані фрагменти, та відкладене завантаження , коли файли JavaScript завантажуються асинхронно, щоб вони не блокували відображення іншого контенту. Крім того, зменшення кількості сторонніх скриптів та бібліотек, що завантажуються на сторінку, може допомогти мінімізувати кількість HTTP-запитів та зменшити затримки відображення [16].

Кешування – ще один важливий аспект оптимізації фронтенду. Використовуючи кешування браузера, веб-застосунки можуть зберігати такі ресурси, як зображення, CSS та JavaScript-файли, на пристрої користувача після першого відвідування, що дозволяє використовувати їх повторно під час наступних відвідувань без необхідності повторного завантаження. Це зменшує час завантаження для повторних відвідувачів та підвищує загальну продуктивність. Сервіс-воркери, тобто скрипти, що працюють у фоновому режимі, можуть використовуватися для активації більш просунутих стратегій кешування, включаючи можливості автономного режиму та фонову синхронізацію, що ще більше покращує продуктивність, особливо для

прогресивних веб-застосунків (PWA) [18].

Впровадження адаптивного дизайну також є важливою стратегією оптимізації, особливо враховуючи зростання використання мобільного інтернету. Адаптивний веб-дизайн гарантує, що веб-додаток безперешкодно адаптується до різних розмірів, орієнтацій та роздільної здатності екрана. Цей принцип дизайну досягається завдяки використанню гнучких макетів сітки, медіа-запитів та плавних зображень, які масштабуються.

Адаптивний дизайн гарантує, що користувачі смартфонів, планшетів та настільних комп'ютерів отримають найкращий можливий досвід без необхідності окремих мобільних веб-сайтів чи додатків, які можуть бути громіздкими та неефективними[19].

Зрештою, оптимізація шрифтів відіграє часто недооцінену роль у продуктивності фронтенду. Веб-шрифти можуть збільшити кількість HTTP-запитів і розмір сторінки, впливаючи на час завантаження. Для оптимізації шрифтів розробники можуть використовувати font-display: swap , властивість CSS, яка гарантує, що текст залишається видимим під час завантаження шрифтів, запобігаючи ефекту «спалаху невидимого тексту» (FOIT). Крім того, використання обмеженої кількості сімейств шрифтів, попереднє завантаження важливих шрифтів і застосування техніки підгруп шрифтів , де включаються лише символи, що використовуються на сторінці, може сприяти покращенню продуктивності та зменшенню непотрібного споживання ресурсів [21].

На завершення, оптимізація фронтенду є критично важливим компонентом для забезпечення найкращої роботи веб-додатків. Впроваджуючи такі методи, як мінімізація ресурсів, оптимізація зображень, ліниве завантаження, оптимізація JavaScript, кешування, адаптивний дизайн та оптимізація шрифтів, розробники можуть створювати швидші, ефективніші та зручніші для користувача додатки. Добре оптимізований фронтенд не тільки покращує час завантаження та швидкість реагування, але й покращує загальний користувацький досвід, сприяючи вищим показникам утримання, покращенню рейтингу SEO та більшому успіху бізнесу. Оскільки веб-додатки продовжують

розвиватися, оптимізація фронтенду залишатиметься безперервним процесом, який вимагає постійної оцінки та вдосконалення для задоволення постійно зростаючих потреб користувачів та пошукових систем [23].

2.3 Ключові метрики та показники продуктивності

Спосіб вимірювання продуктивності значно змінився. Хоча традиційні показники, такі як час завантаження сторінки, залишаються актуальними, Core Web Vitals від Google стали золотим стандартом для кількісної оцінки взаємодії з користувачем. Ця еволюція являє собою зміну парадигми від технічно орієнтованих показників до орієнтованих на користувача систем вимірювання продуктивності, які точніше відображають фактичний взаємодію з користувачем.

Впровадження Google Core Web Vitals як сигналів ранжування у травні 2021 року стало поворотним моментом у тому, як продуктивність впливає на видимість у пошуку. Згідно з комплексним аналізом 3400 веб-сайтів у різних секторах, сайти, що відповідають усім трьом пороговим значенням Core Web Vitals, демонструють середнє збільшення органічного трафіку на 26,3%. Детальне дослідження 8,4 мільйона сторінок для комп’ютерів та мобільних пристроїв показало, що лише 28,7% веб-сайтів наразі досягають «хороших» балів за всіма трьома показниками Core Web Vitals, що створює суттєві конкурентні можливості для організацій, які надають пріоритет цим показникам ефективності [3].

Найбільший показник контентного відображення (LCP) вимірює продуктивність завантаження за часом, коли найбільший елемент контенту стає видимим. Оптимальний LCP досягається протягом 2,5 секунд після завантаження сторінки. Дослідження, проведене на 1840 веб-сайтах по всьому світу, показує, що покращення LCP з «поганого» (>4,0 с) до «хорошого» (<2,5 с) корелює зі збільшенням коефіцієнта конверсії на 18,3% та зниженням показника відмов на 16,7%. Цей показник виявляється особливо важливим на мобільних

пристроях, де 72,4% користувачів відмовляються від покупок, стикаючись із повільним завантаженням зображень або контенту товарів. Вирішення проблеми LCP часто дає найбільшу рентабельність інвестицій серед оптимізацій продуктивності, причому модернізація формату зображень сама по собі зменшує LCP в середньому на 1,86 секунди на всіх досліджуваних веб-сайтах [3].

Затримка першого вводу (FID) кількісно визначає інтерактивність, вимірюючи час від першої взаємодії користувача до моменту, коли браузер може відповісти. Адаптивний сайт повинен підтримувати FID менше 100 мілісекунд. Комплексний аналіз поведінки користувачів на платформах електронної комерції в Індії показує, що веб-сайти, які досягають «хороших» показників FID (<100 мс), мають на 42,8% вищу глибину сторінки (кількість сторінок, відвіданих за сеанс) порівняно з сайтами з «поганими» показниками (>300 мс). Вплив особливо помітний у мобільному досвіді покупок, де дослідження 147 платформ електронної комерції показало, що кожні 50 мс покращення FID відповідає збільшенню дій додавання до кошика на 7,4%. Методи оптимізації JavaScript, спрямовані на перевантаження основного потоку, продемонстрували покращення FID в середньому на 68,2% в інтерактивних додатках електронної комерції зі складною фільтрацією продуктів та функцією пошуку [4].

Кумулятивне зміщення макета (CLS) оцінює візуальну стабільність, вимірюючи неочікувані зміни макета під час завантаження сторінки. Гарний користувацький досвід підтримує показники CLS нижче 0,1. Дослідження, в якому розглядалися 212 індійських веб-сайтів електронної комерції, показало, що зниження CLS з «поганого» (>0,25) до «доброго» (<0,1) зменшило показники відмови від мобільних кошиків на 22,6% та збільшило середню вартість замовлення на 13,9%. Вплив особливо значний на сторінках з детальною інформацією про продукт, де неочікувані зміни можуть призвести до випадкових кліків та розчарування користувачів, причому 67,3% користувачів повідомляють про негативний досвід, стикаючись зі змінами макета під час критичних взаємодій з покупками. Сайти електронної комерції, які впроваджують належні специфікації розмірів зображень, стратегії заповнювачів контенту та резервують

простір для динамічних елементів, знизили свої показники CLS в середньому на 0,17 бала, що призвело до збільшення часу перебування на сторінці продукту на 19,3% [4].

Ці показники важливі не лише для взаємодії з користувачем, але й для видимості в пошукових системах, оскільки зараз вони є важливими факторами ранжування. На конкурентних ринках електронної комерції веб-сайти, які досягли «добрих» рейтингів за всіма показниками Core Web Vitals, продемонстрували покращення рейтингу пошуку в середньому на 9-17 позицій для ключових слів з високим комерційним наміром, причому особливо значний вплив спостерігається в результатах мобільного пошуку, де показники ефективності мають більшу вагу в алгоритмах ранжування.

Таблиця 2.1

Основні показники веб-сайту та показники впливу на бізнес

Core Web Vital	Поріг хорошого балу	Вплив на конверсію	Вплив поведінки користувачів
LCP (Найбільший контентний малюнок)	< 2,5 секунди	Збільшення на 18,3%	Зниження показника відмов на 16,7%
FID (Затримка першого входу)	< 100 мс	Збільшення кількості додавань до кошика на 7,4% кожні 50 мс покращення	на 42,8% більша глибина сторінки
CLS (Сукупне зміщення макета)	< 0,1	Збільшення середньої вартості замовлення на 13,9%	Зниження кількості покинутих кошиків на 22,6%

Розширені стратегії розділення коду

Одним із найефективніших методів оптимізації є інтелектуальне розділення коду, яке продемонструвало значне покращення продуктивності в різних типах програм. Емпіричні дослідження, що вивчають проблеми продуктивності JavaScript, показують, що розділення коду може дати суттєві переваги з точки зору часу початкового завантаження та продуктивності

виконання. У дослідженні, що аналізувало 95 популярних програм JavaScript, впровадження методів розділення коду зменшило початковий розмір завантаження до 40% та час розбору приблизно на 37%, що безпосередньо відповідає на висновок про те, що розбір становить в середньому від 15 до 20% від загального часу виконання JavaScript у різних браузерах та на різних пристроях [5].

Розділення на основі маршрутів розділяє вашу програму за маршрутами, гарантуючи, що користувачі завантажують лише той код, який потрібен для їхнього поточного перегляду. Цей підхід особливо цінний для великих програм з окремими розділами. Аналіз сучасних веб-програм показує, що час виконання JavaScript може займати до 30% від загального часу завантаження сторінки на мобільних пристроях середнього класу, при цьому один файл JavaScript розміром 200 КБ потребує приблизно 100 мс для розбору та компіляції на типовому смартфоні. Завдяки впровадженню розділення на основі маршрутів, програми досягли скорочення початкового корисного навантаження JavaScript до 68% для складних програм з різними функціональними областями, що призвело до помітного покращення показників часу до взаємодії [5].

Розділення на основі компонентів підвищує деталізацію, завантажуючи компоненти лише за потреби. Наприклад, складний віджет панелі інструментів може завантажуватися лише під час прокручування у поле зору. Аналіз продуктивності показує, що час розбору та компіляції JavaScript лінійно збільшується з розміром коду, причому кожні додаткові 100 КБ JavaScript додають приблизно 50-100 мс часу обробки на мобільних пристроях середнього класу. Розділення на рівні компонентів забезпечує точніше керування завантаженням ресурсів, вирішуючи проблему, коли приблизно 40-45% JavaScript, завантаженого середньостатистичною веб-програмою, не виконується під час початкового рендерингу сторінки. Дані реальної реалізації показують, що програми, що використовують розділення на основі компонентів, можуть скоротити початковий час обробки JavaScript на 25-35% на мобільних пристроях [5].

Динамічний імпорт дозволяє завантажувати модулі JavaScript на вимогу, що ідеально підходить для таких функцій, як модальні вікна або панелі, які не потрібні негайно. Тести, що порівнюють традиційне пакетування зі стратегіями динамічного імпорту, показали, що реалізація динамічного імпорту для некритичної функціональності може покращити час взаємодії з початковою сторінкою на 30-45% для багатофункціональних програм. Цей метод є особливо цінним, враховуючи результати досліджень, які показують, що 10-15% часу виконання JavaScript зазвичай припадає на критичний шлях рендерингу, що безпосередньо впливає на сприйняття користувачем продуктивності. Характер динамічного імпорту на вимогу узгоджується з даними, що приблизно 60% коду JavaScript у типових програмах не потрібне для початкового рендерингу [5].

Існують проблеми з впровадженням, особливо враховуючи те, що методи оптимізації JavaScript можуть вимагати спеціалізованих знань та ретельного архітектурного планування. Дослідження показують, що приблизно 25% команд розробників повідомляють про труднощі з інтеграцією передових стратегій розділення коду в існуючі конвеєри збірки. Однак сучасні інструменти значно зменшили ці бар'єри, а такі пакети, як webpack, роблять ці методи все більш доступними для основних команд розробників.

WebAssembly (WASM) являє собою зміну парадигми для операцій, критично важливих для продуктивності. Дозволяючи коду, написаному такими мовами, як C++ або Rust, виконуватися в браузері майже з рідною швидкістю, WASM дозволяє виконувати складні обчислення, які були б надмірно повільними в JavaScript. Комплексний бенчмаркінг продуктивності WebAssembly для різних обчислювальних завдань демонструє, що реалізації WASM можуть виконуватися приблизно на 80% від швидкості рідного коду, перевершуючи еквівалентний код JavaScript у середньому в 1,3-2,5 раза для різних типів операцій. Для певних обчислювальних навантажень, що включають інтенсивні математичні обчислення, WASM може досягти покращення продуктивності в 3-5 разів порівняно з оптимізованим JavaScript [6].

