НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра інформаційних технологій в телекомунікаціях

До захисту допущено:
ВО завідувача кафедри
_____ Валерій ПРАВИЛО
«_____»_____2026 р.

ДИПЛОМНА РОБОТА
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційно-комунікаційні технології»
спеціальності 172 Телекомунікації та радіотехніка

на тему: Метод поетапного покращення Core Web Vitals у web-сайтах зі збереженням функціональності та візуальної цілісності інтерфейсу

Виконав: здобувач вищої освіти 4 курсу, групи ТІ-22
Строков Єгор Русланович _____

Науковий керівник: доцент кафедри ІТТ НН ІТС
кандидат технічних наук, доцент
Суліма Світлана Валеріївна _____

Рецензент: доцент кафедри ТК НН ІТС, кандидат технічних наук, доцент
Міночкін Дмитро Анатолійович
_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____

Київ – 2026 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра інформаційних технологій в телекомунікаціях

Рівень вищої освіти – перший (бакалаврський)
Освітньо-професійна програма «Інформаційно-комунікаційні технології» спеціальності 172 Телекомунікації та радіотехніка

ЗАТВЕРДЖУЮ
ВО завідувача кафедри
_____ Валерій ПРАВИЛО
«_____»_____2026 р.

ЗАВДАННЯ
на дипломну роботу студенту
Строкову Єгору Руслановичу

1. Тема дипломної роботи: Метод поетапного покращення Core Web Vitals у web-сайтах зі збереженням функціональності та візуальної цілісності інтерфейсу, науковий керівник роботи кандидат технічних наук, доцент кафедри ІТТ НН ІТС Суліма Світлана Валеріївна - затверджені наказом по університету від 26.05.2026 р. №1912-с

2. Строк подання студентом дипломної роботи 06.06.2026 р.

3. Об’єкт дослідження: Процес оцінки та покращення продуктивності сучасних веб-сайтів за критеріями якості користувацького досвіду.

4. Вихідні дані до роботи: Офіційна технічна документація та стандарти щодо оцінювання вебпродуктивності (Core Web Vitals), існуючі продуктові та інженерні моделі пріоритезації технічних змін, інструментальні засоби синтетичного аудиту та моніторингу швидкодії, а також вихідний код клієнтської частини шаблонного web-сайту.

5. Перелік завдань, які потрібно розробити:

1. Провести аналіз теоретичних засад оцінки вебпродуктивності, дослідити механіку ключових метрик Core Web Vitals (LCP, CLS, INP) та сучасні інструменти їх вимірювання.
2. Дослідити існуючі продуктові та інженерні моделі пріоритезації технічних змін і обґрунтувати необхідність розробки спеціалізованого методу.
3. Розробити метод поетапного покращення Core Web Vitals на основі критеріїв користі, складності та ризику візуально-функціональної регресії (K-S-R).
4. Провести базове тестування продуктивності клієнтської частини обраного web-сайту.
5. Здійснити практичну реалізацію розробленого методу шляхом ітеративного впровадження оптимізаційних змін.
6. Виконати порівняльний аналіз ефективності впроваджених рішень, оцінити дієвість методу K-S-R порівняно з автоматизованими рекомендаціями та розробити практичні рекомендації щодо підтримки високих показників швидкодії.

6. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація - 14 слайдів, 5 таблиць, 25 рисунків

7. Дата видачі завдання: 16.02.2026 р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Узгодження організаційних питань з науковим керівником	16.02.26 - 18.02.26	Виконано
2	Узгодження теми роботи та початок роботи над першим розділом	19.02.26 - 22.02.26	Виконано
3	Робота над першим розділом	23.02.26 - 15.03.26	Виконано
4	Робота над другим розділом. Планування розробки власного рішення	16.03.26 - 05.04.26	Виконано
5	Розробка власного рішення та його перевірка	06.04.26 - 30.04.26	Виконано
6	Підготовка третього розділу	01.05.26 - 20.05.26	Виконано
7	Висновки про ефективність рішення	21.05.26 - 27.05.26	Виконано
8	Оформлення дипломної роботи	28.05.26 - 07.06.26	Виконано

9	Підготовка презентації до захисту	08.06.26 - 15.06.26	Виконано
10	Захист дипломної роботи	16.06.26 - 19.06.26	Виконано

Здобувач вищої освіти _____ Єгор СТРОКОВ

Науковий керівник роботи _____ Світлана Суліма

РЕФЕРАТ

Дипломна робота «Метод поетапного покращення Core Web Vitals у web-сайтах зі збереженням функціональності та візуальної цілісності інтерфейсу» містить 84 сторінки тексту, 25 рисунків та 5 таблиць, а також використовує 24 літературних джерела посилання.

Актуальність теми: Покращення показників Core Web Vitals є критично важливим аспектом веб-розробки, що безпосередньо визначає конкурентоспроможність web-сайтів, оскільки ці метрики виступають офіційним фактором ранжування Google. В умовах ускладнення клієнтської частини (через великі обсяги медіа, шрифти та блокуючі скрипти) інструменти автоматизованого аудиту, такі як Lighthouse, генерують десятки рекомендацій, проте не визначають раціонального порядку їх впровадження на практиці. Оскільки існуючі моделі пріоритезації слабо враховують специфіку вебпродуктивності та ігнорують ризики порушення інтерфейсу, особливого значення набуває розробка відтворюваного методу поетапного покращення, який ефективно поєднує зростання метрик із збереженням візуально-функціональної цілісності web-сайту.

Метою роботи є підвищення ефективності функціонування сайту шляхом розробки методу пріоритезації покращень показників Core Web Vitals, що забезпечує прискорення завантаження основного контенту та підвищення візуальної стабільності інтерфейсу при мінімізації ризиків візуально-функціональної регресії.

Об'єктом дослідження є процес оцінки та покращення продуктивності сучасних веб-сайтів за критеріями якості користувацького досвіду.

Предметом дослідження є метод поетапного покращення та впровадження технічних змін для покращення показників Core Web Vitals з урахуванням ризиків візуально-функціональної регресії.

Результатом роботи є розроблений метод K-S-R, який дає можливість системно підійти до покращення швидкодії web-сайтів, включаючи до оцінки ризик порушення візуально-функціональної цілісності інтерфейсу. Практична апробація показала, що кероване впровадження технічних змін сприяє позитивній динаміці ключових метрик. На відміну від неструктурованого виконання порад з автоматизованих аудитів, запропонована модель робить процес технічного покращення прогнозованим, знижує ймовірність критичних помилок та ефективніше розподіляє ресурси розробника.

Ключові слова: CORE WEB VITALS, МЕТОД ПРІОРИТЕЗАЦІЇ K-S-R, ПОКРАЩЕННЯ ПРОДУКТИВНОСТІ, WEB-САЙТ.

ABSTRACT

The bachelor's thesis contains 84 pages, 25 figures, 5 tables, and references 24 sources.

Relevance of the topic: Improving Core Web Vitals is a critically important aspect of web development that directly determines the competitiveness of websites, as these metrics serve as an official Google ranking factor. Given the increasing complexity of the client-side (due to large media volumes, fonts, and render-blocking scripts), automated audit tools like Lighthouse generate dozens of recommendations but fail to define a rational order for their practical implementation. Since existing prioritization models poorly account for the specifics of web performance and ignore the risks of interface disruption, the development of a reproducible step-by-step improvement method that effectively combines metric growth with the preservation of the website's visual and functional integrity becomes particularly significant.

The objective of this work is to increase the efficiency of website functioning by developing a method for prioritizing Core Web Vitals improvements, which ensures accelerated loading of main content and increased visual stability of the interface while minimizing the risks of visual and functional regression.

The object of the study is the process of evaluating and improving the performance of modern websites based on user experience quality criteria.

The subject of the study is a method of step-by-step improvement and implementation of technical changes to enhance Core Web Vitals (LCP, CLS, INP) while considering the risks of visual and functional regression.

The result of the work is the developed K-S-R method, which allows for a systematic approach to improving website performance, including the assessment of risks related to the disruption of the interface's visual and functional integrity. Practical testing has shown that the controlled implementation of technical changes promotes positive dynamics in key metrics. Unlike the unstructured execution of automated audit recommendations, the proposed model makes the technical

improvement process predictable, reduces the probability of critical errors, and allocates developer resources more effectively.

Keywords: CORE WEB VITALS, K-S-R PRIORITIZATION METHOD, PERFORMANCE IMPROVEMENT, WEBSITE.

ЗМІСТ

СПИСОК ТЕРМІНІВ І СКОРОЧЕНЬ

API	Application Programming Interface (Інтерфейс прикладного програмування)
AVIF	AV1 Image File Format (Формат стиснення зображень AV1)
CDN	Content Delivery Network (Мережа доставки контенту)
CI	Continuous Integration (Безперервна інтеграція коду)
CLS	Cumulative Layout Shift (Сукупний зсув макета сторінки)
CPU	Central Processing Unit (Центральний процесор)
CrUX	Chrome User Experience Report (Звіт про користувацький досвід Chrome)
CSR	Client-Side Rendering (Відмальовування на боці клієнта / Клієнтський рендеринг)
CSS	Cascading Style Sheets (Каскадні таблиці стилів)
CSSOM	CSS Object Model (Об’єктна модель каскадних таблиць стилів)
CWV	Core Web Vitals (Ключові показники вебпродуктивності)
DOM	Document Object Model (Об’єктна модель документа)
FCP	First Contentful Paint (Час першого відмальовування контенту)
FID	First Input Delay (Затримка першої взаємодії користувача)
GIF	Graphics Interchange Format (Формат обміну графічними даними)
GPU	Graphics Processing Unit (Графічний процесор)
GSC	Google Search Console (Сервіс моніторингу вебсайтів від Google)
HTML	HyperText Markup Language (Мова гіпертекстової розмітки)
HTTP	HyperText Transfer Protocol (Протокол передавання гіпертексту)
ICE	Impact, Confidence, Ease (Вплив, Впевненість, Простота — продуктова модель пріоритезації)
INP	Interaction to Next Paint (Затримка відгуку інтерфейсу на

	взаємодію)
JS	JavaScript (Мова сценаріїв вебсторінок)
K-S-R	Користь, Складність, Ризик (розроблений інженерний метод пріоритезації технічних змін)
LCP	Largest Contentful Paint (Час відображення найбільшого елемента контенту)
PIE	Potential, Importance, Ease (Потенціал, Важливість, Простота — інженерний метод пріоритезації)
PNG	Portable Network Graphics (Портативна мережева графіка)
PSI	PageSpeed Insights (Сервіс аналізу вебпродуктивності сторінок)
RICE	Reach, Impact, Confidence, Effort (Охоплення, Вплив, Впевненість, Трудомісткість — продуктова модель пріоритезації)
RUM	Real User Monitoring (Моніторинг досвіду реальних користувачів)
SEO	Search Engine Optimization (Пошукова оптимізація веб-ресурсів)
SPA	Single Page Application (Односторінковий вебзастосунок)
SVG	Scalable Vector Graphics (Масштабована векторна графіка)
TBT	Total Blocking Time (Загальний час блокування основного потоку браузера)
TTFB	Time to First Byte (Час очікування першого байта відповіді від сервера)
UI	User Interface (Інтерфейс користувача)
URL	Uniform Resource Locator (Уніфікований покажчик ресурсу)
UX	User Experience (Користувацький досвід взаємодії)
W3C	World Wide Web Consortium (Консорціум Всесвітньої павутини)
WebP	Web Picture Format (Сучасний формат стиснення вебзображень)
WSJF	Weighted Shortest Job First (Пріоритет найкоротшої задачі з найбільшою цінністю — продуктова модель пріоритезації)

ВСТУП

У сучасних умовах швидкого розвитку цифрових технологій web-сайти стали основним інструментом комунікації, надання послуг та бізнесу. Зі зростанням конкуренції в інтернет-середовищі вимоги користувачів до якості веб-ресурсів постійно підвищуються. Сучасний користувач очікує миттєвого завантаження сторінок, плавної анімації та швидкого відгуку на свої дії. Забезпечення високого рівня досвіду користувача (UX) перетворилося з бажаної характеристики на критичну необхідність успішного функціонування будь-якого web-сайту.

Особливої актуальності це питання набуло після того, як компанія Google офіційно визнала набір орієнтованих на користувача метрик Core Web Vitals фактором ранжування в пошуковій системі. Це зробило технічне покращення продуктивності суворою бізнес-вимогою. Проте архітектура сучасних web-сайтів стає дедалі складнішою: використання великих обсягів медіаданих, складних веб-шрифтів, масивних UI-бібліотек та рендер-блокуючих скриптів типово призводить до погіршення швидкодії та нестабільності інтерфейсу.

На практиці розробники стикаються із проблемою: інструменти автоматизованого аудиту, такі як Google Lighthouse, генерують десятки технічних рекомендацій, але не визначають раціональний порядок їх впровадження. Існуючі продуктові та інженерні моделі пріоритезації виявляються недостатньо ефективними, оскільки вони слабо враховують специфіку веб-продуктивності та ігнорують ризики візуально-функціональної регресії – ситуації, коли технічна зміна покращує метрику, але руйнує верстку чи логіку роботи клієнтської частини.

Одним з перспективних способів вирішення цієї проблеми є розробка спеціалізованого методу поетапного покращення. Такий підхід дозволить формалізувати процес вибору та впровадження технічних змін, збалансувавши очікувану користь для метрик Core Web Vitals, складність реалізації та ризики порушення цілісності сайту.

Мета роботи – підвищити ефективність функціонування сайту шляхом розробки методу пріоритезації покращень показників Core Web Vitals, що забезпечує прискорення завантаження основного контенту та підвищення візуальної стабільності інтерфейсу при мінімізації ризиків візуально-функціональної регресії.

Об'єктом дослідження є процес оцінки та покращення продуктивності сучасних веб-сайтів за критеріями якості користувацького досвіду.

Предметом дослідження є метод поетапного покращення та впровадження технічних змін для покращення показників Core Web Vitals з урахуванням ризиків візуально-функціональної регресії.

Науковою новизною є метод поетапного покращення Core Web Vitals на web-сайтах, що базується на критеріях K, S і R, враховує ризик візуально-функціональної регресії та задає порядок впровадження технічних змін

Результатом роботи є розроблений метод K-S-R, який дає можливість системно підійти до покращення швидкодії web-сайтів, включаючи до оцінки ризик порушення візуально-функціональної цілісності інтерфейсу. Практична апробація показала, що кероване впровадження технічних змін сприяє позитивній динаміці ключових метрик. На відміну від неструктурованого виконання порад з автоматизованих аудитів, запропонована модель робить процес технічного покращення прогнозованим, знижує ймовірність критичних помилок та ефективніше розподіляє ресурси розробника.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ЗАСАДИ ОЦІНКИ ПРОДУКТИВНОСТІ WEB-САЙТІВ

1.1 Еволюція підходів до оцінки швидкодії та поява метрик Core Web Vitals

У сучасній web-розробці забезпечення високої продуктивності web-сайтів є одним із найпріоритетніших завдань. Однак ця величина не є абсолютною або статичною, оскільки фактична швидкість роботи ресурсу істотно залежить від апаратних можливостей кінцевого пристрою та стабільності з'єднання користувача. Більше того, суб'єктивне сприйняття швидкості часто не збігається з об'єктивним часом завантаження. Завдяки застосуванню методів інкрементного відображення контенту браузер малює елементи поступово замість того щоб утримувати порожній екран. У той самий час виникає проблема "оманливої швидкості": сторінка візуально завантажується швидко, створюючи ілюзію готовності, але за спроби взаємодії (кліка чи скролінгу) інтерфейс залишається заблокованим через перевантаженості фоновими обчисленнями [1].

Враховуючи цю багатогранність, для якісної оцінки та оптимізації необхідні точні об'єктивні метрики, що відображають реальний стан речей. Історично склалося так, що протягом тривалого часу головним критерієм вимірювання продуктивності у веб-середовищі була технічна подія життєвого циклу сторінки – window.onload (або просто load). Ця подія фіксує чітко визначений момент, коли браузер успішно завантажив вихідний HTML-документ, а також усі залежні ресурси (таблиці стилів, зображення, базові скрипти). Незважаючи на технічну однозначність, головна вада цієї події полягає в тому, що момент настання load зовсім не корелює з тим, що дійсно важливо для користувача під час реальної взаємодії з веб-ресурсом [1].

Сучасна архітектура web-сайтів (зокрема Single Page Applications) часто функціонує таким чином, що сервер миттєво повертає мінімальний HTML-каркас (App Shell). У такому випадку подія load генерується майже відразу,

що формально свідчить про "блискавичну" швидкість завантаження. Однак після цього браузер розпочинає тривалий процес виконання JavaScript-коду, асинхронного запиту даних через API та подальшого клієнтського рендерингу (Client-Side Rendering). Протягом усіх цих додаткових секунд після події load користувач продовжує дивитися на порожній екран або індикатор завантаження. Таким чином, суто технічна метрика виявилася відірваною від реального користувальницького досвіду [1].

Усвідомлення цієї проблеми призвело до зміни парадигми. В останні роки інженери команди Google Chrome у тісній співпраці з Робочою групою W3C з веб-продуктивності (Web Performance Working Group) ініціювали глобальний процес стандартизації нового набору API та показників. Головним вектором цього розвитку став перехід до орієнтованих на користувача метрик продуктивності (User-centric performance metrics), які здатні максимально точно вимірювати і оцифровувати суб'єктивне сприйняття веб-сторінки живою людиною [2].

Важливим етапом цієї еволюції став травень 2020 року, коли компанія Google офіційно представила ініціативу Web Vitals, виокремивши з великої кількості показників єдиний стандарт — Core Web Vitals [3]. Вагомість цього кроку підтвердилася влітку 2021 року з виходом оновлення алгоритмів Google Page Experience, коли показники Core Web Vitals стали офіційним фактором ранжування в пошуковій системі. Це перетворило оптимізацію продуктивності з бажаної інженерної практики на критичну бізнес-вимогу [4]. Еволюція підходів до оцінки триває й досі: зважаючи на ускладнення клієнтської логіки сучасних web-сайтів, у березні 2024 року застарілу метрику інтерактивності FID (First Input Delay) було офіційно замінено на більш комплексну INP (Interaction to Next Paint) [5].

Щоб розробити релевантну систему оцінювання, дослідники звернулися до психології сприйняття інтерфейсів і виділили чотири ключові питання, на які підсвідомо шукає відповідь користувач:

Чи відбувається процес?— оцінює базовий рівень: чи розпочалася навігація та чи успішно відповів сервер на запит клієнта.

Чи це корисно? — фіксує момент появи достатнього обсягу візуалізованого контенту, щоб відвідувач міг зрозуміти контекст сторінки та почати споживати інформацію.

Чи можна цим користуватися? — визначає технічну готовність до взаємодії (скролінг, кліки, введення даних), перевіряючи, чи не заблокований головний потік браузера виконанням важких скриптів.

Чи це комфортно? — вимірює візуальну стабільність, відсутність неочікуваних зсувів макета та загальну плавність анімацій без пропуску кадрів [1].

Ця фундаментальна класифікація стала концептуальною базою для відмови від застарілих технічних індикаторів на користь сучасних метрик продуктивності.

Для забезпечення комплексного та об'єктивного аналізу продуктивності web-сайтів застосовується подвійний підхід до збору метрик. Вимірювання у лабораторних умовах (Lab Data) у контрольованому середовищі є критично важливим на етапах розробки для виявлення архітектурних недоліків до моменту релізу. Однак синтетичне тестування не здатне повною мірою врахувати гетерогенність пристроїв, якість мережевого з'єднання та динамічність контенту. Тому для оцінки швидкодії в умовах реальної експлуатації застосовується «Моніторинг реальних користувачів» (RUM). Тільки зібрана таким чином польова статистика (Field Data) враховує весь спектр зовнішніх факторів і дозволяє покращити метрики, орієнтовані на користувача. Оскільки кожен підхід вирішує свої специфічні завдання, жоден із цих методів не є самодостатнім, тому для надійного моніторингу необхідна їхня синергія [1].

Зважаючи на багатовимірність користувацького досвіду, оцінка швидкодії не може обмежуватися єдиним показником і базується на кількох фундаментальних категоріях:

Сприймана швидкість завантаження: визначає, як швидко веб-сторінка відображає ключові візуальні елементи у видимій області екрана.

Швидкість реагування під час завантаження: показує здатність інтерфейсу оперативно реагувати на перші дії користувача, поки фонові ініціалізаційні скрипти ще виконуються.

Чуйність під час виконання: оцінює швидкість реакції на введення даних, кліки та взаємодію після повного завантаження сторінки, що є особливо критичним для складних односторінкових застосунків.

