Державний університет “Одеська політехніка”

(повне найменування вищого навчального закладу)

Інститут штучного інтелекту та робототехніки

(повне найменування інституту, назва факультету)

Комп'ютерні системи

(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

Кваліфікаційна робота магістра

(освітньо-кваліфікаційний рівень)

на тему Дослідження методів оптимізації для підвищення швидкості завантаження веб-сторінки

Виконав: студент 5 курсу, групи УК-162
напряму підготовки (спеціальності)
123 - “Комп’ютерна інженерія”
(шифр і назва напряму підготовки, спеціальності)
Журавльов О.О.
(прізвище та ініціали)
Керівник Ситніков В.С.
(прізвище та ініціали)
Рецензент Стрельцов О.В.
(прізвище та ініціали)

Одеса – 2021 року

ЗМІСТ

Вступ

У сучасному світі багато сервісів та послуг знаходяться у всесвітній мережі інтернет. Багато підприємств, від малих до великих, для надання своїх послуг користувачам, створюють свої веб-сторінки, у зв'язку з цим зростає і конкуренція за перші місця в популярних пошукових системах. Факторів ранжування безліч, одним з них є швидкість завантаження веб-сторінки, тому що виходячи з даних дослідження компанії Google 53%[1] користувачів залишають сайт у випадку, якщо завантаження сторінки перевищує 3 секунди. Також за результатами спільного дослідження аналітичного агентства SOASTA конверсія безпосередньо залежить від того, як швидко у користувача завантажується цільовий сайт. Ті веб-сторінки, які частіше приводили до конверсії, були в середньому на 26% швидше за інших. В агентстві зробили висновок, що швидкі сторінки роблять користування комфортним, створюють умови для подальших дій: замовлення, покупки, дзвінка. Існують різні методи оптимізації та технології, які допомагають зменшити параметр часу завантаження веб-додатку, тим самим підвищуючи комфорт його використання. Найчастіше для підбору необхідного методу або технології розробникі витрачають безліч годин на визначення причини повільного завантаження сайту, а також на дослідження існуючих у цій сфері рішень. Допомогу у виборі надають деякі веб-ресурси, такі як: Google Lighthouse, GTmetrix, які визначають наскільки довго здійснюється завантаження цільового сайту та виявляють можливу причину. Але ці методи не завжди допомагають виявити конкретну причину та дають тільки статистичні дані про веб-сторінку. Виходячи з вищенаведених даних можна зробити висновок, що застосування та дослідження методів оптимізації сайту для підвищення швидкості завантаження веб-додатку - є дуже актуальною темою.

Мета дослідження - дослідити процес оптимізації веб-сторінки для підвищення швидкості її завантаження.

Предмет дослідження - особливості та аспекти методів оптимізації продуктивності веб-сторінки.

Задачі дослідження. Для досягнення поставленої мети визначені наступні задачі, які необхідно вирішити:

1) дослідити існуючі рішення в даній області;
2) дослідити параметри відображення продуктивності веб-ресурсів та методи їх оптимізації;
3) скласти алгоритм перевірки веб-ресурсу з використанням досліджених метрик;
4) перевірка ефективності методів оптимізації.

Інноваційність рішення полягає у комбінуванні методів оптимізації з клієнтської та серверної сторони веб-ресурсу.

Практична цінність - цю оптимізаційну модель можуть використовувати більшість бізнесів у інтернеті, що надасть їм збільшення користувачів які вживатимуть послуги що надає даний бізнес.

По результатам дослідження була написана публікація

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз існуючих рішень сканування web-ресурсів

У сучасному світі безліч веб-ресурсів мають бути оптимізовані під ті пристрої, якими людина користується щодня, а це планшети, мобільні телефони та ноутбуки. Така мобільна техніка часто використовує тільки мобільний трафік, для отримання доступу до глобальної мережі Інтернет, або до різних даних, відповідно, розмір завантажуваного веб-ресурсу повинен бути максимально зменшеним, або мати спеціальні “полегшені” версії.

Процес оптимізації для розробників найчастіше повільна процедура, котра вимагає багато часу та зусиль на відстеження первинної причини малої продуктивності веб-сайту. Саме тому, для аналізу продуктивності веб-ресурсу, розробники використовують різні онлайн-сервіси з обробки даних завантаження цих ресурсів. Такі онлайн-сервіси надають спеціальні метрики, котрі дозволяють розробникам знайти причину повільного завантаження сайту та виправити її. Існує декілька найпопулярніших таких сервісів Google Lighthouse, GTmetrix, та Loader.io. Найчастіше розробники використовують різні комбінації таких веб-сервісів, тому що жоден з перерахованих вище не може надати комплексну статистику всього, що відбувається на веб-сайті. Тому для того, щоб з'ясувати, яким чином можливе компонування та покращення сервісу зі сканування сайтів, з метою оптимізації швидкості завантаження веб-ресурсу, необхідно проаналізувати найпотужніші та найпопулярніші веб-сервіси з таким же призначенням.

1.2 Методи вимірювання продуктивності

Перш ніж приступати до будь-яких оптимізаційних процесів, необхідно визначити конкретні місця веб-сервісу, які цього потребують. Щоб знайти такі місця, потрібні виміри продуктивності. Щоб зрозуміти які саме метрики нам потрібні для їх комбінування треба проаналізувати існуючі рішення, котрі використовуються у сучасних веб-сервісах. Нижче наведено варіант використання, який відстежує подію First Contentful Paint.

```
const performanceObserver = new PerformanceObserver ((performanceEntryList) => {
  const performanceEntries = performanceEntryList.getEntries ();

  for (const performanceEntry of performanceEntries) {
    if (performanceEntry.name === 'first-contentful-paint') {
      const firstContentfulPaint = performanceEntry.startTime;

      console.log ('FCP is', firstContentfulPaint);
    }
  }
});
```

performanceObserver.observe ({type: 'paint', buffered: true}); - Параметр buffered дозволяє отримати інформацію про події, що відбулися до етапу ініціалізації спостерігача.

Далі йде збір показників, які відображають сприйняття ефективності. Бібліотека зосереджується на зборі лише тих показників, які відображають реальне сприйняття ефективності проекту. Ми не збираємо статичні показники, які можна переглянути в інструментах розробника, показники, які погано чи не завжди корелюють із фактичною продуктивністю, не збираємо нічого, що не

стане в нагоді в подальшій діагностиці та оптимізації проекту. Існує безліч показників які відображають продуктивність веб-ресурсу:

- F
CP — First Contentful Paint;
- F
MP — First Meaningful Paint;
- L
CP — Largest Contentful Paint;
- C
LS — Cumulative Layout Shift;
- F
ID — First Input Delay;
- F
CI — First CPU Idle;
- T
TI — Time To Interactive;
- T
BT — Total Blocking Time;
- V
C — Visually Complete;
- S
P — Speed Index и тд.

Сучасні показники ефективності мають певний набір атрибутів, завдяки яким їх можна розділити на категорії. Наприклад, середовище використання. Деякі метрики призначені для використання у виробничому середовищі, тоді як

інші призначені для моніторингу продуктивності в CI/CD. Причин такого поділу декілька :

- Перша пов'язана з відсутністю можливості вимірювань як таких. Наприклад, вимірювання показника First Input Delay за межами виробничого середовища непрактично.
- Інша причина полягає в тому, що алгоритм роботи деяких метрик є трудомістким у обчислювальному відношенні. Тому використання тих чи інших показників у конкретному середовищі може бути недоцільним, а іноді й зовсім неприйнятним.

Єдине середовище, в якому ми працюємо з даними про продуктивність від реальних користувачів, — це синтетичне середовище. Тому таке середовище можемо визначити як клієнтське, а усі інші - синтетичні. Розділивши показники за середовищем використання, ми бачимо таку картину:

Клієнтська середа	Синтетична середа
• Largest Contentful Paint (LCP) • Cumulative Layout Shift (CLS) • Firs Meaningful Paint (FMP) • First Contentful paint (FCP) • First Input Delay (FID)	• Total Blocking Time (TBT) • Time To Interactive (TTI) • Visually Complete (CV) • First CPu Idle (FCI) • Speed Index (SI)

Середовища використання та показників продуктивності

Це не суворий поділ, і деякі показники можна використовувати в обох середовищах одночасно. Нас цікавить перша категорія показників.

1.3 Використання метрик у Google Lighthouse

Lighthouse — це проект із відкритим кодом, який веде спеціальна команда розробників Google Chrome. За останні пару років Lighthouse став стандартним безкоштовним інструментом аналізу ефективності веб-сайтів. Lighthouse фіксує різноманітні показники у своєму опитуванні про ефективність сторінки, які вимірюють те, що користувач бачить і відчуває під час взаємодії зі сторінкою. Для визначення загального показника ефективності використовуються шість показників:

- Time to Interactive (TTI, Час завантаження до взаємодії);
- Speed Index (Індекс швидкості завантаження);
- First Contentful Paint (FCP, Час завантаження першого контенту);
- First CPU Idle (Час закінчення роботи ЦП);
- First Meaningful Paint (FMP, Час завантаження достатньої частини контенту);
- Estimated Input Latency (Приблизний час затримки під час введення).

Кожен із цих показників оцінюється за шкалою від 0 до 100. Оцінка здійснюється шляхом отримання 75-ти та 95-ти перцентилів для мобільних

сторінок із HTTP-архіву та за допомогою функції log normal. Дотримуючись цього алгоритму та дивлячись на дані, які використовуються для обчислення TTI, можна побачити, що якби сторінка стала «інтерактивною», придатною для взаємодії з користувачем за 2,1 секунди, тоді TTI був би 92/100. Після розрахунку кожного з показників йому присвоюється певна вага, яка використовується як модифікатор при розрахунку загального показника. Нижче (таб. 1.1) наведено коефіцієнти ваги різних показників.

Метрика	Вага
Time to Interactive (TTI)	5
Speed Index	4
First Contentful Paint	3
First CPU Idle	2
First Meaningful Paint	1
Estimated Input Latency	0

Таблиця 1.1 - Вага основних показників швидкодії веб-ресурсу

Вагові показники вказують на вплив кожного параметру на роботу сторінки мобільним користувачем. Команда розробників LightHouse розробила калькулятор, в якому э можливість проаналізувати вплив кожного показника на результат оцінки сканованого веб-ресурсу (рис 1.1).

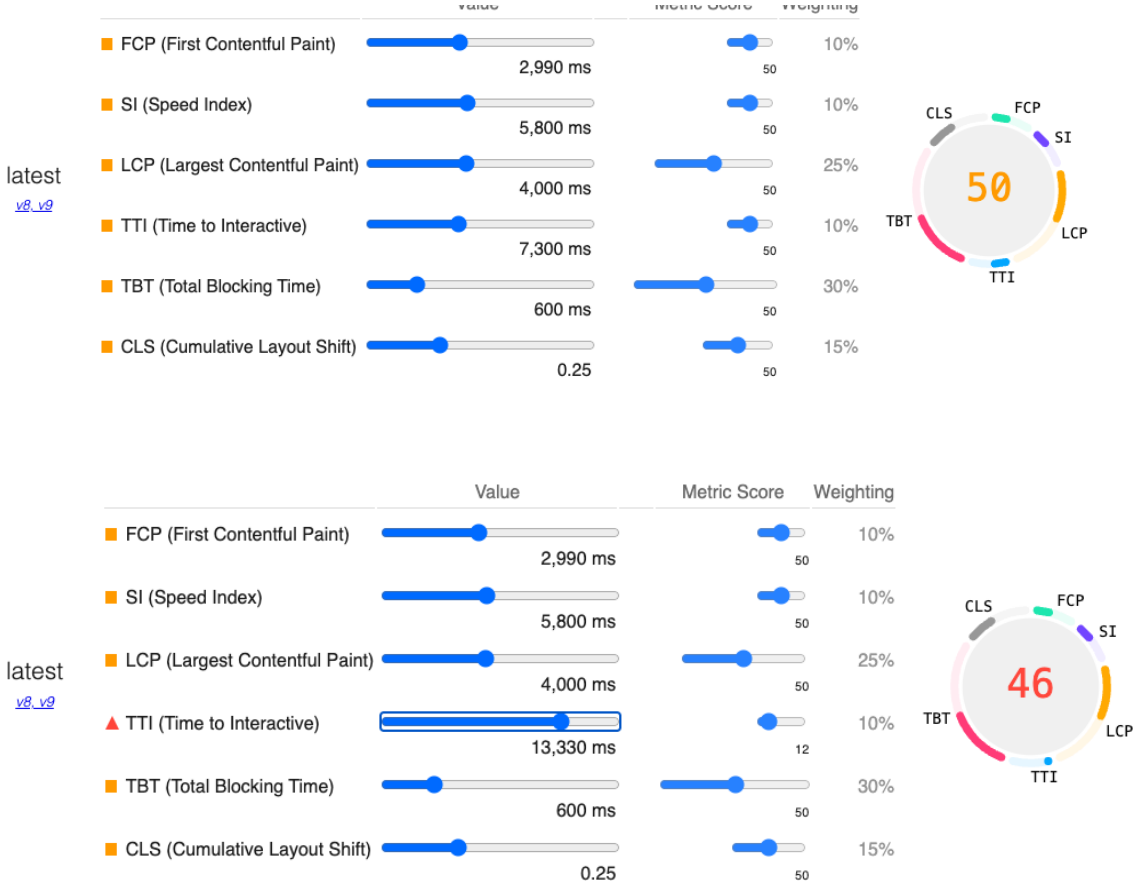


Рисунок 1.1 - Вплив параметру TTI на оцінку сайту

Якщо роздивитися даний приклад (рис 1.1), то маємо зробити висновок що збільшення параметру TTI (нестандартизована метрика «прогресу» веб-продуктивності, яка визначається як момент часу, коли завершилося останнє довге завдання, після чого 5 секунд бездіяльності мережі та основного потоку.) напряму впливає на погіршення продуктивності, якщо у першому випадку параметр був рівен 7,3 секунди продуктивність веб-ресурсу складала 50 відсотків, у випадку збільшення параметру TTI продуктивність впала до 46 відсотків.

У результаті ми можемо зробити висновок, що показник TTI має найбільший вплив на підсумкову оцінку сайту. Підсумок полягає в тому, що для того, щоб отримати високу оцінку PageSpeed, сторінка повинна

продемонструвати пристойний TTI. Розглянутий вище приклад оцінки продуктивності веб-ресурсу має низку своїх переваг і недоліків. Переваги:

- дозволяє миттєво оцінити загальні показники продуктивності сайту;
- алгоритм перевірки є простим і дозволяє зробити статистику серед інших веб-ресурсів;
- зрозумілий та доступний інтерфейс;
- надає поради щодо оптимізації сканованого веб ресурсу.

Недоліки:

- відсутність безперервного відстеження продуктивності;
- відсутність можливості інтегрування у веб-ресурс та створення тестового оточення;
- розкид у метриках, що надаються, через неізольоване оточення;
- немає вибору сервера, з якого відбувається сканування;

Отже цей інструмент візначений для аналізу сайту зі сторони фронтенд частини та має функції для порівняння продуктивності різноманітних веб - ресурсів, але не надає можливості порівняти продуктивність сайту до оптимізації та після. Тим самим розробник або клієнт має шукати ще один інструмент, що може постійно слідкувати за швидкістю веб-ресурса.

