УДК 004.051 DOI: 10.31673/2786-8362.2024.026405

Василенко В.В., к.т.н.; **Котелянець В.В.,** к.т.н.;
Дубовицький Д.С., Волуйко І.В.

ЗНИЖЕННЯ ЗАГАЛЬНОГО ЧАСУ БЛОКУВАННЯ (TBT) ДЛЯ ВЕБ-КОМПОНЕНТІВ ЗА ДОПОМОГОЮ РЕНДЕРУ НА БОЦІ СЕРВЕРУ

Vasylenko V.V., Kotelianets V.V., Dubovytskyi D.S., Voluiko I.V. Reduction of total blocking time (TBT) for web components using server-side rendering. The introduction of web components into the W3C web development standards opens up many opportunities for web application developers to create pages without relying on third-party libraries. While the proliferation of the use of web components provides new opportunities for users to dynamically create pages, changing the page using JavaScript at the start of the application negatively affects the display speed and the time before the page becomes interactive. The above properties are among the main ones by which search engines determine the ranking of a page. Also, the use of JavaScript to display the page makes indexing by search engines impossible. Server-side rendering technology is a promising solution to reduce display time and reduce page interactivity time, and also creates conditions for search engine indexing.

Keywords: web, website, rendering, HTML, server-side rendering, SEO, JavaScript, web components, TBT, total block time, closed block hour

Василенко В.В., Котелянець В.В., Дубовицький Д.С., Волуйко І.В. Зниження загального часу блокування (TBT) для веб-компонентів за допомогою рендеру на боці серверу. Впровадження веб-компонентів в стандарти веб-розробки W3C відкриває чимало можливостей для розробників веб-застосунків створювати сторінки не покладаючись на сторонні бібліотеки. В той час як проліферація використання веб-компонентів дає нові можливості для користувачів стосовно динамічного створення сторінок, зміна сторінки за допомогою JavaScript при старті додатку негативно впливає на швидкість відображення та час до інтерактивності сторінки. Вище наведені властивості є одними з основних, за якими пошукові машини визначають рейтинг сторінки. Також використання JavaScript для відображення сторінки унеможливлює індексацію пошуковою машиною. Технологія рендеру на боці серверу є перспективним рішенням зниження часу відображення та зниження часу до інтерактивності сторінки, і також створює умови для індексації пошуковою машиною.

Ключові слова: веб, веб-сайт, рендер, HTML, рендер на боці серверу, SEO, JavaScript, веб-компонент, TBT, total blocking time, загальний час блокування

Вступ

На сьогоднішній день мати веб-сайт для бізнесу або державної установи вважається необхідним. За допомогою веб-сайтів відбувається продаж та покупка товарів, реклама нових продуктів, наукові дослідження, спілкування з друзями та знайомими та перегляд медіа. З поширенням використання мобільних пристроїв для перегляду веб-сайтів, швидкість завантаження стає все більш важливим критерієм визначаючим успіх або провал веб-сайту, оскільки чим швидше завантажується та відображається веб сайт, тим більша вірогідність того що користувач дочекається і почне з ним взаємодіяти замість того щоб покинути його. Також важливим є процес пошуку необхідного веб-сайту за допомогою пошукових машин таких як Google або Bing.

Написання веб-сторінки відбувається за допомогою динамічного створення її компонентів за допомогою JavaScript і подальшого їх компонування. Специфікація веб-компонентів є універсальною та підтримується більшістю браузерів.

Але для відображення веб-сайту побудованого за допомогою цієї технології, окрім завантаження розмітки HTML потребується завантаження і виконання JavaScript. Необхідність завантаження і виконання JavaScript негативно впливає на швидкість відображення сторінки, та додає час, коли сторінка заблокована для взаємодії.

Також ця потреба унеможливлює індексацію сторінки пошуковою машиною, оскільки вони індексують лише вміст HTML, без виконання скрипту веб-сайту.

© Василенко В.В., Котелянець В.В., Дубовицький Д.С., Волуйко І.В. 2024

В цій роботі ми розглянемо технологію, за допомогою якої вміст сторінки створеної на основі веб-компонентів буде доступний відразу після завантаження, таким чином зменшить загальний блокувальний час та зробить сторінку доступною для індексації.