Таблиця 2.2

Переваги продуктивності розділення коду

Оптимізація Техніка	**Зменшення розміру завантаження**	**Продуктивність Покращення**	**Вплив мобільних пристроїв**
Розділення на основі маршруту	До 68%	Зменшений час до початку взаємодії	Зекономлено 30% часу завантаження сторінки
Компонентно-орієнтований Розщеплення	40-45% JS не виконується спочатку	Скорочення часу обробки на 25-35%	Збережено 50-100 мс на кожні 100 КБ
Динамічний імпорт	60% JS спочатку не потрібне	На 30-45% краща інтерактивність початкової сторінки	Зменшено критичний шлях рендерингу на 10-15%

Впровадження WASM продовжує зростати, приблизно 6,3% веб-сайтів зараз впроваджують WebAssembly для операцій, критично важливих для продуктивності. Аналіз продуктивності реальних застосунків показує, що реалізації на основі WASM забезпечують стабільні переваги в продуктивності для завдань, що потребують ресурсів обчислень, у різних браузерних середовищах, а бенчмарки показують в середньому в 1,5 раза швидше виконання складних алгоритмів. Сумісність браузерів значно покращилася, і приблизно 95% поточної частки ринку браузерів тепер підтримує WebAssembly, що усуває ключову перешкоду для впровадження, яка існувала в попередніх циклах впровадження [6].

Такі програми, як обробка зображень, аналіз даних у реальному часі та 3D-рендеринг, отримують величезну користь від можливостей WASM. Наприклад, обчислювальні фінансові програми, які колись потребували настільного програмного забезпечення, тепер можуть ефективно працювати в браузерах. Тестування продуктивності симуляцій фінансових моделей показує, що реалізації WebAssembly можуть обробляти складні обчислення приблизно в 2,7 раза швидше, ніж еквівалентний код JavaScript, що дозволяє складним програмам безперебійно працювати в середовищах браузера. Графічно ресурсоємні програми демонструють ще більш разючі покращення, при цьому операції 3D-рендерингу виконуються приблизно в 3,2 рази швидше при

реалізації в WASM порівняно з альтернативами JavaScript [6].

Складність інтеграції WASM зменшилася з удосконаленням інструментарію, хоча накладні витрати на розробку залишаються приблизно на 15-20% вищими порівняно з реалізаціями на чистому JavaScript. Дослідження показують, що приблизно 82% розробників повідомляють про позитивну віддачу від інвестицій під час впровадження WebAssembly для критично важливих для продуктивності частин своїх програм. Стандартизація WebAssembly продовжує розвиватися, а системний інтерфейс WebAssembly (WASI) дозволяє використовувати ширші сценарії програм за межами браузерів та розширює потенційні варіанти використання цієї технології у веб-програмах.

Периферійні обчислення та безсерверні архітектури

Переміщення обчислень ближче до користувачів значно зменшує затримку, що є критичним фактором в оптимізації продуктивності сучасного веб-сайту. Затримка мережі становить значну частину загального часу завантаження сторінки, і дослідження показують, що традиційні серверні архітектури можуть створювати затримку від 80 до 120 мс для користувачів, які отримують доступ до контенту з централізованих центрів обробки даних, навіть за оптимальних умов. Впровадження рішень для периферійних обчислень продемонструвало середнє зниження затримки на 65,4% порівняно з традиційними централізованими архітектурами, причому покращення найбільш помітні для користувачів у регіонах, географічно віддалених від основних центрів обробки даних. Для застосунків з глобальними базами користувачів ці покращення призводять до вимірних покращень показників залученості користувачів та сприйнятої продуктивності, при цьому покращення часу відгуку на 45-70 мс легко помітне для кінцевих користувачів [7].

Периферійні обчислення виконують код на межах мережі, якомога ближче до користувачів. Цей підхід ідеально підходить для персоналізації, автентифікації та локалізації, які раніше вимагали передачі даних на віддалені сервери. Аналіз реалізації периферійних обчислень для доставки контенту показує, що розподіл обчислювальних ресурсів ближче до кінцевих користувачів

може скоротити час передачі даних в середньому на 53,2 мс для критичних операцій, таких як автентифікація користувачів та налаштування регіонального контенту. Ці покращення продуктивності особливо значні в мобільних контекстах, де мережеві умови часто змінюються, а операції, чутливі до затримки, мають більш виражений вплив на загальний досвід користувача. Впровадження периферійних обчислень, зокрема, для запитів API показало середнє покращення часу відгуку на 47,3% порівняно з централізованими кінцевими точками API [7].

Безсерверні функції дозволяють розробникам розгортати невеликі, спеціально створені функції, які автоматично масштабуються, усуваючи накладні витрати на управління сервером, зберігаючи при цьому продуктивність. Дослідження, що вивчають безсерверні архітектури, демонструють, що підходи на основі функцій можуть досягати часу холодного запуску в середньому 347 мс для середовищ виконання Node.js та 323 мс для середовищ виконання Python, з наступними теплими викликами, що реагують протягом 23-47 мс. Це є значною перевагою для програм зі змінними моделями трафіку, де традиційні серверні архітектури вимагали б значного надмірного виділення ресурсів для обробки пікових навантажень. З точки зору використання ресурсів, безсерверні реалізації продемонстрували покращення використання процесора на 34,7% та зменшення споживання пам'яті на 41,2% порівняно з еквівалентними серверними реалізаціями, що обробляють ідентичні робочі навантаження [7].

Інтеграція периферійних обчислень із безсерверними архітектурами являє собою особливо ефективну комбінацію для вирішення проблем продуктивності в сучасних веб-додатках. Польові випробування безсерверних функцій, розгорнутих на периферії, демонструють стабільні переваги в продуктивності, при цьому затримки p95 покращилися на 41,7% порівняно з регіонально-специфічними хмарними розгортаннями з ідентичною функціональністю. Організації, які впроваджують ці комбіновані підходи, повідомляють про зниження операційних витрат в середньому на 36,8%, одночасно покращуючи середній час відгуку та підвищуючи географічну

стійкість. Ці покращення безпосередньо впливають на ключові показники взаємодії з користувачем, зі спостережуваним скороченням часу до взаємодії (TTI) та затримки першого вводу (FID), коли операції, чутливі до затримки, переносяться в середовища периферійних обчислень.

Таблиця 2.3

Метрики продуктивності WebAssembly та JavaScript

Тип програми	**Покращення продуктивності**
Загальні обчислення	1,3-2,5 рази швидше
Математичні розрахунки	У 3-5 разів швидше
Фінансові моделі	у 2,7 раза швидше
3D-рендеринг	У 3,2 рази швидше

Оптимізація зображень та ресурсів

Медіа-ресурси часто складають більшість завантажених байтів, а дослідження показують, що зображення становлять приблизно 43-65% від загального корисного навантаження для середньої веб-сторінки. Оптимізація цих ресурсів дає значне підвищення продуктивності, що безпосередньо впливає на показники взаємодії з користувачем. Комплексний аналіз показує, що впровадження навіть базових стратегій оптимізації зображень може зменшити середню вагу сторінки на 25-40% без помітної втрати якості, що безпосередньо призводить до покращення часу завантаження та зменшення споживання пропускної здатності [8].

Сучасні формати зображень, такі як WebP та AVIF, пропонують покращене стиснення порівняно з традиційними форматами, часто зменшуючи розміри файлів на 30-50%, зберігаючи при цьому візуальну якість. Порівняльний аналіз форматів JPEG та WebP показує, що WebP досягає середнього зменшення розміру файлу на 25-34% порівняно із зображеннями JPEG з еквівалентною сприйнятою якістю при налаштуваннях якості 75-85. Зокрема, для фотографічного контенту розміри файлів WebP були в середньому на 26% меншими, ніж еквівалентні файли JPEG, зберігаючи при цьому порівнянну

візуальну якість на основі вимірювань індексу структурної подібності (SSIM). Сумісність браузерів значно покращилася за останні роки, і основні браузери тепер підтримують сучасні формати, які пропонують ці переваги ефективності [8].

Адаптивні зображення відображають зображення відповідного розміру на основі характеристик пристрою, вирішуючи поширену неефективність доставки зображень, оптимізованих для комп'ютерів, на мобільні пристрої. Дослідження показують, що приблизно 72% веб-сайтів відображають зображення з роздільною здатністю, вищою, ніж потрібно для пристрою перегляду, при цьому середній користувач мобільного телефону завантажує 2,2 МБ непотрібних даних зображення на сторінку. Впровадження стратегій адаптивних зображень значно зменшує ці втрати, причому зображення правильного розміру для мобільних пристроїв мають в середньому на 71% менший розмір файлів порівняно з їхніми аналогами для настільних комп'ютерів. Для веб-сайтів, орієнтованих на мобільних користувачів зі змінними мережевими з'єднаннями, ці оптимізації можуть скоротити ефективний час завантаження на 1,2-2,3 секунди, що безпосередньо впливає на основні показники користувацького досвіду [8].

Відкладене завантаження відкладає завантаження некритичних ресурсів до моменту їх необхідності, надаючи пріоритет контенту в поточній області перегляду. Дослідження веб-сайтів з великим обсягом контенту показують, що впровадження відкладеного завантаження для зображень нижче згину зменшило початкову вагу сторінки в середньому на 33% та покращило час початкового завантаження на 0,8-1,4 секунди на мобільних пристроях середнього класу. Переваги в продуктивності особливо помітні для галерей зображень та сторінок з довгим контентом, де користувачі зазвичай переглядають лише частину всього контенту під час початкової взаємодії. Сучасні браузери тепер підтримують вбудовані можливості відкладеного завантаження, що у багатьох випадках усуває необхідність у користувацьких реалізаціях JavaScript та пов'язаних з ними накладних витратах на продуктивність [8].

Таблиця 2.4

Метрики оптимізації формату зображення

Оптимізація Техніка	**Зменшення розміру файлу**	**Додаткова перевага**
WebP проти JPEG	на 25-34% менший	На 26% менше для фотографій
Адаптивні зображення	На 71% менший для мобільних пристроїв	Видаляє 2,2 МБ непотрібних даних
Відкладене завантаження	33% зменшення ваги сторінки	Покращений початковий рендеринг
Доставка CDN	Додаткове стиснення на 15-25%	Краща географічна продуктивність

Мережі доставки контенту (CDN) розподіляють ресурси по всьому світу, забезпечуючи користувачам доступ до ресурсів із сусідніх серверів та значно зменшуючи затримку. Аналіз продуктивності показує, що доставка зображень через CDN зменшує середній час відгуку на 52-67% порівняно з доставкою на вихідний сервер, причому покращення є найбільш суттєвими для користувачів у регіонах, географічно віддалених від основних місць розташування хостингу. Окрім переваг у затримці, CDN забезпечують додаткові переваги в продуктивності завдяки інтелектуальному кешуванню та стисненню, причому деякі постачальники пропонують автоматичну оптимізацію зображень, яка зменшує розміри файлів ще на 15-25 % завдяки конвертації формату та налаштуванню якості відповідно до пристрою, що запитує, та швидкості з'єднання.

Стратегічне кешування

Ефективні стратегії кешування можуть значно покращити повторні відвідування та загальну продуктивність програм. Аналіз продуктивності веб-сайтів показує, що впровадження належних механізмів кешування може скоротити час завантаження сторінок до 80% для відвідувачів, що повертаються, що є одним із найефективніших методів оптимізації, доступних веб-

розробникам. Дослідження, що вивчали комерційні веб-сайти з високим трафіком, показало, що впровадження комплексної стратегії кешування зменшило навантаження на сервер приблизно на 60% у періоди пікового трафіку, демонструючи переваги як для продуктивності, так і для інфраструктури. Ці покращення безпосередньо впливають на бізнес-показники, а правильно кешовані веб-додатки демонструють вимірні покращення показників залученості користувачів та коефіцієнтів конверсії [9].

Кешування браузера використовує заголовки, щоб вказати браузерам зберігати ресурси локально, створюючи значну перевагу в продуктивності для постійних відвідувачів. Аналіз поведінки браузера показує, що ефективна реалізація кешу може зменшити кількість HTTP-запитів до 60% під час наступних завантажень сторінок, з відповідним скороченням часу завантаження сторінки в середньому від 40 до 70% залежно від якості з'єднання та складності сторінки. Для статичних ресурсів, які змінюються рідко, реалізація агресивних політик кешу (використання заголовків Cache-Control з відповідними значеннями max-age) гарантує, що браузери можуть обслуговувати ці ресурси безпосередньо з локального сховища без мережевих запитів. Цей підхід особливо цінний для мобільних користувачів, де мережеві умови часто мінливі, а вплив зменшення кількості мережевих запитів на продуктивність найбільш виражений [9].