1.4 GTmetrix

GTmetrix[2] — це інструмент порівняльного аналізу веб-сайтів, розроблений канадською компанією GT.net. Інструмент GTmetrix дозволяє користувачам легко дізнатися, наскільки швидко працює чи не працює їхній сайт, де він працює добре, а де виникають проблеми. GTmetrix заслужив репутацію найпопулярнішого інструменту порівняльного аналізу в Інтернеті на сьогоднішній день. Інструмент використовується, щоб допомогти клієнтам дізнатися про їхній веб-ресурс, щоб вони могли вносити покращення. GTmetrix відомий як простий у використанні та розумінні, що робить його відмінним як для новачків, так і для досвідчених розробників. Базові функції GTMetrix які доступні для безкоштовного використання це параметри PageSpeed, YSlow та інструмент аналізу Waterfall.

Параметри PageSpeed і YSlow пропонують два різні способи визначення структури сайту та механізмів, що впливають на його швидкість. Результати, отримані за цими параметрами, містять рекомендації, дозволяють збільшити продуктивність сайту, наприклад, використовувати кешування, додавання expires-заголовків, мінімізувати асети, дозволити gzip-зжатие та багато іншого. Інструменти, які використовують GTMetrix.

Waterfall[3] – це корисний інструмент, який відкриває розробку збільшує продуктивність веб-ресурсу. Він схожий на браузерні інструменти типу Firebug Net Panel. Кожен асет вашого сайту запитується, передається і відображається. Кожна шкала у Waterfall показує кроки, включені в обробку асетів, і час, який знадобиться для їх виконання. Асети що використовує Waterfall:

- D

NS Lookup: запит до DNS

- C

onnecting: время, затрачиваемое на установку зв'язку

- B

locking: час, який вимагає браузеру для установки зв'язку

- S

ending: час посылки запита

- W

aiting: час очікування відкликання з сайту (до пересилки першого байта)

- R

eceiving: час, затрачиваемое на загрузку контенту

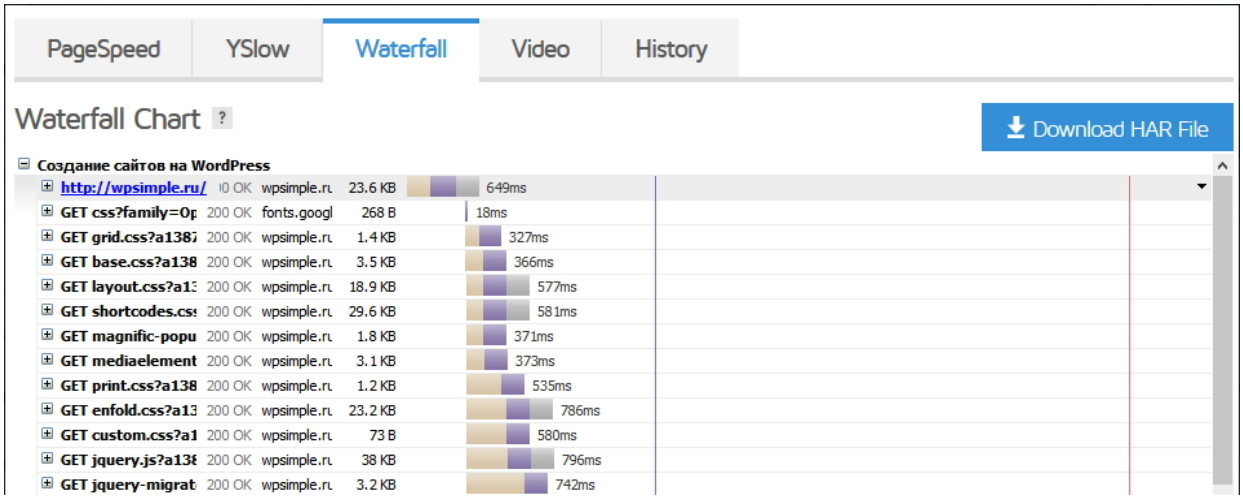


Рисунок 1.2 - Відображення параметрів інструменту Waterfall

Використовуючи інформацію на рис. 1.2, можемо зрозуміти, що відбувається на скануємому веб-ресурсі. Насамперед, потрібно подивитися на голубу лінію вгору, яка представляє собою точку завантаження DOM. Червона лінія представляє час завантаження сторінки. Час завантаження першого байта (TTFB) також є важливим індикатором швидкості дії сервера. Якщо параметр TTFB не змінюється після оптимізації веб-сторінки на стороні фронтенду, тоді можемо зробити висновок, що проблема у сервері.

Цей інструмент також має ряд своїх недоліків:

- немає варіативного тестування сторінок;
- немає порад для оптимізації процесу візуалізації;
- основні інструменти з’являються тільки після оформлення підписки на сервіс.

Проаналізувавши цей веб-ресурс можна зробити висновок, що даний інструмент не універсальний та підходить тільки для первинної оптимізації швидкості завантаження сайту. Для подальшого аналізу знадобиться ще кілька інструментів, які надають більш глибокий аналіз сайту та серверу.

1.5 Loader.io

Однією з найважливіших частин веб-сервісу є сервер. Серверу необхідно приймати безліч запитів до нього і відповідати на них, для перевірки навантаження на сервер існує ще один веб сервіс Loader.io від SendGrid - це повністю керована програма для моніторингу та оцінки даних, створена командою розробників, щоб усунути головні проблеми, що пов’язані із постійною перевіркою та входом на низку платформ для оцінки продуктивності сервера та API. Loader[4] - це повністю централізований веб-сервіс, який збирає всі потоки вашого сервера в одному місці та створює індивідуальні звіти на основі їхньої продуктивності. Loader також визначає потенційні загрози безпеки на скануємому веб-сайті, програмах, продуктивності сервера, і ви можете отримувати сповіщення про те, коли ці загрози близькі до появи. Loader.io працює застосовуючи симулювання реальних користувачів, їх дії та вплив на ваш веб-сайт, даний метод допомагає фокусувати увагу не тільки на коді веб-ресурсу,

а і на його вузькі місця загалом. Це дає більш реалістичний результат, ніж просто аналіз даних параметра TTI та інших параметрів, які надають сервіси розглянуті рані. Але цей сервіс не є інструкцією, що показує, на чому саме повинні зосередитися розробники для підвищення продуктивності.

Даний веб-сервіс виявить проблеми з маштабуванням, але розробникам необхідно буде самим знайти рішення щодо оптимізації цього параметра. Параметри, за якими здійснюється оцінка швидкодії веб-ресурсу (рис 1.3);

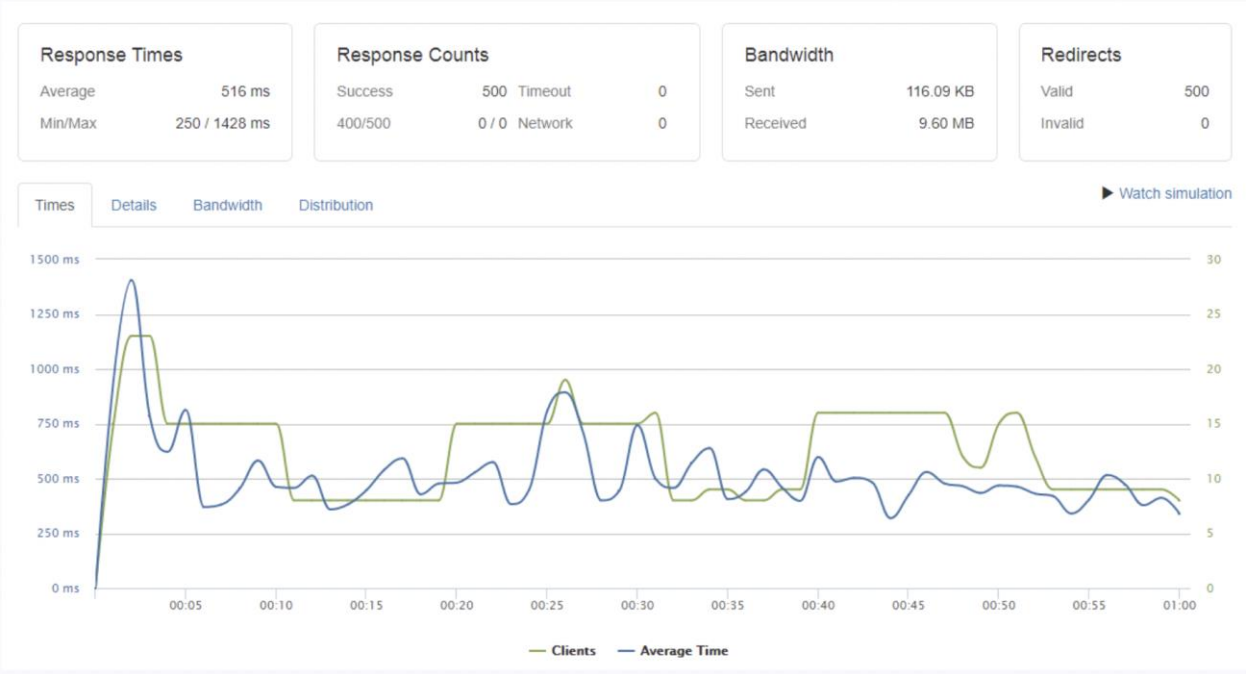


Рисунок 1.3 - відображення графіку метрики Loader.io

- Response time - загальна кількість часу, необхідного для відповіді на запит серверу;
- Response Counts - кількість запитів до серверу;
- Bandwidth Sent - розмір переданих сервером даних;

- Redirects - кількість перенаправлених відвідувачів на потрібний хостинг.

Основним мінусом даного сервісу являється те, що він має вузьку спеціалізацію та надає можливість тільки на тестування завантаженості серверу.

1.6 Аналіз існуючих методів оптимізації

На даному етапі існує безліч способів оптимізації веб-сайтів та в основному вони поділяються на 2 основних частини:

1. серверна частина;
2. клієнтська частина.

1.6.1 Методи оптимізації зі сторони серверу

На сервері в цілому покращується параметр Time To First Byte (TTFB). Цей параметр показує час від початку запиту до першого байта даних від сервера. На початковому етапі час витрачається на:

- затримку мережі;
- отримання IP від сервера DNS;
- встановлення з'єднання.

TTFB складається з часу на:

- передачу пакета даних запиту;
- обробку запиту веб-додатком;
- генерацію відповіді (додавання заголовків, стиснення).

Найбільше цей показник впливає швидкість відгуку на дію користувача. Як приклад – перехід на іншу сторінку, відправлення форми, оформлення замовлення. Зменшити час запиту можна за допомогою налаштування веб-серверу. Розглянемо основні моменти на прикладі Nginx.

Налаштування з'єднання. Для початку необхідно задати кількість "працівників" Nginx. Кожен робочий процес Nginx[10] здатний обробляти безліч сполук і прив'язується до фізичних ядер процесора. Якщо точно знати, скільки ядер у на придбаному сервері, можна задати їх кількість самостійно, або довіритися Nginx. Крім цього необхідно задати кількість з'єднань. Визначення кількості з'єднань на один робочий процес, в межах від 1024 до 4096.

Налаштування запитів. Щоб веб-сервер міг обробляти максимальну кількість запитів, необхідно задіяти директиву multi_accept, вимкнену за замовчуванням: multi_accept on;(Робочі процеси прийматимуть усі з'єднання). Примітно, що функція буде корисною лише за умови великої кількості запитів одночасно. Якщо ж запитів не так багато, є сенс оптимізувати робочі процеси, щоб вони не працювали в холосту команда accept_mutex on надае змогу робочим процесам приймати з'єднання по черзі. Покращення TTFB та часу відгуку сервера безпосередньо залежить від директив tcp_nodelay та tcp_nopush. Якщо не надто вдаватися до подробиць, то обидві функції дозволяють відключити деякі особливості TCP, які були актуальні в 90х, коли Інтернет тільки набирав обертів, але не мають сенсу в сучасних умовах. Перша директива tcp_nodelay надсилає

дані, як тільки вони будуть доступні (минає алгоритм Нейгла). А друга tcp_nopush дає можливість надсилати заголовок відповіді (веб-сторінки) та початок файлу, очікуючи заповнення пакета (тобто включає tcp_cork). Отже, браузер зможе почати відображення веб-сторінки раніше.

На перший погляд, функції суперечать одна одній. Тому директива tcp_nopush має використовуватися разом із sendfile. І тут пакети будуть заповнені до відправки, т.к. директива працює набагато швидше та оптимальніше, ніж метод read+write. Після того, як пакет заповнено, Nginx автоматично відключає tcp_nopush, а tcp_nodelay змушує сокет надіслати дані. Тому комбінація всіх трьох директив знижує навантаження на мережу і прискорює відправлення файлів.

Налаштування буферів [11] - ще одна важлива оптимізація торкається розміру буферів - якщо вони занадто маленькі, то Nginx буде часто звертатися до дисків, занадто великі - швидко заповниться оперативна пам'ять. Для цього потрібно налаштувати чотири директиви. client_body_buffer_size та client_header_buffer_size, що задають розмір буфера для читання тіла та заголовка запиту клієнта відповідно. client_max_body_size задає максимальний розмір запиту клієнта, а large_client_header_buffers задає максимальне число та розмір буферів для читання великих заголовків запитів.

Оптимальні параметри буферів виглядатимуть так:

- client_body_buffer_size 10K;
- client_header_buffer_size 1k;
- client_max_body_size 8m;
- large_client_header_buffers 2 1k;

- Розмір буфера 10 КБ на тіло запиту, 1 КБ на заголовок, 8 МБ на сам запит і 2 буфери для читання великих заголовків.

Кешування [12] - це параметр, що значно покращить час відгуку сервера. Nginx здатний відправляти запит на кешування рідко змінних даних, які часто використовуються, на стороні клієнта. Команда open_file_cache задає максимальну кількість файлів, інформація про які зберігатиметься, та час зберігання. open_file_cache_valid визначає час, після якого потрібно перевірити актуальність інформації, open_file_cache_min_uses визначає мінімальну кількість звернень до файлу з боку клієнтів, а open_file_cache_errors включає кешування помилок пошуку файлів.

Логування - це ще одна функція, яка може трохи знизити продуктивність всього сервера і, відповідно, час відгуку і TTFB. Так що найкращим рішенням буде відключити основний лог, а зберігати інформацію лише про критичні помилки.

Стиснення Gzip[13]. Корисність Gzip важко перебільшити. Стиснення дозволяє значно зменшити трафік та розвантажити канал. Але в нього є і зворотний бік — для компресії потрібен час. Так що для покращення TTFB та часу відгуку сервера його доведеться відключити.

На даному етапі ми не можемо рекомендувати відключення Gzip, оскільки стиснення покращує Time To Last Byte[14], тобто час, необхідний для повного завантаження сторінки. А це здебільшого важливіший параметр.