Візуальна стабільність: фіксує неочікувані зсуви елементів макета під час асинхронного завантаження ресурсів, які можуть призвести до помилкових дій користувача.

Плавність: характеризує стабільну частоту кадрів та відсутність ривків під час відтворення переходів та анімацій [1].

Очевидно, що жодна ізольована метрика не здатна комплексно відобразити швидкодію сторінки. Тому в інженерній практиці застосовується стандартизований набір показників, що охоплюють швидкість завантаження (FCP, LCP, TTFB), візуальну стабільність (CLS) та чуйність системи (TBT, INP). Цей перелік продовжує адаптуватися до нових архітектурних патернів, інколи доповнюючись власними показниками. Детальний технічний аналіз, алгоритми обчислення та характеристика ключової тріади Core Web Vitals (LCP, CLS та INP) будуть наведені у наступному підрозділі роботи.

1.2 Характеристика ключових показників продуктивності (LCP, CLS, INP)

Для комплексного розуміння та подальшого покращення продуктивності веб-сайтів важливо проаналізувати механіку роботи кожної метрики, що входять до стандарту Core Web Vitals: Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS) та Interaction to Next Paint (INP).

1.2.1 Largest Contentful Paint (LCP)

Найбільше відображення контенту (LCP) – це важлива та стабільна метрика Core Web Vitals, призначена для вимірювання сприйнятої швидкості завантаження сторінки. Ця метрика фіксує момент на часовій шкалі завантаження, коли з високим ступенем ймовірності основний вміст веб-ресурсу вже успішно відображено на екрані. Швидке досягнення LCP є критично важливим, оскільки це візуально підтверджує користувачеві, що сторінка корисна та функціональна.

На основі досліджень було встановлено, що найточнішим способом вимірювання часу завантаження основного контенту є фіксація моменту відображення найбільшого елемента. Згідно з цим принципом, метрика LCP визначає час рендерингу (візуалізації) найбільшого зображення, текстового блоку або відео, що знаходиться у видимій області екрана, відносно моменту, коли користувач вперше ініціював перехід на цю веб-сторінку [6].

Загальний час досягнення LCP розбивається на чотири компоненти, кожний з яких відображає окремий етап мережевої взаємодії та обробки даних браузером, що описується формулою (1.1):

$$\text{LCP} = \text{TTFB} + \text{Resource Load Delay} + \text{Resource Load Duration} + \text{Element Render Delay} \quad (1.1)$$

TTFB – проміжок часу від моменту ініціалізації завантаження сторінки користувачем до моменту отримання браузером першого байта відповіді HTML-документа.

Resource load delay – часовий інтервал між завершенням TTFB та початком завантаження ресурсу LCP браузером. Якщо елемент LCP не потребує мережевого завантаження, це значення дорівнює нулю.

Resource load duration – час, необхідний для завантаження самого ресурсу LCP. Якщо елемент не потребує завантаження ресурсів для рендерингу, цей час дорівнює нулю.

Element render delay – час між завершенням завантаження ресурсу LCP та повною відмальовкою елемента LCP на екрані.

Значення LCP для кожної сторінки складається виключно з цих чотирьох підкатегорій. Вони є послідовними, не перетинаються між собою, не мають часових прогалин і в сукупності формують повний час досягнення Largest Contentful Paint [7], що наочно продемонстровано на рисунку 1.1.

Рисунок 1.1 - Каскадна діаграма часових складових метрики LCP [7]

Для забезпечення високоякісного користувацького досвіду веб-ресурси повинні прагнути до того, щоб час відображення найбільшого елемента контенту (LCP) становив 2,5 секунди або менше. Згідно з встановленими критеріями оцінки, цей показник класифікується за трьома пороговими значеннями: показник до 2,5 секунди вважається добрим, діапазон від 2,5 до 4,0 секунд вважається таким, що потребує покращення, а час, що перевищує 4,0 секунди, вважається поганим (рис.1.2).

Для об'єктивного вимірювання та забезпечення досягнення оптимальних показників для більшості аудиторії загальноприйнятим стандартом є оцінка, що базується на 75-му процентилі всіх завантажень сторінок. У цьому випадку аналіз даних необхідно сегментувати окремо для мобільних та настільних пристроїв [6].

Рисунок 1.2 - Граничні значення метрики Largest Contentful Paint (LCP) [6]

Для визначення найбільшого елемента аналізується чітко обмежений набір типів контенту. До цього переліку належать:

- елементи <img> (для анімованого контенту, такого як GIF або анімований PNG, використовується час відтворення першого кадру);
- елементи <image> всередині елемента <svg>;
- елементи <video> (використовується час завантаження зображення-постера або час появи першого кадру відео – залежно від того, що настане раніше);

- елементи з фоновим зображенням, завантаженим за допомогою функції url();
- блокові елементи, які містять текстові вузли або інші дочірні текстові елементи рядкового типу.

Обмеження переліку саме цими базовими типами було впроваджено навмисно для зниження обчислювальної складності процесу вимірювання.

Окрім аналізу конкретних тегів, вимірювання LCP використовує спеціальні евристичні алгоритми для виключення певних елементів, які користувачі можуть вважати такими, що не містять корисного контенту. Для браузерів на базі Chromium до них належать:

- елементи з непрозорістю 0, невидимі для користувача;
- елементи, що охоплюють усю область перегляду, які, найімовірніше, вважаються фоном, а не вмістом;
- зображення-заглушки або інші зображення з низькою ентропією, які, ймовірно, не відображають справжнього змісту сторінки.

Ці евристичні правила роблять метрику LCP значно вибірковою. Якщо показник First Contentful Paint (FCP) може враховувати появу будь-якої візуальної інформації на екрані, навіть зображень-заглушок чи повноекранного фону, то LCP вимірює, коли на екрані відображається саме основний контент. Саме тому критерії відбору кандидатів для цієї метрики є набагато суворішими [6].

Розмір елемента для LCP визначається лише його видимою частиною в межах області перегляду. Якщо елемент виходить за межі екрана, обрізаний або має невидиме переповнення, ці частини не враховуються при розрахунку площі.

Для зображень використовується або видимий розмір на сторінці, або внутрішній розмір самого файлу — залежно від того, яке значення є меншим. Для текстових елементів враховується лише найменший прямокутник, що

охоплює всі текстові вузли, при цьому кожен вузол закріплюється за найближчим блоковим елементом-предком. Характерною особливістю є те, що при розрахунку розміру будь-якого елемента метрика ігнорує зовнішні та внутрішні поля, а також рамки, встановлені за допомогою CSS [6].

Оскільки веб-сторінки завантажуються поетапно, найбільший елемент у видимій області може змінюватися. Щоразу, коли браузер виявляє новий, більший за попередній об'єкт (наприклад, велике зображення після початкового відображення заголовка), він генерує новий запис PerformanceEntry. Цей процес триває до моменту першої взаємодії користувача зі сторінкою (клік, прокрутка або натискання клавіші), після чого фіксація нових кандидатів LCP припиняється. Для аналізу враховується лише останній згенерований запис, оскільки він ідентифікує фінальний та найбільший елемент контенту.

З міркувань безпеки браузери мають певні обмеження при роботі з ресурсами з інших джерел (cross-origin). Якщо сервер не надає заголовок Timing-Allow-Origin, точний час рендерингу зображень тривалий час був недоступним для веб-API — замість нього відображався лише час завантаження ресурсу. Хоча у сучасних версіях браузерів (зокрема Chrome 133 та пізніших) цей захист був пом’якшений для підвищення точності вимірювань, встановлення заголовка Timing-Allow-Origin залишається рекомендованим стандартом для отримання максимально коректних аналітичних даних [6].

1.2.2 Cumulative Layout Shift (CLS)

Cumulative Layout Shift (CLS) – це важливий, орієнтований на користувача стабільний показник стандарту Core Web Vitals, який використовується для вимірювання візуальної стабільності сторінки. Він допомагає кількісно визначити, як часто користувачі стикаються з неочікуваними змінами макета. Низький рівень CLS гарантує комфортну взаємодію з веб-ресурсом [8].

Неочікувані зсуви макета виникають щоразу, коли видимий елемент змінює своє положення від одного відмальованого кадру (візуалізованого) до іншого.

Це може серйозно порушити роботу користувача різними способами: від простої втрати місця під час читання тексту через його раптове переміщення, до випадкового натискання на неправильне посилання чи кнопку. У деяких випадках такі зсуви інтерфейсу (наприклад, під час підтвердження замовлення) можуть завдати серйозної шкоди користувацькому досвіду.

Раптове переміщення вмісту сторінки зазвичай відбувається, коли ресурси завантажуються асинхронно або елементи об'єктної моделі документа (DOM) динамічно додаються на сторінку перед уже існуючим контентом. Згідно з офіційною документацією, основними причинами змін макета є:

- зображення або відео з невідомими розмірами.
- веб-шрифти, які після рендерингу відображаються більшими або меншими за вихідний резервний варіант.
- сторонні рекламні оголошення чи віджети, що динамічно змінюють свій розмір [8].

Проблема візуальної нестабільності часто ускладнюється через різницю між тим, як сайт функціонує в процесі розробки, і тим, як його сприймають реальні користувачі. Наприклад, персоналізований або сторонній контент часто поводить себе по-різному в середовищах розробки та виробництва. Крім того, тестові зображення у розробника зазвичай вже знаходяться в кеші браузера, тоді як їх завантаження у кінцевого користувача займає набагато більше часу. Локальні виклики API також виконуються настільки швидко, що непомітні затримки під час розробки стають суттєвими в реальних умовах використання. Саме метрика CLS допомагає вирішити цю проблему, вимірюючи частоту її виникнення безпосередньо у реальних користувачів [8].

Для розрахунку CLS визначається показник найбільшої кількості змін макета для кожного неочікуваного зсуву, що відбувається протягом усього життєвого циклу сторінки. Сучасний алгоритм обчислення базується на концепції «вікна сеансу» (session window).

Вікно сеансу – це серія змін макета, за якої один або кілька окремих зсувів відбуваються у швидкій послідовності. Для формування такого вікна інтервал між кожним індивідуальним зсувом має становити менше ніж 1 секунду, а загальна максимальна тривалість усього вікна обмежена 5 секундами.

Підсумковий бал CLS сторінки – це вікно сеансу з максимальним сукупним балом усіх зсувів макета, що відбулися саме в цьому вікні. Слід звернути увагу, що раніше метрика CLS вимірювала загальну суму взагалі всіх індивідуальних зсувів макета, що сталися протягом усього терміну життя сторінки, проте алгоритм було оновлено для забезпечення більш об'єктивного порівняння [8].

Для забезпечення якісного користувацького досвіду показник CLS повинен становити 0,1 або менше. Для об'єктивної оцінки та гарантії оптимальної роботи для більшості аудиторії стандартом є вимірювання за 75-м перцентилем завантажень сторінок із сегментацією на мобільні та десктопні пристрої. Відповідно до офіційних рекомендацій, цю метрику класифікують за трьома пороговими значеннями: показник 0,1 або менше вважається хорошим, діапазон від 0,1 до 0,25 — таким, що потребує покращення, а значення, більше за 0,25, розцінюється як погане (рис. 1.3).

Рисунок 1.3 - Граничні значення оцінювання метрики Cumulative Layout Shift (CLS) [8]

З технічної точки зору, зсуви макета фіксуються за допомогою API нестабільності макета (Layout Instability API). Він реєструє подію щоразу, коли видимий елемент змінює свою початкову позицію між двома кадрами рендерингу. Щоб обчислити показник зсуву, браузер використовує добуток

двох значень: частки впливу (impact fraction) та частки відстані (distance fraction).

Частка впливу визначає, яку загальну площу екрана об'єднано займає нестабільний елемент у двох послідовних кадрах (до та після свого зміщення). Частка відстані вимірює максимальну відстань, на яку перемістився цей елемент по горизонталі чи вертикалі, поділену на найбільший вимір області перегляду (її ширину або висоту). Відповідно, бал обчислюється за базовою формулою (1.2):

$$\text{layout shift score} = \text{impact fraction} * \text{distance fraction} \quad (1.2)$$

layout shift score – бал зсуву макета.

impact fraction – частка впливу.

distance fraction – частка відстані. [8]

Водночас алгоритм враховує, що не всі зміни інтерфейсу є негативними. Зсуви, які є очікуваною реакцією на ініціативу користувача (наприклад, клік по посиланню, натискання кнопки або введення тексту), вважаються допустимими. Якщо зміна макета відбувається протягом 500 мілісекунд після взаємодії з користувачем, системі присвоюється внутрішній прапорець hadRecentInput, і такий зсув виключається з обчислення загального балу CLS [8].

Для створення динамічних інтерфейсів без шкоди для продуктивності розробникам рекомендується оптимізувати анімації та переходи. Замість зміни базових геометричних властивостей елементів (таких як width, height, top або left), які провокують зсув макета, ефективною практикою є використання CSS-властивості transform (зокрема функцій scale() та translate()), що дозволяє анімувати об'єкти без перерахунку структури документа [8].

1.2.3 Interaction to Next Paint (INP)

Interaction to Next Paint (INP) – це стабільна метрика Core Web Vitals, призначена для оцінки загальної швидкості відгуку вебсторінки на дії

користувача протягом усього часу її використання. На відміну від показників, зосереджених лише на етапі завантаження (як-от FID), INP аналізує затримку всіх взаємодій: натискань кнопки миші, дотиків до сенсорного екрана та використання клавіатури. Підсумковим значенням метрики є максимальна зафіксована тривалість взаємодії за вирахуванням статистичних викидів [9].

Необхідність впровадження INP зумовлена тим, що близько 90% часу користувач проводить на сторінці вже після її початкового завантаження. Висока чуйність означає, що у відповідь на дію користувача браузер максимально швидко відображає візуальний зворотний зв'язок у наступному кадрі. Це підтверджує користувачеві, що його дія (наприклад, відкриття меню, додавання товару до кошика або обробка форми) була успішно розпізнана системою.

З технічного погляду, затримка INP складається з трьох фаз:

- затримка введення – інтервал до початку виконання обробників подій, спричинений блокуванням основного потоку тривалими завданнями;
- тривалість обробки – час виконання всіх зворотних викликів обробників подій у межах поточного кадру;
- затримка представлення – час від завершення обробки подій до фактичного виведення оновленого кадру на екран.

Основним драйвером інтерактивності виступає JavaScript, проте браузери також обробляють елементи керування на основі CSS. До цільових типів взаємодії належать натискання кнопки миші, дотики до сенсорних екранів та клавіатурне введення. Прокручування або наведення курсору не враховуються безпосередньо, але можуть ініціювати жести, що підлягають реєстрації. Якщо взаємодія складається з кількох подій, у загальну затримку вносить вклад подія з найбільшою тривалістю.

Для оцінки якості відгуку встановлено порогові значення на основі 75-го процентиля сесій:

– добра чуйність – значення менше або дорівнює 200 мс.

– потребує покращення – значення в діапазоні від 200 до 500 мс.

– низька чуйність – значення понад 500 мс (рис. 1.4).

Низький показник INP свідчить про стабільну реакцію інтерфейсу. Відсутність даних за цією метрикою може бути наслідком відсутності активних взаємодій під час сесії або доступу автоматизованих систем (ботів). Найбільш точну оцінку забезпечує збір польових даних (RUM) від реальних користувачів [9].

Рисунок 1.4 - Граничні значення оцінювання метрики Interaction to Next Paint (INP)

1.3 Огляд сучасних інструментів для вимірювання та моніторингу веб-продуктивності

Теоретичне розуміння метрик продуктивності є лише початковим етапом у процесі оптимізації веб-сайтів. Для переходу до практичної реалізації завдань із покращення швидкодії необхідно використовувати спеціалізовані програмні рішення, здатні фіксувати та аналізувати ці показники. Загалом, усі сучасні інструменти та методики оцінювання базуються на двох основних підходах: лабораторних вимірюваннях (синтетичне тестування у контрольованому середовищі) та зборі польових даних (моніторинг реального користувацького досвіду). Такий поділ дозволяє розробникам не лише проводити точковий аудит сторінок на етапі їх створення, але й

здійснювати безперервний контроль стану продуктивності під час фактичної взаємодії з користувачами. У цьому підрозділі зроблено загальний огляд найбільш актуальних інструментів для вимірювання вебпродуктивності, що є необхідним кроком для вибору оптимальних засобів для подальшого дослідження.

1.3.1 Інструменти для лабораторного тестування (Lab Data)

Google Lighthouse – це автоматизований інструмент із відкритим вихідним кодом, призначений для комплексного підвищення якості вебсторінок. Він дозволяє проводити аудит продуктивності, доступності (accessibility), пошукової оптимізації (SEO) та інших параметрів як на загальнодоступних ресурсах, так і на сторінках, що вимагають автентифікації. Запуск інструменту може здійснюватися кількома способами: безпосередньо через панель Chrome DevTools, за допомогою командного рядка або у вигляді модуля Node.js. За заданою URL-адресою Lighthouse виконує серію перевірок і генерує розгорнутий звіт про поточний стан сторінки. Невдалі результати аудиту виступають індикаторами для подальшого покращення, при цьому кожна виявлена проблема супроводжується посиланням із поясненням її важливості та інструкціями щодо усунення. Крім базового інструментарію, передбачена можливість використання Lighthouse CI для забезпечення безперервного контролю та запобігання регресіям продуктивності вебсайтів у процесі їх подальшої розробки [10].

Загальний показник продуктивності в інструменті Google Lighthouse формується виключно на основі значень ключових метрик і розраховується як їхнє середньозважене значення. Вагові коефіцієнти кожної метрики (наприклад, LCP, CLS, TBT) розподіляються залежно від їхнього фактичного впливу на користувацьке сприйняття швидкодії. Алгоритм оцінювання перетворює абсолютні значення (у мілісекундах або секундах) на уніфіковану шкалу від 0 до 100 балів. Ця конвертація базується на логарифмічно нормальному розподілі, сформованому на основі даних реальної продуктивності вебсайтів з бази HTTP Archive. Для забезпечення

максимальної об'єктивності Lighthouse використовує окремі алгоритми та криві оцінювання для мобільних і десктопних пристроїв. Підсумковий бал має стандартизоване кольорове маркування для швидкої інтерпретації: від 0 до 49 (червоний колір) – низька продуктивність; від 50 до 89 (помаранчевий) – потребує покращення; від 90 до 100 (зелений) – висока швидкодія. Оскільки досягнення абсолютного максимуму в 100 балів вимагає експоненційних зусиль, цільовим орієнтиром вважається потрапляння сайту до "зеленої" зони (90+ балів). Для подальшого покращення результатів розробники використовують згенеровані звітом розділи "Можливості" та "Діагностика", які містять детальні рекомендації з оптимізації. При цьому під час проведення лабораторних аудитів варто враховувати, що підсумковий бал може мати незначні коливання через зовнішні фактори, такі як маршрутизація трафіку, апаратна потужність пристрою чи втручання розширень браузера [11].

Оскільки інструменти лабораторного тестування не передбачають реальної взаємодії користувача з інтерфейсом, безпосереднє вимірювання метрики INP у таких умовах є технічно ускладненим. Як альтернатива, у лабораторному середовищі використовується метрика Total Blocking Time (TBT), що слугує непрямим індикатором (проксі-показником) потенційних проблем з INP. Необхідно підкреслити, що TBT не є повноцінною заміною INP, оскільки може фіксувати затримки, з якими користувачі фактично не стикаються за відсутності взаємодії в конкретний момент часу, або ж навпаки – пропускати специфічні проблеми, що виникають лише під час реальних сценаріїв використання. З огляду на це, TBT розглядається як доцільний орієнтир для оптимізації на етапі розробки, тоді як об'єктивна оцінка фактичної швидкості відгуку вимагає вимірювання INP виключно на основі польових даних [12].

На відміну від автоматизованих інструментів аудиту, таких як Lighthouse, які надають загальну оцінку, панель Performance у складі інструментів розробника (Chrome DevTools) призначена для глибокого ручного

профілювання та мікроаналізу продуктивності вебдодатка під час його виконання (runtime performance). Цей інструмент дозволяє записувати та візуалізувати всі процеси, що відбуваються в браузері під час завантаження сторінки або взаємодії з нею, з точністю до мілісекунди.

Використання панелі Performance є критично важливим етапом для діагностики та оптимізації метрики Interaction to Next Paint (INP). Інструмент надає деталізовану візуалізацію активності основного потоку (Main thread) у вигляді каскадних діаграм (flame charts). Це дозволяє розробникам з високою точністю ідентифікувати «тривалі завдання» (Long Tasks) – процеси відмальовування або виконання скриптів JavaScript, які тривають понад 50 мілісекунд. Саме такі завдання блокують основний потік, перешкоджаючи браузеру оперативно реагувати на введення користувача, що безпосередньо призводить до високих значень затримок (Input delay та Processing duration) у рамках INP.