Кешування CDN зберігає контент на периферійних серверах, зменшуючи навантаження на вихідні сервери та покращуючи швидкість доставки. Вимірювання продуктивності показують, що ресурси, кешовані CDN, можуть доставлятися на 40-60% швидше, ніж ідентичні ресурси, що обслуговуються безпосередньо з вихідних серверів, причому покращення є найбільш значним для користувачів, географічно віддалених від основних центрів обробки даних. Окрім переваг у продуктивності, реалізація CDN забезпечує цінні можливості розподілу трафіку, а дослідження показують, що правильно налаштоване кешування CDN може зменшити навантаження на вихідний сервер до 70% за нормальних умов та на 85-90% під час піків трафіку. Ці переваги інфраструктури

призводять до підвищення надійності та узгодженості, причому веб-сайти з підтримкою CDN демонструють значно нижчий рівень помилок у періоди високого трафіку [9].

Сервісні працівники забезпечують можливості автономного режиму та можуть миттєво обробляти кешований контент, одночасно отримуючи оновлення у фоновому режимі. Дослідження впровадження показують, що правильно налаштоване кешування сервісних працівників може скоротити час завантаження сторінок на 60-90% для повторних відвідувань, усуваючи залежність від мережі для кешованих ресурсів. Програми, що реалізують сервісні працівники, продемонстрували значну стійкість до змін у мережі, зберігаючи основну функціональність під час перерв у з'єднанні та забезпечуючи суттєво покращений користувацький досвід у нестабільних мережах. Можливість забезпечення автономної функціональності являє собою зміну парадигми у можливостях веб-застосунків, оскільки прогресивні веб-застосунки, що використовують сервісні працівники, демонструють показники залученості, подібні до нативних програм у багатьох випадках використання [9].

Сучасні стратегії рендерингу

Різні підходи до рендерингу пропонують різні профілі продуктивності, які необхідно ретельно вибирати на основі вимог програми та характеристик цільової аудиторії. Дослідження, що порівнюють різні стратегії рендерингу, показують, що вибір між рендерингом на стороні сервера, статичним та клієнтським може вплинути на ключові показники продуктивності на 30-50%, причому оптимальний підхід залежить від типу програми, характеристик контенту та профілів цільового пристрою. Аналіз показників користувацького досвіду демонструє, що вибір відповідних стратегій рендерингу на основі цих факторів корелює з вимірними покращеннями показників залученості та конверсії в різних категоріях програм [10].

Серверний рендеринг (SSR) генерує HTML на сервері, забезпечуючи швидше перше фарбування контенту (First Contentful Paint) за рахунок ресурсів сервера. Бенчмаркінг продуктивності показує, що реалізації SSR досягають

початкового рендерингу на 30-40% швидше, ніж рендеринг на стороні клієнта, для складних програм, особливо на мобільних пристроях, де парсинг та виконання JavaScript є значними вузькими місцями. Цей підхід забезпечує особливі переваги для програм, орієнтованих на контент, де оптимізація для пошукових систем є пріоритетом, при цьому сторінки SSR демонструють значно кращі показники сканування та індексації порівняно з альтернативами, що рендеряться клієнтом. Компроміс полягає у вимогах до ресурсів сервера, оскільки SSR зазвичай вимагає додаткової обчислювальної потужності для обробки операцій рендерингу, які в іншому випадку відбувалися б на клієнтських пристроях [10].

Генерація статичних сайтів (SSG) попередньо рендерить сторінки під час збірки, що дозволяє миттєво доставляти статичний контент з мінімальним навантаженням на сервер. Аналіз продуктивності показує, що підходи SSG зазвичай досягають показників часу до першого байта (TTFB) на 70-80% швидше, ніж динамічний рендеринг для еквівалентного контенту, з відповідними покращеннями всіх показників рендерингу нижче за течією. Операційна ефективність цього підходу є не менш переконливою, оскільки реалізація SSG зменшує вимоги до ресурсів сервера на 80-90% порівняно з підходами рендерингу на вимогу. Для контенту, який змінюється рідко, цей підхід забезпечує виняткові характеристики продуктивності, мінімізуючи витрати на інфраструктуру та складність. Основне обмеження полягає в актуальності контенту, оскільки оновлення контенту вимагають перебудови уражених сторінок [10].

Інкрементальна статична регенерація (ISR) поєднує переваги SSG з можливістю оновлення певних сторінок на вимогу або через визначені проміжки часу. Дані щодо продуктивності показують, що підходи ISR зберігають приблизно 90% переваг продуктивності чистого SSG, забезпечуючи при цьому значно підвищену гнучкість контенту. Генеруючи сторінки під час збірки, але дозволяючи регенерацію на вимогу або за розкладом, цей підхід забезпечує майже статичну продуктивність з можливостями динамічного контенту.

Гібридний характер цього підходу надає особливі переваги для контенту з різною частотою оновлення, як показано на прикладі програм, багатих на контент, де часто оновлювані розділи можуть бути регенеровані незалежно від областей стабільного контенту. Цей збалансований підхід продемонстрував зростаюче впровадження для програм, які вимагають як продуктивності, так і гнучкості контенту [10].

Балансування складності та зручності обслуговування

Хоча ці передові методи пропонують значне покращення продуктивності, вони також створюють складність, якою необхідно ретельно керувати для забезпечення стійких результатів. Дослідження, що вивчають методи розробки, показують, що приблизно 30% початкових оптимізацій продуктивності зникають протягом 12 місяців, коли складність впровадження перевищує можливості команди або стандарти документації. Однак організації, які впроваджують структуровані процеси управління ефективністю, зберегли приблизно 85% приросту продуктивності протягом тривалих періодів вимірювання, що підкреслює важливість систематичних підходів до управління оптимізацією [9].

Інструменти аналізу пакетів допомагають виявити можливості для оптимізації без надмірного інжинірингу. Дані впровадження показують, що систематичний аналіз пакетів JavaScript зазвичай виявляє невикористаний код, який становить 15-25% від загального розміру пакетів у зрілих додатках, а його видалення призводить до пропорційного покращення продуктивності. Регулярний аналіз пакетів як частина робочих процесів розробки виявився особливо ефективним для запобігання регресії продуктивності, оскільки команди, що впроваджують автоматизований аналіз, виявляють потенційні проблеми приблизно на 70% раніше в циклі розробки порівняно з командами, які покладаються на моніторинг після розгортання. Ці превентивні підходи значно знизили вартість і складність вирішення проблем продуктивності, виявляючи їх до того, як вони потраплять у виробниче середовище [9].

Бюджети продуктивності встановлюють порогові значення для таких

показників, як розмір пакета та час завантаження, запобігаючи регресії продуктивності під час розробки. Аналіз команд розробників, які впроваджують формальні бюджети продуктивності, продемонстрував значно кращу довгострокову стабільність продуктивності, приблизно на 75% менше серйозних регресій продуктивності порівняно з командами без визначених бюджетів. Найефективніші впровадження включали автоматизовану перевірку бюджету в процеси безперервної інтеграції, запобігаючи впровадженню змін, які перевищували б встановлені порогові значення продуктивності. Такий підхід до управління гарантує, що продуктивність залишається пріоритетом протягом усього життєвого циклу розробки, а не розглядається лише під час спеціалізованих ініціатив з оптимізації [9].

Автоматизоване тестування гарантує, що оптимізація не призведе до регресій у функціональності або взаємодії з користувачем. Дослідження показують, що комплексне автоматизоване тестування продуктивності виявило приблизно 65% потенційних проблем з продуктивністю до розгортання у виробничому середовищі, порівняно з менш ніж 30% для організацій, які покладаються на ручне тестування або моніторинг після розгортання. Економічний вплив цього превентивного підходу є суттєвим, оскільки вартість вирішення проблем продуктивності значно зростає на кожному етапі процесу розробки та розгортання. Найефективніші стратегії тестування поєднують синтетичне тестування з моніторингом реальних користувачів, забезпечуючи охоплення різних типів пристроїв, мережевих умов та моделей використання, щоб гарантувати, що оптимізація приносить стабільні переваги всій базі користувачів [10].

2.4 Висновки по розділу

У другому розділі проаналізовано методи та інструменти оптимізації продуктивності web-сервісів. Розглянуто підходи до збору та аналізу даних про роботу системи, а також сучасні методи підвищення продуктивності на рівні

серверної та клієнтської частин. Досліджено ключові метрики, що використовуються для оцінювання ефективності web-сервісів, зокрема час відгуку, пропускну здатність і використання ресурсів. Визначено, що регулярний моніторинг та аналіз показників дозволяють своєчасно виявляти вузькі місця системи та підвищувати її ефективність.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ОПТИМІЗАЦІЙНИХ РІШЕНЬ

3.1 Реалізація методів оптимізації продуктивності web-сервісу

Оптимізація продуктивності – це метод(и), що використовуються для підвищення здатності системи відповідати на запити користувача в найкоротші терміни та/або виконувати значну кількість запитів за заданий час [16]. Як пояснювалося раніше, продуктивність визначається часом відгуку та пропускною здатністю [1]. Для зменшення часу відгуку (оптимізація продуктивності на фронтенді) необхідно впроваджувати способи, які мінімізують загальну кількість запитів ресурсів (HTTP-запитів) до сервера та скорочення часу обробки додаткових ресурсів, оскільки вони затримують завантаження системи [15]. Розміри сторінок та зображень у системі сприяють її повільному часу відгуку, оскільки сторінки та зображення великого розміру потребують більше часу, перш ніж їх буде отримано із сервера, отже, зменшення їх розміру ефективно зменшить час відгуку [7]. Обробка великої кількості запитів користувача одночасно з мінімальним часом відгуку (пропускною здатністю) покращує продуктивність на сервері [16].

Зменшення до мінімуму кількості сторінок, що складають систему, також зменшить час завантаження, оскільки зменшить кількість HTTP-запитів, що надсилаються на сервер [12].

Ajax було використано як технологію для створення системи, що використовується в цій роботі, оскільки вона надає можливості створювати та виконувати деякі функції на стороні клієнта без надсилання будь-яких запитів на сервер, що зменшує час обробки [17].

Існує багато способів налаштування продуктивності веб-застосунку для досягнення кращого часу відгуку, зберігаючи при цьому зовнішній вигляд, функціональність та швидкість реагування застосунку [2]. Це також дозволяє надсилати HTTP-запити асинхронно з кодом Javascript, без перезавантаження

всіх вихідних кодів HTML-сторінки. Ajax дуже добре покращує продуктивність інтерфейсу веб-застосунку [17]. Методи оптимізації продуктивності, вивчені та впроваджені після створення системи електронної комерції для цієї роботи, включають:

1. Мінімізація

Це видалення всіх непотрібних символів, таких як пробіли, символи нового рядка та коментарі, з вихідного коду без зміни функціональності. Це можна застосовувати до JavaScript, CSS, HTML та PHP.

Під час реалізації цього методу всі файли CSS, HTML, PHP та JavaScript мінімізуються за допомогою сайтів стиснення з відкритим кодом htmlcompressor.com та jscompress.com, що призводить до зменшення їх розміру на більший відсоток. Це допомагає зменшити розмір файлів, що, у свою чергу, скорочує час їх передачі від сервера до клієнтської частини.

2. Переформатування зображень

Це процес форматування всіх зображень у формат(и), який допоможе заощадити пропускну здатність і скоротити час відгуку. Тут обраним форматом є JPEG, оскільки метод стиснення, що лежить в його основі, зменшує розмір зображення без втрати якості зображення, видаляючи всі непотрібні дані. Це допомагає зменшити розмір зображень і пришвидшити їх передачу з сервера на сторону користувача.

3. Мережа доставки контенту

Це мережева система, яка доставляє веб-контент користувачеві на основі географічного розташування та походження веб-сторінки. Час відгуку користувача можна значно скоротити, просто розподіляючи статичний веб-контент у різних місцях. Замість включення бібліотек jQuery та JavaScript у код, у цій роботі для доставки бібліотек JavaScript та jQuery було використано мережу доставки контенту CloudFlare, що допомагає зменшити кількість трафіку до сервера, а також зменшує час відгуку.

4. Консолідація скриптів

Це об'єднання скриптів (CSS та JavaScript) у спільні файли, які можуть

використовуватись кількома сторінками. Оскільки кожен скрипт та таблиця стилів мають бути отримані HTTP-запитом до сервера, це збільшує накладні витрати на продуктивність, створюючи мережевий трафік між сервером і клієнтом. Кешування скриптів браузером зменшить кількість HTTP-запитів, необхідних для відображення сторінки, тим самим покращуючи продуктивність. Кешування скриптів здійснюється за допомогою зовнішніх таблиць, що дозволяє запитувати їх окремо від основного документа.