На поліпшення TTFB і часу відгуку сервера істотно вплине масштабне використання HTTP/2, який містить вбудовані способи компресії заголовків і мультиплексування. Так що в майбутньому, можливо, відключення Gzip буде не таким помітним як зараз.

Перевагами серверної оптимізації є:

- з

меншення файлів що завантажуються,

- з

меншенн часу відповіді серверу на запити клієнта,

- з

меншення навантаження на основні процеси, що підвищує стабільність роботи веб-ресурсу.

Основним недоліком тільки серверної оптимізації являється те, що хоча й стабільність роботи серверу буде підвищена та основні дані можуть завантажуватися за менший час, але відвідувач може не побачити цих змін якщо на клієнтській стороні будуть присутні довгі анімації, великий вміст не потрібного для завантаження контенту.

1.6.2 Методи оптимізації на клієнтській стороні

Проаналізувавши метрики які вказують на проблеми, що знижують продуктивність та швидкість завантаження веб-сайту проаналізуємо існуючі способи оптимізації клієнтської частини веб-сайту

Розглянемо способи оптимізації найпоширеніших випадків зміщення макета:

- О

птимізувати всі сторонні скрипти,

- П

рискорити JavaScript – розбити тривалі завдання на дрібніші. У проміжках між обробкою коротких завдань браузер встигне оптимізувати запит користувача,

- М

інімізувати кількість поліфілів, що не використовуються;

- Оптимізувати код CSS - для покращення FID рекомендується мінімізувати, стиснути і видалити CSS, що не використовується.
- Запускати довготривалі задачі поза основного потоку - Це має утримувати основний потік у режимі очікування і, отже, збільшувати затримку першого введення. Для цього ви можете передати дані у Web Worker.
- Налаштувати завантаження лише коду, який необхідний для первинної роботи.

Перевагою такої оптимізації являється підвищення користувацького досвіду при використовуванні сайту.

1.7 Розроблення та опис веб-додатку, що допомагає оптимізувати швидкість завантаження веб-ресурсу

Виходячи з технічного завдання і результатів аналізу подібних рішень в роботі буде розроблен алгоритм роботи веб-додатку, що сканує, та пропонує комбінований процес оптимізації цільового веб-ресурсу. Сканування веб-ресурсу складається з 3 х етапів:

1. Сканування візуальної частини веб-ресурсу.
2. Сканування серверної частини.
3. Аналіз отриманих результатів та відображення графіку продуктивності проскановаого веб-ресурсу

Результатом роботи веб-сервісу буде вивід необхідних метрик, на основі яких буде проведено аналіз існуючих методів прискорення показників, які впливають на швидкість завантаження. Після повторного сканування користувач

зможе побачити статистику до покращень та після, що надасть інформацію про дієздатність обраного методу оптимізації для даного ресурсу.

1.8 Висновок до розділу

Для оптимізації швидкості завантаження веб-ресурсу використовують комплексний вибір інструментів що допомагають аналізувати статистичні дані зібрані після сканування усіх частин сайту та обрати стратегію по оптимізації веб-сайту. Даний підхід дозволяє уникнути недоліків одного веб-ресурсу, та доповнити перевагами іншого. Також, виходячи з цілей для вирішення різних задач можуть знадобитися різні методи.

2. ТЕОРЕТИЧНА ЧАСТИНА

Проаналізувавши існуючі метрики та спосіб їх використання іншими веб-сервісами враховуючи їх недоліки можемо виділити такі найвагоміші метрики:

- Cumulative Layout Shift;
- Largest Contentful Paint;
- First Input Delay;
- Speed Index;
- Time to Interactive.

Для максимального підвищення продуктивності веб-сторінки потрібно враховувати всі дані з цих метрик та провести виправлення недоліків їх застосування у інших веб-сервісах.

2.1 Cumulative Layout Shift

Cumulative Layout Shift (CLS) [5] - це метрика, яка вимірює нестабільність контенту, складаючи зсув всіх елементів, яке відбувається від дій користувача.

Принцип дії алгоритму зрушення - якщо виявлено зрушення, метричний алгоритм намагається визначити область вікна перегляду (viewport), в якій воно відбулося. Використовуючи відмінності між сусідніми фреймами сторінок, він шукає всі зміщені елементи DOM.

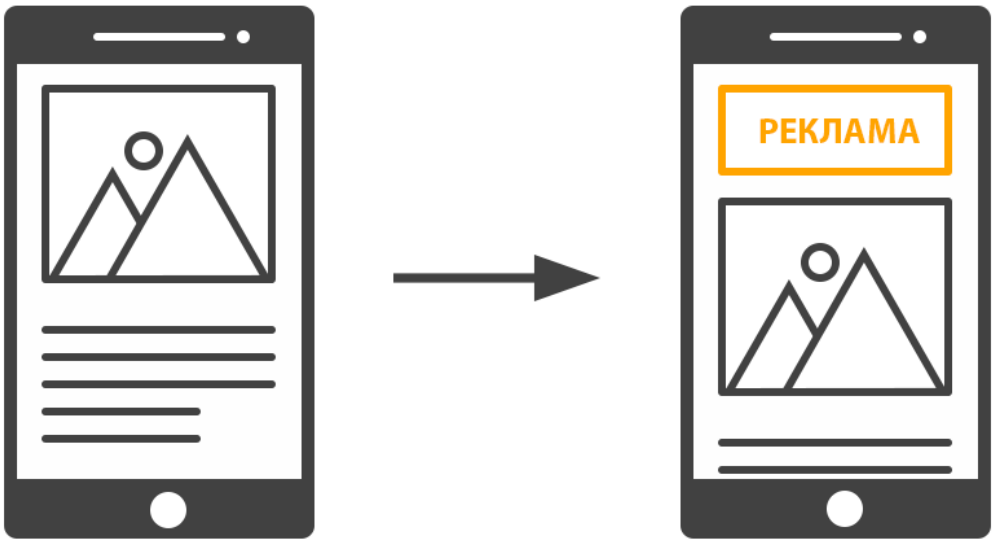


Рисунок 2.1 - Зрушення макету

Далі визначаються їхні попередні та поточні зони розташування (рис 2.3)

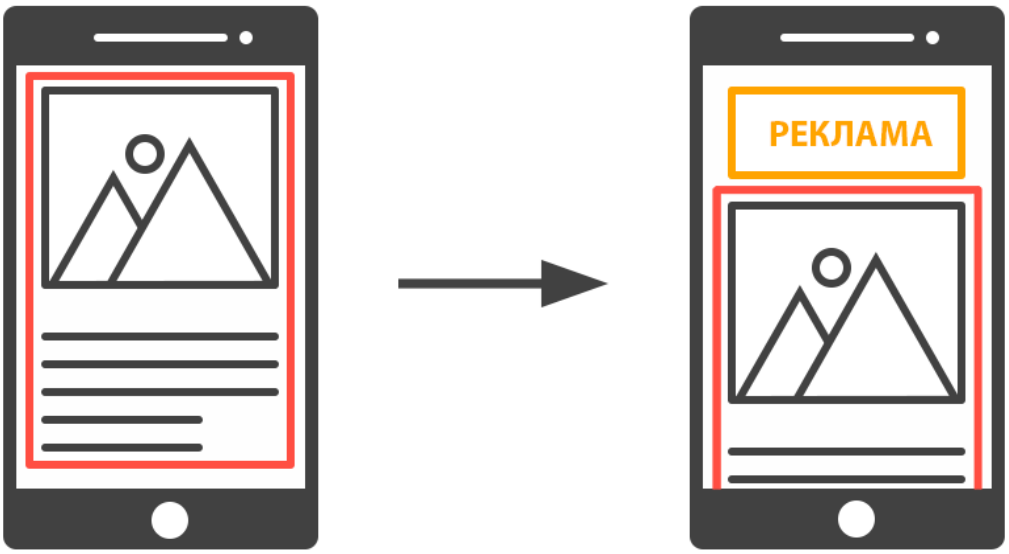


Рисунок 2.2 - Зона розташування зміщених елементів

Об'єднання яких утворює зону зміщення макета:

Рисунок 2.3 - Зміщення області макета

Після того, як область зміщення знайдена, визначається ступінь її впливу по відношенню до розміру вікна перегляду.

Рисунок 2.4 - Розміри вікна перегляду та контенту

Для цього розраховується (2.1) співвідношення площі області зміщення і площі.

Дані взяті з рис. 2.4

$$\textbf{\textit{Область зміщення / Область перегляду}} = (192 \times 308) / (220 \times 320) = 0{,}84 \quad (2.1)$$

Спочатку алгоритм метричної операції не передбачав визначення інших параметрів зміщення. Але проблемою цього методу є те, що відстань не враховується. Коли елемент має зсув на пару пікселів, ефект цього зміщення є непомітним для користувача. Причому, якщо розміри елемента спочатку були великі, то навіть такі незначні зрушення істотно впливають на площу переміщення та, відповідно, на значення метрики.

Тому треба видозмінити формулу для оцінки зсуву макета. Додамо відстань зміщення.

Рисунок 2.5 - Відстань зміщення контенту

Коли відбувається розміщення кількох елементів одночасно вздовж будь-якої з осей компоновки (горизонтальної чи вертикальної), виконується пошук максимальної відстані зміщення.

Далі визначається максимальний розмір вікна перегляду (ширина або висота). У нашому випадку це висота. Після цього розраховується співвідношення цих характеристик (2.2)

Максимальна відстань зміщення / найбільший розмір вікна перегляду = 87/320 = 0,27

2.2

В результаті ми отримали відносні значення двох параметрів зміщення:

- З міщення площі
- В ідстань зміщення

Щоб визначити передбачуваний зсув розкладки, потрібно знайти добуток цих параметрів.

Розрахункове зміщення = Площа зміщення * Відстань зміщення = 0,84 × 0,27 = 0,2268

2.3

Але оскільки ми оперували результатами різниці між сусідніми фреймами сторінки, то отримана оцінка зсуву (Layout Shift) буде індикатором зрушення для окремого кадру. Щоб визначити сукупний зсув макета (Cumulative Layout Shift),

потрібно підрахувати кожен кадр від початку завантаження сторінки до його закриття.

Основною проблемою цього підходу є неузгодженість макета, що є частиною логіки інтерфейсу. Зрештою, коли певні елементи сторінки потрібно змінити у відповідь на взаємодію користувача, вони, очевидно, змінять макет, що в кінцевому підсумку негативно вплине на результати метрики.

Одним з найпростіших рішень було б ігнорувати зміни макета, що відбуваються в обробнику подій. Але події, які спричинили зміщення макета, можуть відбуватися не тільки в обробниках, вони можуть відбуватися після виконання асинхронного запиту або всередині користувацьких планувальників задач (Event Scheduler)[6]. Це означає, що для визначення причини зсуву знадобиться складний аналіз послідовності асинхронних кроків, що являється складною задачею.

Остаточне рішення цієї задачі полягає в тому, щоб забезпечити часове вікно, протягом якого зміни макета не враховуються. Часове вікно з'являється відразу після події, введеної користувачем. Тривалість вікна візьмемо 500 мс. Таким чином, будь-які зміщення макета, що відбуваються протягом 500 мс після умовного кліку миші по вікну веб-ресурсу, не будуть враховуватися до загальної оцінки.

Іншою проблемою є анімація. Анімація властивостей css, які запускають операцію Layout, вплине на кінцевий результат метрики. Однак анімація з використанням властивостей, які створюють окремий контекст змішування, не викликає таких перерахунків. Ці властивості включають перетворення, непрозорість і все, що не має нічого спільного з позиціонуванням і геометрією елементів сторінки.

Часткове вирішенням проблеми - це ігнорування зміщень макета, викликаних властивостями анімації, які не запускали операції перерахунку макета.

2.3 Largest Contentful Paint

Largest Contentful Paint[7] (LCP) визначає, скільки часу потрібно, щоб відобразити найбільший елемент на сторінці. Але перш ніж зупинитися на тому, як працює LCP, варто поговорити про фон, який спонукав команду Chrome SpeedMetrics створити новий показник.

Показники продуктивності клієнта, які ми визначили на самому початку, можна розділити на фази завантаження програми. Я буду називати ці етапи так:

- фаза завантаження - коли користувач увійшов у програму;
- фаза відображення основного контенту – коли на екрані користувача починають відображатися значущі елементи сторінки;
- фаза настання інтерактивності - коли сторінка готова стабільно реагувати на дії користувача.

Якщо розташувати основні показники клієнта за фазою завантаження, представивши зображення, яка нещодавно використовувалася, то отримаємо щось на зразок цього:

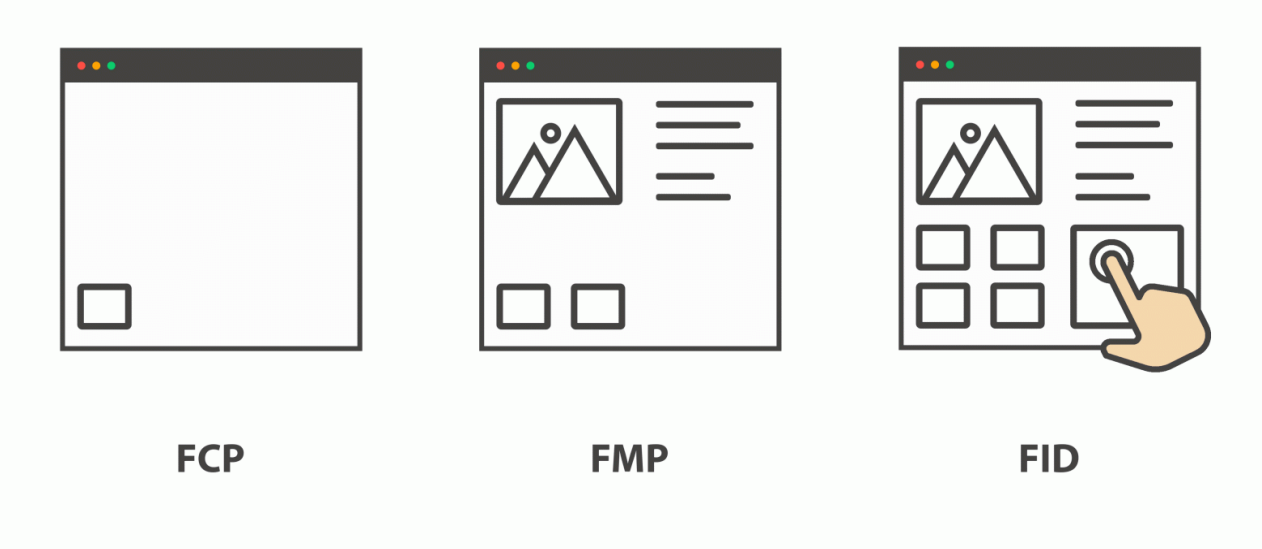


Рисунок 2.6 - Фази завантаження контенту

Тут може виникнути резонне запитання: якщо всі фази завантаження програми охоплюються відповідними показниками, навіщо знадобився інший? Відповідь криється в метриці FMP (First Meaningful Paint), яка має ряд проблем.