Аналіз останніх досліджень. В зв'язку з розвитком односторінкових веб-додатків, з'являється потреба вирішувати проблеми пов'язані з динамічним створенням вмісту, а конкретно відсутністю статично доступної сторінки. Таким чином, рендер сторінки на боці клієнту негативно впливає на метрики First Contentful Paint[1], Cumulative Layout Shift[2], та Total Blocking Time[3]. Вказані характеристики являють собою синтетичні показники оцінки якості веб-сайту, котрі запровадила корпорація Google. Провідні сторонні бібліотеки створення веб-додатків мають свої систему рендеру на боці серверу, такі як Angular[4] або React[5]. Не дивлячись на те, що веб-компоненти надають функціонал близький до наданого сторонніми бібліотеками [5], на даний момент дослідження стосовно рендеру на боці серверу сторінок, які були створені за допомогою веб-компонентів без використання сторонніх бібліотек є недостатніми.

Постановка завдання. Сторінка створена за допомогою веб-компонентів не матиме наповнення в HTML документі до тих пір поки JavaScript сторінки не буде виконано. Після виконання, сторінка заповнюється динамічно згенерованим вмістом. Таким чином на пристроях з повільним інтернетом сторінка буде пуста до тих пір поки не завантажиться та виконається скрипт, час після завантаження сторінки, до того моменту як вона стає інтерактивною називається тотальний час блокування сторінки. Збільшення часу блокування сторінки негативно впливатиме на позицію веб-сайту в видачі результатів пошуку. Також оскільки більшість пошукових машин не виконують скрипти сторінок які індексують, відповідний веб-сайт не буде індексований та відображений в результатах пошуку відповідних систем.

Для того, щоб поєднати всі переваги динамічно створених веб-сайтів, зменшити тотальний час блокування та зробити можливою індексацію сторінки, пропонується використати підхід, коли перед завантаженням веб-сайту спочатку виконується рендер на боці серверу в емульованому браузерному середовищі, і клієнту віддається вже готовий HTML зі статичним вмістом. Такий підхід називається рендер на боці серверу і є популярним рішенням серед готових бібліотек компонентів таких як React та Angular. Але стандарт веб-компонентів не має подібної підтримки.

В роботі вирішується наукове завдання з розробки методу рендеру веб-компонентів на боці серверу, для того щоб зменшити параметр тотального блокованого часу. Також буде розглянуто емуляцію document object model в серверному середовищі, та досліджено проблеми даного підходу та їх потенційні рішення. Спочатку буде створена тестова сторінка за допомогою веб-компонентів, на основі якої будуть проведені заміри тотального часу блокування. Після успішного створення тестової сторінки, необхідно зробити емуляцію браузера на боці серверу.

Для того, щоб згенерувати сторінку на боці серверу необхідно:
- створити емуляцію браузера на боці серверу;
- сторінка має бути завантажена в пам'ять та її відповідні скрипти мають бути виконані;
- після виконання скриптів сторінки, вміст має бути збережений в емульовані вузли HTML сторінки;
- емульовані вузли HTML сторінки необхідно серіалізувати в HTML документ;
- серіалізований HTML документ має бути повернутий на запит клієнта.

Кінцевим продуктом роботи системи має стати статична HTML сторінка, таким чином щоб при її завантаженні, тотальний час блокування став меншим, ніж відображення сторінки напряму в браузері.

Метою роботи є зниження тотального часу блокування сторінок веб-сайтів створених за допомогою веб-компонентів за допомогою методики рендеру на боці серверу. Основна увага при цьому звертається на вирішення проблем, які можуть виникнути через несумісність скрипів написаного для браузерного середовища під час виконання на сервері.

Виклад основного матеріалу дослідження.

Для відображення веб-сайту використовується мова розмітки HTML. Написання і використання мови розмітки полягає в наступному. Спочатку розробник створює розмітку зі всіма елементами веб-сайту і його вмістом, розміщає сторінку на віддаленому сервері і клієнт за URL-адресою веб-сервера може завантажити сторінку, браузер проводить парсинг мови розмітки в візуальні елементи інтерфейсу.