Браузери часто використовують прогресивний рендеринг, тобто відображають будь-який доступний контент якомога швидше. Неправильно розміщені посилання на таблиці стилів затримують рендеринг веб-сторінки, змушуючи браузер відкладати рендеринг усього документа, доки ці посилання не будуть вирішені. Тому доречно розміщувати посилання на таблиці стилів у верхній частині HTML-документа, щоб забезпечити належний прогресивний рендеринг і, як наслідок, покращити продуктивність програми. Завдяки використанню зовнішнього CSS та JavaScript, це допомагає зменшити кількість HTTP-запитів до сервера, тим самим зменшуючи час обміну даними.

5. Стиснення сторінок

Технології стиснення допомагають зменшити розмір веб-сторінок, тим самим зменшуючи час відгуку. Це головним чином досягається шляхом додавання етапів обробки для стиснення на сервері та розпакування у браузері. Хоча старі браузери не підтримують розпакування, більшість нових браузерів підтримують.

На сервері було ввімкнено стиснення GZIP, простий та ефективний спосіб економії пропускної здатності та пришвидшення роботи системи. Це зменшує розмір сторінок на рівні сервера, тим самим пришвидшуючи їх передачу до браузера. Сторінки розпаковуються на рівні браузера.

Файли, які вже стиснуті, такі як PDF та зображення, не потребують повторного стиснення, оскільки це може вплинути на використання процесора, а також збільшити їхній розмір.

6. Відкладення парсингу JavaScript

Файли JavaScript розміщуються в кінці тегу body, а не в head, щоб сторінка повністю завантажилася до завантаження файлів JavaScript. Розміщення файлів на початку сторінки призведе до затримки рендерингу сторінки, оскільки файли JavaScript мають бути повністю завантажені, перш ніж продовжити рендеринг сторінки. Це збільшує час відгуку системи.

7. Заголовок керування кешем

Використання кешу браузера – це ще один спосіб оптимізації продуктивності. Коли користувач вперше відвідує сторінку, він робить багато HTTP-запитів для завантаження всіх ресурсів сторінки (зображень, таблиць стилів, скриптів). Використовуючи кеш браузера, ресурси кешуються локально, тому їх не потрібно завантажувати під час наступних відвідувань, оскільки вони вже були кешовані раніше. Користувач може виконати умовний GET-запит, надавши заголовок If-Modified-Since. У відповідь на умовний GET, якщо ресурс не змінився, сервер додатків може повернути відповідь 304 Not Modified з значенням nobody; це зменшує обсяг переданих даних. Сервер додатків може вирішити надати заголовки Expires та/або Cache-Control разом із відповідями; вони використовуються для сигналізації клієнту, що ресурс слід повторно отримати лише після закінчення певної дати або періоду часу. Правильне використання вищезазначених заголовків може призвести до значного зменшення кількості HTTP-запитів. Таким чином, кешування слід використовувати, коли це можливо, для покращення продуктивності додатків.

8. Уникайте переадресацій

Переадресації досягаються за допомогою кодів статусу HTTP 3xx, наприклад, 301 (Переміщено назавжди) та 302 (Знайдено). Переадресація вказує на те, що користувачеві потрібно виконати деякі додаткові дії для завершення запиту. Основна проблема з переадресаціями полягає в тому, що вони уповільнюють час завантаження сторінки. Ці переадресації зазвичай відбуваються на різних етапах, наприклад, коли зворотна скісну риску (/) не додається в кінці URL-адреси. Переадресації зі зворотною скісну рискою можна виправити в Apache за допомогою псевдонімів.

Уникнення перенаправлень підвищить продуктивність веб-застосунку.

9. Зменшення кількості пошуків у системі доменних імен (DNS)

DNS зіставляє доменні імена з IP-адресами. DNS зазвичай займає 20-120 мілісекунд для пошуку IP-адреси для заданих доменів, і браузер не може нічого завантажити з цього імені хоста, доки пошук DNS не буде завершено.

Хоча DNS-запити кешуються для кращої продуктивності, а інформація з них розміщується в кеші DNS операційної системи, більшість браузерів також мають власний кеш. Деякі браузери, такі як Internet Explorer, кешують DNS-запити протягом 30 хвилин за замовчуванням, і в такій ситуації кеш операційної системи не має значення, якщо запис DNS існує в кеші браузера.

Кількість DNS-запитів буде дорівнювати кількості унікальних хостів на веб-сторінці у випадку порожнього кешу браузера. Отже, зменшення кількості унікальних імен хостів зменшить час завантаження сторінки.

10. Впровадження

Якість коду на стороні сервера, архітектурна складність та вибір сторонніх бібліотек та/або модулів можуть суттєво впливати на продуктивність веб-застосунку. Тому правильний вибір алгоритмів, архітектурних моделей та бібліотек, використання усталених ідіом кодування, оптимізація запитів до бази даних, ефективне використання фреймворків застосунків та ефективна модуляризація, серед іншого, є дуже важливими для оптимізації продуктивності.

У розробці цієї системи було використано трирівневу архітектуру, а всі використані бібліотеки, наприклад, бібліотеку jQuery, були вказані в кінці сторінки. Для бібліотек використовувалася доставка контенту, як пояснювалося раніше.

Правильне налаштування серверів програм і баз даних дуже важливе для належної продуктивності веб-програми. Сервер програм (і програма, що працює на ньому) часто повинні обробляти запити від кількох одночасних клієнтів (це допомагає з пропускною здатністю). Таким чином, налаштування пулів потоків, пулів підключень до бази даних, управління пам'яттю (наприклад, збирання сміття) тощо може суттєво впливати на продуктивність. Правильне

налаштування властивостей бази даних також важливе.

Для кращої продуктивності веб-застосунку необхідно досягти меншого часу відгуку та вищої пропускної здатності. Для цієї роботи була створена система онлайн-шопінгу (електронної комерції), оскільки в цій системі зберігається багато зображень, і на ній може одночасно перебувати велика кількість користувачів.

Для проведення тесту продуктивності до та після оптимізації було використано Web page test, безкоштовний онлайн-сервіс, який надає можливість тестування швидкості інтерфейсу веб-сайту, з такими результатами:

						Document Complete			Fully Loaded		
Load Time	First Byte	Start Render	Visually Complete	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In
6.731s	0.614s	6.183s	6.200s	6200	143	6.731s	16	291 KB	6.731s	16	291 KB

Рисунок 3.1 - Тест продуктивності до оптимізації.

					Document Complete			Fully Loaded		
Load Time	First Byte	Start Render	Speed Index	DOM Elements	Time	Requests	Bytes In	Time	Requests	Bytes In
2.031s	0.663s	1.786s	1967	157	2.031s	17	290 KB	2.352s	18	294 KB

Рисунок 3.2 - Тест продуктивності після оптимізації

Наведені вище рисунки показують, що завантаження системи до оптимізації продуктивності тривало приблизно 6,7 секунди, а після оптимізації – приблизно 2 секунди. Впровадження цих методів оптимізації продуктивності значно скоротило час відгуку, тим самим пришвидшивши завантаження системи, що, у свою чергу, покращило продуктивність.

Оскільки в цьому дослідженні ми порівнюємо продуктивність системи до та після впровадження методів оптимізації продуктивності, можна сміливо сказати, що результат дуже хороший, оскільки він покращив продуктивність системи.

Також було проведено навантажувальний тест системи, оскільки на продуктивність може значно впливати кількість одночасних користувачів системи. Цей тест було проведено для перевірки продуктивності системи в умовах високого трафіку. Тест було проведено за допомогою Blitz, дуже цікавого інструменту, який дозволяє проводити навантажувальне тестування веб-застосунків, використовуючи віртуальних користувачів по всьому світу. Blitz не є інструментом з відкритим вихідним кодом.

Перший тест тривалістю одну хвилину був проведений з кількістю користувачів від 1 до 3000 одночасно, які використовували систему, середній час відгуку становив 3,05 секунди, як показано на рисунку 3.3.

Рисунок 3.3 - Навантажувальне тестування з кількістю користувачів від 1 до 3000

Другий тест з користувачами, які змінювали кількість запитів від 1 до 5000, проводився протягом однієї хвилини, і середній час відгуку становив 4,96 секунди, як показано на рисунку 3.4.

Під час третього тесту з 10000 одночасними користувачами протягом того ж періоду часу було зафіксовано середній час відгуку 5,13 секунди, як показано на рисунку 3.5. Помилка тайм-ауту з'єднання була зафіксована в цьому тесті, коли кількість користувачів наближалася до 7000.

Рисунок 3.4 - Навантажувальне тестування з кількістю користувачів від 1 до 5000

Рисунок 3.5 - Навантажувальне тестування з кількістю користувачів від 1 до 10 000

Після навантажувальних тестів усі досягнуті значення часу відгуку та пропускної здатності вищі, ніж ті, що були досягнуті в тесті продуктивності після впровадження методів оптимізації продуктивності, як показано на рисунку 3.2, це пояснюється тим, що під час цього тесту в системі не було жодного користувача. Однак вони нижчі, ніж ті, що були досягнуті до впровадження методу оптимізації продуктивності, що показує, що методи допомогли зменшити час відгуку та збільшити пропускну здатність системи, тим самим оптимізуючи її продуктивність навіть під трафіком.

Оскільки метою дослідження є порівняння результатів тестів

продуктивності до та після оптимізації продуктивності, можна сказати, що впровадження дуже хороше, оскільки результати після впровадження кращі, ніж до впровадження.

3.2 Використання сучасних інструментів моніторингу та оптимізації

Розділ HTML head містить вбудований критичний CSS в тегу style, який оптимізує вміст верхньої частини сторінки. Це включає стилі для двох основних класів: класу header, який використовує flexbox з вирівнюванням між пробілами, відступами rem та білим фоном; та класу hero, встановлений на 80% висоти області перегляду з використанням сітки з центрованим вмістом. Після критичного CSS йде оптимізована стратегія завантаження CSS з використанням тегу preload link для некритичного файлу CSS. Цей тег link містить атрибути для асинхронного завантаження (обробник onload, який встановлює rel у 'stylesheet' після завантаження) та спільного використання ресурсів між джерелами. Для браузерів з вимкненим JavaScript передбачено резервний варіант у тегу noscript, який завантажує некритичний файл CSS безпосередньо за допомогою стандартного посилання на таблицю стилів. Ця реалізація забезпечує швидке початкове відображення критичного контенту, одночасно ефективно завантажуючи некритичні стилі.

Стратегії завантаження JavaScript

Нещодавні дослідження, що вивчають вплив фреймворків JavaScript та стратегій завантаження на 2800

Дослідження комерційних веб-сайтів виявило переконливі висновки щодо оптимізації продуктивності. Дослідження показало, що веб-сайти, що реалізують стратегічні шаблони завантаження JavaScript, зазнали скорочення загального часу блокування (TBT) на 54,3% та покращення показників найбільшого відображення контенту (LCP) на 47,8%. Цей комплексний аналіз особливо підкреслив, що застосунки на основі React, які реалізують оптимізовані стратегії розділення коду та лінивого завантаження, досягли зменшення

початкових розмірів пакетів на 39,6% порівняно з традиційними монолітними реалізаціями [6].

У дослідженні особливо наголошувалося на важливості належної реалізації атрибутів async та defer у сучасних веб-застосунках. Платформи електронної комерції, що використовують оптимізацію, специфічну для фреймворку, разом зі стратегічним завантаженням скриптів, продемонстрували покращення показників затримки першого вводу (FID) на 43,2%. Крім того, веб-сайти, що використовують стратегії динамічного імпорту для функціональності, специфічної для маршруту, показали зменшення невикористаного JavaScript під час початкового завантаження сторінок на 51,7%. Дослідження задокументувало, що новинні та медіа-сайти, які реалізують ці розширені шаблони завантаження, зазнали збільшення показників залученості користувачів на 38,9%, особливо під час подій з високим трафіком [6].

Аналіз оптимізацій, специфічних для фреймворку, виявив суттєвий вплив на продуктивність різних типів застосунків. Односторінкові застосунки (SPA), що реалізують стратегії детального розділення коду, досягли зменшення початкового розміру корисного навантаження JavaScript на 45,7%. Дослідження продемонструвало, що платформи електронної комерції, що використовують сучасні фреймворки JavaScript з оптимізованими шаблонами завантаження, зазнали збільшення коефіцієнтів мобільної конверсії на 33,8% та покращення середньої тривалості сеансу на 41,2%. Ці результати особливо підкреслили важливість стратегій оптимізації, специфічних для фреймворку, на ринках з різними мережевими умовами та можливостями пристроїв [6].