Команда Chrome SpeedMetrics, яка має великий досвід у створенні показників продуктивності, визначила набір властивостей, якими повинні володіти хороші показники.

На жаль, у метриці FMP відсутня частина з них – зокрема, властивості простоти та гнучкості. Перше пов’язано зі складністю стандартизації використовуваних рішень, а друге – через велику кількість евристик. В результаті від цього показника було вирішено відмовитися. Після відмови від FMP фаза відображення основного вмісту залишається без метрик, отже, без відповідного моніторингу.

Щоб знайти повну заміну, потрібно зробити аналіз декількох кандидатів-метрик, які впроваджуються в браузер Chrome:

- L

argest image paint;

- L

argest text paint;

- L

argest image OR text paint;

- L

ast image paint;

- L

ast text paint;

- L

ast image OR text paint.

Після аналізу даних, зібраних з більш ніж тисячі сайтів, метрика Найбільше зображення АБО затінення тексту була визнана найбільш підходящим кандидатом. Згодом цей показник був названий Largest Contentful Paint.

Принцип дії - як згадувалося раніше, метрика LCP вимірює, скільки часу потрібно, щоб відобразити найбільший елемент на сторінці. А оскільки відображення веб-ресурсу є поступовим, то найбільший компонент на сторінці може змінюватися з часом. Візуально цей процес може виглядати як:

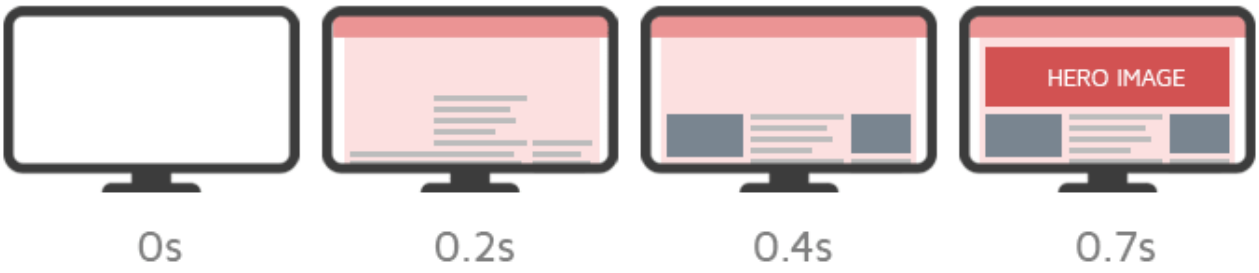


Рисунок 2.7 - фази відображення контенту

Поки невідомо, які типи предметів вважаються, як визначається розмір та час відображення. В даний час алгоритм метрики LCP відстежує такі типи елементів:

- з

ображення:

  - <

img> елементи,

  - е

лементи <image> всередині документів SVG,

  - е

лементи з фоновим зображенням;

- в

ідео із poster-зображенням;

- т

екст у елементі блоковий.

Розмір найбільшого елемента визначається за такою формулою:

Розмір = Ширина × Висота

Зовнішні та внутрішні виїмки, а також рамки елементу не враховуються.

Час відображення визначається так:

- З

ображення показують момент відображення, який настає відразу після події завантаження ресурсу. Для зображень, завантажених із сторонніх джерел, потрібен заголовок Timing-Allow-Origin[8]. В іншому випадку, моментом відтворення елемента буде вважатися момент настання події завантаження ресурсу.

- Для відеозаписів фіксується момент показу зображення плаката. Алгоритм визначення часу відображення ідентичний описаному вище для зображень.
- Для текстових елементів час першого відображення фіксується незалежно від майбутнього завантаження основного шрифту та наступного перемальовування текстового вмісту сторінки.

Як згадувалося раніше, при завантаженні програми статус найбільшого елемента сторінки може змінюватися від одного елемента до іншого. По суті, пошук - це останній за величиною елемент області перегляду. Але питання виникає коли алгоритм метрики вирішує перестати відслідковувати нові елементи. Адже нас цікавить лише етап завантаження програми. Очевидне рішення полягає в тому, що після події введення користувача пошук нових кандидатів для метрики LCP припиняється.

Ще одна особливість – механізм роботи метрики після видалення DOM-елемента. Коли визначається найбільший елемент сторінки видалення з дерева DOM, починається пошук нового кандидата. Найпоширеніший приклад - це відображення попереднього завантажувача, яке потім зникає. Логіка подальшого пошуку обумовлена тим, що елементи, які видаляються в процесі завантаження, швидше за все, не значимі з точки зору користувача, тому враховувати результати їхнього малювання не має особливого сенсу.

Варто згадати найскладніший сценарій, коли на сайті використовується компонент каруселі з автоматичною зміною елементів по таймауту. Більшість реалізацій таких компонентів включають видалення елемента каруселі, що відображається після його заміни іншим. Але якщо після входу на сторінку введення користувача не відбулося, а елементи каруселі були позначені як найбільші і продовжують міняти один одного, то кожен раз при зміні значення

метрики буде оновлюватися, щоб отримувати все більше і більше невірних результатів.

2.4 First Input Delay

First Input Delay[9] (FID) - вимірює тривалість затримки введення користувача при першій взаємодії зі сторінкою. Принцип дії - для початку давайте роздивимося тимчасову шкалу, яка відображає типове завантаження сторінки:

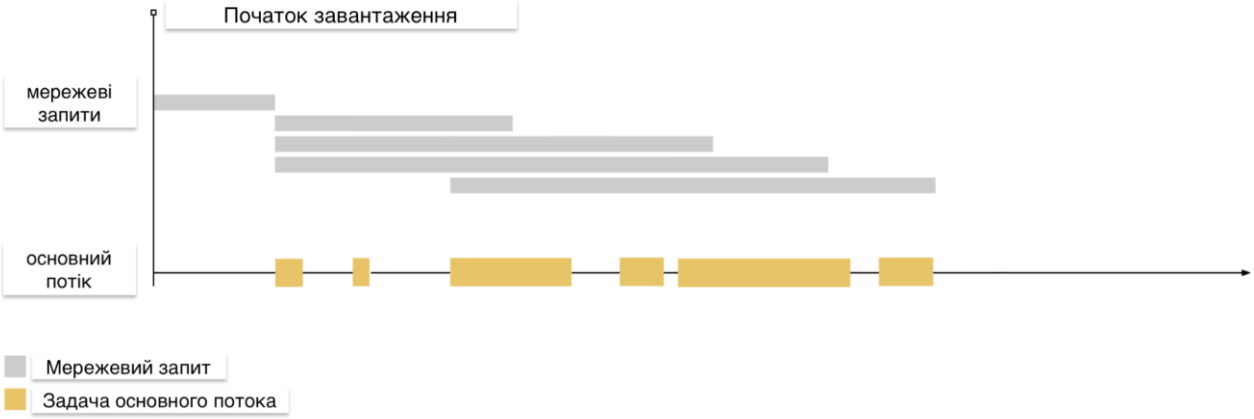


Рисунок 2.8 - Графік відображення завантаження сторінки

На шкалі показано виконання мережевих запитів до основних ресурсів сторінки, файлів CSS та JS (виділені сірим кольором).

Після завантаження ресурс обробляється.Після завантаження ресурс обробляється та запускається в основному потоці браузера (відзначений бежевим кольором).

У більшості випадків існують великі затримки введення (FID) між першим відображенням вмісту (FCP) та часом до взаємодії (TTI) рис. 2.10.

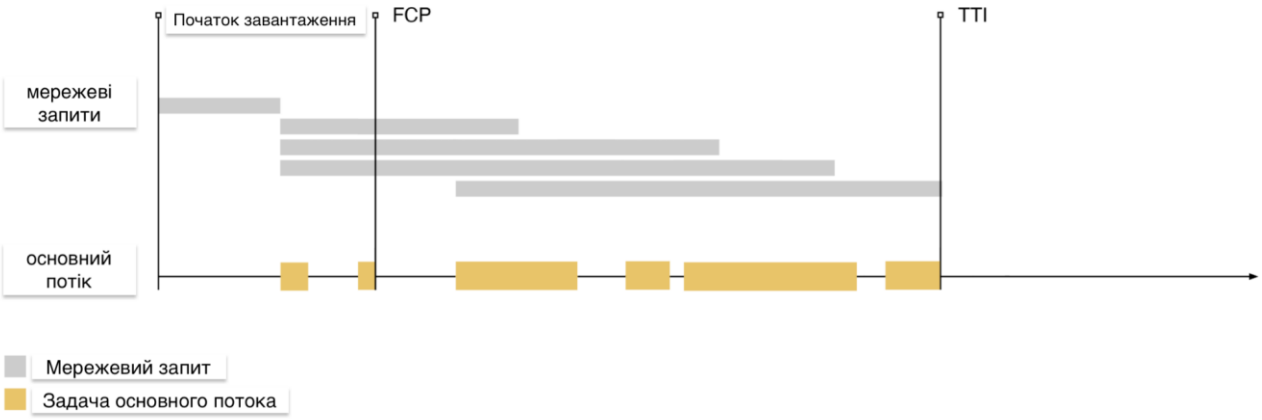


Рисунок 2.9 - Відображення FCP та TTI метрик

У тимчасовому інтервалі між метриками FCP і TTI можна побачити деякі завдання, що виконуються в основному потоці браузера. Будь-яке завдання, яке триває понад 50 мс, називається довгою. Межа 50 мс відповідає одному з вимог моделі продуктивності RAIL. Позначимо довгі завдання відповідним кольором:

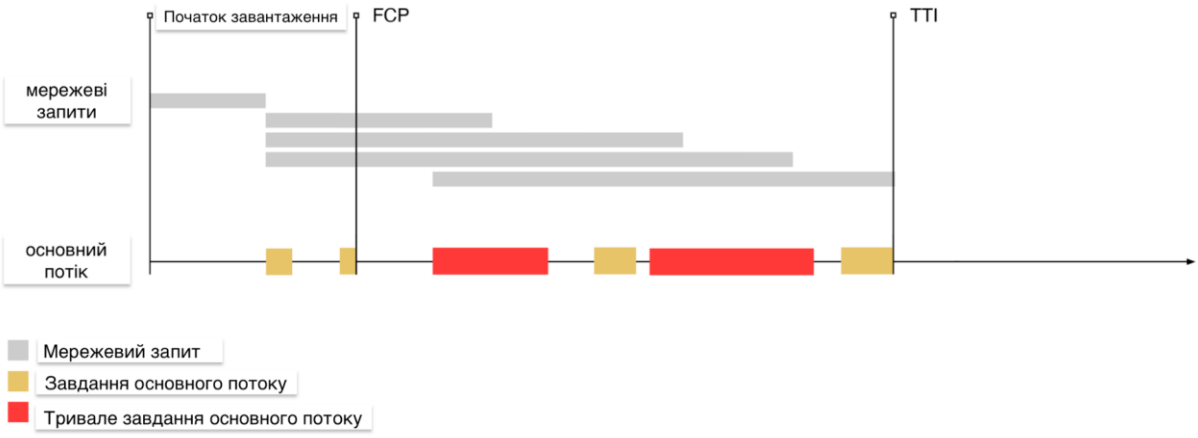


Рисунок 2.10 - Графічне зображення графіку основного потоку

Якщо користувач починає взаємодіяти зі сторінкою при виконанні одного з цих завдань, вони будуть мати видиму затримку. Затримка буде

продовжуватися, поки основний потік браузера не завершить завдання, пов'язане з цим завданням рис. 2.12.

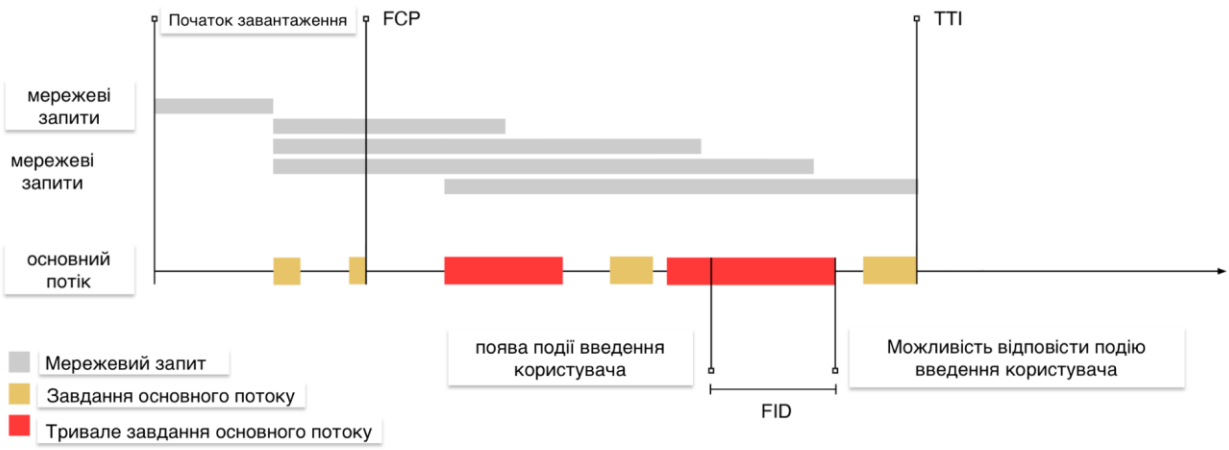


Рисунок 2.11 - графічне відображення FID метрики

Як бачите, введення користувача відбулося під час тривалого завдання. Відповідь на запис надійшла після того, як завдання було виконане. І час, що минув від появи вхідної події до можливості обробки, буде значенням FID.

Цінність моніторингу цієї метрики багаторазово зростає, якщо у веб-додатку використовується SSR (Рендеринг на стороні сервера). Це пов'язано з тим, що користувач одразу отримує сторінку, наповнену контентом, що дає уявне відчуття готовності до взаємодії з нею. Однак сторінка не може відповідати на дані, що вводяться користувачем, доки файли скриптів не будуть завантажені і гідратовані.

Ситуація посилюється, коли такий проект прагне покращити час першого відображення контенту (FCP), незважаючи на час інтерактивності (TTI). Оскільки підвищення продуктивності малювання без підвищення продуктивності інтерактивності збільшить інтервал між відображенням

інтерфейсу та його реальною готовністю до взаємодії. Це може негативно вплинути на ваш FID та досвід роботи з вашим веб-додатком.