З часом, вимоги до веб-сайтів стали все вищим, від них вимагається підтримувати відео, анімації, динамічні зміни інтерфейсу, і замість декількох-сторінок веб-сайти стали повноцінним графічним інтерфейсом для складних систем в таких сферах як банківська, телекомунікація, логістика. Для того, щоб ефективно виконувати роль динамічного графічного інтерфейсу в цих предметних областях використовується мова програмування JavaScript, а для побудови складного інтерфейсу використовується компонування сторінок з декількох компонентів.

Компонент в графічній системі це графічний елемент, який має сталий вигляд та сталий функціонал. Компоненти бувають двох видів: лише графічні, і ті які мають виконувати логіку предметної частини. Компоненти створюються за допомогою компонування інших компонентів і написання додаткової логіки для них за необхідності. Такий підхід до компонування дозволяє ефективно перевикористовувати частини вже готових сторінок в розробці інших сторінок, а також сприяє розділенню кодової бази відповідно до частин предметної області. Все це призводить до підвищення швидкості роботи команди розробників, покращення візуальної послідовності графічної системи. логічного розділення та розбиття системи на невеличкі блоки-компоненти.

Для розробки веб-сторінок на основі компонентів часто використовуються вже готові бібліотеки типу React та Angular. Хоча використання сторонніх бібліотек є популярним рішенням для написання веб-сайту, негативними наслідками полягання на сторонню бібліотеку для написання власної системи може стати скорочений вибір спеціалістів які досвідчені в технології вибраного проекту, або знижена продуктивність спеціаліста іншої технології під час його навчання та ознайомлення з технологією, яка була обрана для проекту.

Також значним негативним наслідком є той факт, що використання сторонніх технологій для написання власного проекту робить проект залежним від рішень іншої компанії. Наприклад, оновлення елементів бібліотеки може спричинити потребу в значній інвестиції часу від розробників проекту для того, щоб впевнитись у стабільності роботи проекту з новою версією бібліотеки та усунення помилок несумісності частин проектів які були написані з розрахунком на старішу версію бібліотеки. Також завжди є ризик того, що оригінальний розробник бібліотеки, від якої наш проект залежить, перестане інвестувати в неї час та розвивати її, не оновлюючи до останніх технологій. В такому випадку перед розробниками постає дилема: інвестувати значний час та переписати проект на іншій технології, або дати системі застаріти.

Альтернативою до використання сторонніх бібліотек є використання технології веб-компонентів, яка була включена в стандарт W3C. Оскільки технології включені в стандарт, має сенс очікувати, що більшість розробників вже з ними ознайомлені та мають досвід, нівелюючи потребу в додатковому навчанні членів команди новим технологіям.

Веб компонент створюється за допомогою поєднання трьох технологій: Custom Element, Shadow Dom та HTML шаблонів. Саме використання цих технологіє дає можливість створити повноцінний веб-компонент.

Для того щоб під час рендеру HTML тегу браузер знав, як він має виглядати, тег має бути або стандартним тегом HTML, або зареєстрованим Custom Element. Для реєстрації нового елементу в середовищі браузера потрібно скористатись скриптом на JavaScript та зареєструвати компонент в глобальному об'єкті customElements за допомогою методу define. Метод define приймає два аргументи: перший це назва нового тегу, другий це класс нового компоненту. Якщо інорідний елемент не було зарєстровано, браузер або не буде його

відображати, а відобразить лише його вміст при наявності. Друга частина веб-компонента це використання технології Shadow Dom.

Для стилізації елементів HTML використовується мова CSS, за її допомогою прописуються візуальні властивості елементів такі як колір, розмір тексту, відступи та інше. Будь який стиль складається з двух частин: selector та самі властивості стилю. В частині selector розробник має прописати умови яким мають виконатись для елемента, в разі успішного виконання умови стиль буде застосовано до компонента. Проблема з цим підходом виникає тоді, коли в під час розробки додатку ми намагаємось застосувати компонент який має власні стилі.

Згідно опису роботи CSS, всі компоненти які попадають під умови, будуть залучені під застосування стилю. Тобто може виникнути колізія між стилями веб-сайту і стилями компоненту, і в такому випадку або компонент буде виглядати не так як має, або частина сторінки матиме застосований до неї стиль з компоненту. Щоб уникнути колізій стилів, використовується технологія Shadow Dom. За допомогою Shadow Dom розробник може створити відокремлене HTML дерево елементів, стилі в середині якого не будуть впливати на загальний глобальний стиль і навпаки, стилі з глобального CSS файлу не будуть впливати на зовнішній вигляд елементів в середині Shadow Dom. Ця технологія дозволяє створювати ізольовані під-дерева з власними стилями без впливу загальних стилів, таким чином, що в незалежності від контексту, компонент буде виглядати коректно.