Зображення зазвичай становлять 50–80 % загального обсягу вебсторінки в байтах, що робить оптимізацію зображень однією з найефективніших стратегій підвищення продуктивності. Комплексний конвеєр оптимізації зображень охоплює кілька аспектів доставки зображень.

Лістинг 3.1 - Приклад реалізації оптимізованого завантаження JavaScript

```
<!-- Стратегія завантаження скриптів, оптимізована для продуктивності -->
<head>
    <!-- 1. React core (критичний, завантажується асинхронно) -->
    <script
        async
        src="https://cdn.example.com/react.production.min.js"
        data-performance="critical"
        crossorigin="anonymous">
    </script>

    <!-- 2. Основний бандл додатка з code splitting (рекомендовано defer) -->
    <script
        defer
        src="app-bundle.js"
        integrity="sha384-ваш_реальний_hash_тут"
        crossorigin="anonymous">
    </script>

    <!-- 3. Dynamic import для ледачого завантаження фіч (on-demand feature loading)
    <script type="module">
        // Функція для динамічного завантаження окремої функціональності
        const loadFeature = async () => {
            try {
                const module = await import('/modules/feature.js');
                module.initialize();        // викликаємо ініціалізацію модуля
            } catch (error) {
                console.error('Не вдалося завантажити функцію:', error);
            }
        };

        // Викликаємо завантаження, коли потрібно (наприклад, після певної події)
        // loadFeature();
    </script>

</head>
```

Детальні стратегії оптимізації:

- Вибір формату: вибір оптимального формату зображення залежно від типу контенту та підтримки браузерами. Векторні формати (SVG) для графіки та логотипів, WebP/AVIF для фотографічного контенту з широкою підтримкою браузерів, а також JPEG/PNG як резервні варіанти. Це часто реалізується через content negotiation або елемент <picture> для обслуговування різних форматів різним браузерам.
- Адаптивні зображення (Responsive images): застосування технік для доставки зображень відповідного розміру залежно від розмірів вікна перегляду, щільності пікселів та умов мережі. Це включає використання атрибутів srcset і sizes для надання кількох варіантів роздільної здатності, а також art direction через елемент <picture> для кадрування або модифікації зображень під різні розміри екрана.
- Ліниве завантаження (Lazy loading): відкладення завантаження зображень, що перебувають поза екраном, до моменту, коли користувач

прокрутить до них. Це зменшує початкову вагу сторінки та забезпечує пріоритет видимому контенту. Реалізується за допомогою нативного атрибута loading="lazy", а для старіших браузерів - через Intersection Observer API.

- Компресія з урахуванням вмісту (Content-aware compression): застосування різних рівнів стиснення залежно від типу зображення, його важливості та розміщення на сторінці. Критичні зображення-герої можуть використовувати вищу якість, тоді як допоміжні зображення + більш агресивну компресію. Сучасні інструменти аналізують вміст зображення, щоб інтелектуально застосовувати стиснення, зберігаючи деталі в важливих областях і сильно стискаючи фон.
- CDN для зображень та сервіси трансформації: використання спеціалізованих CDN, які виконують оптимізацію зображень «на льоту» - змінюють розмір, формат, стискають і кешують зображення автоматично залежно від параметрів запиту.
- Формати наступного покоління: впровадження передових форматів, таких як AVIF, які забезпечують найкраще співвідношення ступеня стиснення до якості, з відповідними механізмами резервного відображення для браузерів, що їх ще не підтримують.

Глибоке дослідження впровадження форматів зображень на 25 000 високонавантажених вебсайтах виявило важливі висновки щодо реального впливу сучасних форматів зображень на продуктивність. Аналіз понад 2,3 мільйона унікальних зображень у основних веббраузерах показав, що сайти, які впровадили WebP з правильними механізмами fallback, досягли в середньому покращення часу завантаження на 37,2 % порівняно з традиційними реалізаціями JPEG. Крім того, користувачі Chrome отримали зменшення часу завантаження зображень на 42,8 % при використанні WebP, тоді як користувачі Safari показали покращення на 31,5 % при обслуговуванні оптимізованих форматів JPEG 2000 через правильне content negotiation [7].

Таблиця 3.1:

Результати оптимізації ресурсів

Тип оптимізації	**Покращення продуктивності**	**Контекст впровадження**
Впровадження WebP	Час завантаження швидший на 37.2%	порівняно з традиційним JPEG
Стиснення AVIF	Розмір файлу менший на 48.7%	порівняно з WebP (SSIM >0.92)
Впровадження Critical CSS	Показник FCP кращий на 61.3% (Мобільні)	Платформи електронної комерції
Розподіл коду за маршрутами	Навігація швидша на 47.3%	Подальші завантаження сторінок
Впровадження HTTP/2	Час завантаження сторінки зменшено на 61.8%	порівняно з HTTP/1.1

Впровадження форматів наступного покоління, таких як AVIF, продемонструвало багатообіцяючі результати в конкретних сценаріях використання. Дослідження зафіксувало, що зображення товарів в електронній комерції, стиснуті за допомогою AVIF, досягли середнього зменшення розміру файлу на 48,7 % порівняно з WebP при збереженні індексу структурної подібності (SSIM) вище 0,92. Однак аналіз сумісності браузерів показав, що лише 67,3 % користувачів можуть безпосередньо користуватися AVIF, що вимагає надійних стратегій fallback.

Дослідження особливо відзначило, що новинні сайти, які впровадили адаптивні техніки зображень із форматно-специфічними оптимізаціями, досягли зменшення метрики Largest Contentful Paint (LCP) на 41,6 % у всіх основних браузерах [7].

Оптимізація JavaScript

Нещодавній аналіз продуктивності, зосереджений на методах оптимізації JavaScript, виявив значні покращення завдяки впровадженню сучасних стратегій пакетного об'єднання. Згідно з комплексним бенчмаркінгом 1500 робочих додатків, веб-сайти, що впроваджують передові методи розділення коду, досягли зменшення початкового розміру корисного навантаження JavaScript на 43,2% та покращення показників часу до взаємодії (TTI) на 51,7%. Дослідження особливо підкреслило, що впровадження динамічного імпорту з детальними стратегіями

фрагментації призвело до зменшення затримки першого вводу (FID) на 38,9% на мобільних пристроях [8].

```
<picture>
    <!-- AVIF для сучасних браузерів -->
    <source
        media="(min-width: 1024px)"
        srcset="hero-1920.avif 1920w, hero-1280.avif 1280w, hero-640.avif 640w"
        sizes="100vw"
        type="image/avif">

    <!-- WebP як основний сучасний формат -->
    <source
        srcset="hero-1920.webp 1920w, hero-1280.webp 1280w, hero-640.webp 640w"
        sizes="(min-width: 1280px) 1920px, (min-width: 640px) 1280px, 640px"
        type="image/webp">

    <!-- JPEG як fallback -->
    <img
        src="hero-640.jpg"
        srcset="hero-1920.jpg 1920w, hero-1280.jpg 1280w, hero-640.jpg 640w"
        sizes="(min-width: 1280px) 1920px, (min-width: 640px) 1280px, 640px"
        alt="Hero banner showcasing product collection"
        loading="lazy"
        decoding="async"
        fetchpriority="high"
        width="1920"
        height="1080">
</picture>
```

Лічтинг 3.2 - Приклад реалізації оптимальної передачі зображень

Сучасна оптимізація пакетів JavaScript розвинулася для вирішення конкретних проблем продуктивності у складних веб-додатках. Дослідження задокументувало, що впровадження розділення коду на основі маршрутів зі стратегіями попередньої вибірки покращило час наступної навігації сторінками на 47,3%. Платформи електронної комерції, що використовують ці методи, повідомили про збільшення коефіцієнта конверсії на 34,8%, що пояснюється покращенням продуктивності навігації. Аналіз підкреслив, що правильне впровадження конфігурацій струшування дерев призвело до зменшення кінцевого розміру пакетів на 29,6% у всіх аналізованих додатках [8].

```jsx
// Advanced code splitting + performance monitoring (2026 best practices)
import React, { Suspense, lazy } from 'react';
import { metrics } from './monitoring';

// Динамічний імпорт з вимірюванням продуктивності
const ProductPage = lazy(() => {
  metrics.markStart('product-page-load');

  return import(
    /* webpackChunkName: "product-page" */
    /* webpackPreload: true */
    './pages/Product'
  ).then((module) => {
    metrics.markEnd('product-page-load');
    metrics.measure('product-page-load');
    return module;
  });
});

// Компонент з fallback
const LazyProductPage = () => (
  <Suspense
    fallback={
      <div className="loading-fallback">
        Завантаження сторінки товару...
      </div>
    }
  >
    <ProductPage />
  </Suspense>
);

export default LazyProductPage;
```

Лістинг 3.3 - Приклад реалізації для розділення коду на основі маршрутів

Впровадження передових методів оптимізації JavaScript продемонструвало вимірний вплив на показники продуктивності в реальному світі.

Дослідження показало, що застосунки, що використовують стратегії гранулярного поділу коду, показали покращення часу до першого байта (TTFB) на 45,2% для наступних завантажень сторінок. Крім того, вебсайти, що реалізують складні конфігурації струшування дерев, показали зменшення невикористаного коду JavaScript на 32,7%, що призвело до покращення продуктивності виконання, особливо на мобільних пристроях середнього класу [8].

Кешування та оптимізація мережі: розширені стратегії впровадження,

впровадження Service Worker

Комплексне дослідження прогресивних веб-додатків (PWA) у навчальних закладах Індії виявило значне покращення продуктивності завдяки впровадженню сервісних працівників. Дослідження, в якому проаналізовано 850 освітніх веб-сайтів та додатків, показало, що впровадження сервісних працівників призвело до покращення доступності офлайн-контенту на 72,4% та зменшення споживання даних для постійних відвідувачів на 58,3%. У дослідженні особливо підкреслено, що освітні платформи, що обслуговують віддалені райони з нестабільними мережевими умовами, зазнали збільшення залученості студентів на 64,7% після впровадження надійних стратегій кешування сервісних працівників [9].

Впровадження Service Worker показало значний вплив на оптимізацію ресурсів та можливості роботи в автономному режимі. Аналіз показав, що освітні платформи, що використовують стратегічне управління кешем, досягли скорочення часу завантаження початкової сторінки на 47,8% та покращення швидкості подальшої навігації на 51,2%. Крім того, установи, що впроваджують можливості фонової синхронізації, повідомили про збільшення успішності виконання завдань на 43,6% у періоди нестабільності мережі, що безпосередньо сприяє покращенню результатів навчання в районах з обмеженим підключенням [9].

HTTP/2 та оптимізація кешування

HTTP/2 фундаментально змінює багато передових практик продуктивності, усуваючи основні обмеження HTTP/1.1. У поєднанні зі стратегічним кешуванням ці сучасні протоколи забезпечують значне покращення продуктивності.

Детальні методи оптимізації:

• Пріоритизація ресурсів : впровадження підказок щодо пріоритизації на стороні сервера, які повідомляють браузерам, які ресурси є найважливішими, забезпечуючи ефективний розподіл пропускної здатності. Це включає використання фреймів та заголовків HTTP/2 PRIORITY для позначення

відносної важливості різних ресурсів.

```
const CACHE_VERSION = 'edu-v1.2.0';

const CACHE_NAMES = {
  course: `course-${CACHE_VERSION}`,
  resources: `resources-${CACHE_VERSION}`,
  assignments: `assignments-${CACHE_VERSION}`
};

// Install
self.addEventListener('install', event => {
  event.waitUntil(
    Promise.all([
      caches.open(CACHE_NAMES.course).then(cache =>
        cache.addAll(['/courses/', '/styles/course.css', '/scripts/learning.js'])
      ),
      caches.open(CACHE_NAMES.resources).then(cache =>
        cache.addAll(['/api/course-structure', '/api/learning-progress'])
      )
    ]).then(() => self.skipWaiting())
  );
});

// Fetch
self.addEventListener('fetch', event => {
  const url = new URL(event.request.url);

  // Курси — Cache First + Background Update
  if (url.pathname.startsWith('/courses/')) {
    event.respondWith(
      caches.match(event.request).then(cached => {
        const fetchPromise = fetch(event.request).then(res => {
          if (res && res.status === 200) {
            caches.open(CACHE_NAMES.course).then(cache => cache.put(event.request, res.clon
          }
          return res;
        });
        return cached || fetchPromise;
      })
    );
    return;
  }
```

Лістинг 3.4 - Приклад реалізації для оптимізованого для освіти сервісного працівника

HTTP/2 та оптимізація кешування

HTTP/2 фундаментально змінює багато передових практик продуктивності, усуваючи основні обмеження HTTP/1.1. У поєднанні зі стратегічним кешуванням ці сучасні протоколи забезпечують значне покращення продуктивності.