2.5 Індекс швидкості

Індекс швидкості (SI) показує, як швидко вміст сторінки відображається під час завантаження програми. Принцип дії - SI заснований на концепції (Visually Complete, VC). Розглянемо два можливі варіанти завантаження сторінки:

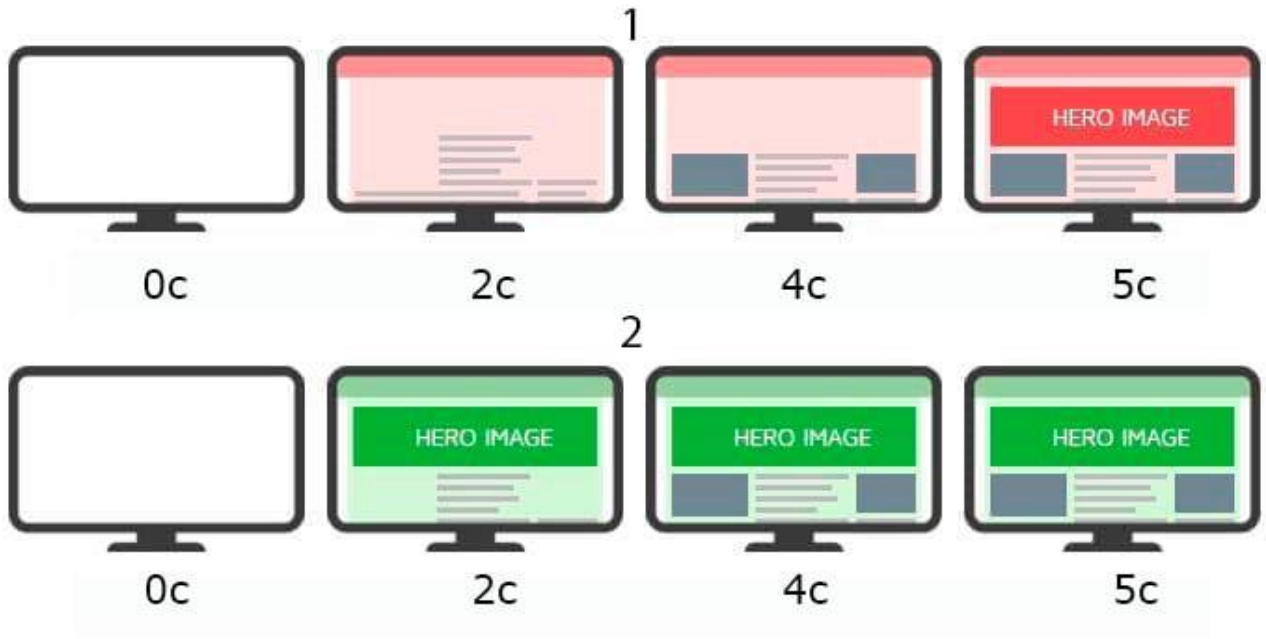


Рисунок 2.12 - Етапи відображення контенту

Обидва варіанти починають завантажуватися та стають візуально завершеними (VC) одночасно (5 секунд). Але, очевидно, варіант 2 набагато кращий з точки зору користувача. Щоб побачити кількісну різницю між такими варіантами навантаження, використовується індикатор SI. SI оцінює відсоток візуального завершення в різний час.

Використовуваний стандартний механізм виявлення VC, наприклад WebPageTest, заснований на аналізі колірної палітри сторінки. Кожні 100 мс кількість відтінків порівнюється між двома сусідніми кадрами. Однак слід зазначити, що Lighthouse використовує іншу модель для визначення візуальної повноти. Замість порівняння кількості відтінків у колірній гамі використовується індекс структурної схожості (SSIM), який базується на оцінці якості сприйманого зображення, включаючи інформацію про зміни яскравості та контрастності. Розібравшись, як розраховується показник VC, представимо графічно розглянуті варіанти завантаження сторінки:

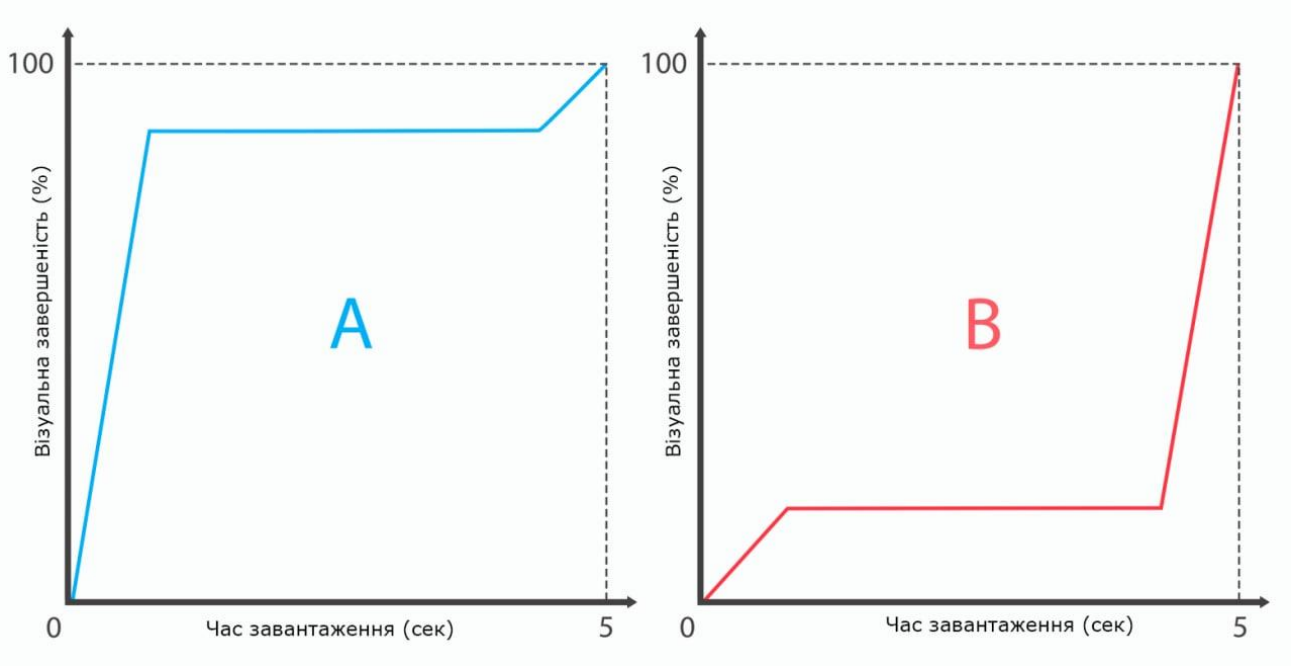


Рисунок 2.13 - Графік завантаження сторінки

На графіку (рис. 2.13) показано співвідношення між часом завантаження та візуальним відсотком завершення. Область графіка під кривою може бути відображена як частина сторінки в певний час (рис. 2.14).

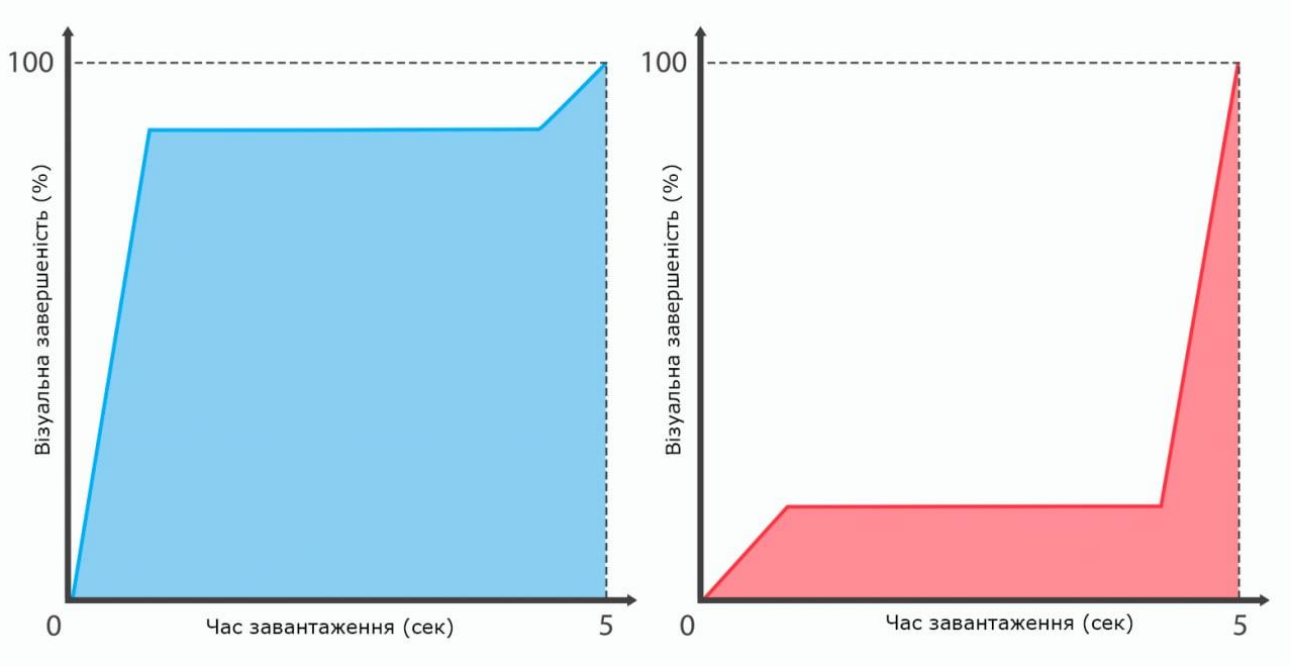


Рисунок 2.14 - Графік завантаження сторінки

Ми могли б використовувати площу цієї області для обчислення CI, але є важливий аспект. Нагадаю, що 5 секунд завантаження, над якими ми працюємо, є часом візуального завершення (VC), але вони не є показником повного завантаження сторінки. Це означає, що площа під кривою буде збільшуватися пропорційно часу завантаження, одночасно збільшуючи результат CI. У той же час використання площі графіка над кривою буде позбавлене таких недоліків, оскільки воно завжди обмежене настанням події VC.

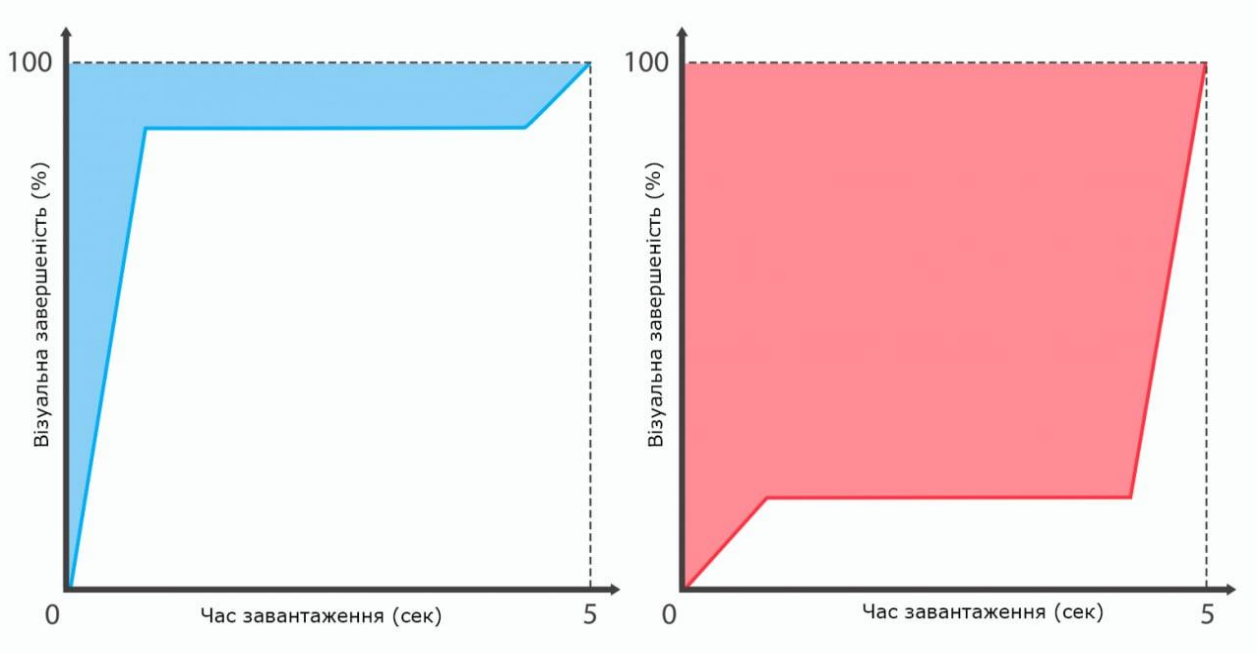


Рисунок 2.15 - Графік завантаження сторінки

Тому алгоритм CI використовує цю область при обчисленні результату. Звичайно, щоб знайти площу зазначеної площі (плоска вигнута фігура), формула розрахунку використовує певний інтеграл і виглядає так 2.4

$$SI = \int_{0}^{end} (1 - \%\ Візуальна\ завершеність\ /100)\, dt$$

2.4

Тому що чим менше площа цієї області, відповідно, і значення CI, тим краще рис. 2.16

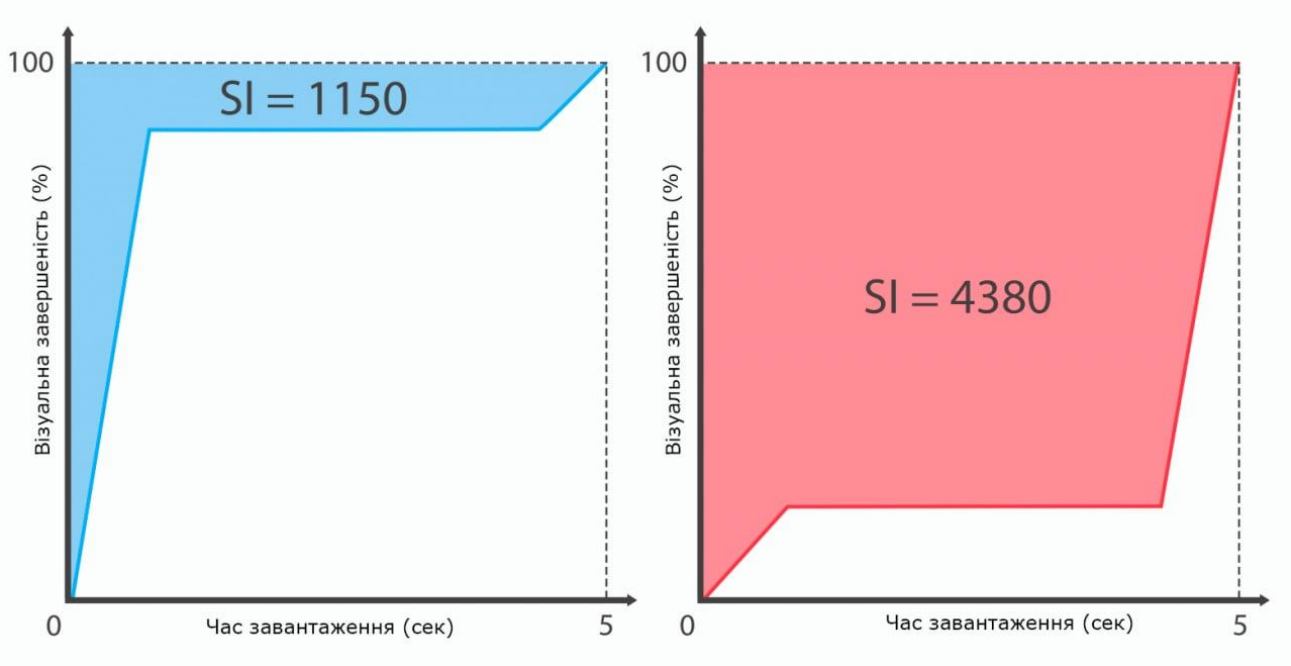


Рисунок 2.16 - Позначення індексу швидкості

2.6 Дослідження клієнтьско-серверних методів оптимізації

Як вже було досліджено є показник на котрий впливають методи оптимізації як з серверної так і з клієнтської сторони. Цей показник є TTI, що вказує як швидко користувач вже може взаємодіяти з сайтом. Отже візьмемо його як основу для комбінування процессів оптимізацій у єдину систему.