Для забезпечення об'єктивності тестування панель Performance має вбудовані механізми троттлінгу (throttling) – штучного обмеження продуктивності центрального процесора (CPU) та швидкості мережевого з'єднання. Ця функція є необхідною, оскільки розробники зазвичай використовують потужне обладнання, тоді як значна частина реальних користувачів взаємодіє з додатком через мобільні пристрої середнього класу зі слабшим процесором. Завдяки симуляції реальних умов та глибокому аналізу стека викликів (Call Tree) інструмент дозволяє не лише виявити факт низької швидкодії, але й локалізувати конкретну функцію або скрипт, що є першопричиною проблеми на рівні вихідного коду [13].

Для проведення розширеного синтетичного тестування та поглибленого аналізу мережевої взаємодії вебдодатків використовується спеціалізована платформа WebPageTest. На відміну від локальних засобів аудиту, ця система дозволяє виконувати тестування з різних географічних локацій, використовуючи реальні браузери та фізичні мобільні пристрої в умовах

суворо контрольованих профілів мережевого з'єднання (наприклад, точна емуляція швидкості та затримок 3G або 4G мереж).

Результатом роботи WebPageTest є вичерпний набір діагностичних артефактів, що включає детальні каскадні діаграми завантаження кожного окремого ресурсу (Waterfall charts), покадрову розгортку процесу відмальовування інтерфейсу (Filmstrip views) та відеозапис екрана. Ці дані дозволяють розробникам візуально оцінити оптимізацію критичного шляху рендерингу (Critical Rendering Path) та ідентифікувати вузькі місця на рівні сервера чи мережі.

Важливою перевагою платформи є підтримка функціоналу користувацьких скриптів (Custom Scripting), який дозволяє тестувати не лише початкове завантаження сторінки, але й багатокрокові сценарії поведінки (наприклад, процеси авторизації або додавання товару до кошика). Саме застосування таких скриптів дає змогу імітувати цільові кліки користувача безпосередньо під час тестування, що надає розробникам унікальну можливість фіксувати, профілювати та оптимізувати показник затримки відгуку (INP) для специфічних елементів інтерфейсу ще на етапі лабораторного середовища [14].

1.3.2 Інструменти на основі польових даних (Field Data / RUM)

Для отримання комплексної оцінки продуктивності вебсторінок, що поєднує як лабораторні метрики, так і показники реального користувацького досвіду, використовується сервіс Google PageSpeed Insights (PSI). Цей інструмент виступає основною сполучною ланкою між синтетичним тестуванням та польовими даними.

Першоджерелом польових даних для PSI є фундаментальний масив Chrome User Experience Report (CrUX). CrUX – це глобальна база даних, яка агрегує анонімізовану статистику взаємодії мільйонів реальних користувачів браузера Google Chrome із вебсайтами. Саме цей набір даних використовується пошуковою системою Google для оцінки показників Core Web Vitals. Під час аналізу сторінки PSI підтягує з CrUX статистику за

останні 28 днів, що дозволяє об'єктивно виміряти метрику інтерактивності INP, яка повноцінно фіксується лише в польових умовах.

Паралельно з відображенням даних CrUX, сервіс генерує звіт у режимі реального часу за допомогою інтегрованого рушія Lighthouse. Така гібридна архітектура інструменту дозволяє розробникам зіставляти змодельовану швидкодію (Lab Data) із фактичним користувацьким досвідом, виявляючи приховані проблеми продуктивності, які можуть не проявлятися у контрольованому середовищі розробки [15].

Для глобального моніторингу продуктивності вебдодатка на рівні всієї його архітектури використовується інструмент Google Search Console (GSC). Ключовим для оцінки якості є вбудований звіт «Основні інтернет-показники» (Core Web Vitals), який агрегує виключно польові дані (Field Data) з масиву Chrome User Experience Report (CrUX). Відмінною рисою GSC, порівняно з інструментами точкового аудиту, є застосування механізму кластеризації. Алгоритми системи автоматично групують URL-адреси зі схожими показниками та архітектурою (наприклад, усі сторінки карток товарів) у єдині пули. Це дозволяє розробникам ідентифікувати системні проблеми на рівні загальних шаблонів чи компонентів, а не ізольованих сторінок.

Система класифікує згруповані URL-адреси за трьома станами: «Добре», «Потребує покращення» та «Погано» для кожної з ключових метрик (LCP, CLS та INP). Такий макрорівневий підхід забезпечує ефективну пріоритезацію завдань з оптимізації, дозволяє оперативно виявляти масові регресії продуктивності після впровадження нового функціоналу та відстежувати загальну динаміку відповідності вебдодатка критеріям алгоритмів ранжування Google щодо якості користувацького досвіду [16].

1.4 Загальна логіка та технічні підходи до покращення Core Web Vitals

У сучасній веб-розробці покращення показників Core Web Vitals не зводиться до одноразового застосування набору правок, а має вигляд безперервного інженерного циклу. Така організація роботи зумовлена тим, що клієнтська частина веб-сайту складається з великої кількості взаємопов’язаних ресурсів

(HTML, CSS, JavaScript, медіа), і будь-яка зміна може одночасно впливати на декілька метрик, як у позитивний, так і в негативний бік. Тому усталеним підходом є послідовне виконання трьох укрупнених етапів: оцінювання поточного стану, налагодження та покращення, моніторингу досягнутих результатів [17].

Перший етап — оцінювання стану веб-сайту — передбачає визначення фактичного рівня продуктивності та виявлення сторінок або метрик, які потребують першочергової уваги. На цьому етапі розробник з'ясовує, які саме показники Core Web Vitals — LCP, CLS чи INP — не відповідають рекомендованим пороговим значенням, і фіксує вихідні значення як точку відліку для подальшого порівняння. У лабораторному дослідженні для оцінки інтерактивності замість польової метрики INP використовується пов'язана з нею Total Blocking Time, оскільки повноцінне вимірювання INP можливе лише на основі реальних взаємодій користувача [17].

Другий етап — налагодження та покращення — є центральним з інженерної точки зору. Після того, як визначено цільову сторінку та критичну метрику, розробник встановлює технічну першопричину проблеми та впроваджує конкретні зміни у клієнтську частину веб-сайту. Для цього послідовно застосовуються три типи дій: ознайомлення зі загальними рекомендаціями автоматизованого аудиту (Lighthouse) для отримання переліку потенційних проблем; використання режиму спостереження за метриками у реальному часі під час завантаження та взаємодії зі сторінкою; виконання детального трасування активності браузера для встановлення причин блокування головного потоку, зсувів макета та інших порушень швидкодії [17].

Принципово важливим є те, що інструменти лабораторного аналізу у цьому циклі виконують виключно діагностичну функцію — вони не реалізують покращень самі по собі, а формують перелік потенційних проблем і кращих практик [17]. Реальне покращення метрик досягається лише застосуванням конкретних технічних прийомів у вихідному коді сторінки, склад яких залежить від цільової метрики Core Web Vitals.

Третій етап — моніторинг змін — забезпечує стабільність досягнутих результатів. Після впровадження виправлень виконується повторне вимірювання у тих самих умовах, що й на першому етапі, з метою підтвердження ефекту. Подальші зміни у проєкті відстежуються через механізми безперервної інтеграції, які автоматично запускають аудит при кожній зміні коду та сигналізують про погіршення метрик. Необхідність цього етапу обумовлена емпірично встановленою тенденцією: значна частина покращень продуктивності схильна до регресій у межах кількох місяців після впровадження, якщо не забезпечено системний контроль [17].

Розглянутий цикл є узагальненою організаційною основою покращення Core Web Vitals і фіксує, що саме робить розробник на кожному з етапів. Конкретний склад технічних змін, що застосовуються на другому етапі циклу для впливу на метрики, формується із систематизованих у практиці веб-розробки прийомів, які розглянуто далі.

Підходи до технічної оптимізації Core Web Vitals структуруються за цільовою метрикою, оскільки LCP, CLS та TBT мають різну фізичну природу — швидкість появи основного вмісту, стабільність макета та обсяг блокування головного потоку — і відповідно потребують різних інженерних дій. Нижче розглянуто прийоми, які усталилися у практиці фронтенд-розробки для кожної з цих метрик.

1.4.1 Підходи до покращення LCP

Метрика LCP визначається часом появи найбільшого елемента у видимій частині першого екрану, тому технічні прийоми її покращення спрямовані на прискорення доставки та відмальовування ключового ресурсу. У межах цього напряму застосовуються:

- пріоритизація LCP-ресурсу — визначення елемента, що є LCP-кандидатом (зазвичай — велике зображення hero-секції, постер відео або об’ємний текстовий блок), і концентрація оптимізації саме на ньому, а не на всіх медіа сторінки;

- раннє завантаження ключового зображення через директиву preload, атрибут fetchpriority="high", а також виключення loading="lazy" для контенту першого екрану — з метою скорочення затримки до початку завантаження ресурсу;
- оптимізація формату та розміру медіа — стиснення зображень, перехід до сучасних форматів (WebP, AVIF), застосування srcset і sizes для адаптивності, що зменшує тривалість завантаження ресурсу;
- скорочення render-blocking CSS — обмеження кількості блокуючих таблиць стилів у секції <head>, виділення мінімального критичного CSS для першого екрану, відкладення або об'єднання некритичних стилів;
- покращення веб-шрифтів (у випадку, коли LCP-кандидатом є текст) — застосування preload для файлу шрифта, директиви font-display: swap, добір fallback-гарнітури з близькими метриками;
- мережеві підказки — використання preconnect і dns-prefetch для зовнішніх джерел (CDN, сервіси шрифтів), що зменшує час встановлення з'єднання з ресурсами, які впливають на LCP [7].

1.4.2 Підходи до покращення CLS

Метрика CLS відображає неочікувані зсуви макета під час завантаження та використання сторінки, тому технічні прийоми її покращення спрямовані на резервування геометрії елементів і запобігання змінам положення вже відмальованого вмісту. Основними прийомами є:

- встановлення явних розмірів медіа — використання атрибутів width і height або CSS-властивості aspect-ratio для зображень і відео, що дозволяє браузеру зарезервувати простір ще до завантаження файлу;

- резервування слотів під динамічний контент — задання фіксованих або мінімальних розмірів для рекламних блоків, банерів, віджетів та інших елементів, які з’являються асинхронно;
- стабілізація відображення шрифтів — добір fallback-гарнітури з метриками, близькими до основного шрифта, для запобігання різкій зміні висоти рядка після підвантаження веб-шрифта;
- безпечні анімації — використання CSS-властивостей transform і opacity замість зміни геометричних властивостей (width, height, top, left), що не провокують перерахунок макета;
- контрольована вставка DOM-елементів — уникнення додавання нового контенту над уже видимим без попередньо зарезервованого простору, застосування placeholder- та skeleton-блоків фіксованого розміру [18].

1.4.3 Підходи до покращення TBT як лабораторного орієнтира інтерактивності

У лабораторному середовищі для оцінки інтерактивності використовується метрика Total Blocking Time, що фіксує сумарний час блокування головного потоку довгими завданнями (понад 50 мс) під час завантаження сторінки. Технічні прийоми її покращення спрямовані на зменшення обсягу та тривалості синхронної роботи JavaScript:

- відкладене виконання скриптів — застосування атрибутів defer або async до некритичних JavaScript-файлів, що дозволяє не блокувати розбір HTML і скоротити роботу головного потоку на старті;
- зменшення обсягу JavaScript — видалення невикористовуваних бібліотек, заміна важких плагінів легшими аналогами, мінімізація файлів, що скорочує час парсингу та виконання коду;

- відкладена ініціалізація компонентів — підключення каруселей, мобільного меню, віджетів та інших інтерактивних елементів не одразу під час завантаження сторінки, а після завершення відмальовування першого екрану або при першій взаємодії користувача;
- розбиття довгих задач — поділ важких обчислень на менші частини за допомогою setTimeout, requestIdleCallback або винесення у Web Workers для запобігання блокуванням головного потоку тривалістю понад 50 мс;
- спрощення обробників подій — покращення функцій обробки кліків, введення даних, прокрутки (зокрема через debounce та throttle), уникнення важкої синхронної роботи в обробнику;
- зменшення зайвого CSS на старті — відкладення таблиць стилів, не потрібних для першого екрану (зокрема стилів іконкових наборів, каруселей, мобільного меню), що скорочує загальний обсяг роботи браузера під час початкового рендерингу [19].

Розглянуті матеріали дозволяють зафіксувати дворівневу структуру існуючих підходів до оптимізації Core Web Vitals. На організаційному рівні робота розробника описується трикроковим циклом «оцінювання — налагодження та покращення — моніторинг», у межах якого інструменти лабораторного аналізу виконують діагностичну функцію та формують перелік потенційних проблем. На технічному рівні для кожної з метрик Core Web Vitals напрацьовано усталений набір прийомів покращення: пріоритизація LCP-ресурсу та покращення критичного шляху рендерингу для LCP; резервування геометрії та контроль шрифтів для CLS; зменшення й відкладення JavaScript для TBT/INP.

Водночас аналіз цих підходів виявляє суттєву методичну прогалину. Загальна логіка роботи програміста та доступні технічні прийоми описують, що саме потрібно робити для покращення показників, проте не визначають, у якій послідовності доцільно впроваджувати численні можливі

зміни у межах одного проєкту. Перелік потенційних покращень, що формується на етапі діагностики, як правило, містить десятки рекомендацій, які відрізняються очікуваним ефектом на метрики, складністю реалізації та ризиком регресії інтерфейсу. Однак ані офіційні методологічні матеріали, ані похідні технічні рекомендації не формалізують єдиного правила черговості їх впровадження.

За цих умов розробник на практиці змушений керуватися неформальними міркуваннями або суб’єктивним досвідом, що знижує відтворюваність процесу покращення та ускладнює об’єктивне порівняння ефекту окремих змін. Це обумовлює необхідність розгляду наявних моделей пріоритезації технічних змін і обґрунтування доцільності розробки спеціалізованого методу для задач оптимізації Core Web Vitals, що становить зміст наступного підрозділу.

1.5 Аналіз моделей пріоритезації та обґрунтування необхідності розробки спеціалізованого методу.

Як показано у підрозділі 1.4, на етапі налагодження та оптимізації Core Web Vitals формується перелік кандидатів на впровадження — дискретних технічних змін, отриманих за результатами лабораторного аудиту та структурного аналізу клієнтської частини web-сайту. Цей перелік зазвичай містить десятки рекомендацій, які відрізняються очікуваним ефектом на метрики (LCP, CLS, INP/TBT), складністю реалізації та ризиком регресії інтерфейсу. У межах одного проєкту з обмеженими ресурсами реалізувати весь набір змін одночасно неможливо, тому виникає задача обґрунтованого вибору черговості їх впровадження. У сучасній практиці для розв'язання таких задач застосовуються моделі пріоритезації двох укрупнених категорій: продуктові моделі, що сформувалися в управлінні розробкою програмного забезпечення для впорядкування ініціатив у backlog проєкту, та інженерні підходи, які використовуються безпосередньо розробниками при покращенні вебпродуктивності. Нижче розглянуто формальні засади кожної з груп та проаналізовано їх застосовність до задач оптимізації Core Web Vitals.

До найпоширеніших продуктових моделей належать RICE, ICE та WSJF, які забезпечують кількісне впорядкування різнорідних ініціатив за єдиним інтегральним показником. Модель RICE описує пріоритет ініціативи через відношення очікуваної користі до зусиль, обчислюваним за залежністю:

$$\text{RICE} = \frac{(\text{Reach} * \text{Impact} * \text{Confidence})}{\text{Effort}} \quad (1.3)$$

Reach відображає охоплення аудиторії,

Impact — силу очікуваного впливу за умовною шкалою,

Confidence — рівень довіри до оцінок у діапазоні від 0 до 1,

Effort — трудомісткість реалізації [20].

Модель ICE є спрощеною схемою без урахування охоплення; замість трудомісткості використовується оцінка простоти реалізації, а узагальнений показник найчастіше обчислюється як добуток трьох складових:

$$\text{ICE} = \text{Impact} * \text{Confidence} * \text{Ease} \quad (1.4)$$

Impact — силу очікуваного впливу за умовною шкалою,

Confidence — рівень довіри до оцінок у діапазоні від 0 до 1,

Ease – простота реалізації.[21].

Модель WSJF (Weighted Shortest Job First) застосовується у середовищі гнучкої розробки великого масштабу і базується на відношенні вартості зволікання до обсягу роботи:

$$WSJF = \frac{Cost\ of\ Delay}{Job\ Size} \quad (1.5)$$

Cost of Delay відображає очікувані втрати цінності внаслідок відкладення задачі,

Job Size — відносну трудомісткість в узгоджених одиницях команди [22].

Розглянуті продуктові моделі є інструментами управління продуктом, а не задачами оптимізації вебпродуктивності, тому при перенесенні у предметну область Core Web Vitals виявляють низку методичних обмежень. Узагальнений показник користі (Impact, Cost of Delay) не передбачає прив'язки до конкретних метрик, складова Reach для змін, які реалізуються в межах однієї контрольної сторінки, фактично є однаковою та не виконує

розрізнювальної функції, а ризик візуально-функціональної регресії інтерфейсу не виокремлюється як самостійний критерій пріоритезації. З огляду на ці обмеження, безпосереднє застосування продуктових моделей до задач оптимізації Core Web Vitals є недоцільним.

Серед інженерних підходів, які застосовуються розробниками безпосередньо при оптимізації вебпродуктивності, найпоширенішим неформальним інструментом є матриця «Impact / Effort», у якій кандидати на впровадження розташовуються у системі координат «вплив — зусилля» з подальшим розподілом на чотири квадранти: швидкі виграші (високий вплив за низьких зусиль), масштабні проєкти (високий вплив за високих зусиль), допоміжні зміни (низький вплив за низьких зусиль) та неефективні задачі (низький вплив за високих зусиль) [23]. Перевагою такого підходу є наочність і простота узгодження у команді, проте він не передбачає кількісної шкали, не диференціює вплив на окремі метрики Core Web Vitals та не містить самостійного критерію ризику регресії.

Більш формалізованою інженерною моделлю є метод PIE (Potential, Importance, Ease), яка обчислюється як середнє арифметичне трьох складових:

$$PIE = \frac{Potential+Importance+Ease}{3} \quad (1.6)$$

Potential відображає потенціал покращення показника,

Importance — важливість сторінки або компонента для проєкту,

Ease — простоту реалізації, причому кожна складова оцінюється у діапазоні від одиниці до десяти [24]. Обмеженням моделі PIE у контексті задач Core Web Vitals є те, що складова Importance для змін, які впроваджуються на одній і тій самій контрольній сторінці, фактично є константою та не виконує розрізнювальної функції, а Potential залишається узагальненою оцінкою без явної прив'язки до конкретної метрики.

Окремим класом підходів є автоматизоване ранжування рекомендацій Lighthouse, у якому потенційні зміни сортуються за оцінкою очікуваної

економії часу (у мілісекундах) або обсягу даних (у кілобайтах), що розраховується на основі вбудованих у інструмент моделей продуктивності [25]. У багатьох випадках цей порядок використовується розробниками як готовий план впровадження, що пояснюється безпосередньою прив’язкою ранжування до конкретної сторінки та автоматичним характером оцінювання. Водночас отримувана оцінка економії наводиться як узагальнене значення без диференціації внеску в окремі метрики Core Web Vitals, не враховує складність реалізації та ризик регресії, а самі рекомендації часто є взаємозалежними і можуть конфліктувати між собою, що зменшує методичну достатність такого ранжування.

Узагальнений аналіз розглянутих моделей за вимогами задач покращення Core Web Vitals наведено у табл. 1.5. У ній порівняння виконується за чотирма критеріями, які є практично значущими у досліджуваній предметній області: прив’язка показника користі до конкретних метрик Core Web Vitals, урахування складності реалізації, наявність окремого критерію ризику регресії інтерфейсу та узгодженість з результатами лабораторного аудиту.

Таблиця 1.5 — Порівняльний аналіз моделей пріоритезації стосовно задач оптимізації Core Web Vitals

Модель	**Прив’язка до метрик CWV**	**Урахування складності реалізації**	**Окремий критерій ризику регресії**	**Узгодженість з лабораторним аудитом**
RICE	відсутня	присутня (Effort)	відсутній	низька
ICE	відсутня	присутня (Ease)	відсутній	низька
WSJF	відсутня	присутня (Job Size)	відсутній	низька
Impact / Effort	відсутня	присутня (якісно)	відсутній	середня

PIE	відсутня	присутня (Ease)	відсутній	середня
Ранжування Lighthouse	часткова	відсутня	відсутній	висока

Наведене порівняння дозволяє зафіксувати методичну прогалину: жодна з розглянутих моделей не задовольняє усім вимогам оптимізації Core Web Vitals. Продуктовим моделям бракує технічної предметності, а інженерним — кількісної формалізації та окремого критерію ризику регресії. Цей ризик є критично важливим, оскільки навіть технічно прості зміни (наприклад, відкладення завантаження CSS) можуть суттєво порушити зовнішній вигляд інтерфейсу.