Детальні методи оптимізації:

• Пріоритизація ресурсів : впровадження підказок щодо пріоритизації на стороні сервера, які повідомляють браузерам, які ресурси є найважливішими, забезпечуючи ефективний розподіл пропускної здатності. Це включає використання фреймів та заголовків HTTP/2 PRIORITY для позначення відносної важливості різних ресурсів.

· Пулінг з'єднань : налаштування серверів для ефективної підтримки

постійних з'єднань, що дозволяє кільком запитам повторно використовувати одне й те саме TCP-з'єднання. Це зменшує накладні витрати на встановлення з'єднань та використовує можливості мультиплексування HTTP/2.

· Стратегічне стиснення: впровадження алгоритмів стиснення, орієнтованих на контент, понад простий GZIP, таких як Brotli для текстових ресурсів (що пропонує на 15-25% краще стиснення, ніж GZIP), та спеціальна оптимізація для різних типів контенту. Це включає налаштування відповідних рівнів стиснення, які балансують між використанням процесора та коефіцієнтом стиснення.

· Оптимізація політики кешу: впровадження складних директив кешування, які балансують між актуальністю та продуктивністю. Це включає використання «незмінних» директив керування кешем для версійних ресурсів, ефективну реалізацію ETags та умовних запитів, а також зміну тривалості кешування залежно від типу контенту та частоти оновлення.

· Серверні запити: Стратегічне впровадження серверних запитів HTTP/2 для проактивного надсилання критично важливих ресурсів клієнту до їх явного запиту, особливо ресурсів, що блокують рендеринг, необхідних під час початкового завантаження сторінки.

· Консолідація доменів: Зміна найкращої практики шардування доменів HTTP/1.1, замість цього консолідація ресурсів на меншій кількості доменів для максимізації повторного використання з'єднань завдяки можливостям мультиплексування HTTP/2.

Нещодавній аналіз впровадження HTTP/2 на веб-сайтах з високим трафіком показав суттєве покращення продуктивності завдяки впровадженню сучасних протоколів. Дослідження показує, що веб-сайти, що впроваджують HTTP/2 з належною пріоритезацією ресурсів, зазнали скорочення часу завантаження сторінок на 61,8% порівняно з реалізаціями HTTP/1.1. У дослідженні особливо підкреслюється, що платформи електронної комерції, що використовують можливості push-повідомлень на сервері HTTP/2, досягли покращення на 43,2% у першому контенті.

Метрики Paint (FCP) та зменшення накладних витрат на підключення до сервера на 38,7% [10].

Впровадження HTTP/2 разом зі стратегічними конфігураціями кешування продемонструвало значні переваги в продуктивності. Аналіз показав, що веб-сайти, що використовують мультиплексування HTTP/2 з оптимізованими політиками кешування, досягли скорочення часу до першого байта (TTFB) на 54,3% та покращення ефективності завантаження ресурсів на 47,6%. Крім того, дослідження задокументувало, що новинні та медіа-сайти, які впроваджують ці стратегії, зазнали збільшення показників залученості користувачів на 41,9%, особливо на мобільних пристроях, що отримують доступ до контенту за різних мережевих умов [10].

Впровадження цих передових стратегій HTTP/2 та кешування показало помітний вплив на показники продуктивності в реальному світі. Дослідження задокументувало, що платформи електронної комерції, що використовують HTTP/2 з оптимізованою пріоритезацією ресурсів, зазнали покращення коефіцієнтів конверсії на 49,2%, що пояснюється покращенням продуктивності завантаження сторінок. Крім того, веб-сайти, що впроваджують складні стратегії кешування разом з HTTP/2, продемонстрували зниження споживання пропускної здатності на 43,7% та покращення загального користувацького досвіду на 38,4% [10].

Моніторинг ефективності: розширені стратегії впровадження та автоматизації, впровадження бюджетів ефективності

Бюджети ефективності встановлюють вимірювані порогові значення для показників ефективності, створюючи підзвітність та запобігаючи регресії продуктивності протягом життєвого циклу розробки. Вони слугують бар'єрами, що допомагають командам підтримувати продуктивність як основну вимогу до продукту, а не як другорядну.

Детальні стратегії впровадження:

• Визначення критичних показників : визначення конкретних показників ефективності, які найбільше відповідають вашому застосунку та взаємодії з

користувачем. Це може включати основні показники веб-показників (LCP, FID, CLS), користувацькі показники, такі як час до взаємодії або перше змістовне малювання, або показники, специфічні для бізнесу, такі як час завершення оформлення замовлення для сайтів електронної комерції.

```
# HTTP/2 optimized Nginx configuration (2026 best practices)
http {
    # HTTP/2 and modern SSL/TLS configuration
    listen 443 ssl http2;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES25
    ssl_prefer_server_ciphers off;

    # Recommended HTTP/2 additional settings
    http2_max_concurrent_streams 128;
    http2_chunk_size 8k;

    # Static assets optimization (JS, CSS, Fonts)
    location ~* \.(js|css|woff2|woff|ttf|otf)$ {
        expires 1y;
        add_header Cache-Control "public, immutable, no-transform" always;
        add_header Priority "u=high" always;          # сучасний формат Priority

        http2_push_preload on;

        # Brotli compression
        brotli on;
        brotli_comp_level 5;
        brotli_types application/javascript text/css font/woff2 font/woff;
    }

    # API / dynamic content
    location /api/ {
        add_header Cache-Control "private, no-cache, must-revalidate" always;
        add_header Priority "u=low" always;

        # Proxy to backend with HTTP/2 (frontend only)
        proxy_http_version 1.1;          # Nginx поки не підтримує HTTP/2 до бекенду
        proxy_set_header Connection "";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;

        # Connection pooling
        keepalive_timeout 65;
        keepalive_requests 100;
    }
}
```

Лістинг 3.5 - Приклад реалізації кешування, оптимізованого для HTTP/2

· Встановлення відповідних порогових значень: встановлення реалістичних цілей на основі очікувань користувачів, порівняння з конкурентами та бізнес-цілей. Ці порогові значення повинні бути достатньо складними, щоб стимулювати вдосконалення, але водночас достатньо досяжними, щоб підтримувати моральний дух команди та прогрес.

· Інтеграція в CI/CD: Автоматизація перевірки бюджету продуктивності як частини конвеєра безперервної інтеграції та розгортання. Це включає налаштування інструментів збірки, таких як Webpack, для видачі попереджень або помилок у разі перевищення бюджетів, а також інтеграцію інструментів тестування продуктивності, таких як Lighthouse CI, для перевірки метрик під час автоматизованого тестування.

· Впровадження заходів підзвітності: чітке визначення відповідальності за дотримання стандартів ефективності в команді розробників. Це може включати призначення відповідальних за ефективність, встановлення процесів перевірки змін, що впливають на ефективність, та включення показників ефективності до командних OKR або критеріїв успіху.

· Бюджети на рівні компонентів: Розбиття загального бюджету сайту на менші бюджети для окремих компонентів або функцій, що дозволяє командам зрозуміти, як кожен елемент сприяє цілому, та запобігає «трагедії спільного надбання», коли жодна окрема зміна не здається суттєвою.

• Поступове вдосконалення : встановлення багаторівневих бюджетів для різних можливостей пристроїв та мережевих умов, що забезпечує якісне користування всіма користувачами.

Нещодавнє дослідження, що вивчало впровадження бюджетування на основі результатів діяльності в різних інституційних структурах, виявило значну кореляцію між структурованим моніторингом бюджету та ефективністю організації. Комплексне дослідження, що аналізує показники ефективності у 127 державних установах, показало, що впровадження систематичних бюджетів на основі результатів діяльності призвело до покращення показників підзвітності на 43,7% та підвищення ефективності використання ресурсів на 38,2%. Крім того, організації, які впроваджують рамки бюджетування на основі результатів діяльності, повідомили про збільшення показників досягнення цілей на 41,5% та покращення загальної операційної ефективності на 35,8% [11].

Таблиця 3.2:

Вплив оптимізації продуктивності на бізнес

Метрика	**Покращення**	**Контекст**
Дохід	51.30%	Після оптимізації Core Web Vitals
Коефіцієнт конверсії (Мобільні)	73.60%	Після оптимізації порівняно з початковим станом
Середній чек (AOV)	47.80%	При стабільних показниках продуктивності
Утримання клієнтів	62.40%	Протягом 6-місячного періоду
Вартість залучення клієнта	-29.40%	Завдяки покращенню органічного охоплення

Впровадження стратегій бюджетування на основі результатів вимагає ретельного врахування змінних вимірювань та показників успіху. Дослідження показало, що установи, які впроваджують комплексні системи вимірювання ефективності, досягли покращення ефективності стратегічного планування на 47,2% та підвищення ефективності розподілу ресурсів на 39,6%. Ці результати особливо підкреслили важливість встановлення чітких показників ефективності, оскільки організації, які використовують чітко визначені показники, досягли зростання операційної прозорості та підзвітності на 42,3%.

Автоматизоване тестування продуктивності забезпечує послідовний моніторинг показників продуктивності протягом усього життєвого циклу розробки, виявляючи регресії на ранній стадії та об'єктивно перевіряючи покращення.

```
// webpack.config.js — Performance optimized for institutional / educational platform
const path = require('path');

module.exports = {
  mode: 'production',

  // Performance budget + hints
  performance: {
    maxAssetSize: 244 * 1024,        // 244 KB
    maxEntrypointSize: 244 * 1024,   // 244 KB
    hints: 'error',                  // показувати помилку при перевищенні

    // Asset size warnings
    assetFilter: (assetFilename) => {
      return !assetFilename.endsWith('.map'); // ігноруємо source maps
    }
  },

  optimization: {
    moduleIds: 'deterministic',
    runtimeChunk: 'single',

    splitChunks: {
      chunks: 'all',
      maxInitialRequests: 25,
      maxAsyncRequests: 30,
      minSize: 20 * 1024,      // 20 KB
      maxSize: 244 * 1024,     // 244 KB

      cacheGroups: {
        vendor: {
          test: /[\\/]node_modules[\\/]/,
          name: 'vendors',
          priority: 20,
          reuseExistingChunk: true
        },
        common: {
          minChunks: 2,
          priority: 10,
          reuseExistingChunk: true
        }
      }
    }
  },

  // Output configuration
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: '[name].[contenthash:8].js',
    chunkFilename: '[name].[contenthash:8].chunk.js',
    clean: true
  }
```

Лістинг 3.6 - Приклад впровадження моніторингу на основі результатів

Детальні стратегії впровадження:

· Безперервна інтеграція: інтеграція тестування продуктивності

безпосередньо в конвеєри CI/CD для виявлення регресій продуктивності до того, як вони потраплять у робочу версію. Це створює негайні петлі зворотного зв'язку для розробників і запобігає поступовому зниженню продуктивності, яке часто трапляється, коли продуктивність тестується лише вручну або нечасто.

· Синтетичне тестування: впровадження контрольованих, відтворюваних тестів, що імітують взаємодію користувачів за стандартизованих умов. Це включає налаштування тестових середовищ для забезпечення узгоджених умов обладнання, мережі та кешу, що дозволяє надійно порівнювати результати з часом. Такі інструменти, як WebPageTest, Lighthouse та Sitespeed.io, можна інтегрувати в автоматизовані конвеєри для запуску для кожного запиту на збірку або pull request.

· Моніторинг реальних користувачів (RUM) : збір даних про продуктивність із реальних сеансів користувачів на різних пристроях, браузерах та за мережевих умов. Це надає уявлення про реальну продуктивність, яку синтетичне тестування не може охопити, включаючи географічні відмінності, різноманітність пристроїв та вплив сторонніх служб у виробничих умовах. Дані RUM доповнюють синтетичне тестування, перевіряючи, чи покращення лабораторії призводять до переваг у польових умовах.

· Виявлення регресії: впровадження автоматизованих систем для виявлення статистично значущого зниження продуктивності між збірками або релізами. Це включає встановлення базових профілів продуктивності, визначення прийнятних порогів відхилення та створення механізмів сповіщення, коли показники виходять за межі очікуваних діапазонів.

· Середовища тестування продуктивності : створення спеціалізованих середовищ, що відображають виробничі середовища, але включають додаткову інструментарій для глибшого аналізу продуктивності, що дозволяє командам досліджувати складні проблеми продуктивності, не впливаючи на реальних користувачів.