Третьою частиною веб-компонента є використання HTML шаблону, елемент template. Елемент template дозволяє створити елемент сторінки який зберігається в пам'яті та не відображається на сторінці. Під час відображення веб-компоненту, замість того щоб браузер виконував парсинг його шаблону, шаблон перетворюється на елемент всього один раз, і кожне відтворення компоненту на сторінці копіює шаблон, що є швидшою операцією ніж парсинг HTML.

Таким чином, за допомогою використання трьох технологій: Custom Elements, Shadow DOM та HTML Template можливе створення веб-сайтів на основі компонентів без використання сторонніх бібліотек. Але оскільки для відтворення веб-компоненту на сторінці треба виконати JavaScript, сайт не підлягає індексації пошуковими машинами, а також залишатиметься порожнім поки JavaScript завантажується та виконується. Для того щоб вирішити ці проблеми, а конкретно щоб сайт міг бути проіндексований пошуковою машиною, а також відображений відразу після завантаження сторінки без додаткових кроків, використовується технологія server side rendering.

Server side rendering це метод відтворення сторінки та її JavaScript в серверному середовищі. Після відпрацювання JavaScript, всі елементи які були створені та додані до віртуальної сторінки серіалізуються в HTML та повертаються клієнту який зробив запит. Таким чином веб-сайт може бути написаний з використанням JavaScript для відображення, а клієнт одразу отримає повну сторінку. Також факт того що сторінка віддається відразу повною дозволяє пошуковим машинам її індексувати. Готові рішення для виконання рендеру на боці серверу є популярними серед бібліотек таких як React та Angular, натомість для веб-компонентів подібний функціонал поки що не підтримується. Оскільки веб-компонент є єдиним способом будування веб-сайту на основі компонентів без застосування сторонніх бібліотек, питання створення технології, яка дозволила б рендерити сайти створені на них в серверному оточенні, таким чином отримавши всі плюси даного підходу.

Під час індексації веб-сайту, пошукова машина оцінює веб-сайт відповідно до декількох параметрів з декількох категорій. Категорії оцінок є наступні: продуктивність, доступність, найкращі практики та оптимізація під пошукову машину. Доступність, найкращі практики та оптимізація можуть бути досягнені за допомогою змін в вмісті веб-сайту. Але на показники категорії продуктивність впливає не тільки вміст сайту, а й технологія за допомогою якої він створений. Оскільки показники даної категорії залежать від використаної для створення веб-сайту технології, в цій роботі буде створено метод їй покращення.

Категорія продуктивність складається з наступних метрик:

- First Contentful Paint (FCP), час від завантаження сторінки до відображення її вмісту;
- Largest Contentful Paint (LCP), час від ініціювання завантаження сторінки до того, як найбільший її елемент намальовано;
- Total Blocking Time (TBT), час від завантаження сторінки, до того як вона стає інтерактивною;
- Cumulative Layout Shift (CLS), кількість та дистанція переміщень елементів сторінки під час завантаження;
- Speed Index (SI), індекс швидкості відображення вмісту під час завантаження.

З наведених вище метрик оцінки веб-сайту пошуковою машиною, в цій роботі зосередимося на First Contentful Paint та Total Blocking Time, оскільки вони разом складають час, за який сторінка буде відображена та стане інтерактивною, таким чином що $T_{total} = T_{FCP} + T_{TBT}$.

Для проведення замірів вище перелічених параметрів, буде використана система Google Lighthouse[6]. Google Lighthouse була обраною оскільки вона є поширеною в галузі, має відкритий вихідний код та є вбудованою в браузер Chrome. Заміри відбуватимуться на пристрої з процесором Intel Core I7, тактова частота якого 2.6ГГц. Браузером для тестування обрано Google Chrome, версія 130 для операційної системи MacOS.