З огляду на виявлені обмеження, доцільною є розробка спеціалізованого методу пріоритезації, який має задовольняти таким вимогам:

- оцінка очікуваного ефекту формулюється безпосередньо у термінах метрик Core Web Vitals (LCP, CLS, TBT як орієнтир INP);
- ризик візуально-функціональної регресії виступає самостійним та рівноправним критерієм пріоритезації;
- перелік кандидатів з лабораторного аудиту транслюється у план впровадження за єдиним кількісним показником;
- критерії допускають оцінювання у межах компактних шкал без потреби у складному додатковому інструментарії.

Зазначений набір вимог обумовлює необхідність розробки власної методики на основі моделі трьох критеріїв — Користі (K), Складності (S) та Ризику регресії (R). Її змістове обґрунтування та апробація на практичному web-проєкті становлять предмет другого розділу кваліфікаційної роботи.

Висновки

1. Надано теоретичне обґрунтування еволюції підходів до оцінки швидкодії веб-ресурсів та охарактеризовано ключові метрики

стандарту Core Web Vitals (LCP, CLS, INP) як головні індикатори якості користувацького досвіду.

2. Розглянуто сучасні інструменти для вимірювання та моніторингу вебпродуктивності, які класифіковано за двома основними підходами: лабораторним синтетичним тестуванням (Lighthouse, Chrome DevTools, WebPageTest) та збором польових даних реальних користувачів (PageSpeed Insights, Search Console).
3. Досліджено загальну логіку та технічні підходи до оптимізації Core Web Vitals, що реалізуються у вигляді безперервного інженерного циклу «оцінювання — налагодження та покращення — моніторинг» для кожної з ключових метрик.
4. Проаналізовано наявні продуктові (RICE, ICE, WSJF) та інженерні (Impact/Effort, PIE, ранжування Lighthouse) моделі пріоритезації технічних змін. Для кожної з них визначено методичні обмеження стосовно задач оптимізації Core Web Vitals, зокрема виявлено відсутність окремого кількісного критерію ризику візуально-функціональної регресії.
5. Результати аналізу підтверджують актуальність розробки власного спеціалізованого методу пріоритезації технічних змін на основі компактної моделі трьох критеріїв — користі (K), складності (S) та ризику регресії (R). Цей метод дозволить системно підходити до впровадження технічних змін, нівелюючи обмеження існуючих підходів.

РОЗДІЛ 2

АНАЛІЗ ОБ'ЄКТА ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДУ ПОКРАЩЕННЯ CORE WEB VITALS

2.1 Обґрунтування вибору веб-сайту для дослідження

Виконання практичної частини дослідження, присвяченого покращенню показників Core Web Vitals, потребує обґрунтованого вибору тестового веб-сайту, який одночасно є типовим для реальних рішень і придатним для відтворюваного експерименту в контрольованих умовах. Критерії відбору сформовано насамперед із вимог до науково-дослідницької частини кваліфікаційної роботи: тестовий веб-сайт має бути чітко ідентифікований, його архітектура має дозволяти фіксувати версію набору файлів, а методика вимірювання — забезпечувати порівняння результатів «до впровадження змін» та «після впровадження змін» без впливу зовнішніх факторів, що не пов'язані з покращенням клієнтської частини.

Першим критерієм є контрольованість середовища. Для статичного веб-сайту, представленого локальною копією проєкту, забезпечується стабільність складу сторінок і ресурсів, що знижує ризик неконтрольованих відмінностей між вимірами, які часто виникають на продакшн-ресурсах із динамічним контентом, сторонніми інтеграціями та змінами конфігурації сервера. Другим критерієм є репрезентативність технологій: тестовий веб-сайт має відображати поширений клас фронтенд-рішень, де продуктивність залежить від порядку завантаження HTML/CSS/JavaScript, роботи сторонніх бібліотек і політики керування медіаресурсами. Третім критерієм є наявність типових факторів впливу на Core Web Vitals: домінуючий вміст першого екрану (LCP), ризики зсувів макета під час завантаження та ініціалізації компонентів (CLS), а також інтерактивність і виконання JavaScript (у лабораторних вимірах також через показники, пов'язані з блокуванням головного потоку, зокрема TBT, як додатковий орієнтир на етапі розробки).

У межах роботи визначено межі дослідження: основний фокус — покращення клієнтської продуктивності завантаження та стабільності відображення веб-сторінок і оцінювання Core Web Vitals за узгодженим лабораторним методом Тестовим веб-сайтом обрано багатосторінковий ресурс на базі шаблонного проєкту Deerhost, який у робочій директорії представлений каталогом deerhost-master. За функціональним призначенням це маркетинговий сайт хостинг-послуг із типовою для корпоративних веб-ресурсів навігацією та набором сторінок, що забезпечують презентацію компанії, опис послуг, тарифів, блогу та контактної інформації. У складі проєкту наявні HTML-сторінки, зокрема index.html, about.html, hosting.html, pricing.html, blog.html, blog-details.html, contact.html, 404.html, а також файл main.html, який може використовуватися залежно від конфігурації як допоміжний шаблон або окрема вхідна сторінка. Багатосторінковість наближає тестовий веб-сайт до реальних проєктів середньої складності, де покращення продуктивності має враховувати загальну структуру ресурсів проєкту (рис. 2.1).

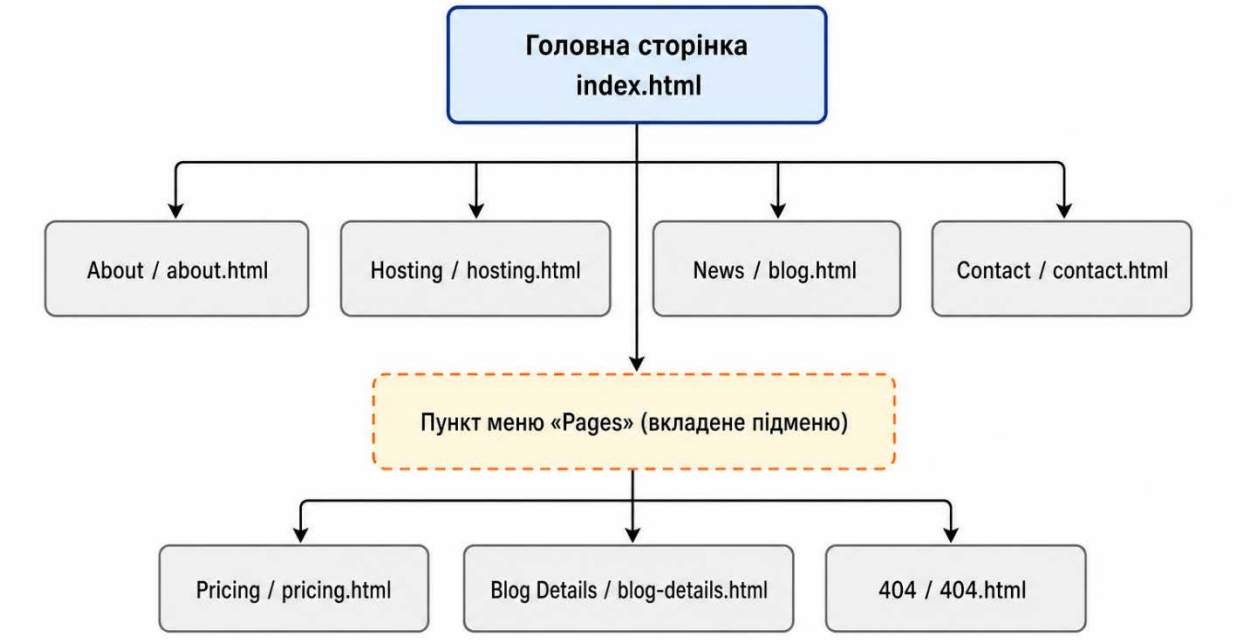

Рисунок 2.1 — Умовна схема структури навігації та зв’язків між HTML-сторінками веб-сайту

Для предметного дослідження Core Web Vitals ключовою є головна сторінка index.html, оскільки вона містить найбільш «навантажений» перший екран із hero-секцією, великими візуальними елементами та ініціалізацією інтерактивних компонентів. Саме на головній сторінці найбільш виразно проявляються залежності між часом відображення основного вмісту, стабільністю макета під час завантаження та навантаженням на головний потік під час ініціалізації сценаріїв. Інші сторінки залишаються складовими цілісного веб-сайту і можуть використовуватися для перевірки узагальнення окремих прийомів покращення, проте базове порівняння показників «до/після» доцільно прив’язати до однієї контрольної сторінки, щоб уникнути змішування ефектів, спричинених різною структурою шаблонів (рис. 2.2).

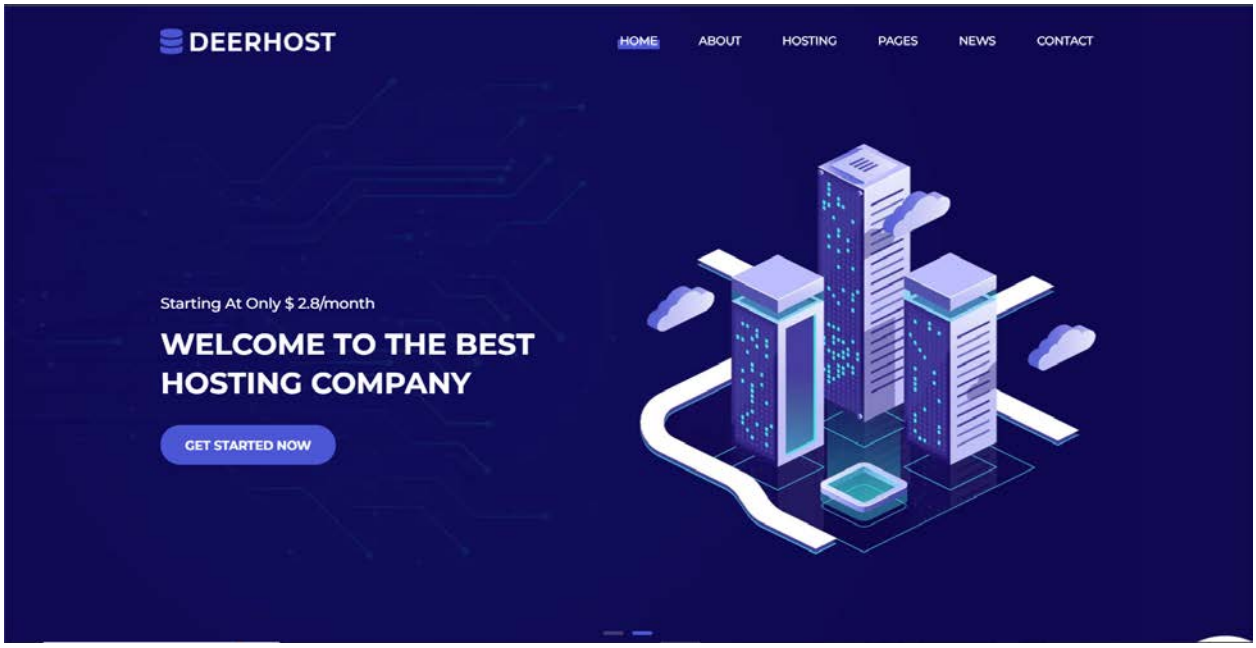

Рисунок 2.2 — Фрагмент інтерфейсу головної сторінки (верхня частина екрану з hero-секцією)

Технологічно проєкт реалізовано як набір статичних HTML-документів із підключенням зовнішніх таблиць стилів і клієнтських сценаріїв. Для реалізації інтерактивних елементів використовується JavaScript із залученням бібліотеки jQuery та відповідних плагінів, зокрема для каруселі на головному екрані та адаптивного мобільного меню. Такий стек є характерним для значної кількості існуючих веб-сайтів і тому є доречним для демонстрації

покращення без обов’язкової повної заміни фреймворку чи відмови від наявних бібліотек (рис. 2.3).

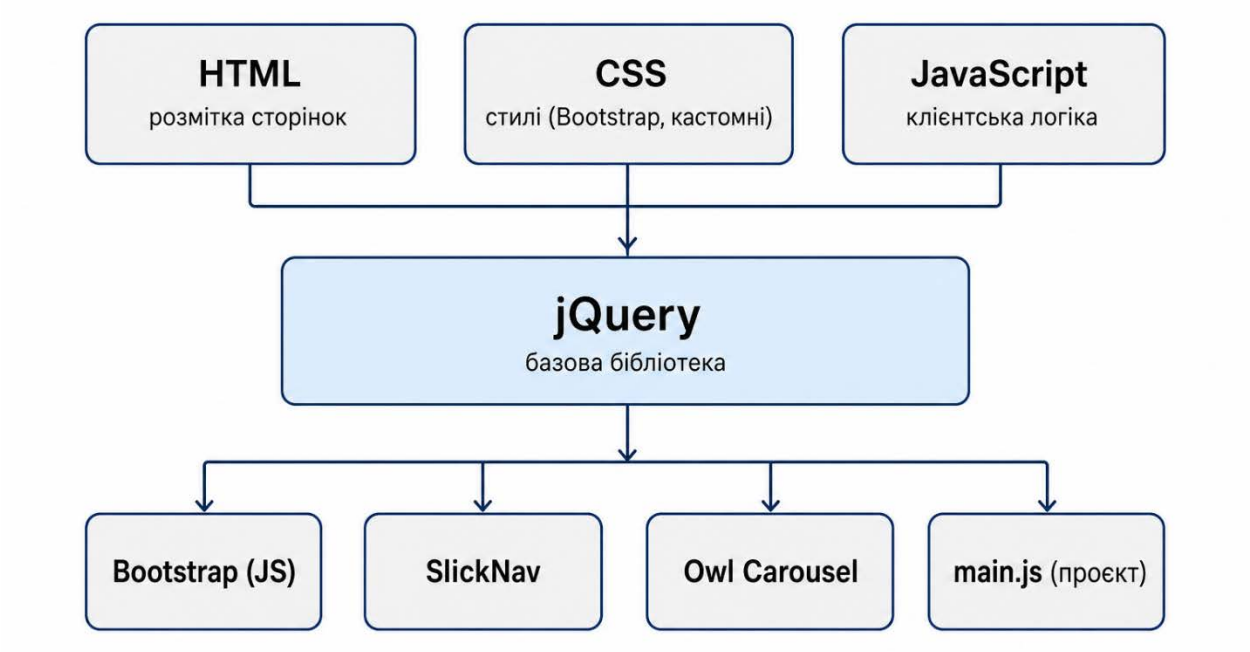

Рисунок 2.3 — Блок-схема технологічного стеку клієнтської частини веб-сайту

Структура каталогів проєкту (HTML, CSS, JavaScript, медіаресурси) забезпечує прозору локалізацію змін і можливість фіксації версії файлів, що підвищує відтворюваність експерименту та спрощує документування впроваджених технічних рішень (рис. 2.4). Основні зведені параметри та межі дослідження обраного веб-сайту наведено в таблиці 2.1.

```
deerhost-master/
├── index.html
├── about.html
├── hosting.html
├── pricing.html
├── blog.html
├── blog-details.html
├── contact.html
├── 404.html
├── main.html
├── css/
│   ├── style.css
│   ├── bootstrap.min.css
│   ├── font-awesome.min.css
│   ├── elegant-icons.css
│   ├── flaticon.css
│   ├── owl.carousel.min.css
│   └── slicknav.min.css
├── js/
│   ├── main.js
│   ├── jquery-3.3.1.min.js
│   ├── bootstrap.min.js
│   ├── jquery.slicknav.js
│   └── owl.carousel.min.js
├── fonts/
│   └── (файли шрифтів та іконок)
└── img/
    └── (медіаресурси сторінок)
```

Рисунок 2.4 — Структура каталогів проєкту (дерево папок і ключових файлів)

Таблиця 2.1 Характеристики обраного веб-додатка

Група характеристик	**Зміст**
Тип рішення	Багатосторінковий статичний веб-сайт (шаблонний проєкт Deerhost)
Призначення	Маркетинговий веб-ресурс хостинг-послуг
Основні сторінки	index.html, about.html, hosting.html, pricing.html, blog.html, blog-details.html, contact.html, 404.html, main.html
Клієнтські технології	HTML, CSS, JavaScript, jQuery, підключаємі бібліотеки UI-компонентів
Контрольна сторінка для вимірювань	index.html
Межі дослідження	Оптимізація клієнтської частини; без аналізу серверної інфраструктури та продакшн-конфігурацій

Коректність порівняння результатів покращення забезпечується однаковістю методики вимірювання на етапі фіксації базового стану та після впровадження змін. У межах роботи доцільно зафіксувати версію браузера на базі Chromium, режим лабораторного аудиту (наприклад, Google Lighthouse у складі Chrome DevTools або еквівалентний сценарій запуску), профіль пристрою (десктоп або мобільна емуляція) та налаштування обмеження мережі й процесора, якщо вони використовуються. Важливо зберегти однаковий спосіб відкриття локальної копії сайту для обох етапів вимірювання (локальний HTTP-сервер або відкриття з файлової системи — залежно від обраної методики). Для зменшення випадкової варіативності доцільно виконувати кілька послідовних прогонів аудиту та узгодити

правило вибору репрезентативного значення (наприклад, медіану серії прогонів), а також мінімізувати вплив сторонніх розширень браузера.

Таким чином, обраний веб-сайт відповідає сформованим критеріям: він є відтворюваним у локальній копії, репрезентативним для типових HTML/CSS/JS-рішень і містить елементи, що дозволяють предметно досліджувати показники Core Web Vitals у межах покращення клієнтської частини. У наступному підрозділі доцільно викласти аналіз структури клієнтської частини та критичного шляху рендерингу обраного тестового веб-сайту як підґрунтя для подальшого базового тестування продуктивності та виявлення вузьких місць.

2.2 Аналіз структури клієнтської частини та критичного шляху рендерингу

Побудова відображення веб-сторінки в браузері описується через DOM (об’єктна модель документа), CSSOM (об’єктна модель стилів) і подальше компонування та малювання (layout / paint). Для контрольної сторінки index.html доцільно виділити три взаємопов’язані площини клієнтської частини: ресурси <head>, які задають раннє оформлення та залежності шрифтів; розмітка та медіа в <body>, що формують видимий вміст; скрипти в кінці документа, що ініціалізують інтерактивні компоненти після розбору основної розмітки.

На критичному шляху суттєве те, які таблиці стилів є блокуючими: типові link rel="stylesheet" змушують браузер враховувати відповідний CSS при побудові дерева відображення, що впливає на момент стабільного показу першого екрану та на ризик зсувів макета. Зовнішнє підключення шрифта через окремий link на сервіс Google Fonts також додає мережевий та обчислювальний етап до раннього рендерингу. Інлайн-скрипт у підвалі сторінки, що оновлює рік у копірайті, має незначний обсяг і не визначає архітектуру завантаження, на відміну від основного блоку підключень у <head> та файлових скриптів у кінці <body> (рис. 2.5).