2.6.1 Методи оптимізації FCP

Першим показником що буде впливати на TTI є FCP отже розглянемо способи оптимізації найпоширеніших випадків зміщення макета:

- А

трибути розміру зображення - завантаження зображень без явного розміру може призвести до зміщення макета. Атрибути ширини та висоти, зазначені в тегу <img>, допоможуть уникнути неочікуваних зміщень.

- Стратегія завантаження шрифтів і відтворення тексту - у процесі завантаження шрифтів браузери відразу або після тайм-ауту відображають текст сторінки за допомогою резервного шрифту. Після завершення завантаження запасний шрифт замінюється основним шрифтом, який перемальовує текстовий вміст сторінки. У більшості випадків такі змінення спричиняють неузгодженість елементів макета, яку можна мінімізувати, чітко визначивши стратегію відтворення тексту. Щоб контролювати стратегію відтворення, правило @ font-face дозволяє визначити дескриптор для відображення шрифту. Дескриптор має багато значень, одне з яких дозволяє мінімізувати перемальовування та зміщення макета. Це необов'язкове значення, головна суть якого полягає в тому, що якщо основний шрифт не завантажується протягом перших 100 мс, то текст відтворюється за допомогою резервного, без будь-якої заміни. Однак наступного разу, коли ви відвідуєте сторінку, текстовий вміст буде відтворено з використанням основного шрифту – звичайно, якщо його було успішно завантажено та додано до кешу з минулого.
- Динамічний вміст сторінки[15] – це вміст, який не відображається безпосередньо у відповідь на введення користувача. Це можуть бути банери, реклама, блоки з проханням підписатися на розсилку або встановити мобільний додаток. Для динамічного вмісту невеликих розмірів рекомендується використовувати абсолютні характеристики позиціонування. А для додаткового – зарезервувати місце на сторінці заздалегідь – наприклад, використовуючи скелетні екрани завантаження.
- Анімація - як згадувалося раніше, властивості анімації, які запускають події

перерахунку макета, слід замінити анімацією за допомогою властивості transform.

Отже для того щоб уникнути зсувів макету знадобиться використати комплексний підхід оптимізації усіх веб ресурсів, які мають вплив на відображення сторінки. Це підвищить рівень комфортного користування веб-ресурсом та знизить ризики того, що клієнт покине сайт.

2.6.2 Методи оптимізації LCP

Як вже було досліджено на показник LCP впливає те, за який час користувач може почати взаємодію зі сторінкою. Для оптимізації такого параметру потрібно:

- зменшити кількість ресурсів, що блокують відображення;
- використовувати стиск текстових файлів;
- оптимізувати зображення, методом їх стиску або змінення формату зображень;
- мініфікувати всі файли що завантажуються (CSS, JS та інші);
- оптимізувати залежності файлів та їх завантаження;
- встановити попереднє з’єднання зі сторонніми доменами та налаштувати пріоритетність завантаження ресурсів.

2.6.3 Методи оптимізації FID

Основні покращення, які можуть впливати на FID:

- Оптимізувати всі сторонні скрипти, такі як кнопки соціальних мереж, аналітику та рекламу, щоб вони не збільшували час завантаження вашого сайту. Як альтернатива ви можете використовувати відкладене завантаження потрібних вам скриптів.
- Прискорити JavaScript – розбити тривалі завдання на дрібніші. У проміжках між обробкою ЦІ коротких завдань браузер встигне оптимізувати запит користувача.
- Мінімізувати кількість поліфілів, що не використовуються;
- Оптимізувати код CSS - для покращення FID рекомендується мінімізувати, стиснути і видалити CSS, що не використовується.
- Запускати довготривалі задачі поза основного потоку - Це має утримувати основний потік у режимі очікування і, отже, збільшувати затримку першого введення. Для цього ви можете передати дані у Web Worker.
- Налаштувати завантаження лише коду, який необхідний для первинної роботи.
- Використання кешування - за допомогою кешування ви можете зберігати раніше завантажений контент, і він не буде перезавантажуватися, коли

користувач відвідує його знову. Завантаження з кешу мінімізує навантаження на сервер та підвищує продуктивність.

2.6.4 Методи оптимізації TTFB

Параметр який буде підвищуватися на серверній стороні покращуючи швидкість відповіді від серверу. Щоб зменшити TTFB, потрібно виконати такі дії:

- Кешування сторінки. Якщо кешування сторінок на сайті вимкнено, сервер буде змушений повторно створити сторінку, до якої звертається кожен відвідувач сайту. Якщо таких сторінок і відвідувачів буде занадто більше, швидкість завантаження сайту зменшиться. Щоб вирішити цю проблему, ви повинні використовувати кешування сервера. При активному кешуванні сторінка не буде створюватися повторно з кожним новим запитом - відвідувачеві буде просто надано раніше створену сторінку.
- Оптимізація бази даних. Якщо час відповіді сервера залишається неприйнятним - значить, настав час зайнятися базою даних. Проблеми з відсутністю оптимізації баз даних виникають на важких сайтах, які мають десятки тисяч і сотні сторінок. Особливо часто їх перевіряють інтернет-магазини. Правило однакове для всіх типів сайтів: чим більше запитів, тим більше часу займе сторінка. Відповідно, час відповіді сервера збільшиться. Для формування кожного блоку на сайті використовуються одночасно десятки запитів. Блок рекомендацій, наприклад, може формуватися дуже довго - для його створення потрібно визначити і розрахувати відразу кілька параметрів. Таким чином, завдання очевидне: необхідно мінімізувати загальну кількість запитів до бази даних. Вам

знадобляться технічні спеціалісти, у тому числі програміст, для налагодження та подальшого визначення найскладніших запитів.

- Р

озширення PHP. Знизити TTFB можна за допомогою прискорювачів. Прискорювач PHP — це спеціальне розширення, яке обробляє скрипти, кешуючі байт-код. Використання прискорювача позитивно впливає на загальну швидкість завантаження сайту. Принцип роботи прискорювача заснований на попередньому кешуванні PHP-коду. Використання прискорювача PHP дозволяє частково звільнити системні ресурси під час обробки файлів PHP. Часто час відповіді сервера страждає, оскільки сервер немає необхідного рівня продуктивності. У цьому випадку потрібно звернути увагу на те, як сервер справляється з максимальним навантаженням. Якщо сайт часто виходить з ладу або завантажується дуже повільно, перехід на більш продуктивний варіант хостингу може бути найбільш продуманим і ефективним сценарієм. Конфігурація сервера повинна бути в налаштована У разі непередбачених ситуацій, наприклад, стрибків руху. Звичайно, хостинг повинен бути платним і передбачати масштабування вашого проекту. Рано чи пізно вам доведеться збільшити наступні обмеження: дискова квота, завантаження бази даних, допустиме статичне навантаження, поштова квота.

2.8 Висновки до розділу

В даному розділі дипломної роботи були досліджені метрики, що відображають продуктивність праці веб-ресурсу.

Було досліджено комплексний підхід оптимізації з клієнтської та серверної сторони які підвищують показники продуктивності та швидкості завантаження веб-сайту.

3. РОЗРОБКА АЛГОРИТМА, МОДЕЛЮВАННЯ АБО ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА

3.1 Опис схеми алгоритму

1. Ініціалізація інтегрованого веб-плагіну метрики.
2. Виконання Get-запиту до сторінки з метою отримати доступ.
3. Перевірка на отримання даних з сервера про веб-ресурс.
4. Перевірка логічної умови: якщо істинно виконати умову «Так», інакше умову «Ні».
5. Запис інформації швидкості отримання первинних даних сторінки.
6. Початок сканування всього веб-ресурсу.
7. Розраховуються усі потрібні метрики (FCP, FMP, FID) задля отримання результатів вимірювання цих метрик.
8. Очікування розрахунку метрики.
9. Перевірка логічної умови: якщо істинно виконати умову «Так», інакше умову «Ні».
10. Запис часу за який сторінка повністю завантажується.
11. Далі запускається функція порівняння результатів метрик, розрахунок розташування даних по їх значенню для веб-ресурсу.
12. Перевірка логічної умови: якщо істинно виконати умову «Так», інакше умову «Ні».
13. Вивід кінцевих результатів сканування та поради по оптимізації проблемних місць.

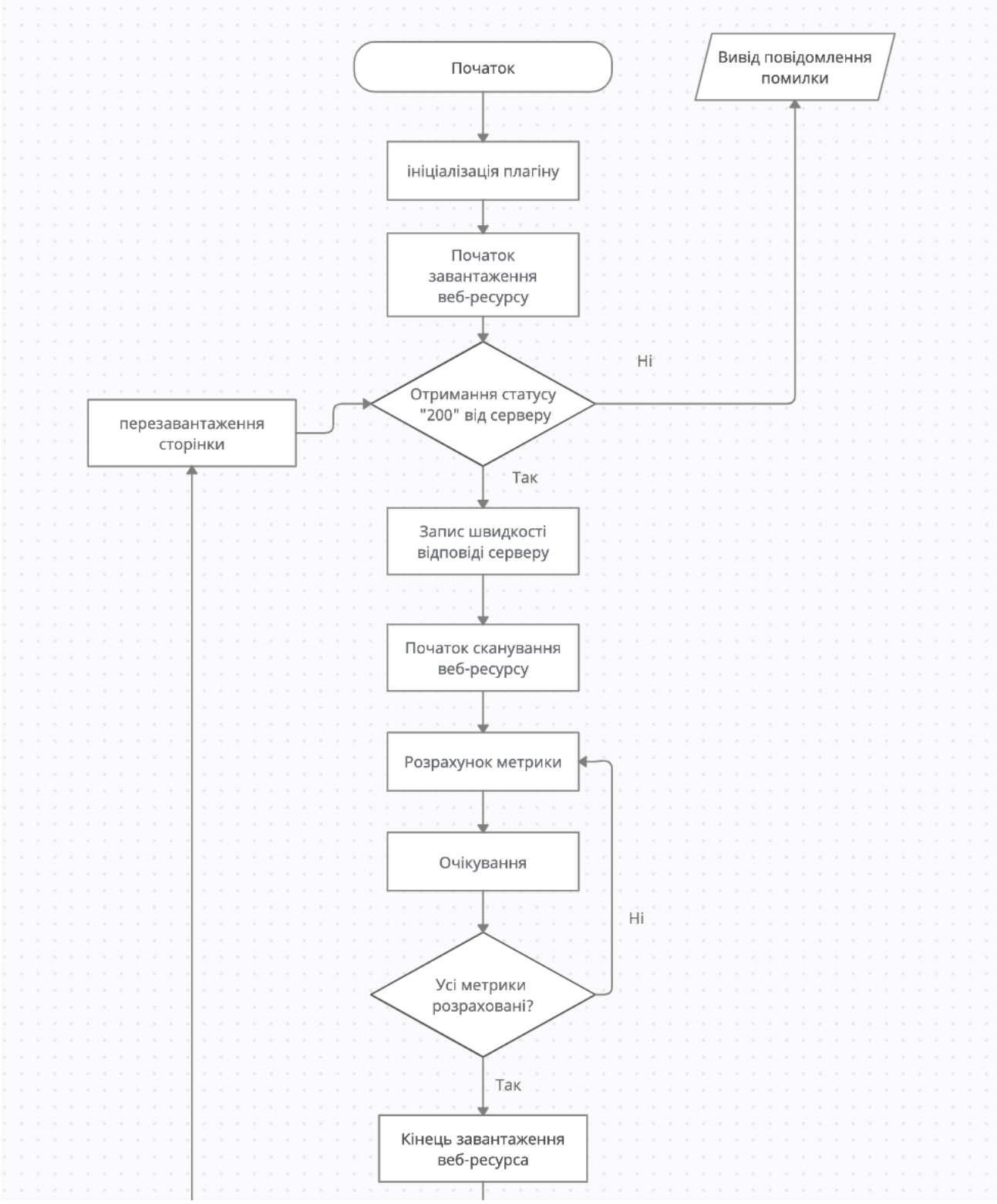
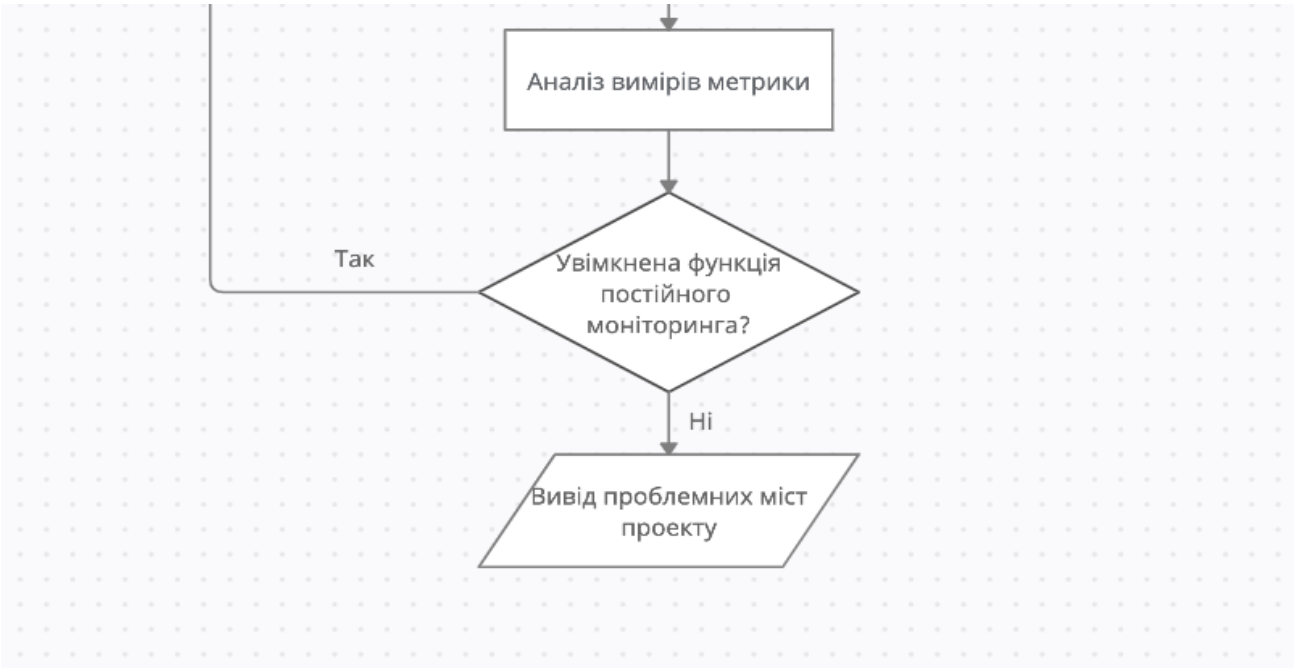