Для проведення замірів буде використано тестову сторінку. Тестова сторінка створена за допомогою технології веб-компонентів, складається з заголовку, списку, і елементів списку. Кількість елементів списку є варіативною. Елементи списку створюються браузером динамічно в циклі, як показано на Рис. 1. На сторінці буде відображено 60 елементів списку.

```
connectedCallback() : void {
    super.connectedCallback();
    const listItem : number = 60;
    for (let i : number = 0; i < listItem; i++) {
        const innerHtml : string = this.getById( id: 'listContainer').innerHTML
        this.getById( id: 'listContainer').innerHTML = innerHtml
            + `<app-list-item title="List item ${i}" subtitle="List item ${i} description"> </app-list-item>\n`
    }
}
```

Рис. 1. Створення елементів списку в циклі

На основі тестової сторінки, використаємо систему Google Lighthouse для проведення замірів. Результати замірів показані на Рис. 2.

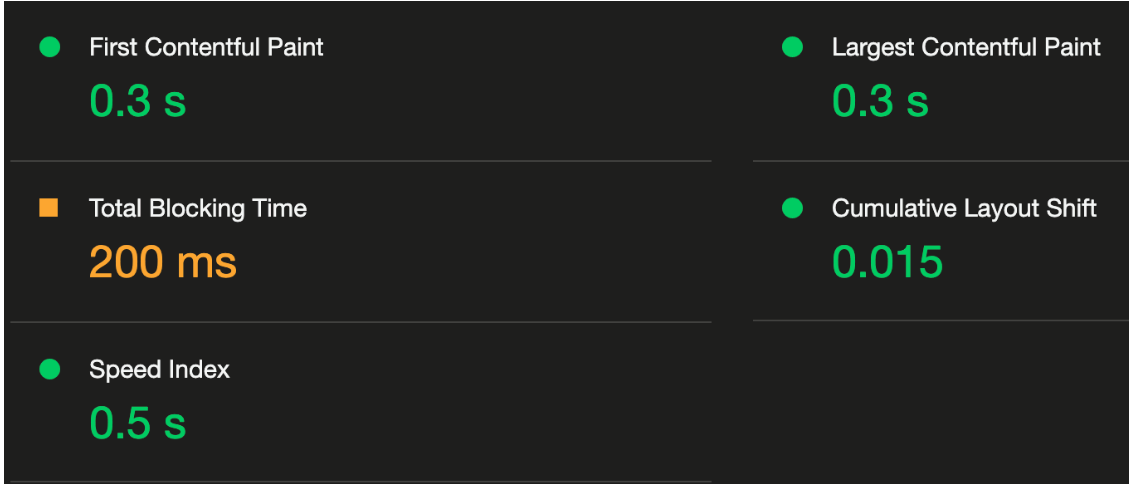

Рис. 2. Результати замірів тестової сторінки

В результаті проведених замірів, такі параметри як First Contentful Paint, Largest Contentful Paint, Cumulative Layout Shit та Speed Index проходять відповідний до параметру поріг, та мають задовільний результат. Однак в результаті замірів, було виявлено що параметр Total

Blocking Time потребує покращення, оскільки результат в 200 мілісекунд не є прийнятним та погіршує загальний результат всієї категорії продуктивність.

Тому, необхідно розробити метод обробки веб-сайту таким чином, щоб після завантаження його Total Blocking Time становив менше 150 мілісекунд. Метод зручно представити у вигляді алгоритму.

На першому кроці, необхідно створити емулятор середовища веб-браузера, а саме: класс HTMLElement, реєстр компонентів customElements, та document, корінь віртуального документа. Для реєстрації компонентів в customElements необхідно реалізувати емуляцію методу define, а тег і класс передані як аргументи зберегти у структурі даних Map. Після того як є успішна емуляція регістру власних елементів, необхідно реалізувати емуляцію класу HTMLElement. Для мінімально робочого прикладу потрібно реалізувати такі методи як: getElementById оскільки в прикладі кожен елемент списку знаходить параграф по id і змінює його наповнення.

Також, оскільки веб-компонент потребує використання shadow dom, потрібно додати емуляцію цієї технології. А конкретно, метод attachShadow. Під час рендеру, shadow dom буде створений за допомогою спеціального HTML елементу template з атрибутом shadowrootmode виставленим у значення "open". Крім того, в клас HTMLElement потрібно додати підтримку cloneNode, оскільки HTML елемент template використовується в основному через копіювання внутрішньої структури. Емуляція атрибутів в клас має бути додана також за допомогою структури Map, де будуть зберігатись атрибути елементу, ключем буде назва атрибуту, а значенням поточна інформація атрибуту.