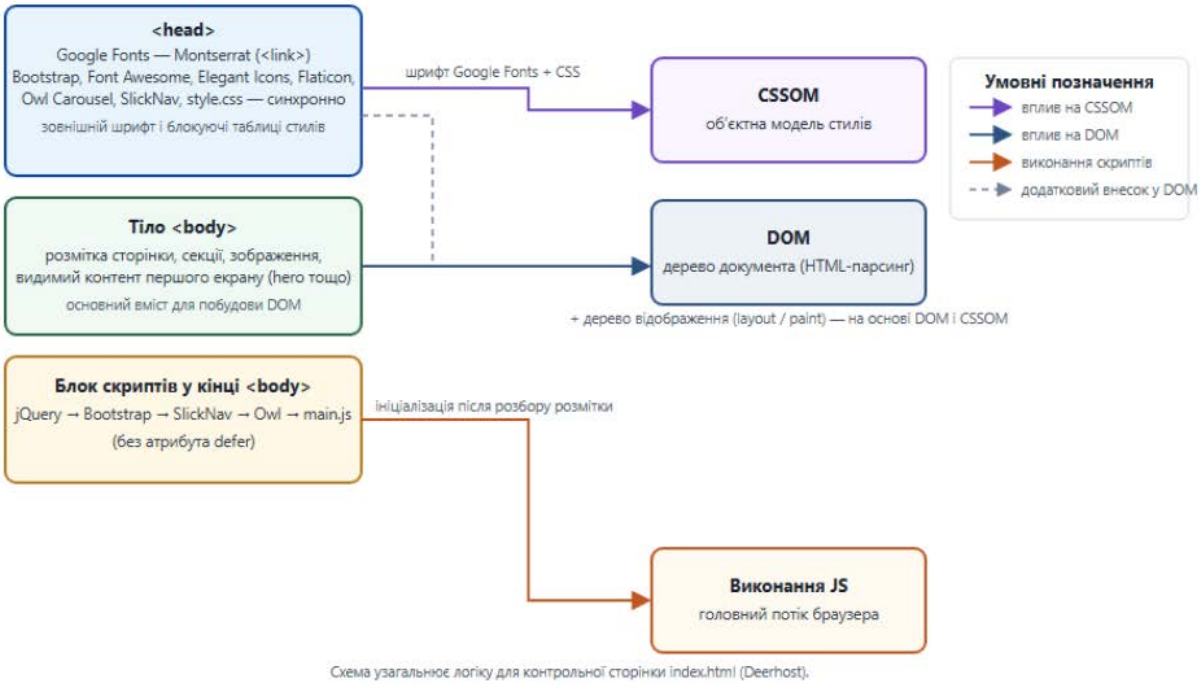

Рисунок 2.5 — Узагальнена схема розподілу ресурсів клієнтської частини сторінки за зонами документа

У секції <head> контрольної сторінки послідовно підключено веб-шрифт Montserrat з зовнішнього джерела та сім зовнішніх таблиць стилів: bootstrap.min.css, font-awesome.min.css, elegant-icons.css, flaticon.css, owl.carousel.min.css, slicknav.min.css, style.css. Усі вони підключені як звичайні стилі без атрибутів відкладеного застосування, тому з точки зору критичного шляху вони утворюють ланцюг блокуючих залежностей на етапі побудови CSSOM. Окремих мережевих підказок (preconnect, preload) та інлайн-стилів у <head> у цій версії розмітки немає, отже прискорення раннього показу за рахунок цих прийомів не реалізовано (рис. 2.6).

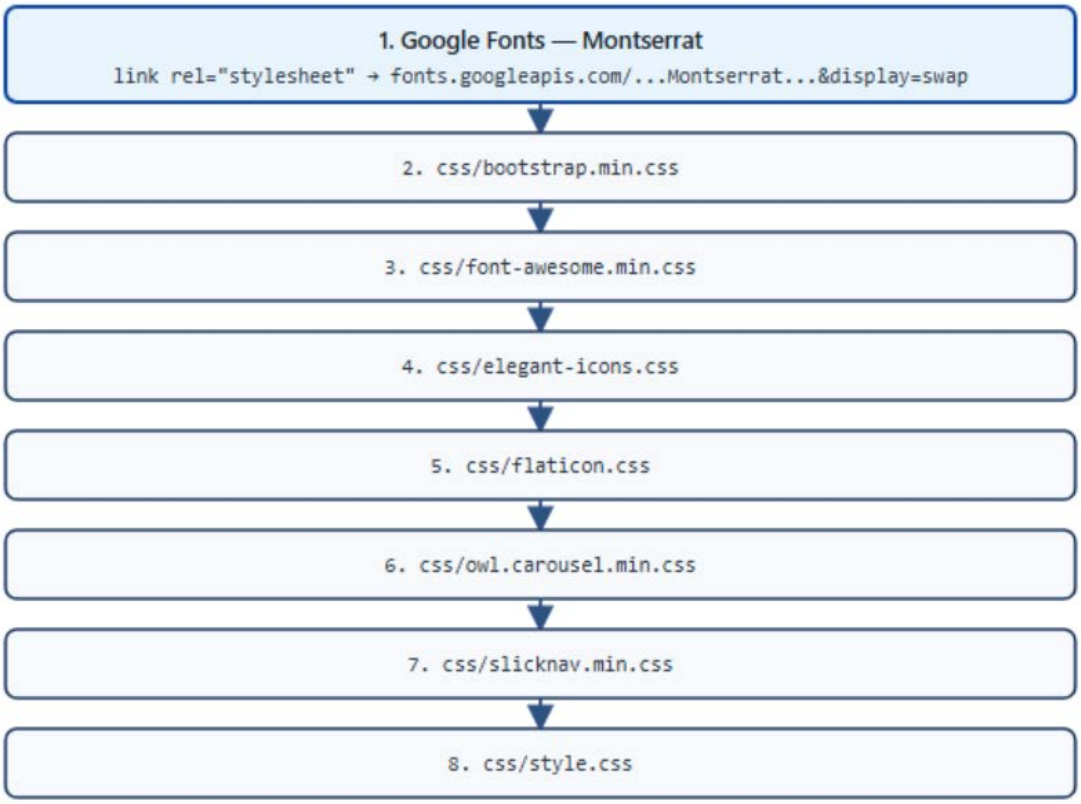

Рисунок 2.6 — Послідовність підключення ключових ресурсів у секції <head> контрольної сторінки

Тіло документа містить, зокрема, hero-секцію з фоновими зображеннями через атрибут data-setbg, зображенням hero-right.png та каруселлю owl-carousel, а також подальші контентні блоки. Для аналізу Core Web Vitals це означає, що LCP-кандидат може бути пов’язаний із великим зображенням або іншим домінуючим елементом першого екрану, а остаточне оформлення hero залежить від повної ініціалізації CSS і скриптів каруселі. Класифікацію ключових ресурсів секції <head> наведено в таблиці 2.2.

Таблиця 2.2 Класифікація ключових ресурсів секції <head> контрольної сторінки index.html

Ресурс	**Функція в документі**	**Роль у критичному шляху (узагальнено)**
link на Google Fonts (Montserrat)	підключення гарнітури	зовнішній CSS; додаткові мережеві запити; типово впливає на етап рендерингу до готовності шрифта
bootstrap.min.css	сітка, базові стилі	блокуючий stylesheet
font-awesome.min.css	набір іконок	блокуючий stylesheet
elegant-icons.css	іконки теми	блокуючий stylesheet
flaticon.css	додаткові іконки	блокуючий stylesheet
owl.carousel.min.css	стилі каруселі	блокуючий stylesheet
slicknav.min.css	стилі мобільного меню	блокуючий stylesheet
style.css	основні стилі шаблону	блокуючий stylesheet

У кінці <body> підключено скрипти у фіксованому порядку: jQuery, Bootstrap, SlickNav, Owl Carousel, main.js. Атрибут відкладеного виконання (defer) для них не використано; розміщення в кінці документа зменшує вплив на початковий розбір розмітки порівняно з підключенням у <head>, проте виконання цих файлів послідовно навантажує головний потік і може корелювати з відгуком на дії користувача після завантаження (рис. 2.7).

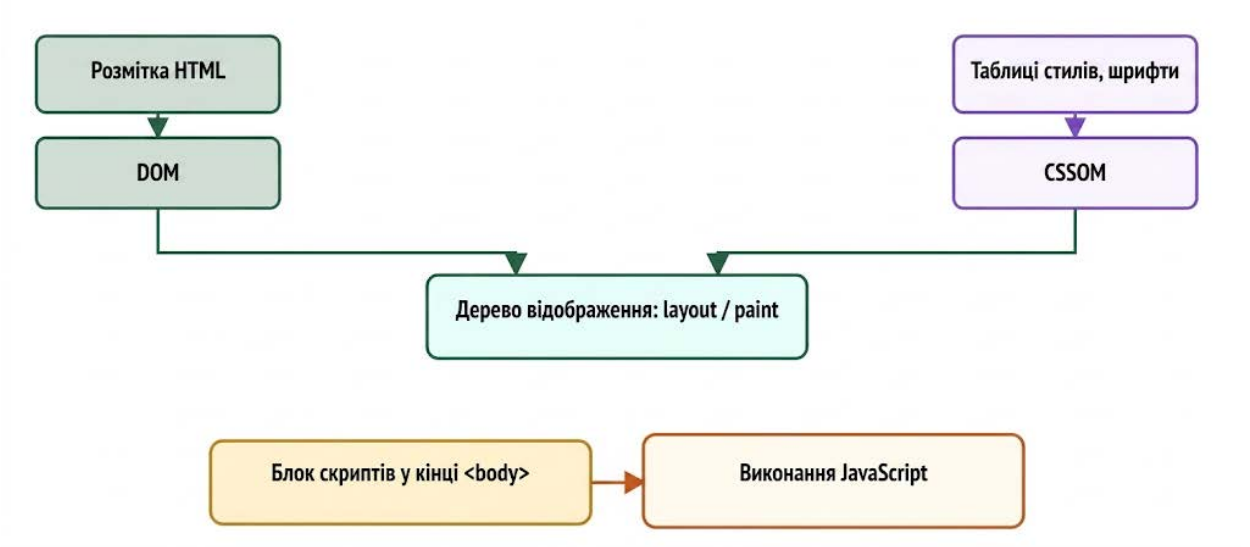

Рисунок 2.7 — Спрощена схема критичного шляху рендерингу веб-сторінки

Отже, підрозділ фіксує технічну модель клієнтської частини контрольної сторінки: блокуючий характер основних стилів і зовнішнього шрифта в <head>, відсутність додаткових мережевих покращень на рівні розмітки та послідовне виконання скриптів бібліотек у кінці документа. Це створює базис для подальшої кількісної оцінки продуктивності та для проєктування заходів покращення з урахуванням Core Web Vitals.

2.3 Базове тестування продуктивності та виявлення вузьких місць

Для кількісної фіксації вихідного рівня продуктивності контрольної сторінки та визначення вузьких місць, пов’язаних із відображенням домінуючого вмісту, стабільністю макета та інтерактивністю, проведено базове лабораторне тестування. Отримані результати тісно узгоджуються з побудованою раніше технічною моделлю клієнтської частини та слугуватимуть відправною точкою для відтворюваного порівняння після реалізації запланованих заходів покращення.

Оцінювання здійснювалося за допомогою інструменту Google Lighthouse (розділ Performance) у двох базових конфігураціях: для мобільного та десктопного профілів. З метою зменшення впливу кешування вимірювання проводилося з очищеним сховищем та із застосуванням симульованого обмеження мережі. Оскільки аудит має синтетичний характер, його

результати розглядаються як модельні орієнтири і базуються на вимірюванні Largest Contentful Paint (LCP) та Cumulative Layout Shift (CLS). Водночас метрика Interaction to Next Paint (INP) потребує фіксації реальних взаємодій користувача, тому для лабораторного порівняння інтерактивності застосовано показник Total Blocking Time (TBT). Він відображає час блокування головного потоку і дозволяє об'єктивно зіставляти різні стани сторінки в однакових умовах (табл. 2.3).

Таблиця 2.3 Лабораторні показники якості завантаження контрольної сторінки (LCP, CLS, TBT)

Показник	**Мобільний профіль**	**Профіль ПК**	**Примітка**
LCP	8,8 с	1,6 с	орієнтир швидкості основного вмісту (CWV)
CLS	1,008	0,621	стабільність макета (CWV)
TBT	100 мс	0 мс	лабораторний орієнтир інтерактивності; не еквівалент INP

Графічне відображення звітів наведено на рисунку 2.8 для мобільного профілю та на рисунку 2.9 для десктопного. Порівняльний аналіз цих конфігурацій виявляє найбільш суттєву різницю для показника LCP: мобільний профіль демонструє значно гірші умови формування домінуючого вмісту порівняно з десктопним, що є очікуваним наслідком жорсткіших параметрів модельованого середовища. Натомість дефіцит візуальної стабільності має системний характер, оскільки показник CLS залишається на незадовільному рівні в обох сценаріях. Показник TBT не виступає домінуючим обмеженням на тлі виявлених проблем із LCP та CLS, що

безпосередньо визначає пріоритетність подальшого аналізу причин зсувів макета та прискорення завантаження ключового вмісту.

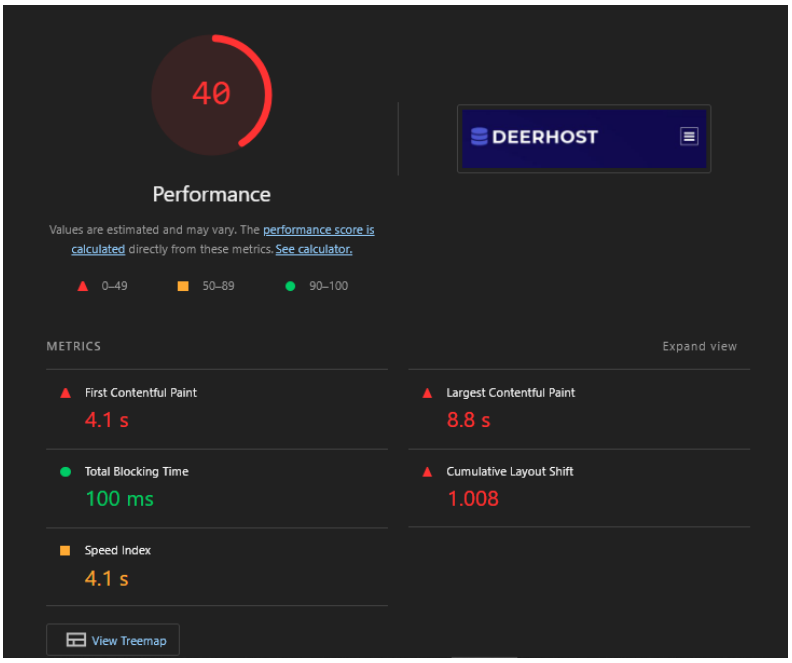

Рисунок 2.8 — Звіт Lighthouse (мобільний профіль)

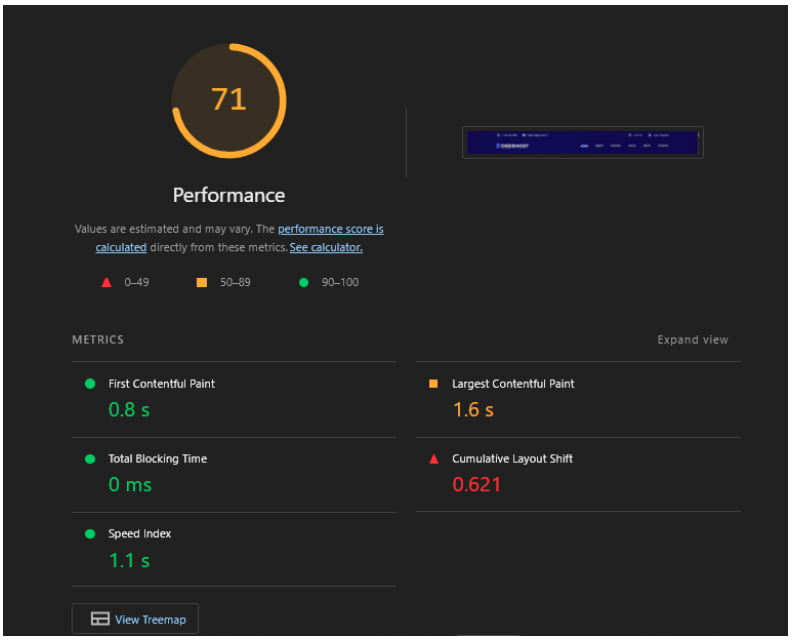

Рисунок 2.9 — Звіт Lighthouse (десктопний профіль)

Для деталізації виявлених проблем виконано перехід від загальних метрик до структурованої діагностики, яку Lighthouse формує у розділах Insights та Diagnostics. Ці дані дозволяють конкретизувати технічні причини затримок.

Зокрема, у розділі Insights пріоритетною проблемою визначено рендер-блокуючі запити, що безпосередньо уповільнюють раннє відображення і критично впливають на показник LCP у мобільному профілі. Інструмент також вказує на необхідність покращення доставки медіаресурсів та містить сповіщення щодо джерел зсувів макета (layout shift culprits) і мережевих ланцюгів завантаження (LCP request discovery, network dependency tree). Окремо зафіксовано проблеми параметрів відображення шрифтів (font display), що є важливим фактором стабільності тексту (рис. 2.10).

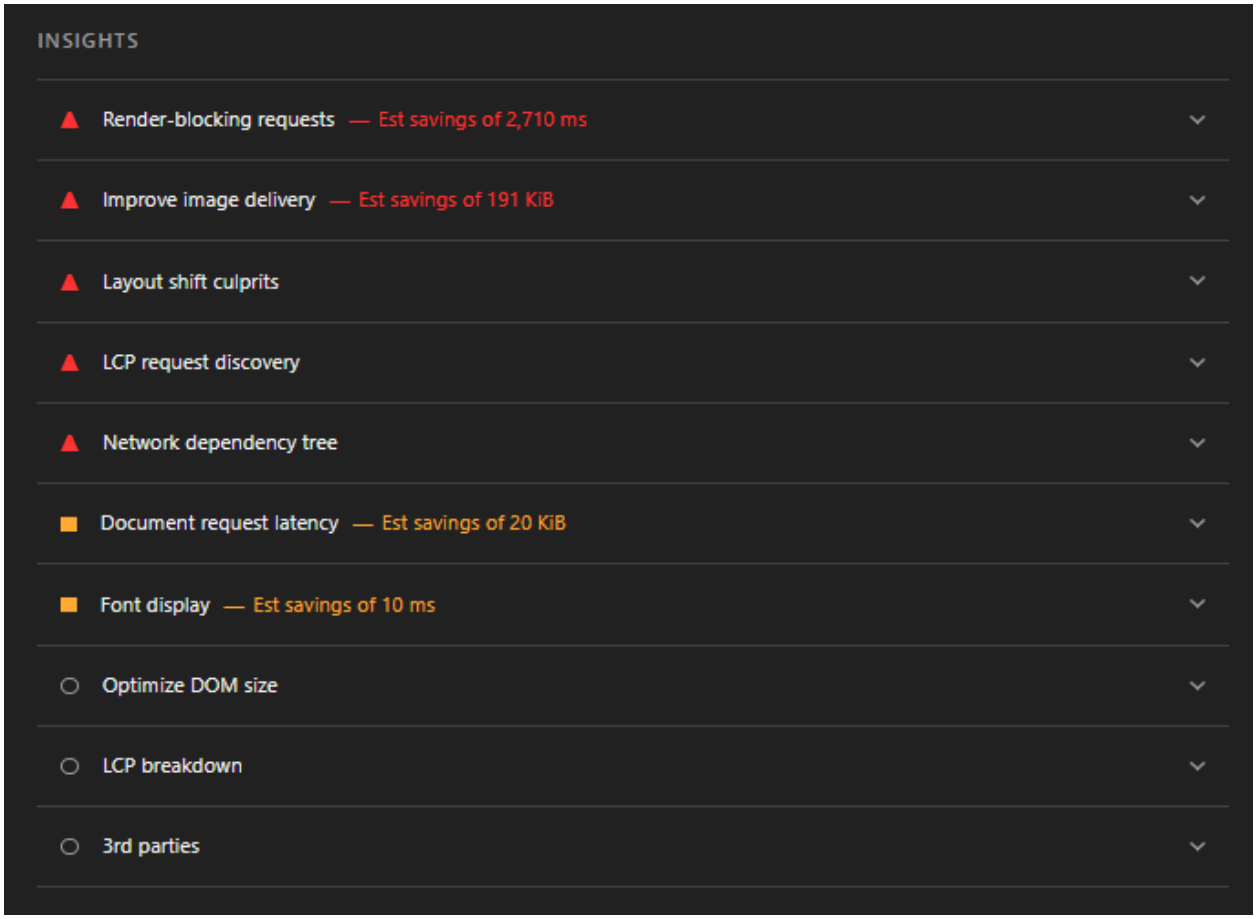

Рисунок 2.10 — Результати автоматизованого аудиту: пріоритетні фактори затримки відображення сторінки

Розділ Diagnostics доповнює загальну картину обчислювальними та ресурсними вузькими місцями. Зафіксовано наявність надлишкового та невикористовуваного коду CSS і JavaScript, а також підвищену тривалість виконання скриптів. Загальна оцінка роботи головного потоку підтверджує значне синхронне навантаження під час завантаження та ініціалізації

клієнтської частини. Для метрики CLS критичним є зауваження щодо відсутності явних атрибутів width та height у зображень, оскільки це є типовою передумовою нестабільності макета (рис. 2.11).