Рисунок 3.1 - блок схема алгоритма моніторингу

3.1 - продовження блок схеми



3.2 Тестування процесу оптимізації веб-сторінки

Для отримання результатів дослідження візьмемо веб-сторінку до та після оптимізаційних процесів. Для отримання первинних даних виміряємо не оптимізовану версію веб-сайту

Значення показників можна отримати за допомогою вбудованого API PerformanceObserver:

нехай cumulativeLayoutShift = 0;

```
let cumulativeLayoutShift = 0;

    const performanceObserver = new PerformanceObserver((performanceEntryList)
=> {
      const performanceEntries = performanceEntryList.getEntries();

      for (const performanceEntry of performanceEntries) {
// Властивість hadRecentInput дозволяє не враховувати події зсуву макета викликані
введенням користувача
        if (!performanceEntry.hadRecentInput) {
```

```
        cumulativeLayoutShift += performanceEntry.value;
          console.log('CLS is', cumulativeLayoutShift);
      }
    }
});

performanceObserver.observe({ type: 'layout-shift', buffered: true });
```

3.1

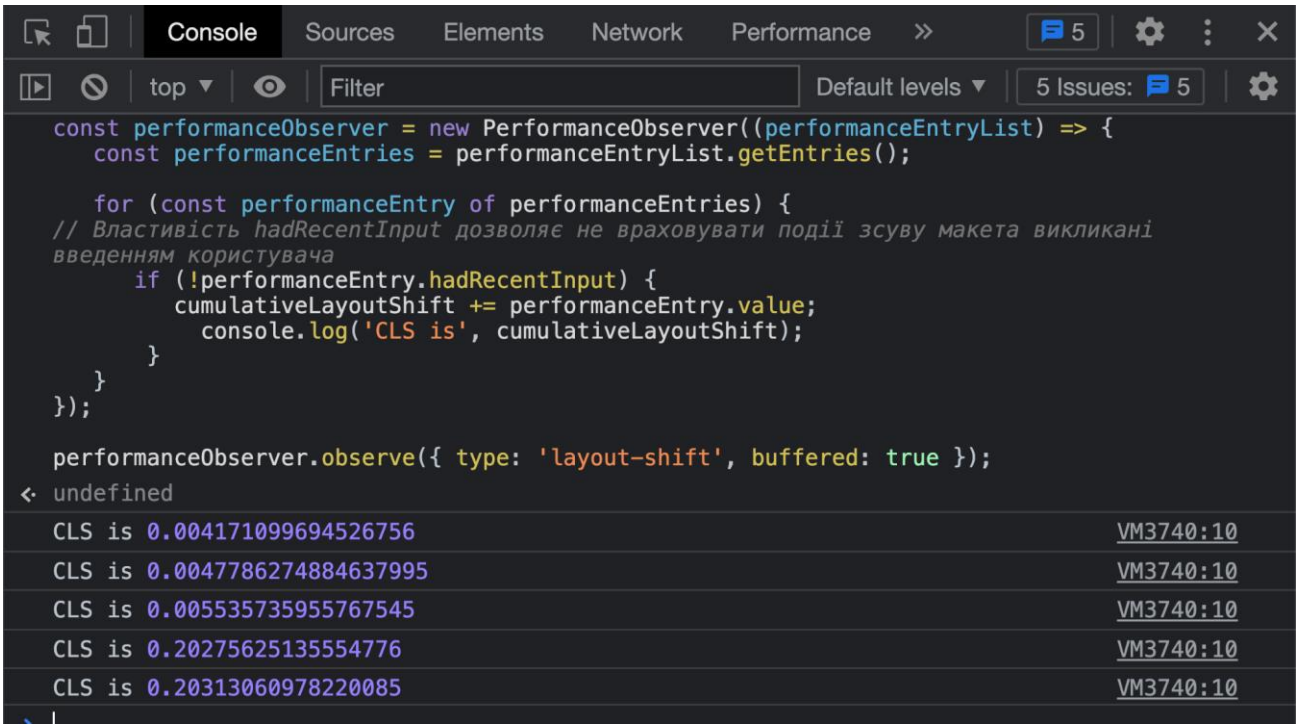


Рисунок 3.2 - результат замірів CLS не оптимізованого веб-ресурсу

Оцінка результатів. В ідеальному світі програми не повинні мати жодного руху макета, але на практиці це не завжди так. Крім того, у будь-якому проекті можуть бути навмисні компроміси в логіці інтерфейсу, які впливають на цінність CLS.

На рис 3.2 можемо побачити результати метрики змінна cumulativeLayoutShift, що дорівнює 0.20. За результатами цієї перевірки та порівняння даних загальноприйнятих градацій рис. 3.3 метрики CLS ми можемо

зробити висновок, що якщо ми маємо результат у діапазоні від 0.1 до 0.25 то вже можна звернути увагу на методи оптимізації шрифтів, анімацій, чи процессу рендерінгу блоків цього веб-сервісу.

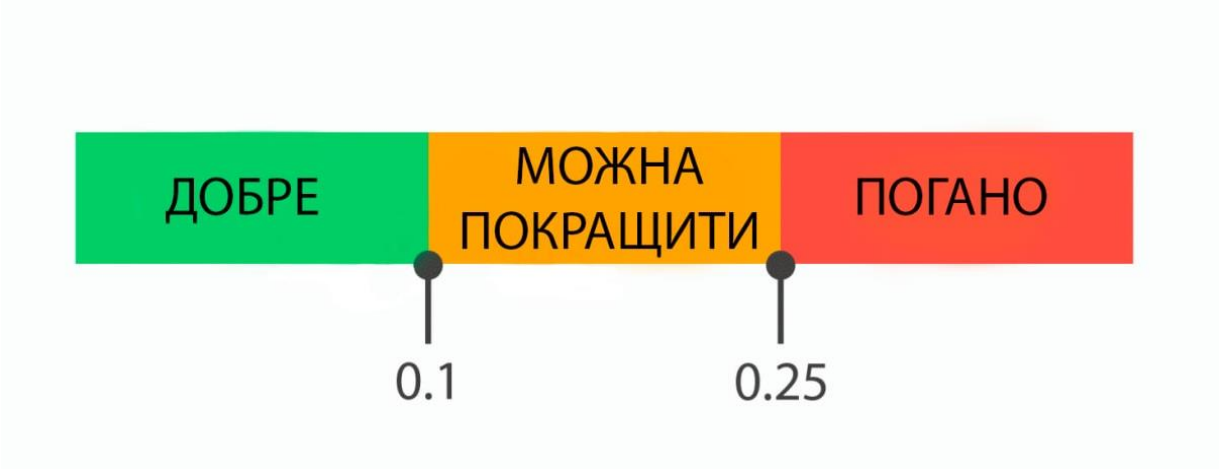


Рисунок 3.3 - Діапазон значень метрики CLS

Також за допомогою API PerformanceObserver можемо отримати результати метрики LCP:

```
const performanceObserver = new PerformanceObserver((performanceEntryList) => {
  const performanceEntries = performanceEntryList.getEntries();
  const lastPerformanceEntry = performanceEntries[performanceEntries.length - 1];

  const largestContentfulPaint = lastPerformanceEntry.startTime;

  console.log('LCP is', largestContentfulPaint);
});

performanceObserver.observe({ type: 'largest-contentful-paint', buffered: true });
```

3.2

Проведено заміри при високошвидкісному інтернеті на десктопній платформі. Результати даного експерименту можна побачити на наступному зображенні рис 3.4.

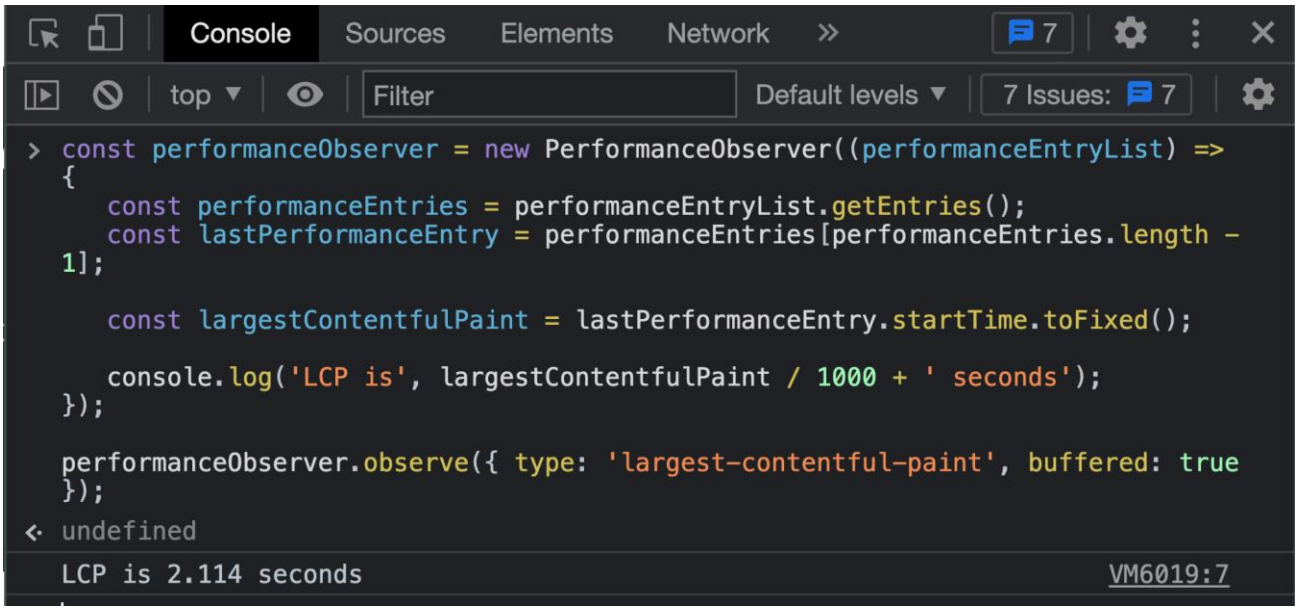


Рисунок 3.4 - Результат заміру метрики LCP

При оцінці результатів метрики рекомендується використовувати такі градації рис. 3.5. За результатами цього тесту можемо побачити, що найбільший елемент на сторінці відбувається менше ніж за 2.1 секунди - це означає що розміри ресурсів що завантажуються досить добре оптимізовані та важать мінімально для десктопної платформи при високогвидкісному інтернеті.

Рисунок - 3.5 - Діапазон значень метрики LCP

Далі за допомогою того ж API виконаємо перевірку наступної важливої клієнтської метрики FID що відобразить через який час користувач вже може

використовувати даний веб-ресурс. Для оцінки результатів використаємо таку градацію рис. 3.7.

Рисунок 3.7 - Діапазон значень метрики FID

Код який виконує перевірку даної метрики 3.3

```
let hiddenTime = document.visibilityState === 'hidden' ? 0 : Infinity;

document.addEventListener('visibilitychange', (event) => {
  hiddenTime = Math.min(hiddenTime, event.timeStamp);
}, { once: true });

new PerformanceObserver(entryList => {
  var fidEntry = entryList.getEntries()[0];
  if (fidEntry.startTime < hiddenTime) {
     var fid = fidEntry.processingStart - fidEntry.startTime;
     console.log("First Input Delay: " + fid);
  }
}).observe({ type: "first-input", buffered: true });

try {
  new PerformanceObserver(entryList => {
```

```
        var fidEntry = entryList.getEntries()[0];
        var fid = fidEntry.processingStart - fidEntry.startTime;
        console.log("First Input Delay: " + fid);
    })
        // Some browsers throw when 'type' is passed:
        .observe({ type: "first-input", buffered: true });
}
catch (e) { console.log(e); }
```

3.3

Результати перевірки рис 3.8 - по цим даним маємо результат 1.39 мілісекунди. Звернувшись до діапазону значень рис 3.7 можна зробити висновок що даний результат нас задовольняє та немає необхідності оптимізувати веб-ресурс.

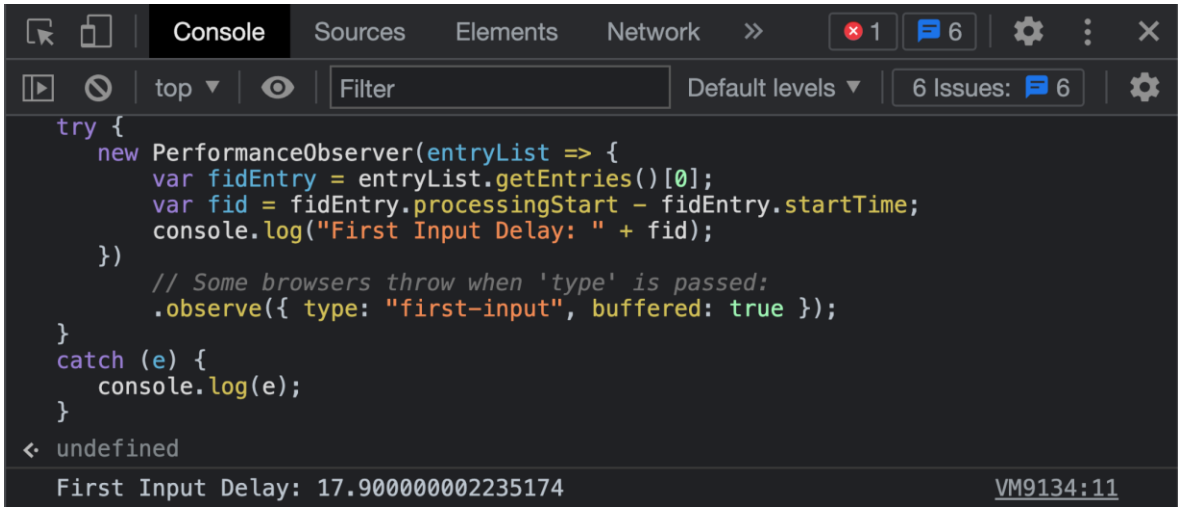


Рисунок 3.8 - Результат перевірки FID

У наведеному прикладі значення затримки запису першого введення вимірюється шляхом взяття різниці між відмітками часу startTime і

processingStart запису. Найчастіше це і буде значенням FID. Однак, не всі записи першого введення дійсні для вимірювання FID.

Далі наведено різницю між тим, що повідомляє API, і тим, як розраховується метрика.

Результатом виконання цього коду буде відображення точної метрики FID, як і кожна із метрик цей параметр допоможе виводити інформацію про сканований веб-ресурс та зробити висновки які методи оптимізації необхідно використовувати для покращення його продуктивності.

Вимір Speed Index та TTI буде проводитися за допомогою веб-сервісу Gtmetrix рис 3.9.