Другим кроком має стати завантаження HTML, прикріплених скриптів і їх виконання. Оскільки рендеринг відбувається на боці серверу і система має прямий доступ до файлу, буде прочитано кореневий HTML файл напряму з файлової системи.

Коли файл завантажено, в ньому знайдемо підключені до нього скрипти за допомогою наступного регулярного виразу /(<script.*src=".)(.*)(?=")/. Після того як отримано масив шляхів до скриптів HTML файла, потрібно завантажити їх, для чого використовується метод import. Метод import завантажує скрипт за його шляхом виконує його та також завантажує скрипти від якого поточний залежний і виконує їх також. Після виконання всіх скриптів система має перейти в стан, в якому всі веб-компоненти зареєстровані в емуляторі customElements, і в Map зберігається назва власного елементу і відповідний йому клас.

Третім кроком алгоритму має стати заміна тегу веб-компонента на його вміст. Коли в результаті виконання другого кроку відповідність між назвою тегу і його класом встановлена, потрібно знайти в тексті всі теги використані для поточного HTML, для кожного тега, який збережено в customElements, створити відповідний компонент, оновити його та виконати рендер компонента, отримавши DOM-дерево зі стандартним HTML елементів. Після чого, замінити тег створене DOM-дерево. Цей процес необхідно повторити для кожного компоненту.

Оскільки сторінки будують за принципом композиції компонентів, кожен компонент може мати в собі інший компонент і він в собі теж може мати наступний, тож пошук тегів, створення компоненту і заміна тегу на результат повноцінного рендеру компонента має відбуватись рекурсивно.

Так як традиційний метод створення Shadow Dom використовується імперативно за допомогою JavaScript, а результат має бути статична сторінка відображена без вживання JavaScript ми маємо використати декларативну версію створення Shadow Dom, а точніше використати тег template з атрибутом shadowrootmode в "open". Заміна має відбутись таким чином, що кожен компонент який створив Shadow Root за допомогою методу attachShadow, має відмічатися як такий, що має Shadow Root і під час рендеру цього компоненту весь його вміст має знаходитись в середені тегу template з відповідним атрибутом.

Таким чином під час відображення в бразуері клієнта HTML сторінки вміст компонентів буде захищений від впливу сторонніх стилів навіть без використання JavaScript. Таким чином, після створення відповідної середи і виконання вище наведених операцій веб-сайти які вже

були створені за допомогою веб-компонентів можуть використати технологію рендеру на боці серверу без будь-якої інвестиції з боку розробників в зміну та адаптацію коду веб-сторінки.

Емуляція елементів браузера має бути достатньою для того, щоб покрити всі випадки звернення до браузера і змінювати стан емуляції сторінки відповідно до того, що мало б статися на реальній сторінці. Єдиною різницею між операціями над елементами які було емульовано та реальним браузером, це те що зміна елемента в браузері має оновити стан компонента і його візуально відображення, коли на боці серверу єдина зміна яка відбувається це зміна даних.

Оскільки для стандартного відображення веб-сайту, браузеру потрібно виконати завантаження, парсинг, рендер та скрипти веб-сайту, таким чином що $T_{default} = T_{load} + T_{parse} + T_{render} + T_{execute}$, де $T_{default}$ є загальним часом потрібним для відображення веб-сайту без застосування запропонованого методу обробки. В результаті обробки веб-сайту за допомогою системи, крок виконання скриптів був завчасно виконаний на сервері, і відповідно не буде виконаний в браузері клієнту, тож час відображення сайту, не матиме останнього кроку, тобто тепер загальний час відображення веб-сайту становитиме $T_{optimized=T_{load}+T_{parse}+T_{render}}$.

Перевіримо систему за допомогою системи тестування Google Lighthouse. Після успішної обробки проведемо заміри на сторінці обробленій системою, результати замірів показані на Рис. 4.