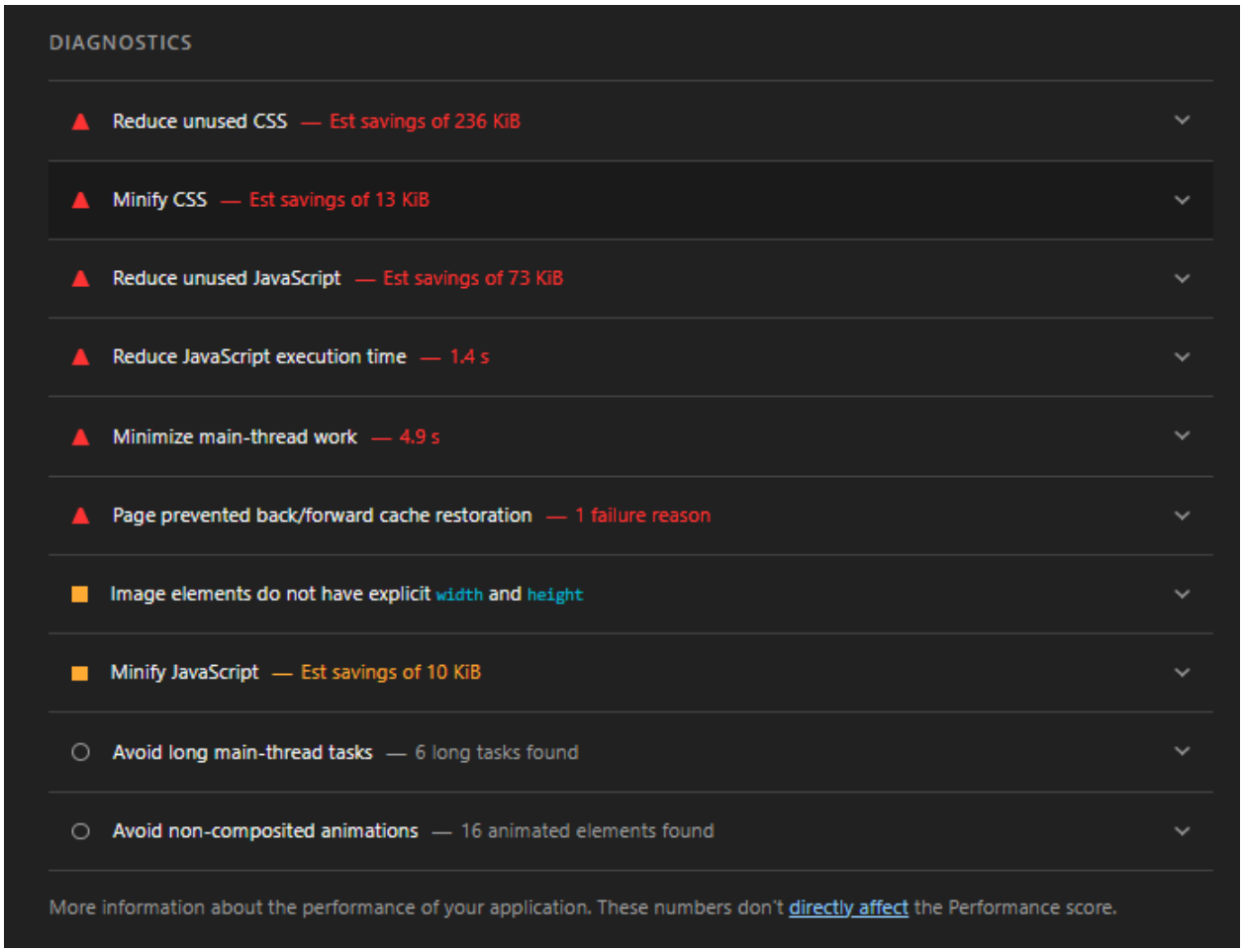

Рисунок 2.11 — Розділ Diagnostics звіту Lighthouse: ресурсні та обчислювальні вузькі місця

Виявлені інструментом тенденції (зокрема рендер-блокуючі таблиці стилів у <head> та зовнішнє підключення шрифтів) повністю підтверджують архітектурні особливості вихідної сторінки, розглянуті у попередньому підрозділі. Це створює надійне підґрунтя для переходу до наступного етапу, де буде формалізовано порядок впровадження технічних рішень для поетапного покращення показників Core Web Vitals.

2.4 Метод поетапного покращення Core Web Vitals

Практична реалізація заходів із покращення Core Web Vitals часто ускладнюється хаотичністю рекомендацій автоматизованих аудитів.

Оскільки різні технічні зміни можуть мати конфліктуючий вплив на метрики та нести неоднакові загрози для стабільності інтерфейсу, виникає потреба в чітко регламентованому процесі. Для розв'язання цієї задачі запропоновано ітераційний метод поетапного покращення. Він базується на кількісному оцінюванні кожної потенційної зміни за трьома незалежними векторами: очікуваною користю K, технічною складністю S та ризиком регресії R.

Критерій K (шкала 0–5) формалізує прогнозований позитивний вплив на цільові показники продуктивності. Замість абстрактної продуктової користі, він обчислюється як зважена сума очікуваних ефектів для кожної окремої метрики:

$$\mathrm{K} = \mathrm{w_{LCP}} * \mathrm{K_{LCP}} + \mathrm{w_{CLS}} * \mathrm{K_{CLS}} + \mathrm{w_{INP}} * \mathrm{K_{INP}} \quad (2.1)$$

$K_{LCP,}\, K_{CLS,}\, K_{INP}$ — це прогнозовані значення покращення для кожної відповідної метрики. Дискретні вагові коефіцієнти $w_{LCP}, w_{CLS}, w_{INP}$ у сумі дорівнюють одиниці і дозволяють гнучко адаптувати формулу до поточного стану проєкту. Наприклад, якщо стабільність макета є найгострішою проблемою веб-сайту, вага w_{CLS} штучно збільшується, що надає пріоритет саме тим змінам, які вирішують цю проблему.

Критерій S (шкала від 0 до 2) характеризує витрати зусиль на реалізацію. Формула розбиває загальну складність на дві окремі складові:

$$S = w_{dev} * S_{dev} + w_{arch} * S_{arch} \quad (2.2)$$

S_{dev} відповідає безпосередньо за трудомісткість написання або модифікації коду розробником, а S_{arch} — за архітектурну складність (глибину втручання у структуру проєкту, зміну шаблонів та порядку підключення залежностей). Компактна шкала від 0 до 2 застосовується навмисно для швидкого відносного ранжування, уникаючи складних і часто неточних розрахунків у людино-годинах.

Критерій R (шкала від 0 до 2) оцінює ймовірність деградації якості веб-сайту після технічного втручання. Він також складається з двох дискретних частин:

$$R = w_{vis} * R_{vis} + w_{func} * R_{func} \quad (2.3)$$

де R_{vis} фіксує ризик візуальної регресії (порушення верстки, неочікувані зсуви компонентів), а R_{func} — ризик функціональної регресії (помилки ініціалізації каруселей, поломка меню чи форм). Відокремлення цих ризиків від загальної «складності» є принциповим: навіть зміна одного рядка коду може повністю зруйнувати логіку відображення, і модель має це враховувати.

Підсумковий показник пріоритету кандидата обчислюється за загальною формулою:

$$Priority = K - \frac{(S+R)}{2} \quad (2.4)$$

Математичний зміст формули полягає у балансуванні: очікувана користь K виступає стимулом для раннього впровадження, а складність S та ризик R — стримуючими штрафними факторами. Поділ на 2 виконує роль нормалізації: він гарантує, що негативні критерії не перекриватимуть користь лише через те, що їх у формулі два. Кандидат із найвищим значенням Priority обирається для першочергової реалізації. Для зручності використання методу всі його складові — критерії, формули розрахунку та шкали оцінювання — візуалізовано у вигляді загальної структури математичної моделі, яку представлено на рисунку 2.12.

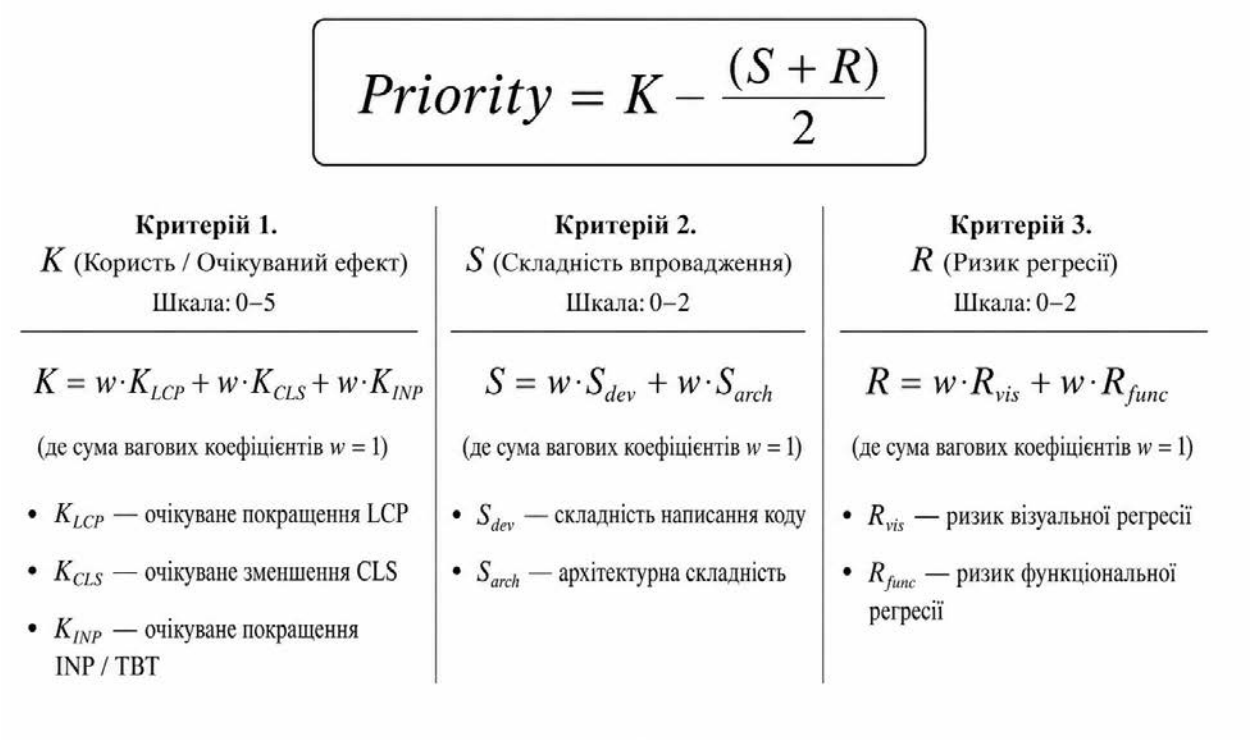

Рисунок 2.12 — Математична модель пріоритезації технічних змін.

Запропонована математична модель вбудовується в чіткий ітераційний алгоритм впровадження змін (рис. 2.13). Цей процес розбивається на чотири послідовні кроки:

1. Фіксація вихідного стану: визначається контрольна сторінка, обирається єдиний профіль лабораторного вимірювання (мобільний або десктопний), встановлюються правила для повторних прогонів (наприклад, серія вимірювань для уникнення похибок) та фіксуються базові значення цільових метрик (LCP, CLS, INP/TBT).
2. Формування пулу кандидатів: на основі результатів первинного аудиту та аналізу критичного шляху рендерингу складається перелік потенційних технічних рішень для покращення швидкодії.
3. Пріоритезація: кожен кандидат із сформованого списку отримує експертну оцінку за критеріями K, S та R, після чого для нього розраховується підсумковий показник Priority. На основі цих значень формується впорядкована черга на впровадження.
4. Ітераційна реалізація: з черги обирається кандидат із найвищим показником Priority, після чого впроваджується суворо одна технічна

зміна. Одразу після цього проводиться повторне вимірювання метрик у тотожних лабораторних умовах, а також перевірка візуальної та функціональної цілісності сторінки. Якщо фіксується регресія, зміна скасовується або коригується, а оцінка ризику R для цієї категорії правок переглядається з огляду на виявлені факти.

У випадку успішного впровадження (коли регресії немає, а цільові метрики покращуються або залишаються в допустимих межах), результат остаточно фіксується. Після цього метод передбачає повернення до оновленого списку кандидатів і перехід до наступної ітерації. Цей цикл безперервно повторюється доти, доки не буде вичерпано перелік доцільних кандидатів або не буде досягнуто цільового рівня метрик, визначеного межами дослідження.

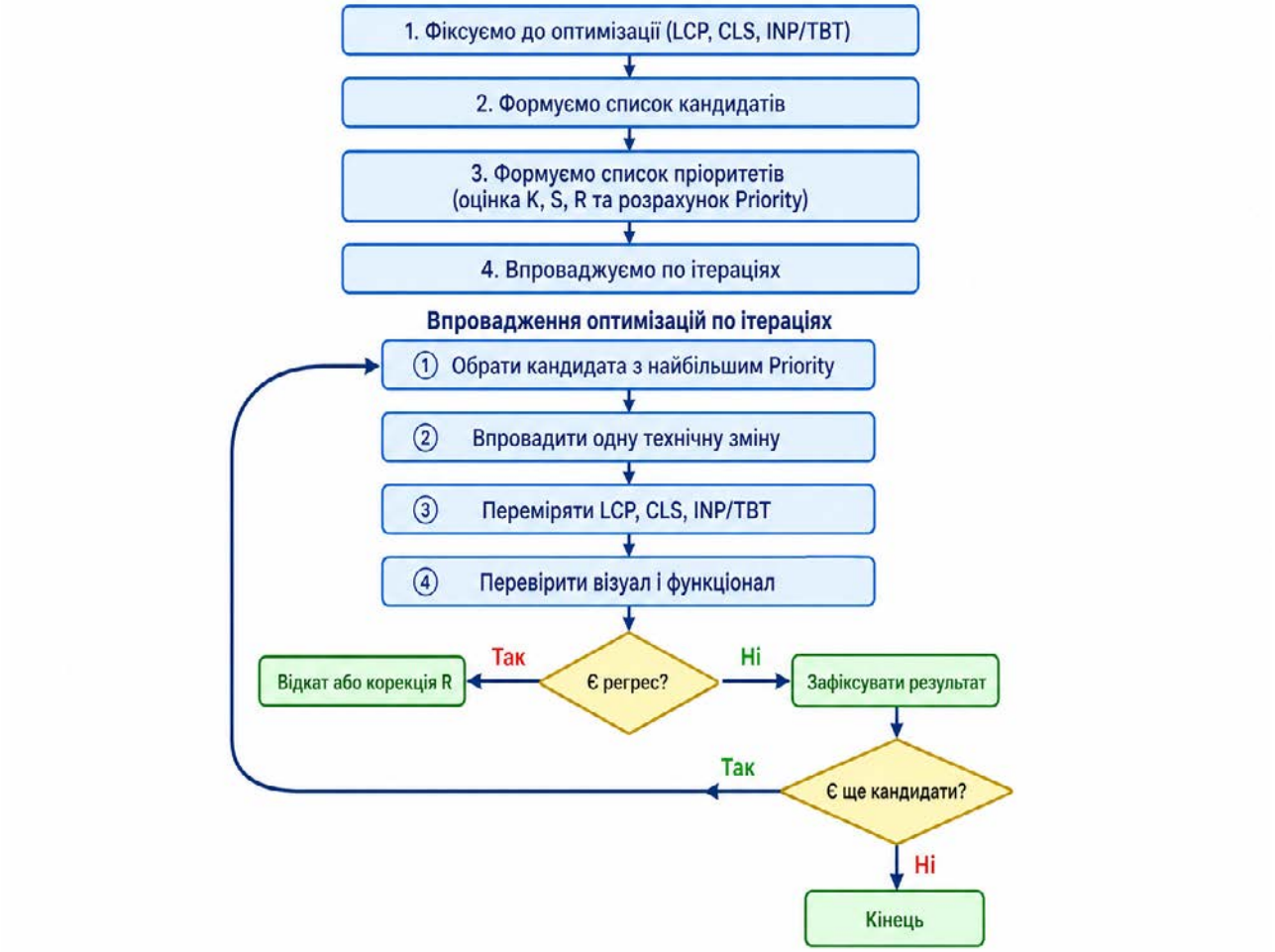

Рисунок 2.13 — Алгоритм поетапного покращення Core Web Vitals за критеріями K, S, R

З огляду на вищезазначене, запропонований метод поетапного покращення успішно поєднує строгу лабораторну вимірюваність результатів із формалізованою пріоритезацією технічних втручань. Він враховує не лише потенційний виграш у швидкодії, а й архітектурну складність реалізації та приховані ризики для користувацького досвіду. Такий підхід гарантує кероване впровадження покращень, мінімізує вплив суб'єктивного фактора під час вибору черговості робіт і формує надійну основу для безпосередньої практичної реалізації цього методу.

2.5 Практична реалізація оптимізацій Core Web Vitals (LCP, CLS, INP/TBT)

Відповідно до розробленого алгоритму (підрозділ 2.4), практична апробація здійснюється шляхом ітераційного впровадження технічних змін на тестовому веб-сайті. Першим кроком реалізації є формування загального пулу непріоритезованих кандидатів на покращення.

Джерелом для формування цього списку слугують результати базового лабораторного тестування (підрозділ 2.3). Інструмент Google Lighthouse генерує великий масив рекомендацій, які згруповані у розділах Insights та Diagnostics. Проте ці рекомендації подаються інструментом без урахування архітектурної складності їх впровадження S та потенційних ризиків регресії R. Такий лінійний підхід робить пряме послідовне виконання порад не лише небезпечним для візуальної та функціональної стабільності веб-сайту, але й суттєво знижує загальну ефективність процесу покращення. Сліпе слідування стандартному списку Lighthouse часто призводить до витрачання ресурсу на складні завдання з мінімальним виграшем у швидкодії. Натомість застосування запропонованого методу пріоритезації (K, S, R) дозволяє виявити найбільш раціональні точки втручання — зміни, що забезпечують максимальний приріст метрик за мінімальних зусиль та ризиків. Це гарантує значно швидше досягнення цільових показників Core Web Vitals порівняно з неструктурованим впровадженням рекомендацій автоматизованого аудиту.

Для підготовки до застосування математичної моделі, усі виявлені технічні зауваження формалізуються та зводяться у єдиний реєстр. Кожному дискретному кандидату (від C1 до C17) присвоюється унікальний ідентифікатор для забезпечення трасованості під час подальших розрахунків. Крім того, для кожного кандидата фіксується його джерело у звіті та головна цільова метрика (LCP, CLS, FCP або TBT), на яку він потенційно впливає. Загальний перелік ідентифікованих кандидатів до початку процедури їх експертного оцінювання та розрахунку підсумкового показника пріоритету (Priority) наведено на рисунку 2.14.

Code	Candidate	Lighthouse Section	Main Metric
C1	Reduce render-blocking requests	Insights	LCP, FCP
C2	Improve image delivery	Insights	LCP
C3	Eliminate layout shift culprits	Insights	CLS
C4	LCP request discovery (preload)	Insights	LCP
C5	Network dependency tree	Insights	LCP
C6	Document request latency	Insights	LCP
C7	font-display / web fonts	Insights	LCP, CLS
C8	Reduce unused CSS	Diagnostics	LCP, CLS, LBI
C9	Minimize CSS	Diagnostics	LCP
C10	Reduce unused JS	Diagnostics	TBT
C11	Shorten JS execution time	Diagnostics	TBT
C12	Reduce main thread work	Diagnostics	TBT
C13	Eliminate bfcache blockers	Diagnostics	INP/TBT
C14	width/height attributes for images	Diagnostics	CLS, LCP
C15	Minimize JavaScript	Diagnostics	TBT
C16	Avoid long tasks	Diagnostics	TBT
C17	Composite animations	Diagnostics	CLS, TBT

Рисунок 2.14 — Непріоритезований перелік кандидатів на впровадження технічних змін за даними аудиту Lighthouse

Для автоматизації математичних розрахунків та унеможливлення суб’єктивних похибок під час обробки масиву ідентифікованих кандидатів, у межах дослідження розроблено спеціалізований програмний скрипт мовою Python. Цей інструмент реалізує раніше описану інтегральну модель оцінювання шляхом інтерактивного збору дискретних підоцінок для кожного кандидата та їх подальшого зведення.

Відповідно до поточного стану тестового веб-сайту та виявленого системного дефіциту візуальної стабільності, вагові коефіцієнти для розрахунку параметрів моделі у скрипті зафіксовано на такому рівні:

- Для критерію користі (K): $w_{LCP} = 0.4, w_{CLS} = 0.4, w_{INP} = 0.2$. Це забезпечує рівноцінний пріоритет вирішенню проблем раннього рендерингу та стабільності макета.
- Для критерію складності (S): $w_{dev} = 0.4, w_{arch} = 0.6$. Зміни архітектури (шаблонів, порядку підключення) штрафуються сильніше, ніж просто обсяг написаного коду.
- Для критерію ризику (R): $w_{vis} = 0.6, w_{func} = 0.4$. Ризик порушення верстки визнано більш імовірним і критичним для контентного веб-сайту, ніж поломка інтерактивних скриптів.