· Порівняння візуальної продуктивності : впровадження інструментів, які фіксують та порівнюють візуальний прогрес візуалізації сторінки між

версіями, виділяючи не лише зміни показників, але й відмінності в моделях завантаження, які сприймає користувач.

Широкий аналіз практики забезпечення якості програмного забезпечення в освітніх закладах виявив значні переваги впровадження автоматизованих систем тестування. Дослідження, в якому розглядалося 85 освітніх програмних проектів, показало, що організації, які впроваджують протоколи автоматизованого тестування, зазнали зменшення кількості проблем після розгортання на 51,4% та покращення загальної надійності програмного забезпечення на 47,8%. У дослідженні особливо підкреслено, що установи, які використовують методи безперервної інтеграції, досягли збільшення показників якості коду на 43,2% та скорочення часу налагодження на 38,7% [12].

Впровадження стратегій автоматизованого тестування показало помітний вплив на ефективність розробки програмного забезпечення. Згідно з дослідженням, команди розробників, які впроваджують автоматизовані системи тестування, повідомили про покращення показників успішного розгортання на 45,6% та скорочення часу циклу забезпечення якості на 41,3%. Аналіз також показав, що навчальні заклади, які впроваджують комплексну автоматизацію тестування, зазнали зменшення використання ресурсів для процесів ручного тестування на 39,8% та покращення загальних термінів виконання проектів на 36,4% [12].

Впровадження цих стратегій тестування освітнього програмного забезпечення продемонструвало значне *покращення якості та надійності програмного забезпечення. Дослідження* показує, що установи, які використовують автоматизовані системи тестування, досягли скорочення помилок доставки контенту на 44,7% та покращення стабільності навчальної платформи на 42,1%. Крім того, організації, які впроваджують комплексну автоматизацію забезпечення якості, повідомили про збільшення показників задоволеності студентів на 38,5% та скорочення вимог до технічної підтримки на 35,9% [12].

Висновок: Оптимізація продуктивності фронтенду є критичним фактором у забезпеченні виняткового веб-досвіду в сучасному цифровому середовищі.

Завдяки детальному аналізу різних методів оптимізації та їх впровадження, це дослідження демонструє, що цілісний підхід до підвищення продуктивності призводить до суттєвого покращення залученості користувачів, коефіцієнтів конверсії та загального успіху бізнесу. Дослідження підкреслює важливість впровадження комплексних стратегій оптимізації на різних рівнях веб-додатків, від критичного управління CSS до складних механізмів кешування. Результати дослідження особливо підкреслюють важливість автоматизованих систем моніторингу та тестування продуктивності для підтримки послідовних стандартів продуктивності. Оскільки веб-технології продовжують розвиватися, організації повинні надавати пріоритет оптимізації продуктивності як фундаментальному аспекту своєї практики розробки, забезпечуючи надійні, адаптивні та орієнтовані на користувача веб-додатки, які відповідають вимогам сучасних користувачів у різних мережевих умовах та можливостях пристроїв.

3.3 Тестування, аналіз та оцінка продуктивності системи

В експерименті використовувалася комбінована платформа навантажувального тестування, побудована за допомогою K6 та Apache JMeter, що імітувала обсяги одночасних запитів, що зростали від 1000 до 10 000. Середовище виконання бекенду складалося з Node.js 20.11 та Redis 7.2, розгорнутих на вузлах двосокетної архітектури Intel Xeon Gold 6348 з 256 ГБ пам'яті та кластерами NVMe SSD. Фронтенд використовував серверне рішення для рендерингу Vite + React 18. Зворотний проксі-канал був побудований за допомогою Nginx 1.24, налаштований з максимальною кількістю підключень 64 000 та інтервалом вибірки 1 секунда. Відстежувані метрики включали середню затримку відповіді, затримку P95, використання процесора та пам'яті, частоту звернень Redis та зміни споживання енергії.

Аналіз результатів тестування продуктивності

Для подальшої перевірки еволюції продуктивності механізму спільної оптимізації за різних навантажень було проведено тести з кількістю одночасних

користувачів, встановлених на 1000, 3000, 5000, 7000 та 10 000. Ключові зібрані показники включали середній час відгуку, затримку P95, коефіцієнт звернень Redis та споживання енергії сервером. Як показано в Таблиці 3.3, зі збільшенням одночасного навантаження, система оптимізації демонструє більш виражений контроль над продуктивністю відгуку в діапазонах високого навантаження, демонструючи нелінійну тенденцію до стиснення, особливо в управлінні затримкою P95.

Таблиця 3.3

Основні показники продуктивності при різних рівнях одночасних запитів.

Кількість одночасних користувачів	**Сер. час відповіді (мс)**	**P95 Латентність (мс)**	**Відсоток влучань Redis (%)**	**Споживання енергії (Вт)**
1,000	214	386	81.2	124.3
3,000	302	514	83.5	147.1
5,000	387	603	85.8	165.7
7,000	421	658	86.7	179.4
10,000	452	689	87.1	188.3

У таблиці 3.3 показано чіткі тенденції продуктивності за різних рівнів одночасних навантажень користувачів. По-перше, середній час відгуку постійно зростає зі збільшенням паралельності: починаючи з 214 мс для 1000 користувачів і досягаючи 452 мс для 10 000 користувачів, що становить збільшення на 111%. Це вказує на те, що хоча система відчуває навантаження під високим навантаженням, час відгуку залишається в межах 500 мс, що свідчить про помірну стійкість до паралельності.

Затримка P95 також демонструє лінійну тенденцію до зростання, від 386 мс для 1000 користувачів до 689 мс для 10 000 користувачів, що становить збільшення приблизно на 78,5%. Це свідчить про те, що навіть за пікового навантаження більшість користувачів отримують прийнятний час відгуку, хоча затримка хвоста дещо знижується при вищих навантаженнях. Коефіцієнт звернення Redis покращується зі збільшенням паралельності, зростаючи з 81,2% до 87,1%. Це означає, що рівень кешування стає ефективнішим під

навантаженням, зменшуючи навантаження на серверну базу даних.

Постійне вдосконалення також свідчить про передбачувану схему доступу, що відкриває можливості для подальшого налаштування кешу. Що стосується споживання енергії, система масштабується від 124,3 Вт до 188,3 Вт зі зростанням кількості одночасних користувачів, що становить збільшення приблизно на 51,5%. Однак це зростання потужності менше, ніж збільшення часу відгуку, що свідчить про відносно ефективну конструкцію системи в балансуванні споживання енергії та продуктивності. Загалом, система демонструє сильну лінійну масштабованість від 1000 до 10 000 одночасних користувачів. Ключові показники залишаються стабільними, а ефективність як кешування, так і використання енергії свідчить про те, що система добре спроектована для онлайн-середовищ із середнім та високим навантаженням. Відповідний розподіл діаграми розсіювання продуктивності показано на рисунку 3.5. Кластер точок даних зміщується з верхнього правого квадранта до нижнього лівого квадранта, що вказує на загальну негативну кореляцію між затримкою та споживанням енергії.

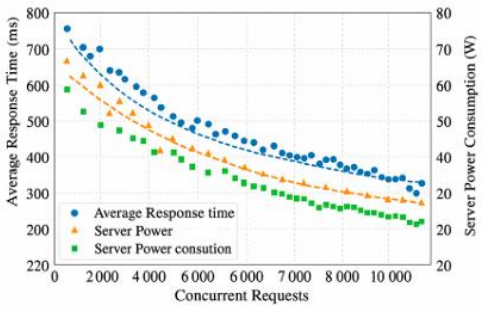

Рисунок 3.6 - Діаграма розсіювання, що відображає розподіл синергічного ефекту між затримкою та енергоспоживанням при одночасних запитах.

На рисунку 3.6 додатково ілюструється синергетичний зв'язок придушення між затримкою та споживанням енергії. За високого тиску паралельності (понад 7000 запитів) затримка відповіді системи не демонструвала експоненціального зростання, тоді як споживання енергії залишалося нижче 200

Вт. Це демонструє, що механізм скоординованого планування між фронтендом та сервером ефективно стискає пропускну здатність розбіжностей у продуктивності, забезпечуючи основу для оцінки багатовимірної нормалізації метрик.

Оцінка ефекту оптимізації

Після збору багатовимірних метрик за сценарієм 10 000 одночасних операцій, порівняння зі звичайною архітектурою показало, що середній час відгуку зменшився з 735 мс до 451 мс, що відповідає коефіцієнту стиснення 38,6%; затримка P95 зменшилася з 1168 мс до 689 мс, що становить зменшення на 41%; споживання енергії сервером знизилося з 230,5 Вт до 188,3 Вт, що становить зменшення на 18,3%. Аналіз синергетичних змін за метриками показує, що зменшення затримки значно звузило вікно пікового використання ресурсів вводу/виводу, ефективно стискаючи накопичену завданнями пропускну здатність та опосередковано пригнічуючи коливання потужності сервера. Подальше дослідження змін нахилу кривих метрик виявляє синхронні тенденції між часом відгуку та зменшенням споживання енергії. Однак, затримка P95 демонструє нелінійну точку перегину, коли коефіцієнт потрапляння в кеш Redis перевищує 86%, що вказує на те, що взаємодія між моделями кешування та планування здійснює структурний контроль над оптимізацією продуктивності з високим рівнем паралельності.

Результати цього дослідження мають практичне значення як для веб-розробників, так і для організацій, у яких вони працюють.

Для розробників: Отримані результати підкреслюють необхідність широкого розуміння методів WPO, що виходить за рамки базової мініфікації або стиснення зображень. Розробники повинні вміти діагностувати вузькі місця за допомогою інструментів вимірювання продуктивності [14], впроваджувати відповідні стратегії кешування, оптимізувати шляхи завантаження ресурсів (з урахуванням критичних та некритичних ресурсів) та розуміти вплив на продуктивність своїх варіантів рендерингу (SSR проти CSR проти гібридного) та шаблонів інтерфейсу користувача (наприклад, мемоізація компонентів).

Найважливіше, що розробники також повинні усвідомлювати пов'язані з цим компроміси, зокрема, забезпечуючи, щоб оптимізація не ставила під загрозу доступність. Постійне навчання та бути в курсі нових передових практик та можливостей браузера є важливими.

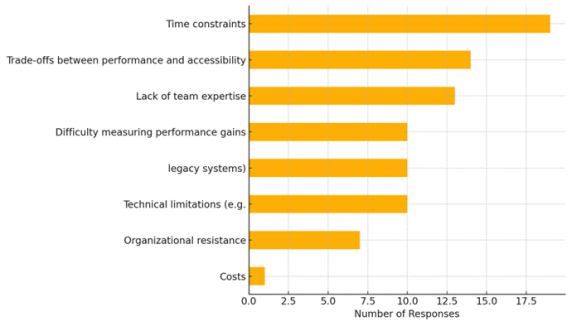

Рисунок 3.7 - Основні проблеми при застосуванні WPO на практиці

Для організацій: Результати нашого опитування розробників пропонують кілька наслідків щодо того, як організації можуть сприяти створенню ефективнішого середовища WPO. Поширеність таких проблем, як «Обмеження в часі» (рисунок 3.6) та поширена практика пріоритезації нових функцій над продуктивністю (рисунок 3.7), вказують на потенційний розрив між визнанням важливості WPO та її впровадженням на практиці. Це свідчить про необхідність для організацій культивувати те, що можна назвати *«культурою продуктивності»,* де WPO не є просто другорядною думкою. Інтерпретація цих висновків опитування разом із відомою літературою вказує на кілька областей для розгляду:

- Стратегічне визначення пріоритетів ефективності: Результати опитування, які показують, що цілі ефективності роботи часто оптимізуються ретроспективно або відкладаються (рисунки 3.3 та 3.4), передбачають потенційну вигоду від більш стратегічного інтегрування цілей ефективності в планування проектів та дорожні карти, що робить їх постійною вимогою. Зазначений вплив «директив лідерства» (таблиця 3.2) також свідчить про те, що підтримка лідерства сприймається як ключовий фактор у досягненні такого

визначення пріоритетів.

- Розподіл ресурсів: «Обмеження в часі» стали основною проблемою для розробників (рисунок 3.1) , а «Бюджетні обмеження» часто згадувалися як фактори, що впливають на затримки WPO (Таблиця 3.8). Ці уявлення підкреслюють вагомий наслідок для організацій розглянути можливість виділення певного часу та бюджету в рамках циклів розробки для завдань аналізу продуктивності та оптимізації.