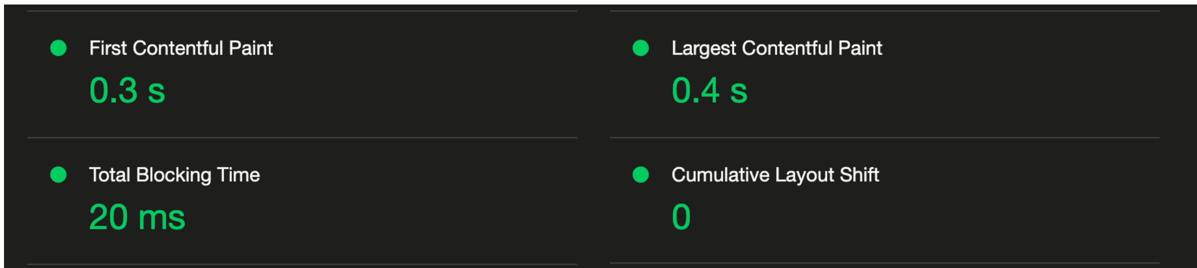

Рис. 4. Результати замірів статично згенерованої сторінки

Відповідно до замірів, тотальний час блокування було знижено з 200 мілісекунд до 20. Це було досягнуто за допомогою перенесення виконання JavaScript на бік серверу, сторінка віддана клієнту в готовому вигляді, виконання JavaScript не потребується, тобто сторінка доступна відразу після завантаження без додаткових дій, чим і є обумовлений низький час блокування.

Висновки

Проведено аналіз продуктивності відображення сторінок зроблених на основі технології веб-компонентів, в результаті аналізу було виявлено, що час виконання скриптів сторінки має значний негативний вплив на параметр Total Blocking Time.

Розроблено метод виконання та рендеру веб-сайту на боці серверу. Оскільки скрипти необхідні для відображення виконуються на сервері, браузеру клієнта віддається вже готовий HTML, загальний час відображення сторінки знижується з $T_{default} = T_{load} + T_{parse} + T_{render} + T_{execute}$ до $T_{optimized=T_{load}+T_{parse}+T_{render}}$.

Даний метод дав змогу завчасно виконати скрипти необхідні для відображення сторінки на сервері, таким чином, що виконання скриптів на боці клієнту перестало бути необхідним. В результаті замірів проведених на тестовій сторінці, до обробки системою і після було продемонстровано покращення параметру Total Blocking Time з 200 мілісекунд до 20.

Подальші дослідження можуть зосередитися на оптимізації JavaScript для інтерактивності сторінок після завантаження, що дозволить поєднати швидке відображення з високою продуктивністю.

Список використаної літератури:

1. First Contentful Paint | Lighthouse | Chrome for Developers. URL: https://developer.chrome.com/docs/lighthouse/performance/first-contentful-paint
2. Cumulative Layout Shift (CLS). URL: https://web.dev/articles/cls
3. Total Blocking Time (TBT). URL: https://web.dev/articles/tbt
4. Angular - Server-side render. URL: angular.io/guide/ssr
5. React Server Components. URL: https://react.dev/reference/rsc/server-components
6. Strazzullo, F. (2023). Web Components. In: Frameworkless Front-End Development. Apress, Berkeley, CA. URL: https://doi.org/10.1007/978-1-4842-9351-5_5

Автори статті

Василенко Володимир – кандидат технічних наук, доцент, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.
ORCID: 0000-0001-8465-6178
Котелянець Віталій – кандидат технічних наук, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.
ORCID: 0009-0000-1874-8322
Дубовицький Денис – аспірант, Державний університет інформаційно-комунікаційних технологій, Київ, Україна.
ORCID: 0009-0004-2830-9197
Волуйко Ігор – студент, Державний університет інформаційно-комунікаційних технологій Київ, Україна.
ORCID: 0009-0003-3035-2840

Authors of the article

Vasylenko Volodymyr – Candidate of Science (technic), Associate Professor, State University of Information and Communication Technology, Kyiv, Ukraine.
ORCID: 0000-0001-8465-6178
Vitalii Kotelianets – Candidate of Science (technic), State University of Information and Communication Technology, Kyiv, Ukraine.
ORCID: 0009-0000-1874-8322
Dubovytskyi Denys – postgraduate, State University of Information and Communication Technology, Kyiv, Ukraine.
ORCID: 0009-0004-2830-9197
Voluiko Ihor – student, State University of Information and Communication Technologies, Kyiv, Ukraine.
ORCID: 0009-0003-3035-2840