Крім безпосереднього розрахунку значення Priority, скрипт виконує автоматичну класифікацію кандидатів на три групи доцільності: високий пріоритет (при Priority ≥ 2.0), середній (1.0 ≤ Priority < 2.0) та низький (Priority < 1.0). Приклад роботи скрипта в інтерактивному режимі під час експертного оцінювання наведено на рисунку 2.15.

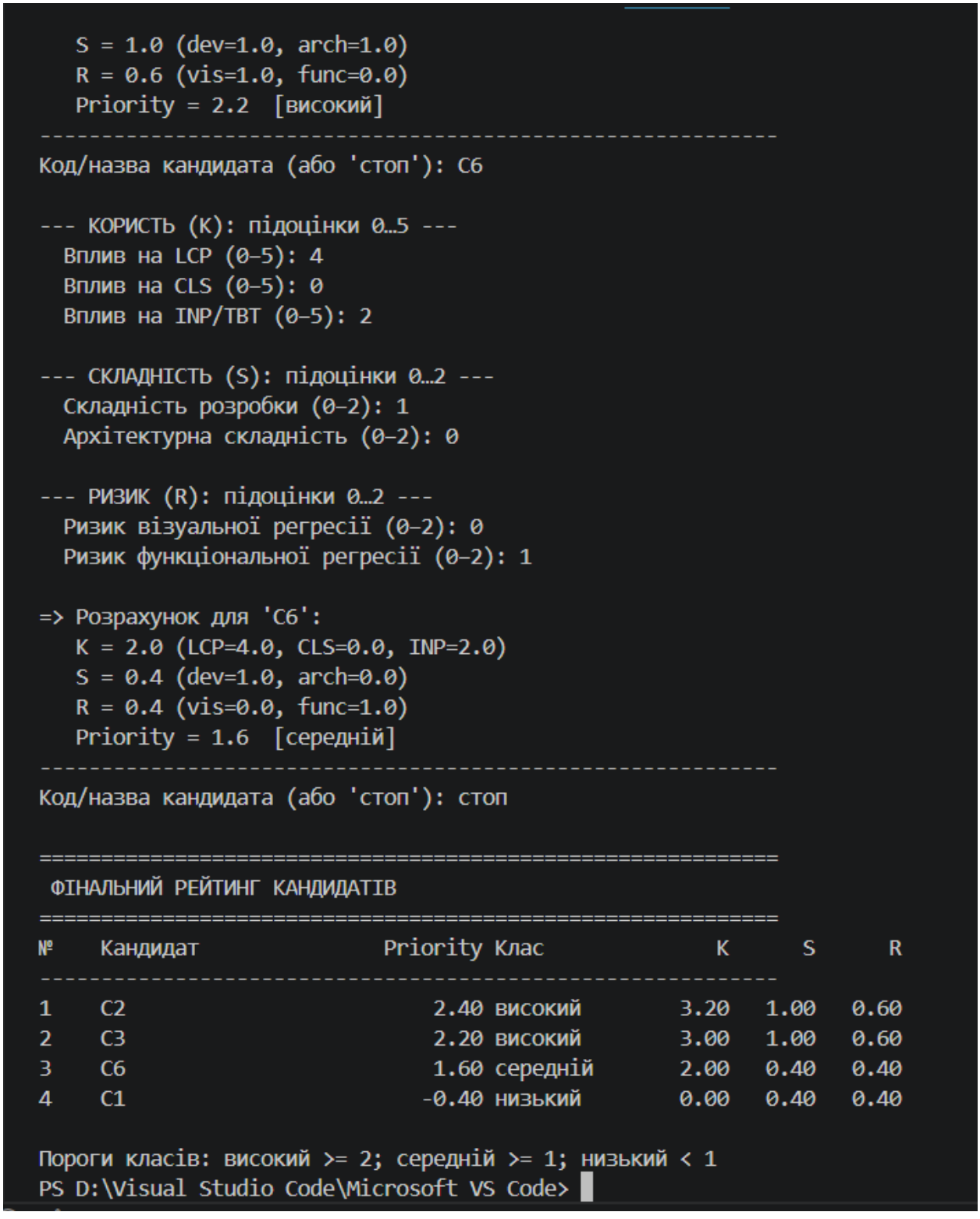

Рисунок 2.15 — Інтерфейс командного рядка скрипта для розрахунку пріоритету технічних покращень

За допомогою розробленого скрипта було проведено експертне оцінювання всіх 17 ідентифікованих кандидатів (від C1 до C17). Кожен з них отримав дискретні бали від 0 до 5 для метрик користі та від 0 до 2 для метрик

складності й ризику. Зведені результати роботи алгоритму, відсортовані за спаданням показника Priority, представлені на рисунку 2.16.

№	ID	Кандидат	K_LCP	K_CLS	K_INP	S_dev	S_arch	R_vis	R_func	K	S	R	Priority	Клас
1	C2	Покращення доставки зображень	5	2	2	1	1	1	0	3,2	1	0,6	2,4	високий
2	C3	Усунення layout shift culprits	2	5	1	1	1	1	0	3	1	0,6	2,2	високий
3	C16	Уникнення long tasks	3	0	5	1	0	0	1	2,2	0,4	0,4	1,8	середній
4	C6	Document request latency	4	0	2	1	0	0	1	2	0,4	0,4	1,6	середній
5	C7	font-display / веб-шрифти	2	2	0	0	0	1	0	1,6	0	0,6	1,3	середній
6	C4	LCP request discovery (preload)	3	0	0	0	0	0	0	1,2	0	0	1,2	середній
7	C12	Зменшення роботи головного потоку	2	0	5	1	1	0	1	1,8	1	0,4	1,1	середній
8	C11	Скорочення часу виконання JS	1	0	5	1	0	0	1	1,4	0,4	0,4	1	середній
9	C10	Зменшення невикористовуваного JS	1	0	4	1	0	0	1	1,2	0,4	0,4	0,8	низький
10	C14	Атрибути width/height	0	2	0	0	0	0	0	0,8	0	0	0,8	низький
11	C13	Усунення перешкод bfcache	0	0	5	1	1	1	0	1	1	0,6	0,2	низький
12	C9	Мінімізація CSS	1	0	0	1	0	0	1	0,4	0,4	0,4	0	низький
13	C8	Зменш. невикористовуваного CSS	2	1	0	2	1	2	1	1,2	1,4	1,6	-0,3	низький
14	C1	Зменшення render-blocking	0	0	0	1	0	0	1	0	0,4	0,4	-0,4	низький
15	C15	Мінімізація JavaScript	0	0	1	1	1	0	1	0,2	1	0,4	-0,5	низький
16	C17	Композитні анімації	0	1	1	2	1	2	0	0,6	1,4	1,2	-0,7	низький
17	C5	Network dependency tree	1	0	0	2	1	1	1	0,4	1,4	1	-0,8	низький

Рисунок 2.16 — Фінальний рейтинг кандидатів на впровадження технічних покращень за моделлю K-S-R

Аналіз отриманого рейтингу наочно демонструє принципову різницю між запропонованим методом пріоритезації та лінійним виконанням рекомендацій Lighthouse. Наприклад, кандидат C1, який інструмент аудиту ставив на перше місце, отримав від’ємний пріоритет (–0.4) та потрапив до категорії низької доцільності через повну відсутність фактичної користі для цільових метрик у реальних умовах даного веб-сайту, що робить навіть мінімальні технічні зусилля та супутні ризики невиправданими.

Натомість найвищі показники Priority отримали кандидати C2 та C3. Відповідно до логіки алгоритму, саме ці два кандидати високого класу визначаються як першочергові цілі для початку ітераційного процесу покращення тестового веб-сайту.

Відповідно до затвердженого рейтингу (рис. 2.16), практична реалізація здійснювалася ітераційно — від кандидата з найвищим показником Priority до найнижчого. Після кожної ітерації проводився лабораторний та візуальний контроль для виявлення можливих регресій.

1. Кандидат C2 (Покращення доставки зображень). Перший етап був спрямований на покращення найбільшого елемента на екрані (LCP). Було проведено масову конвертацію растрової графіки у сучасний формат WebP із

налаштуванням адаптивної подачі зображень залежно від пристрою. Для цільового LCP-зображення встановлено найвищий пріоритет завантаження на рівні браузера, а для всієї графіки, що знаходиться поза межами першого екрана, активовано нативний механізм відкладеного завантаження (lazy loading). Також усунено залежність відображення фонових зображень від виконання JavaScript-коду.

2. Кандидат C3 (Усунення layout shift culprits). Для вирішення проблеми нестабільності макета (CLS) було повністю переглянуто механізм завантаження сторінки. Усунено скрипти повноекранних прелоадерів, які викликали зсуви контенту під час зникнення. Для динамічних блоків та слайдерів реалізовано жорстке резервування простору (через фіксацію мінімальної висоти та пропорцій) ще до моменту їх ініціалізації. Це гарантувало, що підвантаження контенту не "штовхатиме" сусідні елементи сторінки.

3. Кандидат C16 (Уникнення long tasks). Для зменшення блокування головного потоку під час початкового парсингу, об’ємні скрипти було переведено в асинхронний режим. Ініціалізацію складних інтерактивних компонентів (лічильників, анімацій статистики) розділено та прив'язано до появи цих елементів у видимій зоні екрана користувача, що суттєво розвантажило процес первинного рендерингу.

4. Кандидат C6 (Document request latency). Покращення на рівні сервера та розмітки документа. Налаштовано довгострокове кешування для статики, активовано сучасні алгоритми стиснення даних та додано інструкції попереднього підключення до важливих доменів. Структуру HTML-документа очищено від зайвих метаданих для прискорення отримання першого байта (TTFB).

5. Кандидат C7 (font-display / веб-шрифти). Для усунення затримок у відображенні тексту скасовано підключення шрифтів зі сторонніх серверів. Файли шрифтів перенесено на локальний сервер, оптимізовано їх формати та впроваджено CSS-властивість swap, яка гарантує негайне відображення

тексту системним шрифтом до моменту завантаження дизайнерського, мінімізуючи вплив на CLS.

6. Кандидат C4 (LCP request discovery). Підвищено ймовірність раннього виявлення цільових ресурсів браузером. У заголовок документа додано прямі інструкції попереднього завантаження (preload) для критичної графіки, а деякі фонові зображення перенесено безпосередньо в HTML-структуру, щоб парсер міг розпочати їх завантаження до повної обробки таблиць стилів.

7. Кандидат C12 (Зменшення роботи головного потоку). Реалізовано стратегію динамічного підвантаження (динамічний імпорт). Важкі модулі інтерфейсу (наприклад, скрипти мобільного меню чи складних каруселей) ініціалізуються виключно в момент фактичної потреби або взаємодії користувача. Для прихованих секцій застосовано пропуск фази рендерингу до моменту скролінгу.

8. Кандидат C11 (Скорочення часу виконання JS). Проведено жорстке відокремлення критичного JavaScript-коду, необхідного для первинного відображення, від сторонніх бібліотек. Сторонні залежності завантажуються лише під час простою браузера (idle state).

9. Кандидат C10 (Зменшення невикористовуваного JS) — ВІДХИЛЕНО. Здійснено спробу заміни об'ємної бібліотеки слайдера на легке власне рішення. На етапі тестування зафіксовано візуальну регресію: порушилася анімація появи тексту та перестав коректно працювати інтерфейс кнопок. Згідно з математичною моделлю, зміну було повністю відкочено до попереднього стабільного стану.

10. Кандидат C14 (Атрибути width/height) — ВІДХИЛЕНО. Масове автоматизоване додавання атрибутів розміру до всіх зображень шаблону з метою покращення CLS призвело до порушення пропорцій та руйнування адаптивної сітки на внутрішніх сторінках веб-сайту. Зміну визнано архітектурно небезпечною та відхилено.

11. Кандидат C13 (Усунення перешкод bfcache). Код веб-сайту адаптовано для коректної роботи з механізмом миттєвого кешування при навігації

браузера «назад/вперед». Вилучено застарілі обробники подій вивантаження сторінки, що блокували збереження знімка стану браузером.

12. Кандидат C9 (Мінімізація CSS). Виконано стиснення базових таблиць стилів проекту шляхом видалення коментарів та непотрібних пробілів.

13. Кандидат C8 (Зменшення невикористовуваного CSS) — ВІДХИЛЕНО. Спроба автоматизованого видалення невикористаних CSS-правил з готових фреймворків (Bootstrap) призвела до критичного руйнування сітки макета та втрати базових стилів для окремих компонентів. Зміну негайно скасовано.

14. Кандидат C1 (Зменшення render-blocking). Таблиці стилів оптимізовано шляхом розділення на невеликий критичний блок (інтегрований безпосередньо в HTML) та основний масив стилів, що підвантажується асинхронно без блокування первинного відображення екрана.

15. Кандидат C15 (Мінімізація JavaScript). Виконано фінальне стиснення та обфускацію власного коду проекту для зменшення обсягу файлів.

16. Кандидат C17 (Композитні анімації). Усі декоративні анімації інтерфейсу переведено на використання безпечних CSS-властивостей (трансформація та зміна прозорості). Це дозволило передати обчислення анімацій на графічний процесор (GPU), усунувши навантаження на головний потік браузера.

17. Кандидат C5 (Network dependency tree). На завершальному етапі впорядковано ієрархію підключення ресурсів. Оптимізовано паралельність запитів, щоб уникнути конкуренції за пропускну здатність між шрифтами, стилями та LCP-зображеннями під час перших мілісекунд завантаження.

У процесі поетапного виконання технічних змін здійснювався безперервний моніторинг цільових показників Core Web Vitals (LCP, CLS) та лабораторної метрики TBT (Total Blocking Time). Усі вимірювання проводилися в ідентичних лабораторних умовах (мобільний профіль Google Lighthouse) для забезпечення чистоти експерименту. Детальну динаміку зміни показників після кожної ітерації зафіксовано в таблиці 2.4.

Таблиця 2.4 — Динаміка зміни метрик Core Web Vitals у процесі ітераційного покращення

№	ID	Статус	LCP	CLS	TBT	Performance
-	baseline	-	8.8 s	1,008	100 ms	40
1	C2	Виконано	8.8 s → 5.4 s	1,008 → 1,012	100 ms → 110 ms	40 → 48 (+8)
2	C3	Виконано	5.4 s → 5.4 s	1,012 → 0	110 ms → 40 ms	48 → 70 (+22)
3	C16	Виконано	5.4 s → 5.1 s	0 → 0	40 ms → 10 ms	70 → 71 (+1)
4	C6	Виконано	5.1 s → 4.7 s	0 → 0	10 ms → 60 ms	71 → 75 (+4)
5	C7	Виконано	4.7 s → 5.4 s	0 → 0	60 ms → 10 ms	75 → 76 (+1)
6	C4	Виконано	5.4 s → 4.9 s	0 → 0	10 ms → 20 ms	76 → 73 (-3)
7	C12	Виконано	4.9 s → 4.8 s	0 → 0	20 ms → 10 ms	73 → 73 (0)
8	C11	Виконано	4.8 s → 4.6 s	0 → 0	10 ms → 0 ms	73 → 75 (+2)
9	C10	Відхилено	4.6 s → 4.6 s	0 → 0	0 ms → 0 ms	75 → 75 (0)
10	C14	Відхилено	4.6 s → 4.6 s	0 → 0	0 ms → 0 ms	75 → 75 (0)
11	C13	Виконано	4.6 s → 4.4 s	0 → 0	0 ms → 10 ms	75 → 76 (+1)
12	C9	Виконано	4.4 s → 4.5 s	0 → 0	10 ms → 20 ms	76 → 76 (0)
13	C8	Відхилено	4.5 s → 4.5 s	0 → 0	20 ms → 20 ms	76 → 76 (0)
14	C1	Виконано	4.5 s → 4.5 s	0 → 0	20 ms → 10 ms	76 → 77 (+1)
15	C15	Виконано	4.5 s → 4.4 s	0 → 0	10 ms → 10 ms	77 → 78 (+1)
16	C17	Виконано	4.4 s → 4.5 s	0 → 0	10 ms → 10 ms	78 → 77 (-1)
17	C5	Виконано	4.5 s → 4.4 s	0 → 0	10 ms → 20 ms	77 → 78 (+1)
-	фінал	-	4,4 s	0	20 ms	78

Аналіз зафіксованих результатів (таблиця 2.4) демонструє таку динаміку цільових метрик. По-перше, відбулася зміна показника швидкодії LCP: на початку експерименту він становив 8.8 с, а після завершення всіх ітерацій скоротився до 4.4 с (зменшення на 50%). По-друге, стабілізувався макет сторінки: вихідний показник CLS, що дорівнював 1.004, був зведений до 0 після ітерації C3 і залишався незмінним під час усіх наступних втручань. По-третє, час блокування головного потоку браузера (TBT) скоротився зі 100 мс до 20 мс. Також під час роботи було зафіксовано регресії на етапах впровадження кандидатів C10, C14 та C8, унаслідок чого ці зміни були скасовані для збереження початкової цілісності веб-сайту.

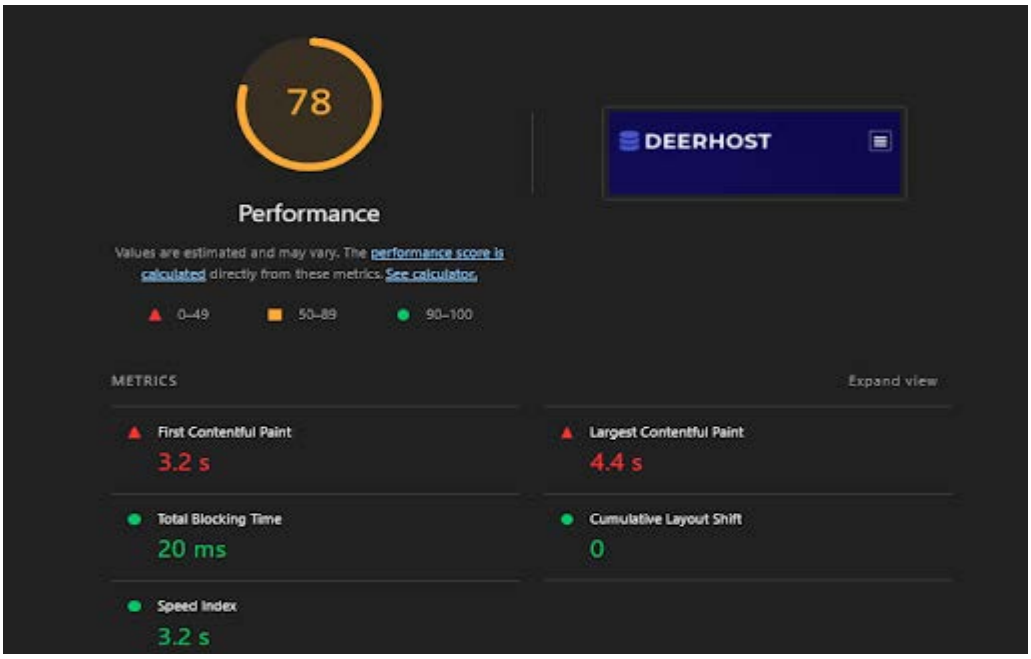

Рисунок 2.17 — Фінальні результати мобільного аудиту Google Lighthouse після впровадження змін

Згідно з фінальним звітом інструмента Google Lighthouse, показники стабільності макета (CLS = 0) та часу блокування (TBT = 20 ms) знаходяться у зеленій зоні. Загальна інтегральна оцінка продуктивності (Performance Score) становить 78 балів.

Висновки

1. Обґрунтовано вибір багатосторінкового статичного веб-сайту на базі шаблонного проєкту Deerhost як об'єкта дослідження, що забезпечило

контрольованість середовища та репрезентативність використання типових клієнтських технологій (HTML, CSS, JavaScript, jQuery). Проаналізовано архітектуру клієнтської частини, що дозволило встановити послідовність підключення ресурсів та побудову критичного шляху рендерингу.