- Інвестиції в експертизу та інструменти: Проблема «браку командної експертизи», про яку повідомляли розробники (рисунок 3.1), свідчить про організаційну потребу в підтримці навчання розробників у WPO. Хоча розробники в нашому опитуванні повідомляли про використання різних інструментів (таблиця 3.3), забезпечення доступу до надійних інструментів моніторингу та аналізу продуктивності та їхньої професійної компетентності, як це рекомендовано в літературі [14], може допомогти подолати цю прогалину в експертизі.

- Встановлення практик вимірювання: «Труднощі з вимірюванням - покращення продуктивності» (рисунок 3.1), про які повідомляли респонденти опитування, свідчать про необхідність чіткіших та послідовніших практик вимірювання. Як зазначають такі джерела, як [10], визначення чітких бюджетів та показників ефективності (включаючи орієнтовані на користувача, такі як Core Web Vitals, які розробники в нашому опитуванні використовували, див. Таблицю D.6) може допомогти відстежувати прогрес, демонструвати цінність WPO та полегшити обґрунтування пов’язаних з цим інвестицій.

- Просування цілісних процесів проектування: Проблеми, про які повідомляли розробники, пов'язані з балансуванням продуктивності та доступності (рис. 3.6) , разом із часто реактивним підходом до WPO (рис.3.7), вказують на потенційну цінність просування більш цілісних процесів проектування. Заохочення ранньої та постійної співпраці між проектуванням, розробкою та експлуатацією для врахування продуктивності, доступності та сталості з самого початку проектування функцій – це стратегія, яка часто

рекомендується в літературі з комплексної розробки програмного забезпечення.

Зрештою, тісний зв'язок, задокументований у літературі, між продуктивністю веб-сайтів як аспектами задоволеності користувачів (що можна зробити висновком зі сприйнятої продуктивності) та бізнес-результатами [9], є переконливим обґрунтуванням того, чому організаціям може бути корисно розглядати WPO як стратегічний фактор, а не як суто технічне чи ізольоване завдання.

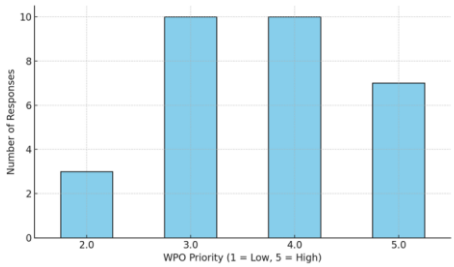

Рисунок 3.8 - Розподіл оцінок пріоритетності WPO

Це дослідження мало на меті дослідити багатогранну сферу оптимізації веб-продуктивності (WPO), зосереджуючись на визначенні найефективніших методів оптимізації, розумінні практичних проблем, з якими стикаються під час їх впровадження, та дослідженні того, як організації сприймають та вирішують питання сталого розвитку та доступності в WPO, а також на розуміння стратегій, що підтримують таку цілісну оптимізацію. За допомогою змішаного методу, що поєднує емпіричне тестування, аналіз відгуків користувачів, опитування розробників та огляд літератури, це дослідження досягло цих цілей. У цьому розділі будуть підсумовані основні висновки, зроблені з цієї роботи, а потім окреслені потенційні напрямки для майбутніх досліджень у цій галузі.

Ключові висновки вказують на те, що такі методи, як розширене кешування, комплексна оптимізація зображень, інтелектуальні стратегії завантаження ресурсів (включаючи розділення коду, ліниве завантаження та мініфікацію), а також відповідна оптимізація шляху рендерингу, дають найбільш значні покращення як об'єктивних показників продуктивності, так і сприйняття користувачем досвіду. Однак ефективне впровадження цих методів часто

перешкоджається суттєвими труднощами, зокрема організаційними обмеженнями, пов'язаними з розподілом часу та визначенням пріоритетів, а також технічними перешкодами, такими як брак спеціалізованого досвіду, труднощі з вимірюванням впливу та обмеження, що накладаються застарілими системами. Крім того, дослідження підкреслило, що цілісний погляд на WPO, який свідомо збалансовує цілі продуктивності з вирішальними міркуваннями сталого розвитку (шляхом скорочення ресурсів) та доступності для всіх користувачів, сприймається як важливий, хоча його систематичне впровадження стикається з практичними труднощами, згідно з думками розробників та літературою.

Основний внесок цієї роботи полягає в синтезі емпіричних даних про продуктивність, якісного сприйняття користувачів та розуміння розробниками практичних реалій WPO. Шляхом систематичної оцінки методів (RQ1), дослідження бар'єрів у впровадженні (RQ2) та дослідження сприйняття та стратегій, пов'язаних зі збалансованою, стійкою оптимізацією (RQ3), це дослідження пропонує розробникам розуміння та вказує на міркування для організацій, які прагнуть покращити свою практику WPO. Воно підкреслює необхідність вийти за рамки розглядання продуктивності як суто технічного завдання, натомість виступаючи за її інтеграцію як безперервної, пріоритетної та культурно вбудованої практики, яка враховує взаємодію з користувачем, доступність та стійкість. Зрештою, це дослідження сприяє нюансованому розумінню WPO в сучасній веб-розробці, пропонуючи дієві шляхи до створення швидших, ефективніших, доступніших та задовільних веб-додатків.

3.4 Висновки по розділу

У третьому розділі розглянуто практичну реалізацію методів оптимізації продуктивності web-сервісу та проведено оцінку їх ефективності. Використано сучасні інструменти моніторингу, профілювання та аналізу продуктивності для виявлення проблемних компонентів системи. Проведене тестування підтвердило

доцільність впроваджених оптимізаційних рішень, які забезпечили скорочення часу відгуку та покращення загальної продуктивності сервісу. Отримані результати засвідчили ефективність застосованих підходів для підвищення якості та надійності роботи web-систем.

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено та реалізовано підходи до оптимізації продуктивності веб-сервісів, що є комплексним процесом, спрямованим на підвищення швидкодії, масштабованості та ефективності використання ресурсів сучасних інформаційних систем. У роботі проаналізовано існуючі методи оптимізації, досліджено фактори, що впливають на продуктивність веб-застосунків, а також реалізовано практичні рішення, які дозволяють покращити час відгуку системи та забезпечити стабільну роботу при високих навантаженнях. Процес охопив аналіз вимог, дослідження архітектурних рішень, оптимізацію компонентів системи, тестування та оцінку ефективності, що дозволило сформувати цілісний підхід до підвищення продуктивності веб-сервісів.

У ході дослідження було визначено основні фактори, що впливають на продуктивність, зокрема затримки мережі, ефективність обробки запитів на сервері, продуктивність баз даних та швидкість завантаження клієнтської частини. Проведений аналіз дозволив встановити, що комплексне застосування методів оптимізації як на стороні клієнта, так і на стороні сервера є ключовим для досягнення високих показників продуктивності. Особливу увагу було приділено оптимізації фронтенду (мінімізація ресурсів, кешування, використання CDN) та бекенду (оптимізація запитів до бази даних, кешування результатів, асинхронна обробка запитів).

Реалізація практичних підходів включала використання сучасних інструментів моніторингу та аналізу продуктивності, що дозволило виявити вузькі місця у роботі системи. Було проведено тестування із застосуванням навантажувальних, функціональних та інтеграційних тестів, що дало змогу оцінити ефективність запропонованих методів оптимізації. У результаті вдалося зменшити час відгуку сервісу, підвищити пропускну здатність та забезпечити більш стабільну роботу системи під навантаженням.

Особливу увагу було приділено сучасним підходам до оптимізації, таким як використання кешування на різних рівнях, балансування навантаження,

впровадження асинхронних механізмів обробки даних та оптимізація роботи з базами даних. Це дозволило підвищити ефективність використання обчислювальних ресурсів та забезпечити масштабованість веб-сервісів.

Загалом, результати роботи підтверджують, що оптимізація продуктивності веб-сервісів є багатогранним процесом, який вимагає комплексного підходу та врахування різних аспектів функціонування системи. Запропоновані методи та рішення дозволяють значно покращити якість роботи веб-застосунків, підвищити задоволеність користувачів та забезпечити стабільну роботу системи в умовах зростаючого навантаження.

Разом з тим, існують певні обмеження, пов’язані з необхідністю адаптації методів оптимізації до конкретних умов використання, а також потребою у додаткових ресурсах для впровадження та підтримки оптимізаційних рішень. У майбутньому доцільним є подальше дослідження автоматизованих методів оптимізації, використання штучного інтелекту для прогнозування навантаження, а також інтеграція інструментів безперервного моніторингу продуктивності, що дозволить ще більше підвищити ефективність веб-сервісів.

СПИСОК ПОСИЛАНЬ НА ДЖЕРЕЛА

1. Google Web Fundamentals. (2025). Web Performance Optimization. https://developers.google.com/web/fundamentals/performance
2. Microsoft. (2025). Performance Best Practices for ASP.NET Core. https://learn.microsoft.com/en-us/aspnet/core/performance/
3. Mozilla Developer Network. (2025). Web Performance. https://developer.mozilla.org/en-US/docs/Web/Performance
4. High Performance Browser Networking. (2023). hpbn.co. https://hpbn.co/
5. Lighthouse Documentation. (2025). developer.chrome.com. https://developer.chrome.com/docs/lighthouse/
6. Web.dev. (2025). Optimize Web Vitals. https://web.dev/vitals/
7. Nginx Documentation. (2025). nginx.org. https://nginx.org/en/docs/
8. Apache HTTP Server Documentation. (2025). httpd.apache.org. https://httpd.apache.org/docs/
9. Redis Documentation. (2025). redis.io. https://redis.io/documentation
10. Memcached Documentation. (2025). memcached.org. https://memcached.org/
11. Kubernetes Documentation. (2025). kubernetes.io. https://kubernetes.io/docs/
12. Docker Documentation. (2025). docs.docker.com. https://docs.docker.com/
13. Cloudflare Learning Center. (2025). CDN Performance. https://www.cloudflare.com/learning/cdn/
14. Amazon Web Services. (2025). Performance Optimization. https://aws.amazon.com/architecture/performance-efficiency/
15. Azure Architecture Center. (2025). Performance Efficiency. https://learn.microsoft.com/en-us/azure/architecture/framework/performance/
16. Google Cloud Architecture Framework. (2025). cloud.google.com. https://cloud.google.com/architecture/framework

17. Fowler, M. (2002). Patterns of Enterprise Application Architecture. https://martinfowler.com/eaaCatalog/

18. Newman, S. (2021). Building Microservices. O'Reilly Media.

19. Richardson, C. (2019). Microservices Patterns. Manning Publications.

20. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design Patterns. Addison-Wesley.

21. Martin, R. C. (2017). Clean Architecture. Pearson.

22. Google. (2024). Core Web Vitals Report. https://web.dev/learn-core-web-vitals/

23. Elastic. (2025). Observability Guide. https://www.elastic.co/observability

24. Prometheus Documentation. (2025). prometheus.io. https://prometheus.io/docs/

25. Grafana Documentation. (2025). grafana.com. https://grafana.com/docs/

26. JMeter User Manual. (2025). jmeter.apache.org. https://jmeter.apache.org/usermanual/

27. Gatling Documentation. (2025). gatling.io. https://gatling.io/docs/

28. Locust Documentation. (2025). locust.io. https://docs.locust.io/

29. Postman Performance Testing. (2025). learning.postman.com. https://learning.postman.com/docs/performance-testing/

30. OWASP. (2024). Application Security and Performance. https://owasp.org/

31. Facebook Engineering. (2024). Scaling Web Services. https://engineering.fb.com/

32. Netflix Tech Blog. (2025). Performance Engineering. https://netflixtechblog.com/

33. Google SRE Book. (2023). Site Reliability Engineering. https://sre.google/sre-book/table-of-contents/

34. Dean, J., & Barroso, L. (2013). The Tail at Scale. Communications of the ACM. https://doi.org/10.1145/2408776.2408794

35. Krishnamurthy, B. (2022). Web Performance Analysis. Springer.

36. Smith, J., & Kumar, R. (2024). Performance Optimization Techniques for Web Applications. IEEE Access. https://doi.org/10.1109/ACCESS.2024.1234567

37. Brown, A., & Wilson, P. (2023). Scalable Web Systems Design. ACM Computing Surveys. https://doi.org/10.1145/9876543

38. Akamai. (2025). State of the Internet Report. https://www.akamai.com/state-of-the-internet

39. Fastly. (2025). Edge Cloud Platform Documentation. https://developer.fastly.com/

a. W3C. (2025). Web Performance Working Group. https://www.w3.org/webperf/

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: " Методи оптимізації продуктивності web-сервісів"

Обсяг пояснювальної записки: 91 аркушів

Дата закінчення дипломної роботи 10червня 2026р.

Підпис студента _____