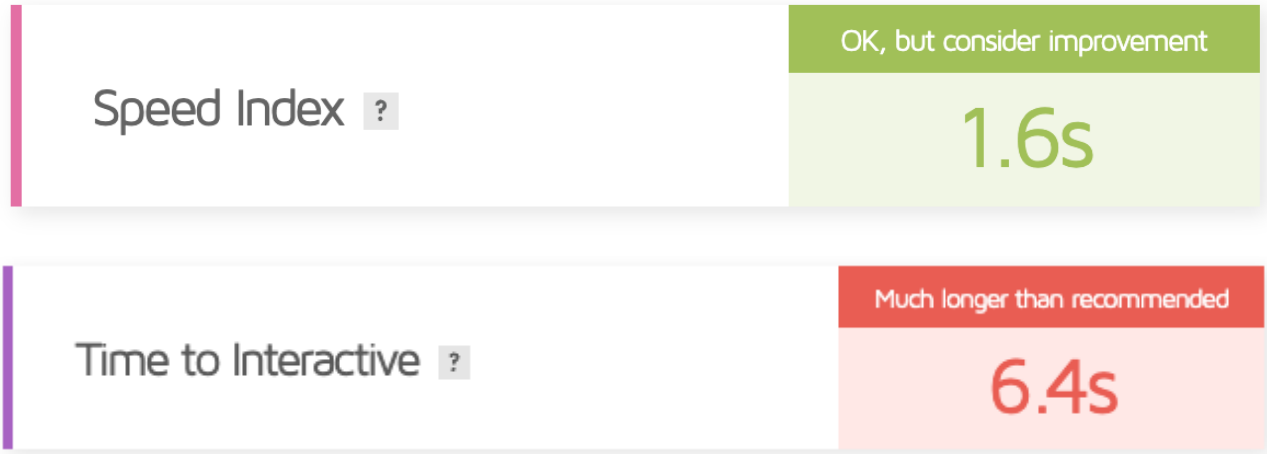


Рисунок 3.9 - результат виміру SI та TTI

По результатам даних вимірювань оптимізованого ресурсу маємо такі результати:

- C
LS = 0.2
- L
CP = 2.1 сек
- F
ID = 17.9 мс

- S

I = 1.6 сек

- T

TI = 6.4 сек

Використаємо досліджений процес оптимізації та зробимо виміри рис. 3.10, 3.11, 3.12, 3.13 для отримання результату покращення.

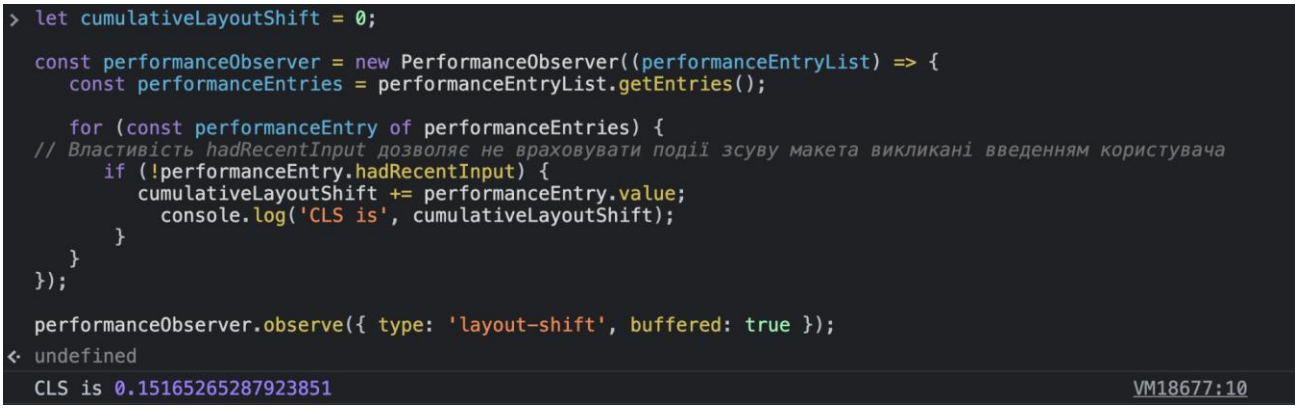


Рисунок 3.10 - Результат виміру CLS

Рисунок 3.11 - Результат виміру LCP

```
> let hiddenTime = document.visibilityState === 'hidden' ? 0 : Infinity;

  document.addEventListener('visibilitychange', (event) => {
     hiddenTime = Math.min(hiddenTime, event.timeStamp);
  }, { once: true });

  new PerformanceObserver(entryList => {
     var fidEntry = entryList.getEntries()[0];
     if (fidEntry.startTime < hiddenTime) {
         var fid = fidEntry.processingStart - fidEntry.startTime;
         console.log("First Input Delay: " + fid);
     }
  }).observe({ type: "first-input", buffered: true });

  try {
     new PerformanceObserver(entryList => {
         var fidEntry = entryList.getEntries()[0];
         var fid = fidEntry.processingStart - fidEntry.startTime;
         console.log("First Input Delay: " + fid);
     })
         // Some browsers throw when 'type' is passed:
         .observe({ type: "first-input", buffered: true });
  }
  catch (e) {
     console.log(e);
  }
< undefined
  First Input Delay: 2.300000000745058                    VM19491:11
```

Рисунок 3.12 - Результат виміру FID

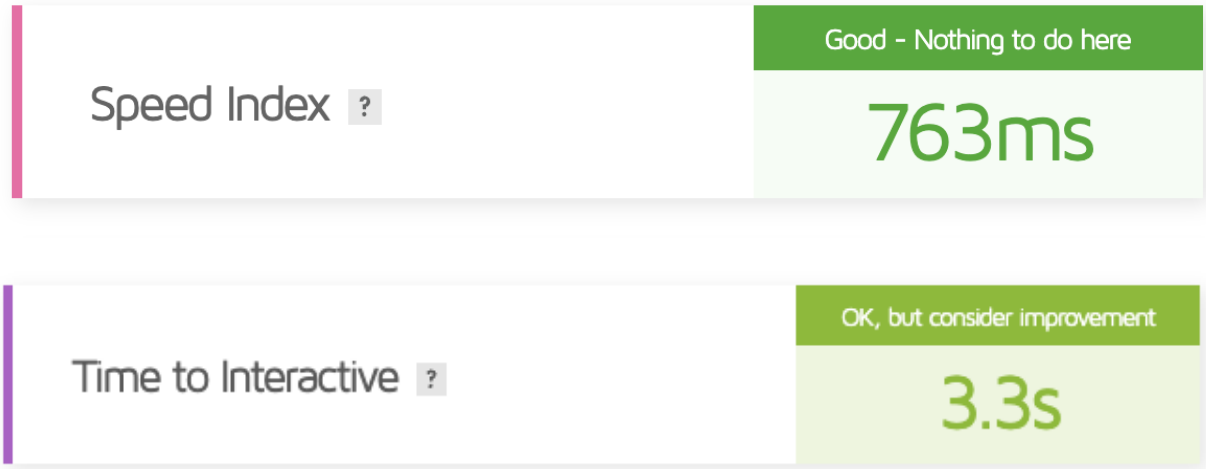


Рисунок 3.13 - Результат виміру SI та TTI

По результатам вимірювань бачимо такі результати:

- CLS = 0.15
- LCP = 1.09 сек
- FID = 2.3 мс
- SI = 763 мс
- TTI = 3.3 сек

Для повної оцінки використаємо Google Page speed calculation не враховуючи First Contentful Paint та Total Blocking Time (рис 3.14, 3.15).

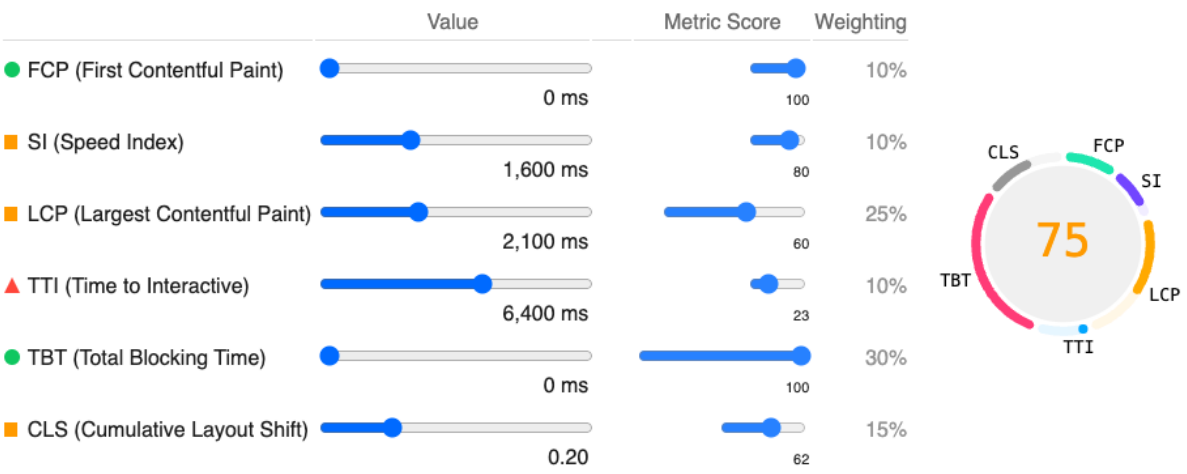


Рисунок 3.14 - Оцінка результатів виміру до оптимізації

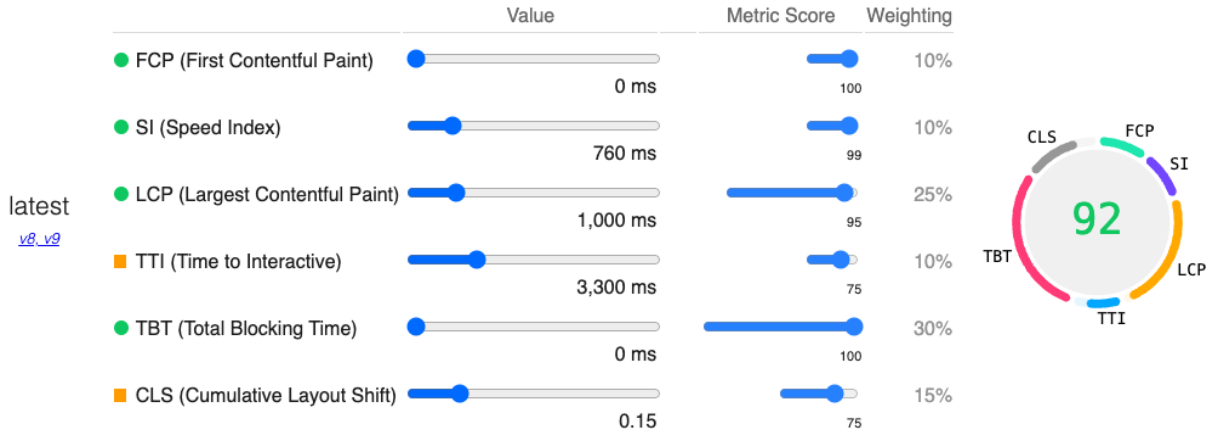


Рисунок 3.15 - Оцінка результатів вимірів після оптимізації

Отже як можемо побачити що при використанні комбінованого методу оптимізації ми маємо різницю у 17% продуктивності веб-сайту. Отже цей метод працює та дає продуктивний результат.

3.3 Висновки до розділу

В ході дослідження вдалося скомбінувати методи оптимізації веб-сайтів. В ході експерименту визначили що такий процесс підвищує продуктивність на 17%. Було перевірено декілька методів по скануванню веб-ресурсів та зроблено висновки які саме методи напряму впливають на швидкість завантаження веб-ресурсу.

ВИСНОВКИ

В ході виконання дипломної роботи був проведений аналітичний огляд методів сканування та проектуваня веб-сервісу сканування веб-ресурсів. Була виконана постановка задачі, зокрема описано необхідний функціонал, передбачені можливі недоліки. Побудована блок-схема алгоритму роботи плагіну сканування веб-ресурсів.

Було проведено тестування роботи системи, наведені дані залежностей. Знайдено універсальні метрики та методи оптимізації. Збільшено точність перевірки інтегрованої метрики FID.

Що стосується майбутньої роботи, планується розширити можливості плагіну інтегруванням функціональності стресового тестування синтетичним трафіком, для отримання розширеного графіку продуктивності. Планується провести дослідження ефективності системи автоматичного розподілення веб-ресурсів на подібні. Оскільки це питання є найбільш актуальним в сучасному світі розробки будь яких веб-сайтів.

5 СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] дослідження компанії Google [Електронний ресурс] – Режим доступу до ресурса: https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/consumer-mobile-brand-content-interaction/

[2] GTmetrix [Електронний ресурс] - Режим доступу до ресурса: https://gtmetrix.com/blog/

[3] Waterfall[Електронний ресурс] - Режим доступу до ресурса: https://gtmetrix.com/blog/how-to-read-a-waterfall-chart-for-beginners/

[4] Loader [Електронний ресурс] - Режим доступу до ресурса: https://falcon.web-automation.ru/blog/provedenie-nagruzochnogo-testirovaniya-cherez-loader-io

[5] Cumulative Layout Shift [Електронний ресурс] - Режим доступу до ресурса:https://defront.ru/posts/2020/06-june/11-optimize-cumulative-layout-shift/#:~:text=Cumulative%20Layout%20Shift%20(CLS)

[6] Event Scheduler [Електронний ресурс] - Режим доступу до ресурса: https://weatherless.ru/javascript/jquery/event-scheduler-jquery/

[7] Про Largest Contentful Paint [Електронний ресурс] - Режим доступу до ресурса: https://rockcontent.com/blog/largest-contentful-paint/#:~:text=Largest%20Contentful%20Paint%20is%20the%20metric%20that%20measures%20the%20time,everything%20that%20appears%20without%20scrolling

[8] Використання Timing-Allow-Origin [Електронний ресурс] - Режим доступу до ресурса: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Timing-Allow-Origin

[9] First Input Delay [Електронний ресурс] - Режим доступу до ресурса: https://web.dev/fid/?gclid=Cj0KCQiAnuGNBhCPARIsACbnLzr-

dIxlca7Wua_fJO_3yXX0EFgJ-q4ItsXVuWap86-CphwGEwjCWFQaAtxXEALw_wcB

[10] робочий процес Nginx [Електронний ресурс] - Режим доступу до ресурса: http://nginx.org/ru/docs/control.html

[11] Налаштування буферів [Електронний ресурс] - Режим доступу до ресурса: https://freexbcodes.com/cat-nginx/nalashtuvannja-nginx-dlja-produktivnosti-ta/

[12] Кешування [Електронний ресурс] - Режим доступу до ресурса: https://highload.today/keshirovanie-s-nginx/

[13] Стиснення Gzip [Електронний ресурс] - Режим доступу до ресурса: https://www.8host.com/blog/povyshenie-proizvoditelnosti-sajta-s-pomoshhyu-gzip-i-nginx/

[14] Time To Last Byte [Електронний ресурс] - Режим доступу до ресурса:

https://www.dareboost.com/en/doc/website-speed-test/metrics/time-to-last-byte-ttlb#:~:text=The%20Time%20to%20Last%20Byte,to%20First%20Byte%20(TTFB).

[15] Динамічний вміст сторінки [Електронний ресурс] - Режим доступу до ресурса: https://armedsoft.com/ua/blog/statychni-ta-dynamichni-web-sayty