2. Проведено базове лабораторне тестування контрольної сторінки за допомогою інструменту Google Lighthouse, яке зафіксувало критичні показники продуктивності для мобільного профілю (LCP = 8,8 с; CLS = 1,008; TBT = 100 мс). Ідентифіковано основні вузькі місця, серед яких домінуючими виявилися рендер-блокуючі таблиці стилів, відсутність явних розмірів зображень та неоптимізована доставка медіаресурсів.
3. Формалізовано метод поетапного покращення метрик Core Web Vitals, в основу якого покладено математичну модель пріоритезації технічних змін. Модель обчислює підсумковий пріоритет на базі трьох незалежних критеріїв: очікуваної користі (K), технічної складності (S) та ризику візуально-функціональної регресії (R).
4. Здійснено практичну реалізацію розробленого методу шляхом ітераційного впровадження 17 технічних кандидатів. Під час апробації було успішно відкочено зміни для кандидатів C10, C14 та C8, що на практиці підтвердило ефективність критерію ризику (R) для своєчасного виявлення та ізолювання втручань, які руйнують верстку або функціонал.
5. Зафіксовано суттєву позитивну динаміку цільових метрик після завершення всіх ітерацій: час відображення основного контенту (LCP) скоротився на 50% (до 4,4 с), показник візуальної нестабільності (CLS) був зведений до ідеального значення (0), а час блокування головного потоку браузера (TBT) зменшився до 20 мс. Загальна інтегральна оцінка продуктивності (Performance Score) досягла 78 балів, що емпірично доводить ефективність запропонованого алгоритму K-S-R.

РОЗДІЛ 3

ОЦІНКА ЕФЕКТИВНОСТІ ВПРОВАДЖЕНИХ РІШЕНЬ ТА РЕКОМЕНДАЦІЇ ВИКОРИСТАННЯ МЕТОДУ

3.1 Порівняльний аналіз метрик продуктивності та оцінка візуально-функціональної цілісності веб-сайту

Після завершення практичного етапу ітераційного впровадження технічних змін, критично важливим є проведення детального порівняльного аналізу отриманих результатів. Основною метою цього підрозділу є оцінка динаміки ключових показників продуктивності (LCP, CLS, TBT) на кожному кроці застосування методу K-S-R, а також підтвердження безпомилкової роботи та коректного відображення тестового веб-сайту.

Найбільш показовою метрикою швидкості завантаження та відображення основного контенту є Largest Contentful Paint (LCP). Динаміку зміни цього показника від базового стану до фінальної конфігурації проілюстровано на рисунку 3.1.

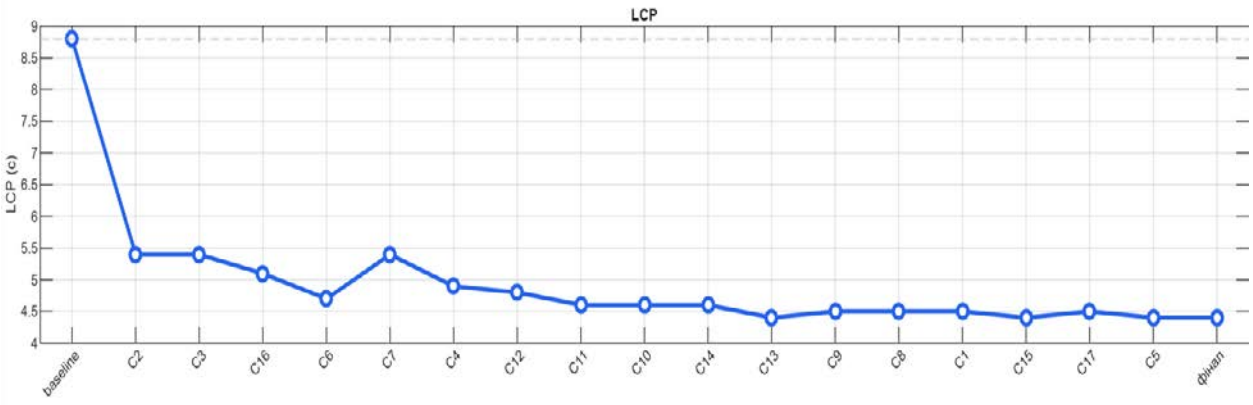

Рисунок 3.1 — Динаміка метрики LCP у процесі застосування методу K-S-R

Як видно з наведеного графіка, початкове (базове) значення LCP перебувало у критичній зоні та становило 8,8 с. Оскільки запропонований алгоритм пріоритезації вивів на перше місце кандидата C2 (покращення доставки зображень), найбільший стрибок продуктивності відбувся вже на першій ітерації — час завантаження різко скоротився до 5,4 с. Наступні кроки, зокрема оптимізація критичного шляху (C16, C6) та мережеві підказки (C7,

C4), забезпечили плавне, хоч і менш стрімке, покращення показника. Незначні коливання графіка на проміжних етапах пояснюються перерозподілом пріоритетів браузера під час завантаження ресурсів, проте загальний спадний тренд зберігся, що дозволило досягти фінального результату в 4,4 с (загальне прискорення рівно вдвічі).

Не менш важливим аспектом користувацького досвіду є стабільність макета сторінки, яка вимірюється метрикою Cumulative Layout Shift (CLS). Графік зміни цього показника наведено на рисунку 3.2.

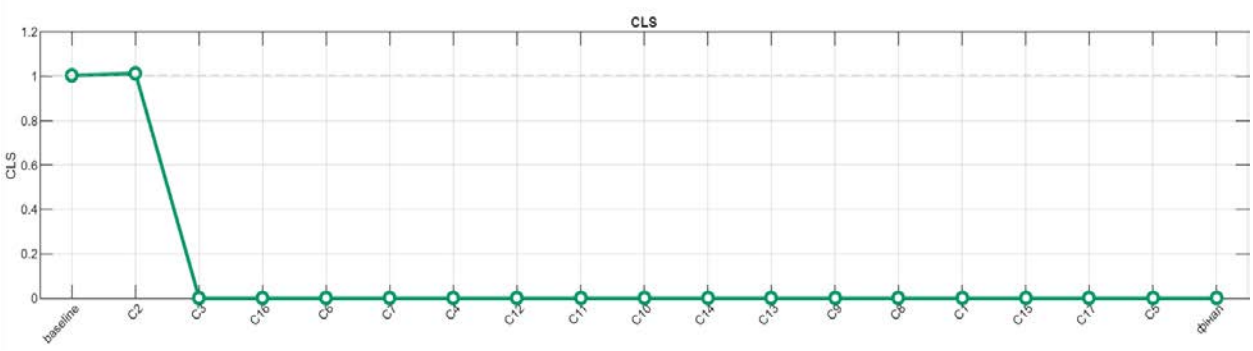

Рисунок 3.2 — Динаміка метрики CLS у процесі застосування методу K-S-R

Базовий стан веб-сайту характеризувався абсолютною візуальною нестабільністю (показник становив близько 1,0). На етапі впровадження кандидата C2 ситуація залишалася незмінною, оскільки ці зміни стосувалися виключно швидкості доставки файлів. Проте вже на наступній ітерації, показник різко впав до ідеального значення 0. Графік наочно демонструє фундаментальну перевагу керованого впровадження: протягом усіх наступних 15 ітерацій значення CLS жодного разу не відхилилося від нульової позначки. Це підтверджує, що оптимізації швидкодії, впроваджені на пізніших етапах, не конфліктували зі стабільністю верстки.

Оцінка впливу технічних змін на інтерактивність та розвантаження головного потоку браузера виконувалася за допомогою лабораторної метрики Total Blocking Time (TBT). Відповідний графік представлено на рисунку 3.3.

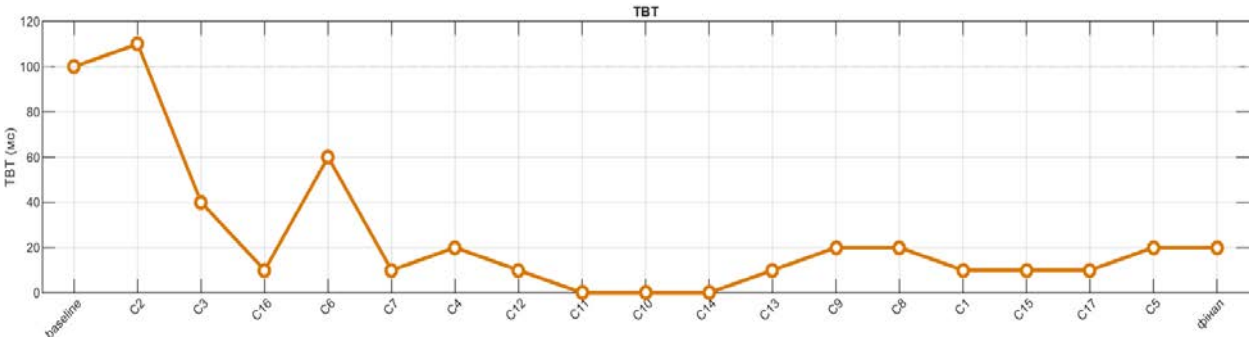

Рисунок 3.3 — Динаміка метрики TBT у процесі застосування методу K-S-R

На відміну від метрик візуального відображення, графік TBT демонструє помітну волатильність. Після початкового значення у 100 мс спостерігалося незначне зростання до 110 мс на етапі C2 (що є наслідком більш раннього парсингу HTML через прискорення завантаження), після чого відбулося стрімке падіння до 40 мс (C3) та 10 мс (C16). Локальний сплеск до 60 мс на етапі C6 пояснюється зміною ієрархії кешування та запитів документа, що тимчасово збільшило синхронну роботу парсера. Однак завдяки подальшій оптимізації виконання JavaScript (C12, C11), загальний тренд вдалося вирівняти: до фінального етапу метрику стабілізовано на рівні 20 мс, що є відмінним показником чуйності (покращення на 80%).

3.2 Оцінка ефективності методу пріоритезації K–S–R порівняно з автоматизованими рекомендаціями

Для об'єктивного доведення практичної цінності розробленого методу K–S–R доцільно порівняти його результативність із класичним лінійним підходом, який передбачає виконання рекомендацій автоматизованого аудиту строго у запропонованому інструментом порядку (від C1 до C17). Щоб коректно провести таке порівняння, було побудовано математичну модель альтернативного сценарію. Для цього використано фактичні значення зміни продуктивності, зафіксовані під час ітераційного впровадження кандидатів (див. рис. 2.17). У змодельованому лінійному сценарії ці зміни послідовно додавалися до базового показника (40 балів) саме в тому порядку, в якому інструмент Google Lighthouse вивів їх у своєму звіті. Порівняльну гістограму

динаміки Performance Score для фактичного застосування методу K–S–R (сині стовпці) та змодельованого лінійного підходу C1→C17 (сірі стовпці) наведено на рисунку 3.4.

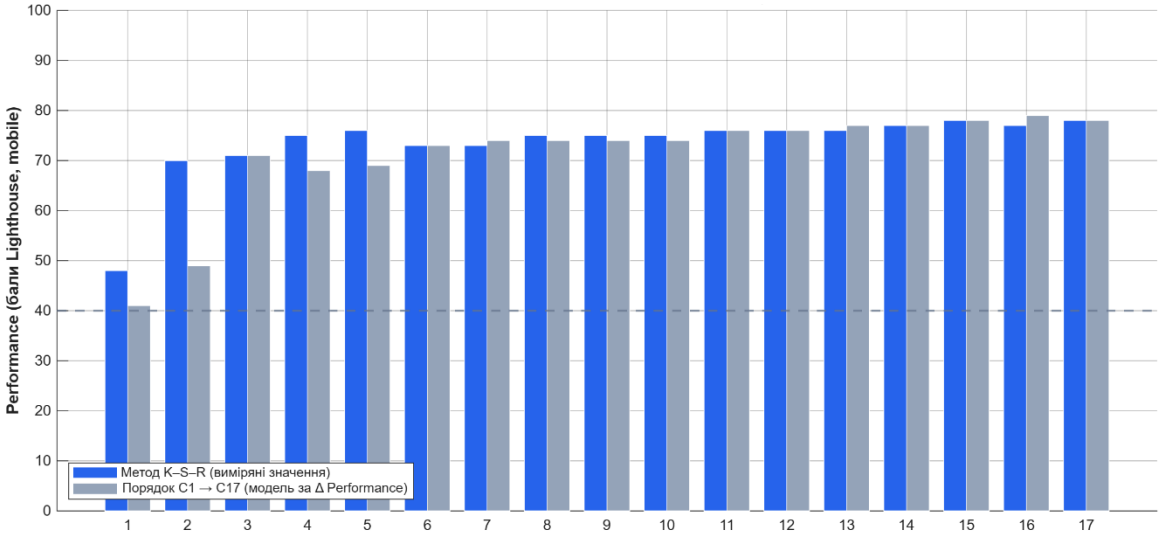

Рисунок 3.4 — Динаміка Performance Score: метод K–S–R у порівнянні з лінійним порядком C1→C17

Аналіз наведеної гістограми дозволяє чітко простежити ключову закономірність: метод K–S–R забезпечує значно швидший прогрес на початкових етапах роботи. Завдяки тому, що розроблений алгоритм першочергово виділяє кандидатів із найвищим співвідношенням користі до складності, інтегральний результат наростає набагато раніше. Як видно з графіка, вже на перших ітераціях метод K–S–R демонструє різкий стрибок продуктивності, дозволяючи системі швидко вийти на стабільно високий рівень оцінки.

Натомість лінійний порядок впровадження C1→C17 стартує значно повільніше. Це пояснюється тим, що автоматизовані звіти часто ставлять на перші місця завдання, які мають високу теоретичну економію ресурсів, але на практиці є надзвичайно трудомісткими або дають мінімальний фактичний приріст для конкретної архітектури веб-сайту. Проте, в міру накопичення впроваджених технічних змін, лінійна модель поступово долає відставання і з часом повністю наздоганяє показники оптимізованого алгоритму.

Фундаментальним висновком цього порівняння є те, що у фіналі (на 17-му кроці) обидва підходи досягають абсолютно однакового результату. Різниця між застосуванням методу K–S–R та неструктурованим виконанням рекомендацій полягає виключно в темпі досягнення мети, а не в кінцевій якості покращення. В умовах реального інженерного процесу саме цей темп є критичним фактором: запропонований метод дозволяє розробнику швидко досягти основної частки результату за мінімальний час, враховуючи при цьому відсутність регресій, що робить процес технічного покращення веб-сайтів раціональним.

Висновки

1. Здійснено загальний аналіз динаміки ключових метрик вебпродуктивності (LCP, CLS, TBT) у результаті застосування розробленого методу K-S-R. Зафіксовано суттєве комплексне покращення швидкодії тестового ресурсу впродовж усіх етапів покращення.
2. Проведено порівняльну оцінку результативності алгоритму K-S-R із класичним лінійним виконанням рекомендацій автоматизованого аудиту Google Lighthouse. Доведено, що розроблений метод забезпечує значно швидший темп зростання інтегральної продуктивності на початкових етапах робіт.
3. Встановлено, що за однакового кінцевого результату обох підходів, застосування методу K-S-R сприяє раціональнішому розподілу робочого часу. Це підтверджує доцільність використання розробленої моделі для структурування результатів автоматизованих аудитів з метою покращення процесу розробки та зниження технічних ризиків.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1. Проведено аналіз теоретичних засад оцінки вебпродуктивності, досліджено механіку ключових метрик стандарту Core Web Vitals (LCP, CLS, INP) та оглянуто сучасні інструментальні засоби їх вимірювання.
2. Виконано аналіз існуючих продуктових та інженерних моделей пріоритезації технічних змін. Обґрунтовано необхідність створення спеціалізованого підходу.
3. Розроблено метод поетапного покращення показників Core Web Vitals, що базується на інтегральній математичній моделі з урахуванням трьох критеріїв: очікуваної користі (K), технічної складності впровадження (S) та ризику (R).
4. Проведено базове тестування продуктивності клієнтської частини багатосторінкового веб-сайту. Зафіксовано початкові показники швидкодії (LCP = 8,8 с, CLS = 1,008, TBT = 100 мс) та виявлено основні технічні вузькі місця.
5. Здійснено практичну реалізацію розробленого методу шляхом ітераційного впровадження 17 технічних кандидатів. Завдяки застосуванню критерію ризику (R) було успішно виявлено та відкочено деструктивні зміни (кандидати C10, C14, C8), що дозволило зберегти коректне відображення макета.
6. Проведено оцінку ефективності впроваджених рішень, що продемонструвала:
   - Скорочення часу відображення основного контенту (LCP) на 50 % (із 8,8 с до 4,4 с).
   - Зниження показника візуальної нестабільності (CLS) до ідеального значення 0.
   - Зменшення часу блокування потоку (TBT) зі 100 мс до 20 мс.

- Зростання загальної інтегральної оцінки продуктивності (Performance Score) до 78 балів.

7. Виконано порівняльний аналіз ефективності розробленого методу K-S-R та лінійного підходу на основі автоматизованих рекомендацій Google Lighthouse. Доведено, що запропонований алгоритм забезпечує значно швидший темп зростання продуктивності на початкових етапах робіт.
8. Розроблений метод є дієвим підходом, який дозволяє раціональніше розподіляти робочий час розробника, системно підходити до покращення показників Core Web Vitals та ефективно мінімізувати ризики порушення візуальної стабільності інтерфейсу веб-сайтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. User-centric performance metrics [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/user-centric-performance-metrics (дата звернення: 21.04.2026).
2. Web Performance Working Group [Електронний ресурс] / W3C. – Режим доступу: https://www.w3.org/webperf/ (дата звернення: 21.04.2026).
3. Introducing Web Vitals: Essential metrics for a healthy site [Електронний ресурс] / The Chromium Blog. – 2020. – Режим доступу: https://blog.chromium.org/2020/05/introducing-web-vitals-essential-metrics.html (звернено 22 квітня 2026 р.).
4. More time, tools, and details on Page Experience [Електронний ресурс] / Google Search Central Blog. – 2021. – Режим доступу: https://developers.google.com/search/blog/2021/04/more-details-page-experience (переглянуто: 23.04.2026).
5. INP is officially a Core Web Vital [Електронний ресурс] / Web.dev Blog. – Режим доступу: https://web.dev/blog/inp-cwv (дата звернення: 23.04.2026).
6. Largest Contentful Paint (LCP) [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/lcp (звернено 27.04.2026).
7. Optimize Largest Contentful Paint [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/optimize-lcp (дата звернення — 28.04.2026).
8. Cumulative Layout Shift (CLS) [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/cls (переглянуто 28 квітня 2026 р.).
9. Interaction to Next Paint (INP) [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/inp (дата звернення: 30.04.2026).
10. Lighthouse overview [Електронний ресурс] / Chrome for Developers. – Режим доступу: https://developer.chrome.com/docs/lighthouse/overview (звернено 30.04.2026).

11. Lighthouse performance scoring [Електронний ресурс] / Chrome for Developers. – Режим доступу: https://developer.chrome.com/docs/lighthouse/performance/performance-scoring (дата звернення: 04.05.2026).
12. Total Blocking Time (TBT) [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/tbt (переглянуто: 05.05.2026).
13. Performance features reference [Електронний ресурс] / Chrome DevTools Documentation. – Режим доступу: https://developer.chrome.com/docs/devtools/performance/overview (дата звернення: 05.05.2026).
14. Getting Started with WebPageTest [Електронний ресурс] / WebPageTest Documentation. – Режим доступу: https://docs.webpagetest.org/getting-started/ (звернено 7 травня 2026 р.).
15. About PageSpeed Insights API [Електронний ресурс] / Google for Developers. – Режим доступу: https://developers.google.com/speed/docs/insights/v5/about (дата звернення: 11.05.2026).
16. Core Web Vitals report in Search Console [Електронний ресурс] / Google Search Central Help. – Режим доступу: https://support.google.com/webmasters/answer/9205520 (переглянуто 11.05.2026).
17. Tools to measure Core Web Vitals [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/vitals-tools (звернення: 13.05.2026).
18. Optimize Cumulative Layout Shift [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/optimize-cls (дата звернення: 16.05.2026).
19. Optimize Interaction to Next Paint [Електронний ресурс] / Web.dev. – Режим доступу: https://web.dev/articles/optimize-inp (звернено 16 травня 2026 р.).

20. RICE Scoring Model [Електронний ресурс] / ProductPlan Glossary. – Режим доступу: https://www.productplan.com/glossary/rice-scoring-model (дата звернення: 18.05.2026).
21. ICE Score Prioritization Method [Електронний ресурс] / SaaS Funnel Lab. – Режим доступу: https://www.saasfunnellab.com/essay/ice-score-prioritization-method/ (переглянуто: 21.05.2026).
22. Weighted Shortest Job First (WSJF) [Електронний ресурс] / SAFe Framework. – Режим доступу: https://framework.scaledagile.com/wsjf/ (дата звернення: 21.05.2026).
23. Prioritization Matrices [Електронний ресурс] / Nielsen Norman Group. – Режим доступу: https://www.nngroup.com/articles/prioritization-matrices/ (звернено 24.05.2026).
24. PIE Framework [Електронний ресурс] / Growth Method. – Режим доступу: https://growthmethod.com/pie-framework/ (дата звернення: 26.05.2026).